J2ME CLDC API

1.1

Connected Limited Device Configuration (CLDC) Specification (“Specification”)
Version: 1.1
Status: FCS
Release: March 4, 2003

NOTICE; LIMITED LICENSE GRANTS

Sun Microsystems, Inc. ("Sun") hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under the Sun's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation, which shall be understood to include developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

Sun also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or patent rights it may have in the Specification to create and / or distribute an Independent Implementation of the Specification that: (i) fully implements the Spec(s) including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required / authorized by the Specification or Specifications being implemented; and (iii) passes the TCK (including satisfying the requirements of the applicable TCK Users Guide) for such Specification. The foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose.

You need not include limitations (i)-(iii) from the previous paragraph or any other particular "pass through" requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to implementations of the Specification (and products derived from them) that satisfy limitations (i)-(iii) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Sun's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Spec in question.

For the purposes of this Agreement: "*Independent Implementation*" shall mean an implementation of the Specification that neither derives from any of Sun's source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun's source code or binary code materials; and "*Licensor Name Space*" shall mean the public class or interface declarations whose names begin with "java", "javax", "com.sun" or their equivalents in any subsequent naming convention adopted by Sun through the Java Community Process, or any recognized successors or replacements thereof.

This Agreement will terminate immediately without notice from Sun if you fail to comply with any material provision of or act outside the scope of the licenses granted above.

TRADEMARKS
No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, J2ME, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

THE SPECIFICATION IS PROVIDED "AS IS". SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND
If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT
You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification (“Feedback”). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.
(LFI#123888/Form ID#011801)

Contents

java.io 1

ByteArrayInputStream 3
ByteArrayOutputStream 8
DataInput 12
DataInputStream 19
DataOutput 29
DataOutputStream 35
EOFException 42
InputStream 44
InputStreamReader 50
InterruptedIOException 54
IOException 56
OutputStream 58
OutputStreamWriter 61
PrintStream 65
Reader 73
UnsupportedEncodingException 78
UTFDataFormatException 80
Writer 82

java.lang 87

ArithmeticException 90
ArrayIndexOutOfBoundsException 92
ArrayStoreException 94
Boolean 96
Byte 99
Character 102
Class 107
ClassCastException 112
ClassNotFoundException 114
Double 116
Error 125
Exception 127
Float 129
IllegalAccessException 138
IllegalArgumentException 140
IllegalMonitorStateException 142
IllegalThreadStateException 144
IndexOutOfBoundsException 146
InstantiationException 148
Integer 150
InterruptedException 158
Long 160
Math 166
NegativeArraySizeException 174
NoClassDefFoundError 176
NullPointerException 178
NumberFormatException 180
Object 182
OutOfMemoryError 188

Runnable 190
Runtime 191
RuntimeException 194
SecurityException 196
Short 198
String 201
StringBuffer 220
StringIndexOutOfBoundsException 234
System 236
Thread 241
Throwable 247
VirtualMachineError 250

java.util 253

Calendar 254
Date 265
EmptyStackException 269
Enumeration 271
Hashtable 273
NoSuchElementException 279
Random 281
Stack 287
TimeZone 290
Vector 293

javax.microedition.io 303

Connection 304
ConnectionNotFoundException 305
Connector 307
ContentConnection 312
Datagram 314
DatagramConnection 319
InputConnection 324
OutputConnection 326
StreamConnection 328
StreamConnectionNotifier 329

Constant Field Values 331

CHAPTER 1

Package java.io

Description

Provides classes for input and output through data streams.

Since: CLDC 1.0

Class Summary	
Interfaces	
DataInput	The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.
DataOutput	The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.
Classes	
ByteArrayInputStream	A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.
DataOutputStream	A data output stream lets an application write primitive Java data types to an output stream in a portable way.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.
InputStreamReader	An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters.
OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes.
OutputStreamWriter	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are translated into bytes.
PrintStream	A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
Reader	Abstract class for reading character streams.
Writer	Abstract class for writing to character streams.
Exceptions	
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.

Class Summary	
`InterruptedIOException`	Signals that an I/O operation has been interrupted.
`IOException`	Signals that an I/O exception of some sort has occurred.
`UnsupportedEncodingException`	The Character Encoding is not supported.
`UTFDataFormatException`	Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface.

java.io
ByteArrayInputStream

Declaration

```
public class ByteArrayInputStream extends InputStream
```

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--java.io.ByteArrayInputStream
```

Description

A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the `read` method.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

protected byte[]	buf
protected int	count
protected int	mark
protected int	pos

Constructors

	ByteArrayInputStream(byte[] buf)
	ByteArrayInputStream(byte[] buf, int offset, int length)

Methods

int	available()
void	close()
void	mark(int readAheadLimit)
boolean	markSupported()
int	read()
int	read(byte[] b, int off, int len)
void	reset()
long	skip(long n)

Inherited Member Summary

Methods inherited from class `InputStream`

`read(byte[])`

Methods inherited from class `Object`

Inherited Member Summary
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

buf

Declaration:
`protected byte[] buf`

Description:
An array of bytes that was provided by the creator of the stream. Elements `buf[0]` through `buf[count-1]` are the only bytes that can ever be read from the stream; element `buf[pos]` is the next byte to be read.

pos

Declaration:
`protected int pos`

Description:
The index of the next character to read from the input stream buffer. This value should always be nonnegative and not larger than the value of `count`. The next byte to be read from the input stream buffer will be `buf[pos]`.

mark

Declaration:
`protected int mark`

Description:
The currently marked position in the stream. ByteArrayInputStream objects are marked at position zero by default when constructed. They may be marked at another position within the buffer by the `mark()` method. The current buffer position is set to this point by the `reset()` method.

Since: JDK1.1

count

Declaration:
`protected int count`

Description:
The index one greater than the last valid character in the input stream buffer. This value should always be nonnegative and not larger than the length of `buf`. It is one greater than the position of the last byte within `buf` that can ever be read from the input stream buffer.

Constructors

ByteArrayInputStream(byte[])

Declaration:
```
public ByteArrayInputStream(byte[] buf)
```

Description:
Creates a `ByteArrayInputStream` so that it uses `buf` as its buffer array. The buffer array is not copied. The initial value of `pos` is `0` and the initial value of `count` is the length of `buf`.

Parameters:
`buf` - the input buffer.

ByteArrayInputStream(byte[], int, int)

Declaration:
```
public ByteArrayInputStream(byte[] buf, int offset, int length)
```

Description:
Creates `ByteArrayInputStream` that uses `buf` as its buffer array. The initial value of `pos` is `offset` and the initial value of `count` is `offset+length`. The buffer array is not copied.

Note that if bytes are simply read from the resulting input stream, elements `buf[pos]` through `buf[pos+len-1]` will be read; however, if a `reset` operation is performed, then bytes `buf[0]` through `buf[pos-1]` will then become available for input.

Parameters:
`buf` - the input buffer.

`offset` - the offset in the buffer of the first byte to read.

`length` - the maximum number of bytes to read from the buffer.

Methods

read()

Declaration:
```
public int read()
```

Description:
Reads the next byte of data from this input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned.

This `read` method cannot block.

Overrides: `read` in class `InputStream`

Returns: the next byte of data, or `-1` if the end of the stream has been reached.

read(byte[], int, int)

Declaration:
```
public int read(byte[] b, int off, int len)
```

Description:
Reads up to `len` bytes of data into an array of bytes from this input stream. If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the number `k` of bytes read is equal to the smaller of `len` and `count-pos`. If `k` is positive, then bytes `buf[pos]` through `buf[pos+k-1]` are copied into `b[off]` through `b[off+k-1]` in the manner performed by `System.arraycopy`. The value `k` is added into `pos` and `k` is returned.

This `read` method cannot block.

Overrides: `read` in class `InputStream`

Parameters:
`b` - the buffer into which the data is read.

`off` - the start offset of the data.

`len` - the maximum number of bytes read.

Returns: the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

skip(long)

Declaration:
```
public long skip(long n)
```

Description:
Skips `n` bytes of input from this input stream. Fewer bytes might be skipped if the end of the input stream is reached. The actual number `k` of bytes to be skipped is equal to the smaller of `n` and `count-pos`. The value `k` is added into `pos` and `k` is returned.

Overrides: `skip` in class `InputStream`

Parameters:
`n` - the number of bytes to be skipped.

Returns: the actual number of bytes skipped.

available()

Declaration:
```
public int available()
```

Description:
Returns the number of bytes that can be read from this input stream without blocking. The value returned is `count - pos`, which is the number of bytes remaining to be read from the input buffer.

Overrides: `available` in class `InputStream`

Returns: the number of bytes that can be read from the input stream without blocking.

markSupported()

Declaration:
```
public boolean markSupported()
```

Description:
Tests if ByteArrayInputStream supports mark/reset.

Overrides: `markSupported` in class `InputStream`

Returns: `true` if this true type supports the mark and reset method; `false` otherwise.

Since: JDK1.1

mark(int)

Declaration:
```
public void mark(int readAheadLimit)
```

Description:
Set the current marked position in the stream. ByteArrayInputStream objects are marked at position zero by default when constructed. They may be marked at another position within the buffer by this method.

Overrides: `mark` in class `InputStream`

Parameters:
`readlimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

Since: JDK1.1

reset()

Declaration:
```
public void reset()
```

Description:
Resets the buffer to the marked position. The marked position is the beginning unless another position was marked. The value of `pos` is set to 0.

Overrides: `reset` in class `InputStream`

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Closes this input stream and releases any system resources associated with the stream.

Overrides: `close` in class `InputStream`

Throws:
`IOException` - if an I/O error occurs.

java.io
ByteArrayOutputStream

Declaration

```
public class ByteArrayOutputStream extends OutputStream
```

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.ByteArrayOutputStream
```

Description

This class implements an output stream in which the data is written into a byte array. The buffer automatically grows as data is written to it. The data can be retrieved using `toByteArray()` and `toString()`.

Since: JDK1.0, CLDC 1.0

Member Summary	
Fields	
protected byte[]	buf
protected int	count
Constructors	
	ByteArrayOutputStream()
	ByteArrayOutputStream(int size)
Methods	
void	close()
void	reset()
int	size()
byte[]	toByteArray()
java.lang.String	toString()
void	write(byte[] b, int off, int len)
void	write(int b)

Inherited Member Summary
Methods inherited from class `Object`
equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()
Methods inherited from class `OutputStream`
flush(), write(byte[])

Fields

buf

Declaration:
```
protected byte[] buf
```

Description:
The buffer where data is stored.

count

Declaration:
```
protected int count
```

Description:
The number of valid bytes in the buffer.

Constructors

ByteArrayOutputStream()

Declaration:
```
public ByteArrayOutputStream()
```

Description:
Creates a new byte array output stream. The buffer capacity is initially 32 bytes, though its size increases if necessary.

ByteArrayOutputStream(int)

Declaration:
```
public ByteArrayOutputStream(int size)
```

Description:
Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

Parameters:
`size` - the initial size.

Throws:
`java.lang.IllegalArgumentException` - if size is negative.

Methods

write(int)

Declaration:
```
public void write(int b)
```

Description:
Writes the specified byte to this byte array output stream.

Overrides: `write` in class `OutputStream`

write(byte[], int, int)

Parameters:

`b` - the byte to be written.

write(byte[], int, int)

Declaration:

```
public void write(byte[] b, int off, int len)
```

Description:

Writes `len` bytes from the specified byte array starting at offset `off` to this byte array output stream.

Overrides: `write` in class `OutputStream`

Parameters:

`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

reset()

Declaration:

```
public void reset()
```

Description:

Resets the `count` field of this byte array output stream to zero, so that all currently accumulated output in the output stream is discarded. The output stream can be used again, reusing the already allocated buffer space.

See Also: `ByteArrayInputStream.count`

toByteArray()

Declaration:

```
public byte[] toByteArray()
```

Description:

Creates a newly allocated byte array. Its size is the current size of this output stream and the valid contents of the buffer have been copied into it.

Returns: the current contents of this output stream, as a byte array.

See Also: `size()`

size()

Declaration:

```
public int size()
```

Description:

Returns the current size of the buffer.

Returns: the value of the `count` field, which is the number of valid bytes in this output stream.

See Also: `count`

toString()

Declaration:

```
public java.lang.String toString()
```

Description:
Converts the buffer's contents into a string, translating bytes into characters according to the platform's default character encoding.

Overrides: `toString` in class `Object`

Returns: String translated from the buffer's contents.

Since: JDK1.1

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Closes this output stream and releases any system resources associated with this stream. A closed stream cannot perform output operations and cannot be reopened.

Overrides: `close` in class `OutputStream`

Throws:
`IOException` - if an I/O error occurs.

java.io
DataInput

Declaration
`public interface` **`DataInput`**

All Known Subinterfaces: `javax.microedition.io.Datagram`

All Known Implementing Classes: `DataInputStream`

Description
The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types. There is also a facility for reconstructing a `String` from data in Java modified UTF-8 format.

It is generally true of all the reading routines in this interface that if end of file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end of file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the input stream has been closed.

Since: JDK1.0, CLDC 1.0

See Also: `DataInputStream`, `DataOutput`

Member Summary	
Methods	
`boolean`	`readBoolean()`
`byte`	`readByte()`
`char`	`readChar()`
`double`	`readDouble()`
`float`	`readFloat()`
`void`	`readFully(byte[] b)`
`void`	`readFully(byte[] b, int off, int len)`
`int`	`readInt()`
`long`	`readLong()`
`short`	`readShort()`
`int`	`readUnsignedByte()`
`int`	`readUnsignedShort()`
`java.lang.String`	`readUTF()`
`int`	`skipBytes(int n)`

Methods

readFully(byte[])

Declaration:

```
public void readFully(byte[] b)
            throws IOException
```

Description:

Reads some bytes from an input stream and stores them into the buffer array `b`. The number of bytes read is equal to the length of `b`.

This method blocks until one of the following conditions occurs:

- `b.length` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If `b.length` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. If an exception is thrown from this method, then it may be that some but not all bytes of `b` have been updated with data from the input stream.

Parameters:

`b` - the buffer into which the data is read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readFully(byte[], int, int)

Declaration:

```
public void readFully(byte[] b, int off, int len)
            throws IOException
```

Description:

Reads `len` bytes from an input stream.

This method blocks until one of the following conditions occurs:

- `len` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown. If `len` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`.

Parameters:

`b` - the buffer into which the data is read.

`off` - an int specifying the offset into the data.

`len` - an int specifying the number of bytes to read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

skipBytes(int)

Declaration:

```
public int skipBytes(int n)
            throws IOException
```

Description:

Makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes. However, it may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned.

Parameters:

`n` - the number of bytes to be skipped.

Returns: the actual number of bytes skipped.

Throws:

`IOException` - if an I/O error occurs.

readBoolean()

Declaration:

```
public boolean readBoolean()
            throws IOException
```

Description:

Reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero. This method is suitable for reading the byte written by the `writeBoolean` method of interface `DataOutput`.

Returns: the `boolean` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readByte()

Declaration:

```
public byte readByte()
            throws IOException
```

Description:

Reads and returns one input byte. The byte is treated as a signed value in the range `-128` through `127`, inclusive. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput`.

Returns: the 8-bit value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readUnsignedByte()

Declaration:

```
public int readUnsignedByte()
            throws IOException
```

Description:

Reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range `0` through `255`. This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput` if the argument to `writeByte` was intended to be a value in the range `0` through `255`.

Returns: the unsigned 8-bit value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readShort()

Declaration:

```
public short readShort()
            throws IOException
```

Description:

Reads two input bytes and returns a `short` value. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(short)((a << 8) | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput`.

Returns: the 16-bit value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readUnsignedShort()

Declaration:

```
public int readUnsignedShort()
            throws IOException
```

Description:

Reads two input bytes, zero-extends it to type `int`, and returns an `int` value in the range `0` through `65535`. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(((a & 0xff) << 8) | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput` if the argument to `writeShort` was intended to be a value in the range `0` through `65535`.

Returns: the unsigned 16-bit value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readChar()

Declaration:

```
public char readChar()
            throws IOException
```

Description:

Reads an input `char` and returns the `char` value. A Unicode `char` is made up of two bytes. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(char)((a << 8) | (b & 0xff))
```

This method is suitable for reading bytes written by the `writeChar` method of interface `DataOutput`.

Returns: the Unicode `char` read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readInt()

Declaration:

```
public int readInt()
            throws IOException
```

Description:

Reads four input bytes and returns an `int` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, and `d` be the fourth byte. The value returned is:

```
 (((a & 0xff) << 24) | ((b & 0xff) << 16) |
 &#32;((c & 0xff) << 8) | (d & 0xff))
```

This method is suitable for reading bytes written by the `writeInt` method of interface `DataOutput`.

Returns: the `int` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readLong()

Declaration:

```
public long readLong()
            throws IOException
```

Description:

Reads eight input bytes and returns a `long` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, `d` be the fourth byte, `e` be the fifth byte, `f` be the sixth byte, `g` be the seventh byte, and `h` be the eighth byte. The value returned is:

```
 (((long)(a & 0xff) << 56) |
  ((long)(b & 0xff) << 48) |
  ((long)(c & 0xff) << 40) |
  ((long)(d & 0xff) << 32) |
  ((long)(e & 0xff) << 24) |
  ((long)(f & 0xff) << 16) |
  ((long)(g & 0xff) <<  8) |
  ((long)(h & 0xff)))
```

This method is suitable for reading bytes written by the `writeLong` method of interface `DataOutput`.

Returns: the `long` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readFloat()

Declaration:

```
public float readFloat()
             throws IOException
```

Description:

Reads four input bytes and returns a `float` value. It does this by first constructing an `int` value in exactly the manner of the `readInt` method, then converting this `int` value to a `float` in exactly the manner of the method `Float.intBitsToFloat`. This method is suitable for reading bytes written by the `writeFloat` method of interface `DataOutput`.

Returns: the `float` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

readDouble()

Declaration:

```
public double readDouble()
             throws IOException
```

Description:

Reads eight input bytes and returns a `double` value. It does this by first constructing a `long` value in exactly the manner of the `readlong` method, then converting this `long` value to a `double` in exactly the manner of the method `Double.longBitsToDouble`. This method is suitable for reading bytes written by the `writeDouble` method of interface `DataOutput`.

Returns: the `double` value read.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

readUTF()

Declaration:

```
public java.lang.String readUTF()
            throws IOException
```

Description:

Reads in a string that has been encoded using a modified UTF-8 format. The general contract of `readUTF` is that it reads a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`.

First, two bytes are read and used to construct an unsigned 16-bit integer in exactly the manner of the `readUnsignedShort` method . This integer value is called the *UTF length* and specifies the number of additional bytes to be read. These bytes are then converted to characters by considering them in groups. The length of each group is computed from the value of the first byte of the group. The byte following a group, if any, is the first byte of the next group.

If the first byte of a group matches the bit pattern `0xxxxxxx` (where `x` means "may be `0` or `1`"), then the group consists of just that byte. The byte is zero-extended to form a character.

If the first byte of a group matches the bit pattern `110xxxxx`, then the group consists of that byte `a` and a second byte `b`. If there is no byte `b` (because byte `a` was the last of the bytes to be read), or if byte `b` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
(char)(((a& 0x1F) << 6) | (b & 0x3F))
```

If the first byte of a group matches the bit pattern `1110xxxx`, then the group consists of that byte `a` and two more bytes `b` and `c`. If there is no byte `c` (because byte `a` was one of the last two of the bytes to be read), or either byte `b` or byte `c` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
 (char)(((a & 0x0F) << 12) | ((b & 0x3F) << 6) | (c & 0x3F))
```

If the first byte of a group matches the pattern `1111xxxx` or the pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown.

If end of file is encountered at any time during this entire process, then an `EOFException` is thrown.

After every group has been converted to a character by this process, the characters are gathered, in the same order in which their corresponding groups were read from the input stream, to form a `String`, which is returned.

The `writeUTF` method of interface `DataOutput` may be used to write data that is suitable for reading by this method.

Returns: a Unicode string.

Throws:

`EOFException` - if this stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

`UTFDataFormatException` - if the bytes do not represent a valid UTF-8 encoding of a string.

java.io
DataInputStream

Declaration
```
public class DataInputStream extends InputStream implements DataInput
```

```
java.lang.Object
  |
  +--java.io.InputStream
        |
        +--java.io.DataInputStream
```

All Implemented Interfaces: DataInput

Description
A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

Since: JDK1.0, CLDC 1.0

See Also: DataOutputStream

Member Summary

Fields

protected InputStream	in

Constructors

	DataInputStream(InputStream in)

Methods

int	available()
void	close()
void	mark(int readlimit)
boolean	markSupported()
int	read()
int	read(byte[] b)
int	read(byte[] b, int off, int len)
boolean	readBoolean()
byte	readByte()
char	readChar()
double	readDouble()
float	readFloat()
void	readFully(byte[] b)
void	readFully(byte[] b, int off, int len)
int	readInt()
long	readLong()
short	readShort()

in

Member Summary	
int	readUnsignedByte()
int	readUnsignedShort()
java.lang.String	readUTF()
static	readUTF(DataInput in)
java.lang.String	
void	reset()
long	skip(long n)
int	skipBytes(int n)

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

in

Declaration:
```
protected java.io.InputStream in
```

Description:
The input stream.

Constructors

DataInputStream(InputStream)

Declaration:
```
public DataInputStream(java.io.InputStream in)
```

Description:
Creates a `DataInputStream` and saves its argument, the input stream `in`, for later use.

Parameters:
`in` - the input stream.

Methods

read()

Declaration:
```
public int read()
            throws IOException
```

Description:
Reads the next byte of data from this input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

This method simply performs `in.read()` and returns the result.

Overrides: `read` in class `InputStream`

Returns: the next byte of data, or `-1` if the end of the stream is reached.

Throws:
`IOException` - if an I/O error occurs.

read(byte[])

Declaration:
```
public final int read(byte[] b)
            throws IOException
```

Description:
See the general contract of the `read` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Overrides: `read` in class `InputStream`

Parameters:
`b` - the buffer into which the data is read.

Returns: the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:
`IOException` - if an I/O error occurs.

See Also: `InputStream.read(byte[], int, int)`

read(byte[], int, int)

Declaration:
```
public final int read(byte[] b, int off, int len)
            throws IOException
```

Description:
Reads up to `len` bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

This method simply performs `in.read(b, off, len)` and returns the result.

Overrides: `read` in class `InputStream`

Parameters:
`b` - the buffer into which the data is read.

`off` - the start offset of the data.

`len` - the maximum number of bytes read.

Returns: the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

readFully(byte[])

Throws:

`IOException` - if an I/O error occurs.

readFully(byte[])

Declaration:

```
public final void readFully(byte[] b)
            throws IOException
```

Description:

See the general contract of the `readFully` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readFully` in interface `DataInput`

Parameters:

`b` - the buffer into which the data is read.

Throws:

`EOFException` - if this input stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

readFully(byte[], int, int)

Declaration:

```
public final void readFully(byte[] b, int off, int len)
            throws IOException
```

Description:

See the general contract of the `readFully` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readFully` in interface `DataInput`

Parameters:

`b` - the buffer into which the data is read.

`off` - the start offset of the data.

`len` - the number of bytes to read.

Throws:

`EOFException` - if this input stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

skipBytes(int)

Declaration:

```
public final int skipBytes(int n)
            throws IOException
```

Description:

See the general contract of the `skipBytes` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `skipBytes` in interface `DataInput`

Parameters:
n - the number of bytes to be skipped.

Returns: the actual number of bytes skipped.

Throws:
IOException - if an I/O error occurs.

readBoolean()

Declaration:
```
public final boolean readBoolean()
            throws IOException
```

Description:
See the general contract of the readBoolean method of DataInput.

Bytes for this operation are read from the contained input stream.

Specified By: readBoolean in interface DataInput

Returns: the boolean value read.

Throws:
EOFException - if this input stream has reached the end.

IOException - if an I/O error occurs.

readByte()

Declaration:
```
public final byte readByte()
            throws IOException
```

Description:
See the general contract of the readByte method of DataInput.

Bytes for this operation are read from the contained input stream.

Specified By: readByte in interface DataInput

Returns: the next byte of this input stream as a signed 8-bit byte.

Throws:
EOFException - if this input stream has reached the end.

IOException - if an I/O error occurs.

readUnsignedByte()

Declaration:
```
public final int readUnsignedByte()
            throws IOException
```

Description:
See the general contract of the readUnsignedByte method of DataInput.

Bytes for this operation are read from the contained input stream.

Specified By: readUnsignedByte in interface DataInput

Returns: the next byte of this input stream, interpreted as an unsigned 8-bit number.

Throws:
`EOFException` - if this input stream has reached the end.

`IOException` - if an I/O error occurs.

readShort()

Declaration:
```
public final short readShort()
            throws IOException
```

Description:
See the general contract of the `readShort` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readShort` in interface `DataInput`

Returns: the next two bytes of this input stream, interpreted as a signed 16-bit number.

Throws:
`EOFException` - if this input stream reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

readUnsignedShort()

Declaration:
```
public final int readUnsignedShort()
            throws IOException
```

Description:
See the general contract of the `readUnsignedShort` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readUnsignedShort` in interface `DataInput`

Returns: the next two bytes of this input stream, interpreted as an unsigned 16-bit integer.

Throws:
`EOFException` - if this input stream reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

readChar()

Declaration:
```
public final char readChar()
            throws IOException
```

Description:
See the general contract of the `readChar` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readChar` in interface `DataInput`

Returns: the next two bytes of this input stream as a Unicode character.

Throws:
`EOFException` - if this input stream reaches the end before reading two bytes.

`IOException` - if an I/O error occurs.

readInt()

Declaration:

```
public final int readInt()
            throws IOException
```

Description:
See the general contract of the `readInt` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readInt` in interface `DataInput`

Returns: the next four bytes of this input stream, interpreted as an `int`.

Throws:

`EOFException` - if this input stream reaches the end before reading four bytes.

`IOException` - if an I/O error occurs.

readLong()

Declaration:

```
public final long readLong()
            throws IOException
```

Description:
See the general contract of the `readLong` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readLong` in interface `DataInput`

Returns: the next eight bytes of this input stream, interpreted as a `long`.

Throws:

`EOFException` - if this input stream reaches the end before reading eight bytes.

`IOException` - if an I/O error occurs.

readFloat()

Declaration:

```
public final float readFloat()
            throws IOException
```

Description:
See the general contract of the `readFloat` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readFloat` in interface `DataInput`

Returns: the next four bytes of this input stream, interpreted as a `float`.

Throws:

`EOFException` - if this input stream reaches the end before reading four bytes.

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

See Also: `readInt()`, `java.lang.Float.intBitsToFloat(int)`

readDouble()

Declaration:
```
public final double readDouble()
            throws IOException
```

Description:
See the general contract of the `readDouble` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readDouble` in interface `DataInput`

Returns: the next eight bytes of this input stream, interpreted as a `double`.

Throws:
`EOFException` - if this input stream reaches the end before reading eight bytes.

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

See Also: `readLong()`, `java.lang.Double.longBitsToDouble(long)`

readUTF()

Declaration:
```
public final java.lang.String readUTF()
            throws IOException
```

Description:
See the general contract of the `readUTF` method of `DataInput`.

Bytes for this operation are read from the contained input stream.

Specified By: `readUTF` in interface `DataInput`

Returns: a Unicode string.

Throws:
`EOFException` - if this input stream reaches the end before reading all the bytes.

`IOException` - if an I/O error occurs.

See Also: `readUTF(DataInput)`

readUTF(DataInput)

Declaration:
```
public static final java.lang.String readUTF(java.io.DataInput in)
            throws IOException
```

Description:
Reads from the stream `in` a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`. The details of the modified UTF-8 representation are exactly the same as for the `readUTF` method of `DataInput`.

Parameters:
`in` - a data input stream.

Returns: a Unicode string.

Throws:

`EOFException` - if the input stream reaches the end before all the bytes.

`IOException` - if an I/O error occurs.

`UTFDataFormatException` - if the bytes do not represent a valid UTF-8 encoding of a Unicode string.

See Also: `readUnsignedShort()`

skip(long)

Declaration:

```
public long skip(long n)
            throws IOException
```

Description:

Skips over and discards `n` bytes of data from the input stream. The `skip` method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly `0`. The actual number of bytes skipped is returned.

This method simply performs `in.skip(n)`.

Overrides: `skip` in class `InputStream`

Parameters:

`n` - the number of bytes to be skipped.

Returns: the actual number of bytes skipped.

Throws:

`IOException` - if an I/O error occurs.

available()

Declaration:

```
public int available()
            throws IOException
```

Description:

Returns the number of bytes that can be read from this input stream without blocking.

This method simply performs `in.available()` and returns the result.

Overrides: `available` in class `InputStream`

Returns: the number of bytes that can be read from the input stream without blocking.

Throws:

`IOException` - if an I/O error occurs.

close()

Declaration:

```
public void close()
            throws IOException
```

Description:

Closes this input stream and releases any system resources associated with the stream. This method simply performs `in.close()`.

Overrides: `close` in class `InputStream`

mark(int)

Throws:
`IOException` - if an I/O error occurs.

mark(int)

Declaration:
```
public void mark(int readlimit)
```

Description:
Marks the current position in this input stream. A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The `readlimit` argument tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

This method simply performs `in.mark(readlimit)`.

Overrides: `mark` in class `InputStream`

Parameters:
`readlimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

reset()

Declaration:
```
public void reset()
            throws IOException
```

Description:
Repositions this stream to the position at the time the `mark` method was last called on this input stream.

This method simply performs `in.reset()`.

Stream marks are intended to be used in situations where you need to read ahead a little to see what's in the stream. Often this is most easily done by invoking some general parser. If the stream is of the type handled by the parse, it just chugs along happily. If the stream is not of that type, the parser should toss an exception when it fails. If this happens within readlimit bytes, it allows the outer code to reset the stream and try another parser.

Overrides: `reset` in class `InputStream`

Throws:
`IOException` - if the stream has not been marked or if the mark has been invalidated.

markSupported()

Declaration:
```
public boolean markSupported()
```

Description:
Tests if this input stream supports the `mark` and `reset` methods. This method simply performs `in.markSupported()`.

Overrides: `markSupported` in class `InputStream`

Returns: `true` if this stream type supports the `mark` and `reset` method; `false` otherwise.

java.io
DataOutput

Declaration
`public interface DataOutput`

All Known Subinterfaces: `javax.microedition.io.Datagram`

All Known Implementing Classes: `DataOutputStream`

Description
The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream. There is also a facility for converting a `String` into Java modified UTF-8 format and writing the resulting series of bytes.

For all the methods in this interface that write bytes, it is generally true that if a byte cannot be written for any reason, an `IOException` is thrown.

Since: JDK1.0, CLDC 1.0

See Also: `DataInput`, `DataOutputStream`

Member Summary	
Methods	
void	`write(byte[] b)`
void	`write(byte[] b, int off, int len)`
void	`write(int b)`
void	`writeBoolean(boolean v)`
void	`writeByte(int v)`
void	`writeChar(int v)`
void	`writeChars(java.lang.String s)`
void	`writeDouble(double v)`
void	`writeFloat(float v)`
void	`writeInt(int v)`
void	`writeLong(long v)`
void	`writeShort(int v)`
void	`writeUTF(java.lang.String s)`

Methods

write(int)

Declaration:

```
public void write(int b)
            throws IOException
```

Description:
Writes to the output stream the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

Parameters:
`b` - the byte to be written.

Throws:
`IOException` - if an I/O error occurs.

write(byte[])

Declaration:

```
public void write(byte[] b)
            throws IOException
```

Description:
Writes to the output stream all the bytes in array `b`. If `b` is `null`, a `NullPointerException` is thrown. If `b.length` is zero, then no bytes are written. Otherwise, the byte `b[0]` is written first, then `b[1]`, and so on; the last byte written is `b[b.length-1]`.

Parameters:
`b` - the data.

Throws:
`IOException` - if an I/O error occurs.

write(byte[], int, int)

Declaration:

```
public void write(byte[] b, int off, int len)
            throws IOException
```

Description:
Writes `len` bytes from array `b`, in order, to the output stream. If `b` is `null`, a `NullPointerException` is thrown. If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown. If `len` is zero, then no bytes are written. Otherwise, the byte `b[off]` is written first, then `b[off+1]`, and so on; the last byte written is `b[off+len-1]`.

Parameters:
`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:
`IOException` - if an I/O error occurs.

writeBoolean(boolean)

Declaration:

```
public void writeBoolean(boolean v)
            throws IOException
```

Description:

Writes a `boolean` value to this output stream. If the argument `v` is `true`, the value `(byte)1` is written; if `v` is `false`, the value `(byte)0` is written. The byte written by this method may be read by the `readBoolean` method of interface `DataInput`, which will then return a `boolean` equal to `v`.

Parameters:

`v` - the boolean to be written.

Throws:

`IOException` - if an I/O error occurs.

writeByte(int)

Declaration:

```
public void writeByte(int v)
            throws IOException
```

Description:

Writes to the output stream the eight low- order bits of the argument `v`. The 24 high-order bits of `v` are ignored. (This means that `writeByte` does exactly the same thing as `write` for an integer argument.) The byte written by this method may be read by the `readByte` method of interface `DataInput`, which will then return a `byte` equal to `(byte)v`.

Parameters:

`v` - the byte value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeShort(int)

Declaration:

```
public void writeShort(int v)
            throws IOException
```

Description:

Writes two bytes to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the `readShort` method of interface `DataInput`, which will then return a `short` equal to `(short)v`.

Parameters:

`v` - the `short` value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeChar(int)

Declaration:

```
public void writeChar(int v)
            throws IOException
```

Description:

Writes a `char` value, which is comprised of two bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the `readChar` method of interface `DataInput`, which will then return a `char` equal to `(char)v`.

Parameters:

`v` - the `char` value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeInt(int)

Declaration:

```
public void writeInt(int v)
            throws IOException
```

Description:

Writes an `int` value, which is comprised of four bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 24))
(byte)(0xff & (v >> 16))
(byte)(0xff & (v >> &#32; &#32;8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the `readInt` method of interface `DataInput`, which will then return an `int` equal to `v`.

Parameters:

`v` - the `int` value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeLong(long)

Declaration:

```
public void writeLong(long v)
            throws IOException
```

Description:

Writes an `long` value, which is comprised of four bytes, to the output stream. The byte values to be written, in the order shown, are:

```
(byte)(0xff & (v >> 56))
(byte)(0xff & (v >> 48))
(byte)(0xff & (v >> 40))
(byte)(0xff & (v >> 32))
(byte)(0xff & (v >> 24))
(byte)(0xff & (v >> 16))
(byte)(0xff & (v >>  8))
(byte)(0xff & v)
```

The bytes written by this method may be read by the `readLong` method of interface `DataInput`, which will then return a `long` equal to `v`.

Parameters:

`v` - the `long` value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeFloat(float)

Declaration:

```
public void writeFloat(float v)
            throws IOException
```

Description:

Writes a `float` value, which is comprised of four bytes, to the output stream. It does this as if it first converts this `float` value to an `int` in exactly the manner of the `Float.floatToIntBits` method and then writes the `int` value in exactly the manner of the `writeInt` method. The bytes written by this method may be read by the `readFloat` method of interface `DataInput`, which will then return a `float` equal to `v`.

Parameters:

`v` - the `float` value to be written.

Throws:

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

writeDouble(double)

Declaration:

```
public void writeDouble(double v)
            throws IOException
```

Description:

Writes a `double` value, which is comprised of eight bytes, to the output stream. It does this as if it first converts this `double` value to a `long` in exactly the manner of the `Double.doubleToLongBits` method and then writes the `long` value in exactly the manner of the `writeLong` method. The bytes written by this method may be read by the `readDouble` method of interface `DataInput`, which will then return a `double` equal to `v`.

Parameters:

`v` - the `double` value to be written.

Throws:

`IOException` - if an I/O error occurs.

Since: CLDC 1.1

writeChars(String)

Declaration:

```
public void writeChars(java.lang.String s)
            throws IOException
```

Description:

Writes every character in the string `s`, to the output stream, in order, two bytes per character. If `s` is `null`, a `NullPointerException` is thrown. If `s.length` is zero, then no characters are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character written is `s[s.length-1]`. For each character, two bytes are actually written, high-order byte first, in exactly the manner of the `writeChar` method.

Parameters:

`s` - the string value to be written.

Throws:

`IOException` - if an I/O error occurs.

writeUTF(String)

Declaration:

```
public void writeUTF(java.lang.String s)
            throws IOException
```

Description:

Writes two bytes of length information to the output stream, followed by the Java modified UTF representation of every character in the string `s`. If `s` is `null`, a `NullPointerException` is thrown. Each character in the string `s` is converted to a group of one, two, or three bytes, depending on the value of the character.

If a character `c` is in the range `&#92;u0001` through `&#92;u007f`, it is represented by one byte:

```
(byte)c
```

If a character `c` is `&#92;u0000` or is in the range `&#92;u0080` through `&#92;u07ff`, then it is represented by two bytes, to be written in the order shown:

```
 (byte)(0xc0 | (0x1f & (c >> 6)))
 (byte)(0x80 | (0x3f & c))
```

If a character `c` is in the range `&#92;u0800` through `uffff`, then it is represented by three bytes, to be written in the order shown:

```
 (byte)(0xe0 | (0x0f & (c >> 12)))
 (byte)(0x80 | (0x3f & (c >>  6)))
 (byte)(0x80 | (0x3f & c))
```

First, the total number of bytes needed to represent all the characters of `s` is calculated. If this number is larger than `65535`, then a `UTFDataFormatError` is thrown. Otherwise, this length is written to the output stream in exactly the manner of the `writeShort` method; after this, the one-, two-, or three-byte representation of each character in the string `s` is written.

The bytes written by this method may be read by the `readUTF` method of interface `DataInput`, which will then return a `String` equal to `s`.

Parameters:

`s` - the string value to be written.

Throws:

`IOException` - if an I/O error occurs.

java.io
DataOutputStream

Declaration
```
public class DataOutputStream extends OutputStream implements DataOutput
```

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.DataOutputStream
```

All Implemented Interfaces: DataOutput

Description
A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

Since: JDK1.0, CLDC 1.0

See Also: DataInputStream

Member Summary	
Fields	
protected OutputStream	out
Constructors	
	DataOutputStream(OutputStream out)
Methods	
void	close()
void	flush()
void	write(byte[] b, int off, int len)
void	write(int b)
void	writeBoolean(boolean v)
void	writeByte(int v)
void	writeChar(int v)
void	writeChars(java.lang.String s)
void	writeDouble(double v)
void	writeFloat(float v)
void	writeInt(int v)
void	writeLong(long v)
void	writeShort(int v)
void	writeUTF(java.lang.String str)

out

Inherited Member Summary

Methods inherited from interface `DataOutput`
`write(byte[])`
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `OutputStream`
`write(byte[])`

Fields

out

Declaration:
```
protected java.io.OutputStream out
```

Description:
The output stream.

Constructors

DataOutputStream(OutputStream)

Declaration:
```
public DataOutputStream(java.io.OutputStream out)
```

Description:
Creates a new data output stream to write data to the specified underlying output stream.

Parameters:
`out` - the underlying output stream, to be saved for later use.

Methods

write(int)

Declaration:
```
public void write(int b)
            throws IOException
```

Description:
Writes the specified byte (the low eight bits of the argument `b`) to the underlying output stream.

Implements the `write` method of `OutputStream`.

Specified By: `write` in interface `DataOutput`

Overrides: `write` in class `OutputStream`

Parameters:
`b` - the `byte` to be written.

Throws:
`IOException` - if an I/O error occurs.

write(byte[], int, int)

Declaration:
```
public void write(byte[] b, int off, int len)
            throws IOException
```

Description:
Writes `len` bytes from the specified byte array starting at offset `off` to the underlying output stream.

Specified By: `write` in interface `DataOutput`

Overrides: `write` in class `OutputStream`

Parameters:
`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:
`IOException` - if an I/O error occurs.

flush()

Declaration:
```
public void flush()
            throws IOException
```

Description:
Flushes this data output stream. This forces any buffered output bytes to be written out to the stream.

The `flush` method of `DataOutputStream` calls the `flush` method of its underlying output stream.

Overrides: `flush` in class `OutputStream`

Throws:
`IOException` - if an I/O error occurs.

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Closes this output stream and releases any system resources associated with the stream.

The `close` method calls its `flush` method, and then calls the `close` method of its underlying output stream.

Overrides: `close` in class `OutputStream`

Throws:
`IOException` - if an I/O error occurs.

writeBoolean(boolean)

Declaration:

```
public final void writeBoolean(boolean v)
            throws IOException
```

Description:
Writes a `boolean` to the underlying output stream as a 1-byte value. The value `true` is written out as the value `(byte)1`; the value `false` is written out as the value `(byte)0`.

Specified By: `writeBoolean` in interface `DataOutput`

Parameters:
`v` - a `boolean` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeByte(int)

Declaration:

```
public final void writeByte(int v)
            throws IOException
```

Description:
Writes out a `byte` to the underlying output stream as a 1-byte value.

Specified By: `writeByte` in interface `DataOutput`

Parameters:
`v` - a `byte` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeShort(int)

Declaration:

```
public final void writeShort(int v)
            throws IOException
```

Description:
Writes a `short` to the underlying output stream as two bytes, high byte first.

Specified By: `writeShort` in interface `DataOutput`

Parameters:
`v` - a `short` to be written.

Throws:
`IOException` - if an I/O error occurs.

writeChar(int)

Declaration:

```
public final void writeChar(int v)
            throws IOException
```

Description:
Writes a `char` to the underlying output stream as a 2-byte value, high byte first.

Specified By: `writeChar` in interface `DataOutput`

Parameters:
`v` - a `char` value to be written.

Throws:
`IOException` - if an I/O error occurs.

writeInt(int)

Declaration:
```
public final void writeInt(int v)
            throws IOException
```

Description:
Writes an `int` to the underlying output stream as four bytes, high byte first.

Specified By: `writeInt` in interface `DataOutput`

Parameters:
`v` - an `int` to be written.

Throws:
`IOException` - if an I/O error occurs.

writeLong(long)

Declaration:
```
public final void writeLong(long v)
            throws IOException
```

Description:
Writes a `long` to the underlying output stream as eight bytes, high byte first.

Specified By: `writeLong` in interface `DataOutput`

Parameters:
`v` - a `long` to be written.

Throws:
`IOException` - if an I/O error occurs.

writeFloat(float)

Declaration:
```
public final void writeFloat(float v)
            throws IOException
```

Description:
Converts the float argument to an `int` using the `floatToIntBits` method in class `Float`, and then writes that `int` value to the underlying output stream as a 4-byte quantity, high byte first.

Specified By: `writeFloat` in interface `DataOutput`

Parameters:
`v` - a `float` value to be written.

Throws:
`IOException` - if an I/O error occurs.

Since: CLDC 1.1

See Also: `java.lang.Float.floatToIntBits(float)`

writeDouble(double)

Declaration:

```
public final void writeDouble(double v)
            throws IOException
```

Description:
Converts the double argument to a `long` using the `doubleToLongBits` method in class `Double`, and then writes that `long` value to the underlying output stream as an 8-byte quantity, high byte first.

Specified By: `writeDouble` in interface `DataOutput`

Parameters:
`v` - a `double` value to be written.

Throws:
`IOException` - if an I/O error occurs.

Since: CLDC 1.1

See Also: `java.lang.Double.doubleToLongBits(double)`

writeChars(String)

Declaration:

```
public final void writeChars(java.lang.String s)
            throws IOException
```

Description:
Writes a string to the underlying output stream as a sequence of characters. Each character is written to the data output stream as if by the `writeChar` method.

Specified By: `writeChars` in interface `DataOutput`

Parameters:
`s` - a `String` value to be written.

Throws:
`IOException` - if an I/O error occurs.

See Also: `writeChar(int)`

writeUTF(String)

Declaration:

```
public final void writeUTF(java.lang.String str)
            throws IOException
```

Description:
Writes a string to the underlying output stream using UTF-8 encoding in a machine-independent manner.

First, two bytes are written to the output stream as if by the `writeShort` method giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the UTF-8 encoding for the character.

Specified By: `writeUTF` in interface `DataOutput`

Parameters:

`str` - a string to be written.

Throws:

`IOException` - if an I/O error occurs.

java.io
EOFException

Declaration

```
public class EOFException extends IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.EOFException
```

Description

Signals that an end of file or end of stream has been reached unexpectedly during input.

This exception is mainly used by data input streams, which generally expect a binary file in a specific format, and for which an end of stream is an unusual condition. Most other input streams return a special value on end of stream.

Note that some input operations react to end-of-file by returning a distinguished value (such as `-1`) rather than by throwing an exception.

Since: JDK1.0, CLDC 1.0

See Also: `DataInputStream`, `IOException`

Member Summary	
Constructors	
	`EOFException()`
	`EOFException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

EOFException()

Declaration:
```
public EOFException()
```

Description:
Constructs an `EOFException` with `null` as its error detail message.

EOFException(String)

Declaration:
```
public EOFException(java.lang.String s)
```

Description:
Constructs an `EOFException` with the specified detail message. The string `s` may later be retrieved by the `java.lang.Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:
`s` - the detail message.

java.io
InputStream

Declaration

```
public abstract class InputStream

java.lang.Object
  |
  +--java.io.InputStream
```

Direct Known Subclasses: `ByteArrayInputStream`, `DataInputStream`

Description

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since: JDK1.0, CLDC 1.0

See Also: `ByteArrayInputStream`, `DataInputStream`, `read()`, `OutputStream`

Member Summary

Constructors

	`InputStream()`

Methods

`int`	`available()`
`void`	`close()`
`void`	`mark(int readlimit)`
`boolean`	`markSupported()`
`abstract int`	`read()`
`int`	`read(byte[] b)`
`int`	`read(byte[] b, int off, int len)`
`void`	`reset()`
`long`	`skip(long n)`

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

InputStream()

Declaration:
```
public InputStream()
```

Methods

read()

Declaration:
```
public abstract int read()
            throws IOException
```

Description:
Reads the next byte of data from the input stream. The value byte is returned as an `int` in the range `0` to `255`. If no byte is available because the end of the stream has been reached, the value `-1` is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns: the next byte of data, or `-1` if the end of the stream is reached.

Throws:
`IOException` - if an I/O error occurs.

read(byte[])

Declaration:
```
public int read(byte[] b)
            throws IOException
```

Description:
Reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown. If the length of `b` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. The number of bytes read is, at most, equal to the length of `b`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[0]` through `b[`*k*`-1]`, leaving elements `b[`*k*`]` through `b[b.length-1]` unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

The `read(b)` method for class `InputStream` has the same effect as:

```
read(b, 0, b.length)
```

Parameters:
`b` - the buffer into which the data is read.

Returns: the total number of bytes read into the buffer, or `-1` is there is no more data because the end of the stream has been reached.

Throws:
`IOException` - if an I/O error occurs.

See Also: `read(byte[], int, int)`

read(byte[], int, int)

Declaration:
```
public int read(byte[] b, int off, int len)
            throws IOException
```

Description:
Reads up to `len` bytes of data from the input stream into an array of bytes. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and `0` is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value `-1` is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let *k* be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+`*k*`-1]`, leaving elements `b[off+`*k*`]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed.

The `read(b, off, len)` method for class `InputStream` simply calls the method `read()` repeatedly. If the first such call results in an `IOException`, that exception is returned from the call to the `read(b, off, len)` method. If any subsequent call to `read()` results in a `IOException`, the exception is caught and treated as if it were end of file; the bytes read up to that point are stored into `b` and the number of bytes read before the exception occurred is returned. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:
`b` - the buffer into which the data is read.

`off` - the start offset in array `b` at which the data is written.

`len` - the maximum number of bytes to read.

Returns: the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws:
`IOException` - if an I/O error occurs.

See Also: `read()`

skip(long)

Declaration:

```
public long skip(long n)
            throws IOException
```

Description:
Skips over and discards `n` bytes of data from this input stream. The `skip` method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly `0`. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If `n` is negative, no bytes are skipped.

The `skip` method of `InputStream` creates a byte array and then repeatedly reads into it until `n` bytes have been read or the end of the stream has been reached. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:
`n` - the number of bytes to be skipped.

Returns: the actual number of bytes skipped.

Throws:
`IOException` - if an I/O error occurs.

available()

Declaration:

```
public int available()
            throws IOException
```

Description:
Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream. The next caller might be the same thread or another thread.

The `available` method for class `InputStream` always returns `0`.

This method should be overridden by subclasses.

Returns: the number of bytes that can be read from this input stream without blocking.

Throws:
`IOException` - if an I/O error occurs.

close()

Declaration:

```
public void close()
            throws IOException
```

Description:
Closes this input stream and releases any system resources associated with the stream.

The `close` method of `InputStream` does nothing.

Throws:
`IOException` - if an I/O error occurs.

mark(int)

Declaration:
```
public void mark(int readlimit)
```

Description:
Marks the current position in this input stream. A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The `readlimit` arguments tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

The general contract of `mark` is that, if the method `markSupported` returns `true`, the stream somehow remembers all the bytes read after the call to `mark` and stands ready to supply those same bytes again if and whenever the method `reset` is called. However, the stream is not required to remember any data at all if more than `readlimit` bytes are read from the stream before `reset` is called.

The `mark` method of `InputStream` does nothing.

Parameters:
`readlimit` - the maximum limit of bytes that can be read before the mark position becomes invalid.

See Also: `reset()`

reset()

Declaration:
```
public void reset()
            throws IOException
```

Description:
Repositions this stream to the position at the time the `mark` method was last called on this input stream.

The general contract of `reset` is:

- If the method `markSupported` returns `true`, then:
  - If the method `mark` has not been called since the stream was created, or the number of bytes read from the stream since `mark` was last called is larger than the argument to `mark` at that last call, then an `IOException` might be thrown.
  - If such an `IOException` is not thrown, then the stream is reset to a state such that all the bytes read since the most recent call to `mark` (or since the start of the file, if `mark` has not been called) will be resupplied to subsequent callers of the `read` method, followed by any bytes that otherwise would have been the next input data as of the time of the call to `reset`.
- If the method `markSupported` returns `false`, then:
  - The call to `reset` may throw an `IOException`.
  - If an `IOException` is not thrown, then the stream is reset to a fixed state that depends on the particular type of the input stream and how it was created. The bytes that will be supplied to subsequent callers of the `read` method depend on the particular type of the input stream.

The method `reset` for class `InputStream` does nothing and always throws an `IOException`.

Throws:
`IOException` - if this stream has not been marked or if the mark has been invalidated.

See Also: `mark(int)`, `IOException`

markSupported()

Declaration:
```
public boolean markSupported()
```

Description:
Tests if this input stream supports the `mark` and `reset` methods. The `markSupported` method of `InputStream` returns `false`.

Returns: `true` if this true type supports the mark and reset method; `false` otherwise.

See Also: `mark(int)`, `reset()`

java.io
InputStreamReader

Declaration

```
public class InputStreamReader extends Reader
```

```
java.lang.Object
  |
  +--java.io.Reader
        |
        +--java.io.InputStreamReader
```

Description

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

Each invocation of one of an InputStreamReader's read() methods may cause one or more bytes to be read from the underlying byte input stream. To enable the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

Since: CLDC 1.0

See Also: `Reader`, `UnsupportedEncodingException`

Member Summary

Constructors

```
                        InputStreamReader(InputStream is)
                        InputStreamReader(InputStream is, java.lang.String enc)
```

Methods

```
                   void  close()
                   void  mark(int readAheadLimit)
                boolean  markSupported()
                    int  read()
                    int  read(char[] cbuf, int off, int len)
                boolean  ready()
                   void  reset()
                   long  skip(long n)
```

Inherited Member Summary

Fields inherited from class `Reader`

```
lock
```

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `Reader`

`read(char[])`

Constructors

InputStreamReader(InputStream)

Declaration:
```
public InputStreamReader(java.io.InputStream is)
```

Description:
Create an InputStreamReader that uses the default character encoding.

Parameters:
`is` - An InputStream

InputStreamReader(InputStream, String)

Declaration:
```
public InputStreamReader(java.io.InputStream is, java.lang.String enc)
            throws UnsupportedEncodingException
```

Description:
Create an InputStreamReader that uses the named character encoding.

Parameters:
`is` - An InputStream

`enc` - The name of a supported character encoding

Throws:
`UnsupportedEncodingException` - If the named encoding is not supported

Methods

read()

Declaration:
```
public int read()
            throws IOException
```

Description:
Read a single character.

Overrides: `read` in class `Reader`

Returns: The character read, or -1 if the end of the stream has been reached

read(char[], int, int)

Throws:
`IOException` - If an I/O error occurs

read(char[], int, int)

Declaration:
```
public int read(char[] cbuf, int off, int len)
            throws IOException
```

Description:
Read characters into a portion of an array.

Overrides: `read` in class `Reader`

Parameters:
`cbuf` - Destination buffer

`off` - Offset at which to start storing characters

`len` - Maximum number of characters to read

Returns: The number of characters read, or -1 if the end of the stream has been reached

Throws:
`IOException` - If an I/O error occurs

skip(long)

Declaration:
```
public long skip(long n)
            throws IOException
```

Description:
Skip characters.

Overrides: `skip` in class `Reader`

Parameters:
`n` - The number of characters to skip

Returns: The number of characters actually skipped

Throws:
`java.lang.IllegalArgumentException` - If `n` is negative.

`IOException` - If an I/O error occurs

ready()

Declaration:
```
public boolean ready()
            throws IOException
```

Description:
Tell whether this stream is ready to be read.

Overrides: `ready` in class `Reader`

Returns: True if the next read() is guaranteed not to block for input, false otherwise. Note that returning false does not guarantee that the next read will block.

Throws:
`IOException` - If an I/O error occurs

markSupported()

Declaration:
```
public boolean markSupported()
```

Description:
Tell whether this stream supports the mark() operation.

Overrides: `markSupported` in class `Reader`

Returns: true if and only if this stream supports the mark operation.

mark(int)

Declaration:
```
public void mark(int readAheadLimit)
            throws IOException
```

Description:
Mark the present position in the stream.

Overrides: `mark` in class `Reader`

Parameters:
`readAheadLimit` - Limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail.

Throws:
`IOException` - If the stream does not support mark(), or if some other I/O error occurs

reset()

Declaration:
```
public void reset()
            throws IOException
```

Description:
Reset the stream.

Overrides: `reset` in class `Reader`

Throws:
`IOException` - If an I/O error occurs

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Close the stream. Closing a previously closed stream has no effect.

Overrides: `close` in class `Reader`

Throws:
`IOException` - If an I/O error occurs

java.io
InterruptedIOException

Declaration

```
public class InterruptedIOException extends IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.InterruptedIOException
```

Description

Signals that an I/O operation has been interrupted. An `InterruptedIOException` is thrown to indicate that an input or output transfer has been terminated because the thread performing it was terminated. The field `bytesTransferred` indicates how many bytes were successfully transferred before the interruption occurred.

Since: JDK1.0, CLDC 1.0

See Also: `InputStream`, `OutputStream`

Member Summary	
Fields	
`int`	`bytesTransferred`
Constructors	
	`InterruptedIOException()`
	`InterruptedIOException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Fields

bytesTransferred

Declaration:
```
public int bytesTransferred
```

Description:
Reports how many bytes had been transferred as part of the I/O operation before it was interrupted.

Constructors

InterruptedIOException()

Declaration:
```
public InterruptedIOException()
```

Description:
Constructs an `InterruptedIOException` with `null` as its error detail message.

InterruptedIOException(String)

Declaration:
```
public InterruptedIOException(java.lang.String s)
```

Description:
Constructs an `InterruptedIOException` with the specified detail message. The string `s` can be retrieved later by the `java.lang.Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:
`s` - the detail message.

java.io
IOException

Declaration

```
public class IOException extends java.lang.Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
```

Direct Known Subclasses: `javax.microedition.io.ConnectionNotFoundException`, `EOFException`, `InterruptedIOException`, `UnsupportedEncodingException`, `UTFDataFormatException`

Description

Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

Since: JDK1.0, CLDC 1.0

See Also: `InputStream`, `OutputStream`

Member Summary	
Constructors	
	`IOException()`
	`IOException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

IOException()

Declaration:
```
public IOException()
```

Description:
Constructs an `IOException` with `null` as its error detail message.

IOException(String)

Declaration:
```
public IOException(java.lang.String s)
```

Description:
Constructs an `IOException` with the specified detail message. The error message string `s` can later be retrieved by the `java.lang.Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:
`s` - the detail message.

java.io
OutputStream

Declaration

```
public abstract class OutputStream

java.lang.Object
  |
  +--java.io.OutputStream
```

Direct Known Subclasses: `ByteArrayOutputStream`, `DataOutputStream`, `PrintStream`

Description

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since: JDK1.0, CLDC 1.0

See Also: `ByteArrayOutputStream`, `DataOutputStream`, `InputStream`, `write(int)`

Member Summary

Constructors

	`OutputStream()`

Methods

void	`close()`
void	`flush()`
void	`write(byte[] b)`
void	`write(byte[] b, int off, int len)`
abstract void	`write(int b)`

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

OutputStream()

Declaration:
```
public OutputStream()
```

Methods

write(int)

Declaration:
```
public abstract void write(int b)
            throws IOException
```

Description:
Writes the specified byte to this output stream. The general contract for `write` is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

Subclasses of `OutputStream` must provide an implementation for this method.

Parameters:
`b` - the `byte`.

Throws:
`IOException` - if an I/O error occurs. In particular, an `IOException` may be thrown if the output stream has been closed.

write(byte[])

Declaration:
```
public void write(byte[] b)
            throws IOException
```

Description:
Writes `b.length` bytes from the specified byte array to this output stream. The general contract for `write(b)` is that it should have exactly the same effect as the call `write(b, 0, b.length)`.

Parameters:
`b` - the data.

Throws:
`IOException` - if an I/O error occurs.

See Also: `write(byte[], int, int)`

write(byte[], int, int)

Declaration:
```
public void write(byte[] b, int off, int len)
            throws IOException
```

Description:
Writes `len` bytes from the specified byte array starting at offset `off` to this output stream. The general contract for `write(b, off, len)` is that some of the bytes in the array `b` are written to the output stream in order; element `b[off]` is the first byte written and `b[off+len-1]` is the last byte written by this operation.

The `write` method of `OutputStream` calls the write method of one argument on each of the bytes to be written out. Subclasses are encouraged to override this method and provide a more efficient implementation.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

Parameters:
`b` - the data.

`off` - the start offset in the data.

`len` - the number of bytes to write.

Throws:
`IOException` - if an I/O error occurs. In particular, an `IOException` is thrown if the output stream is closed.

flush()

Declaration:
```
public void flush()
            throws IOException
```

Description:
Flushes this output stream and forces any buffered output bytes to be written out. The general contract of `flush` is that calling it is an indication that, if any bytes previously written have been buffered by the implementation of the output stream, such bytes should immediately be written to their intended destination.

The `flush` method of `OutputStream` does nothing.

Throws:
`IOException` - if an I/O error occurs.

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Closes this output stream and releases any system resources associated with this stream. The general contract of `close` is that it closes the output stream. A closed stream cannot perform output operations and cannot be reopened.

The `close` method of `OutputStream` does nothing.

Throws:
`IOException` - if an I/O error occurs.

java.io
OutputStreamWriter

Declaration

```
public class OutputStreamWriter extends Writer

java.lang.Object
  |
  +--java.io.Writer
        |
        +--java.io.OutputStreamWriter
```

Description

An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are translated into bytes. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

Each invocation of a write() method causes the encoding converter to be invoked on the given character(s). The resulting bytes are accumulated in a buffer before being written to the underlying output stream. The size of this buffer may be specified, but by default it is large enough for most purposes. Note that the characters passed to the write() methods are not buffered.

Since: CLDC 1.0

See Also: `Writer`, `UnsupportedEncodingException`

Member Summary

Constructors

	`OutputStreamWriter(OutputStream os)`
	`OutputStreamWriter(OutputStream os, java.lang.String enc)`

Methods

void	`close()`
void	`flush()`
void	`write(char[] cbuf, int off, int len)`
void	`write(int c)`
void	`write(java.lang.String str, int off, int len)`

Inherited Member Summary

Fields inherited from class `Writer`

`lock`

Methods inherited from class `Object`

Inherited Member Summary
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Writer`
`write(String)`, `write(String)`

Constructors

OutputStreamWriter(OutputStream)

Declaration:
```
public OutputStreamWriter(java.io.OutputStream os)
```

Description:
Create an OutputStreamWriter that uses the default character encoding.

Parameters:
`os` - An OutputStream

OutputStreamWriter(OutputStream, String)

Declaration:
```
public OutputStreamWriter(java.io.OutputStream os, java.lang.String enc)
            throws UnsupportedEncodingException
```

Description:
Create an OutputStreamWriter that uses the named character encoding.

Parameters:
`os` - An OutputStream

`enc` - The name of a supported

Throws:
`UnsupportedEncodingException` - If the named encoding is not supported

Methods

write(int)

Declaration:
```
public void write(int c)
            throws IOException
```

Description:
Write a single character.

Overrides: `write` in class `Writer`

Parameters:
`c` - int specifying a character to be written.

Throws:
`IOException` - If an I/O error occurs

write(char[], int, int)

Declaration:
```
public void write(char[] cbuf, int off, int len)
            throws IOException
```

Description:
Write a portion of an array of characters.

Overrides: `write` in class `Writer`

Parameters:
`cbuf` - Buffer of characters to be written

`off` - Offset from which to start reading characters

`len` - Number of characters to be written

Throws:
`IOException` - If an I/O error occurs

write(String, int, int)

Declaration:
```
public void write(java.lang.String str, int off, int len)
            throws IOException
```

Description:
Write a portion of a string.

Overrides: `write` in class `Writer`

Parameters:
`str` - String to be written

`off` - Offset from which to start reading characters

`len` - Number of characters to be written

Throws:
`IOException` - If an I/O error occurs

flush()

Declaration:
```
public void flush()
            throws IOException
```

Description:
Flush the stream.

Overrides: `flush` in class `Writer`

Throws:
`IOException` - If an I/O error occurs

close()

Declaration:

```
public void close()
            throws IOException
```

Description:
Close the stream.

Overrides: `close` in class `Writer`

Throws:
`IOException` - If an I/O error occurs

java.io
PrintStream

Declaration

```
public class PrintStream extends OutputStream
```

```
java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--java.io.PrintStream
```

Description

A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a `PrintStream` never throws an `IOException`; instead, exceptional situations merely set an internal flag that can be tested via the `checkError` method.

All characters printed by a `PrintStream` are converted into bytes using the platform's default character encoding.

Since: JDK1.0, CLDC 1.0

Member Summary

Constructors

	`PrintStream(OutputStream out)`

Methods

`boolean`	`checkError()`
`void`	`close()`
`void`	`flush()`
`void`	`print(boolean b)`
`void`	`print(char c)`
`void`	`print(char[] s)`
`void`	`print(double d)`
`void`	`print(float f)`
`void`	`print(int i)`
`void`	`print(long l)`
`void`	`print(java.lang.Object obj)`
`void`	`print(java.lang.String s)`
`void`	`println()`
`void`	`println(boolean x)`
`void`	`println(char x)`
`void`	`println(char[] x)`
`void`	`println(double x)`
`void`	`println(float x)`
`void`	`println(int x)`
`void`	`println(long x)`
`void`	`println(java.lang.Object x)`

Member Summary	
void	println(java.lang.String x)
protected void	setError()
void	write(byte[] buf, int off, int len)
void	write(int b)

Inherited Member Summary
Methods inherited from class Object equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait() **Methods inherited from class OutputStream** write(byte[])

Constructors

PrintStream(OutputStream)

Declaration:
```
public PrintStream(java.io.OutputStream out)
```

Description:
Create a new print stream. This stream will not flush automatically.

Parameters:
`out` - The output stream to which values and objects will be printed

Methods

flush()

Declaration:
```
public void flush()
```

Description:
Flush the stream. This is done by writing any buffered output bytes to the underlying output stream and then flushing that stream.

Overrides: `flush` in class `OutputStream`

See Also: `OutputStream.flush()`

close()

Declaration:
```
public void close()
```

Description:
Close the stream. This is done by flushing the stream and then closing the underlying output stream.

Overrides: `close` in class `OutputStream`

See Also: `OutputStream.close()`

checkError()

Declaration:
```
public boolean checkError()
```

Description:
Flush the stream and check its error state. The internal error state is set to `true` when the underlying output stream throws an `IOException`, and when the `setError` method is invoked.

Returns: True if and only if this stream has encountered an `IOException`, or the `setError` method has been invoked

setError()

Declaration:
```
protected void setError()
```

Description:
Set the error state of the stream to `true`.

Since: JDK1.1

write(int)

Declaration:
```
public void write(int b)
```

Description:
Write the specified byte to this stream.

Note that the byte is written as given; to write a character that will be translated according to the platform's default character encoding, use the `print(char)` or `println(char)` methods.

Overrides: `write` in class `OutputStream`

Parameters:
`b` - The byte to be written

See Also: `print(char)`, `println(char)`

write(byte[], int, int)

Declaration:
```
public void write(byte[] buf, int off, int len)
```

Description:
Write `len` bytes from the specified byte array starting at offset `off` to this stream.

Note that the bytes will be written as given; to write characters that will be translated according to the platform's default character encoding, use the `print(char)` or `println(char)` methods.

Overrides: `write` in class `OutputStream`

Parameters:

`buf` - A byte array

`off` - Offset from which to start taking bytes

`len` - Number of bytes to write

print(boolean)

Declaration:

```
public void print(boolean b)
```

Description:

Print a boolean value. The string produced by `java.lang.String.valueOf(boolean)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`b` - The `boolean` to be printed

print(char)

Declaration:

```
public void print(char c)
```

Description:

Print a character. The character is translated into one or more bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`c` - The `char` to be printed

print(int)

Declaration:

```
public void print(int i)
```

Description:

Print an integer. The string produced by `java.lang.String.valueOf(int)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`i` - The `int` to be printed

See Also: `java.lang.Integer.toString(int)`

print(long)

Declaration:

```
public void print(long l)
```

Description:

Print a long integer. The string produced by `java.lang.String.valueOf(long)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`l` - The `long` to be printed

See Also: `java.lang.Long.toString(long)`

print(float)

Declaration:
```
public void print(float f)
```

Description:
Print a floating point number. The string produced by `java.lang.String.valueOf(float)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:
`f` - The `float` to be printed

Since: CLDC 1.1

See Also: `java.lang.Float.toString(float)`

print(double)

Declaration:
```
public void print(double d)
```

Description:
Print a double-precision floating point number. The string produced by `java.lang.String.valueOf(double)` is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:
`d` - The `double` to be printed

Since: CLDC 1.1

See Also: `java.lang.Double.toString(double)`

print(char[])

Declaration:
```
public void print(char[] s)
```

Description:
Print an array of characters. The characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:
`s` - The array of chars to be printed

Throws:
`java.lang.NullPointerException` - If `s` is `null`

print(String)

Declaration:
```
public void print(java.lang.String s)
```

Description:
Print a string. If the argument is `null` then the string `"null"` is printed. Otherwise, the string's characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:
`s` - The `String` to be printed

print(Object)

Declaration:
```
public void print(java.lang.Object obj)
```

Description:
Print an object. The string produced by the `java.lang.String.valueOf(Object)` method is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:
`obj` - The `Object` to be printed

See Also: `java.lang.Object.toString()`

println()

Declaration:
```
public void println()
```

Description:
Terminate the current line by writing the line separator string. The line separator string is defined by the system property `line.separator`, and is not necessarily a single newline character (`'\n'`).

println(boolean)

Declaration:
```
public void println(boolean x)
```

Description:
Print a boolean and then terminate the line. This method behaves as though it invokes `print(boolean)` and then `println()`.

Parameters:
`x` - The `boolean` to be printed

println(char)

Declaration:
```
public void println(char x)
```

Description:
Print a character and then terminate the line. This method behaves as though it invokes `print(char)` and then `println()`.

Parameters:
`x` - The `char` to be printed.

println(int)

Declaration:
```
public void println(int x)
```

Description:
Print an integer and then terminate the line. This method behaves as though it invokes `print(int)` and then `println()`.

Parameters:
`x` - The `int` to be printed.

println(long)

Declaration:
```
public void println(long x)
```

Description:
Print a long and then terminate the line. This method behaves as though it invokes `print(long)` and then `println()`.

Parameters:
`x` - The `long` to be printed.

println(float)

Declaration:
```
public void println(float x)
```

Description:
Print a float and then terminate the line. This method behaves as though it invokes `print(float)` and then `println()`.

Parameters:
`x` - The `float` to be printed.

Since: CLDC 1.1

println(double)

Declaration:
```
public void println(double x)
```

Description:
Print a double and then terminate the line. This method behaves as though it invokes `print(double)` and then `println()`.

Parameters:
`x` - The `double` to be printed.

Since: CLDC 1.1

println(char[])

Declaration:
```
public void println(char[] x)
```

Description:
Print an array of characters and then terminate the line. This method behaves as though it invokes `print(char[])` and then `println()`.

println(String)

Parameters:

`x` - an array of chars to print.

println(String)

Declaration:

```
public void println(java.lang.String x)
```

Description:

Print a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

Parameters:

`x` - The `String` to be printed.

println(Object)

Declaration:

```
public void println(java.lang.Object x)
```

Description:

Print an Object and then terminate the line. This method behaves as though it invokes `print(Object)` and then `println()`.

Parameters:

`x` - The `Object` to be printed.

java.io
Reader

Declaration
```
public abstract class Reader
```

```
java.lang.Object
  |
  +--java.io.Reader
```

Direct Known Subclasses: `InputStreamReader`

Description
Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since: JDK1.1, CLDC 1.0

See Also: `InputStreamReader`, `Writer`

Member Summary	
Fields	
protected	lock
java.lang.Object	
Constructors	
protected	Reader()
protected	Reader(java.lang.Object lock)
Methods	
abstract void	close()
void	mark(int readAheadLimit)
boolean	markSupported()
int	read()
int	read(char[] cbuf)
abstract int	read(char[] cbuf, int off, int len)
boolean	ready()
void	reset()
long	skip(long n)

Inherited Member Summary
Methods inherited from class `Object`

Inherited Member Summary
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

lock

Declaration:
```
protected java.lang.Object lock
```

Description:
The object used to synchronize operations on this stream. For efficiency, a character-stream object may use an object other than itself to protect critical sections. A subclass should therefore use the object in this field rather than `this` or a synchronized method.

Constructors

Reader()

Declaration:
```
protected Reader()
```

Description:
Create a new character-stream reader whose critical sections will synchronize on the reader itself.

Reader(Object)

Declaration:
```
protected Reader(java.lang.Object lock)
```

Description:
Create a new character-stream reader whose critical sections will synchronize on the given object.

Parameters:
`lock` - The Object to synchronize on.

Methods

read()

Declaration:
```
public int read()
            throws IOException
```

Description:
Read a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

Subclasses that intend to support efficient single-character input should override this method.

Returns: The character read, as an integer in the range 0 to 65535 (`0x00-0xffff`), or -1 if the end of the stream has been reached

Throws:
`IOException` - If an I/O error occurs

read(char[])

Declaration:
```
public int read(char[] cbuf)
            throws IOException
```

Description:
Read characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Parameters:
`cbuf` - Destination buffer

Returns: The number of bytes read, or -1 if the end of the stream has been reached

Throws:
`IOException` - If an I/O error occurs

read(char[], int, int)

Declaration:
```
public abstract int read(char[] cbuf, int off, int len)
            throws IOException
```

Description:
Read characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Parameters:
`cbuf` - Destination buffer

`off` - Offset at which to start storing characters

`len` - Maximum number of characters to read

Returns: The number of characters read, or -1 if the end of the stream has been reached

Throws:
`IOException` - If an I/O error occurs

skip(long)

Declaration:
```
public long skip(long n)
            throws IOException
```

Description:
Skip characters. This method will block until some characters are available, an I/O error occurs, or the end of the stream is reached.

Parameters:
`n` - The number of characters to skip

Returns: The number of characters actually skipped

Throws:
`java.lang.IllegalArgumentException` - If `n` is negative.

`IOException` - If an I/O error occurs

ready()

Declaration:
```
public boolean ready()
            throws IOException
```

Description:
Tell whether this stream is ready to be read.

Returns: True if the next read() is guaranteed not to block for input, false otherwise. Note that returning false does not guarantee that the next read will block.

Throws:
`IOException` - If an I/O error occurs

markSupported()

Declaration:
```
public boolean markSupported()
```

Description:
Tell whether this stream supports the mark() operation. The default implementation always returns false. Subclasses should override this method.

Returns: true if and only if this stream supports the mark operation.

mark(int)

Declaration:
```
public void mark(int readAheadLimit)
            throws IOException
```

Description:
Mark the present position in the stream. Subsequent calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support the mark() operation.

Parameters:
`readAheadLimit` - Limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail.

Throws:
`IOException` - If the stream does not support mark(), or if some other I/O error occurs

reset()

Declaration:
```
public void reset()
            throws IOException
```

Description:
Reset the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream, for example by repositioning it to its starting point. Not all character-input streams support the reset() operation, and some support reset() without supporting mark().

Throws:

`IOException` - If the stream has not been marked, or if the mark has been invalidated, or if the stream does not support reset(), or if some other I/O error occurs

close()

Declaration:

```
public abstract void close()
            throws IOException
```

Description:

Close the stream. Once a stream has been closed, further read(), ready(), mark(), or reset() invocations will throw an IOException. Closing a previously-closed stream, however, has no effect.

Throws:

`IOException` - If an I/O error occurs

java.io
UnsupportedEncodingException

Declaration

```
public class UnsupportedEncodingException extends IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.UnsupportedEncodingException
```

Description

The Character Encoding is not supported.

Since: JDK1.1, CLDC 1.0

Member Summary	
Constructors	
	UnsupportedEncodingException() UnsupportedEncodingException(java.lang.String s)

Inherited Member Summary
Methods inherited from class Object equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait() **Methods inherited from class Throwable** getMessage(), printStackTrace(), toString()

Constructors

UnsupportedEncodingException()

Declaration:

```
public UnsupportedEncodingException()
```

Description:

Constructs an UnsupportedEncodingException without a detail message.

UnsupportedEncodingException(String)

Declaration:

```
public UnsupportedEncodingException(java.lang.String s)
```

Description:

Constructs an UnsupportedEncodingException with a detail message.

Parameters:

`s` - Describes the reason for the exception.

java.io
UTFDataFormatException

Declaration

```
public class UTFDataFormatException extends IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--java.io.UTFDataFormatException
```

Description

Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface. See the `writeUTF` method for the format in which UTF-8 strings are read and written.

Since: JDK1.0, CLDC 1.0

See Also: `DataInput`, `DataInputStream.readUTF(DataInput)`, `IOException`

Member Summary	
Constructors	
	`UTFDataFormatException()`
	`UTFDataFormatException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

UTFDataFormatException()

Declaration:

```
public UTFDataFormatException()
```

Description:
Constructs a `UTFDataFormatException` with `null` as its error detail message.

UTFDataFormatException(String)

Declaration:
```
public UTFDataFormatException(java.lang.String s)
```

Description:
Constructs a `UTFDataFormatException` with the specified detail message. The string `s` can be retrieved later by the `java.lang.Throwable.getMessage()` method of class `java.lang.Throwable`.

Parameters:
`s` - the detail message.

java.io
Writer

Declaration
```
public abstract class Writer
```

```
java.lang.Object
  |
  +--java.io.Writer
```

Direct Known Subclasses: `OutputStreamWriter`

Description
Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since: JDK1.1, CLDC 1.0

See Also: `OutputStreamWriter`, `Reader`

Member Summary

Fields

protected	lock
java.lang.Object	

Constructors

protected	Writer()
protected	Writer(java.lang.Object lock)

Methods

abstract void	close()
abstract void	flush()
void	write(char[] cbuf)
abstract void	write(char[] cbuf, int off, int len)
void	write(int c)
void	write(java.lang.String str)
void	write(java.lang.String str, int off, int len)

Inherited Member Summary

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

lock

Declaration:
```
protected java.lang.Object lock
```

Description:
The object used to synchronize operations on this stream. For efficiency, a character-stream object may use an object other than itself to protect critical sections. A subclass should therefore use the object in this field rather than `this` or a synchronized method.

Constructors

Writer()

Declaration:
```
protected Writer()
```

Description:
Create a new character-stream writer whose critical sections will synchronize on the writer itself.

Writer(Object)

Declaration:
```
protected Writer(java.lang.Object lock)
```

Description:
Create a new character-stream writer whose critical sections will synchronize on the given object.

Parameters:
`lock` - Object to synchronize on.

Methods

write(int)

Declaration:
```
public void write(int c)
            throws IOException
```

Description:
Write a single character. The character to be written is contained in the 16 low-order bits of the given integer value; the 16 high-order bits are ignored.

Subclasses that intend to support efficient single-character output should override this method.

Parameters:
`c` - int specifying a character to be written.

Throws:
`IOException` - If an I/O error occurs

write(char[])

Declaration:

```
public void write(char[] cbuf)
            throws IOException
```

Description:
Write an array of characters.

Parameters:
`cbuf` - Array of characters to be written

Throws:
`IOException` - If an I/O error occurs

write(char[], int, int)

Declaration:

```
public abstract void write(char[] cbuf, int off, int len)
            throws IOException
```

Description:
Write a portion of an array of characters.

Parameters:
`cbuf` - Array of characters

`off` - Offset from which to start writing characters

`len` - Number of characters to write

Throws:
`IOException` - If an I/O error occurs

write(String)

Declaration:

```
public void write(java.lang.String str)
            throws IOException
```

Description:
Write a string.

Parameters:
`str` - String to be written

Throws:
`IOException` - If an I/O error occurs

write(String, int, int)

Declaration:

```
public void write(java.lang.String str, int off, int len)
            throws IOException
```

Description:
Write a portion of a string.

Parameters:
`str` - A String

`off` - Offset from which to start writing characters

`len` - Number of characters to write

Throws:

`IOException` - If an I/O error occurs

flush()

Declaration:

```
public abstract void flush()
            throws IOException
```

Description:

Flush the stream. If the stream has saved any characters from the various write() methods in a buffer, write them immediately to their intended destination. Then, if that destination is another character or byte stream, flush it. Thus one flush() invocation will flush all the buffers in a chain of Writers and OutputStreams.

Throws:

`IOException` - If an I/O error occurs

close()

Declaration:

```
public abstract void close()
            throws IOException
```

Description:

Close the stream, flushing it first. Once a stream has been closed, further write() or flush() invocations will cause an IOException to be thrown. Closing a previously-closed stream, however, has no effect.

Throws:

`IOException` - If an I/O error occurs

Writer java.io

close()

CHAPTER 2

Package java.lang

Description

Provides classes that are fundamental to the Java programming language.

Since: CLDC 1.0

Class Summary	
Interfaces	
`Runnable`	The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread.
Classes	
`Boolean`	The Boolean class wraps a value of the primitive type `boolean` in an object.
`Byte`	The Byte class is the standard wrapper for byte values.
`Character`	The Character class wraps a value of the primitive type `char` in an object.
`Class`	Instances of the class `Class` represent classes and interfaces in a running Java application.
`Double`	The Double class wraps a value of the primitive type `double` in an object.
`Float`	The Float class wraps a value of primitive type `float` in an object.
`Integer`	The Integer class wraps a value of the primitive type `int` in an object.
`Long`	The Long class wraps a value of the primitive type `long` in an object.
`Math`	The class `Math` contains methods for performing basic numeric operations.
`Object`	Class `Object` is the root of the class hierarchy.
`Runtime`	Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.
`Short`	The Short class is the standard wrapper for short values.
`String`	The `String` class represents character strings.
`StringBuffer`	A string buffer implements a mutable sequence of characters.
`System`	The `System` class contains several useful class fields and methods.
`Thread`	A *thread* is a thread of execution in a program.
`Throwable`	The `Throwable` class is the superclass of all errors and exceptions in the Java language.

Class Summary

Exceptions

ArithmeticException	Thrown when an exceptional arithmetic condition has occurred.
ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been accessed with an illegal index.
ArrayStoreException	Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.
ClassCastException	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
ClassNotFoundException	Thrown when an application tries to load in a class through its string name using the `forName` method in class `Class` but no definition for the class with the specified name could be found.
Exception	The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.
IllegalAccessException	Thrown when an application tries to load in a class, but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.
IllegalArgumentException	Thrown to indicate that a method has been passed an illegal or inappropriate argument.
IllegalMonitorStateException	Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.
IllegalThreadStateException	Thrown to indicate that a thread is not in an appropriate state for the requested operation.
IndexOutOfBoundsException	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.
InstantiationException	Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.
InterruptedException	Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it.
NegativeArraySizeException	Thrown if an application tries to create an array with negative size.
NullPointerException	Thrown when an application attempts to use `null` in a case where an object is required.
NumberFormatException	Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.
RuntimeException	`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.
SecurityException	Thrown by the system to indicate a security violation.
StringIndexOutOfBoundsException	Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.

Errors

Class Summary	
`Error`	An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch.
`NoClassDefFoundError`	Thrown if the Java Virtual Machine tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the `new` expression) and no definition of the class could be found.
`OutOfMemoryError`	Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.
`VirtualMachineError`	Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

java.lang
ArithmeticException

Declaration

```
public class ArithmeticException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ArithmeticException
```

Description

Thrown when an exceptional arithmetic condition has occurred. For example, an integer “divide by zero” throws an instance of this class.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`ArithmeticException()`
	`ArithmeticException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ArithmeticException()

Declaration:

```
public ArithmeticException()
```

Description:
Constructs an `ArithmeticException` with no detail message.

ArithmeticException(String)

Declaration:
```
public ArithmeticException(java.lang.String s)
```

Description:
Constructs an `ArithmeticException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
ArrayIndexOutOfBoundsException

Declaration
```
public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
                          |
                          +--java.lang.ArrayIndexOutOfBoundsException
```

Description
Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`ArrayIndexOutOfBoundsException()`
	`ArrayIndexOutOfBoundsException(int index)`
	`ArrayIndexOutOfBoundsException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ArrayIndexOutOfBoundsException()

Declaration:
```
public ArrayIndexOutOfBoundsException()
```

Description:
Constructs an `ArrayIndexOutOfBoundsException` with no detail message.

ArrayIndexOutOfBoundsException(int)

Declaration:
```
public ArrayIndexOutOfBoundsException(int index)
```

Description:
Constructs a new `ArrayIndexOutOfBoundsException` class with an argument indicating the illegal index.

Parameters:
`index` - the illegal index.

ArrayIndexOutOfBoundsException(String)

Declaration:
```
public ArrayIndexOutOfBoundsException(java.lang.String s)
```

Description:
Constructs an `ArrayIndexOutOfBoundsException` class with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
ArrayStoreException

Declaration

```
public class ArrayStoreException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ArrayStoreException
```

Description

Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an `ArrayStoreException`:

```
Object x[] = new String[3];
x[0] = new Integer(0);
```

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`ArrayStoreException()`
	`ArrayStoreException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ArrayStoreException()

Declaration:
```
public ArrayStoreException()
```

Description:
Constructs an `ArrayStoreException` with no detail message.

ArrayStoreException(String)

Declaration:
```
public ArrayStoreException(java.lang.String s)
```

Description:
Constructs an `ArrayStoreException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Boolean

Declaration
```
public final class Boolean

java.lang.Object
  |
  +--java.lang.Boolean
```

Description
The Boolean class wraps a value of the primitive type `boolean` in an object. An object of type `Boolean` contains a single field whose type is `boolean`.

Since: JDK1.0, CLDC 1.0

Member Summary	
Fields	
`static Boolean`	`FALSE`
`static Boolean`	`TRUE`
Constructors	
	`Boolean(boolean value)`
Methods	
`boolean`	`booleanValue()`
`boolean`	`equals(Object obj)`
`int`	`hashCode()`
`String`	`toString()`

Inherited Member Summary
Methods inherited from class `Object`
`getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

TRUE

Declaration:
```
public static final java.lang.Boolean TRUE
```

Description:
The `Boolean` object corresponding to the primitive value `true`.

FALSE

Declaration:
```
public static final java.lang.Boolean FALSE
```

Description:
The `Boolean` object corresponding to the primitive value `false`.

Constructors

Boolean(boolean)

Declaration:
```
public Boolean(boolean value)
```

Description:
Allocates a `Boolean` object representing the `value` argument.

Parameters:
`value` - the value of the `Boolean`.

Methods

booleanValue()

Declaration:
```
public boolean booleanValue()
```

Description:
Returns the value of this `Boolean` object as a boolean primitive.

Returns: the primitive `boolean` value of this object.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String object representing this Boolean's value. If this object represents the value `true`, a string equal to `"true"` is returned. Otherwise, a string equal to `"false"` is returned.

Overrides: `toString` in class `Object`

Returns: a string representation of this object.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hash code for this `Boolean` object.

Overrides: `hashCode` in class `Object`

Returns: the integer `1231` if this object represents `true`; returns the integer `1237` if this object represents `false`.

equals(Object)

Declaration:

```
public boolean equals(java.lang.Object obj)
```

Description:

Returns `true` if and only if the argument is not `null` and is a `Boolean` object that represents the same `boolean` value as this object.

Overrides: `equals` in class `Object`

Parameters:

`obj` - the object to compare with.

Returns: `true` if the Boolean objects represent the same value; `false` otherwise.

java.lang
Byte

Declaration
```
public final class Byte
```

```
java.lang.Object
  |
  +--java.lang.Byte
```

Description
The Byte class is the standard wrapper for byte values.

Since: JDK1.1, CLDC 1.0

Member Summary	
Fields	
static byte	MAX_VALUE
static byte	MIN_VALUE
Constructors	
	Byte(byte value)
Methods	
byte	byteValue()
boolean	equals(Object obj)
int	hashCode()
static byte	parseByte(String s)
static byte	parseByte(String s, int radix)
String	toString()

Inherited Member Summary
Methods inherited from class Object
getClass(), notify(), notifyAll(), wait(), wait(), wait()

Fields

MIN_VALUE

Declaration:
```
public static final byte MIN_VALUE
```

MAX_VALUE

Description:
The minimum value a Byte can have.

MAX_VALUE

Declaration:
```
public static final byte MAX_VALUE
```

Description:
The maximum value a Byte can have.

Constructors

Byte(byte)

Declaration:
```
public Byte(byte value)
```

Description:
Constructs a Byte object initialized to the specified byte value.

Parameters:
`value` - the initial value of the Byte

Methods

parseByte(String)

Declaration:
```
public static byte parseByte(java.lang.String s)
            throws NumberFormatException
```

Description:
Assuming the specified String represents a byte, returns that byte's value. Throws an exception if the String cannot be parsed as a byte. The radix is assumed to be 10.

Parameters:
`s` - the String containing the byte

Returns: the parsed value of the byte

Throws:
`NumberFormatException` - If the string does not contain a parsable byte.

parseByte(String, int)

Declaration:
```
public static byte parseByte(java.lang.String s, int radix)
            throws NumberFormatException
```

Description:
Assuming the specified String represents a byte, returns that byte's value. Throws an exception if the String cannot be parsed as a byte.

Parameters:
`s` - the String containing the byte

`radix` - the radix to be used

Returns: the parsed value of the byte

Throws:
`NumberFormatException` - If the String does not contain a parsable byte.

byteValue()

Declaration:
```
public byte byteValue()
```

Description:
Returns the value of this Byte as a byte.

Returns: the value of this Byte as a byte.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String object representing this Byte's value.

Overrides: `toString` in class `Object`

Returns: a string representation of the object.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hashcode for this Byte.

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this object to the specified object.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with

Returns: true if the objects are the same; false otherwise.

java.lang
Character

Declaration

```
public final class Character
```

```
java.lang.Object
  |
  +--java.lang.Character
```

Description

The Character class wraps a value of the primitive type `char` in an object. An object of type `Character` contains a single field whose type is `char`.

In addition, this class provides several methods for determining the type of a character and converting characters from uppercase to lowercase and vice versa.

Character information is based on the Unicode Standard, version 3.0. However, in order to reduce footprint, by default the character property and case conversion operations in CLDC are available only for the ISO Latin-1 range of characters. Other Unicode character blocks can be supported as necessary.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

static int	MAX_RADIX
static char	MAX_VALUE
static int	MIN_RADIX
static char	MIN_VALUE

Constructors

	Character(char value)

Methods

char	charValue()
static int	digit(char ch, int radix)
boolean	equals(Object obj)
int	hashCode()
static boolean	isDigit(char ch)
static boolean	isLowerCase(char ch)
static boolean	isUpperCase(char ch)
static char	toLowerCase(char ch)
String	toString()
static char	toUpperCase(char ch)

Inherited Member Summary
Methods inherited from class `Object` `getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

MIN_RADIX

Declaration:
```
public static final int MIN_RADIX
```

Description:
The minimum radix available for conversion to and from Strings.

See Also: `Integer.toString(int, int)`, `Integer.valueOf(String)`

MAX_RADIX

Declaration:
```
public static final int MAX_RADIX
```

Description:
The maximum radix available for conversion to and from Strings.

See Also: `Integer.toString(int, int)`, `Integer.valueOf(String)`

MIN_VALUE

Declaration:
```
public static final char MIN_VALUE
```

Description:
The constant value of this field is the smallest value of type `char`.

Since: JDK1.0.2

MAX_VALUE

Declaration:
```
public static final char MAX_VALUE
```

Description:
The constant value of this field is the largest value of type `char`.

Since: JDK1.0.2

Constructors

Character(char)

Declaration:
```
public Character(char value)
```

Description:
Constructs a `Character` object and initializes it so that it represents the primitive `value` argument.

Parameters:
`value` - value for the new `Character` object.

Methods

charValue()

Declaration:
```
public char charValue()
```

Description:
Returns the value of this Character object.

Returns: the primitive `char` value represented by this object.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hash code for this Character.

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and is a `Character` object that represents the same `char` value as this object.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String object representing this character's value. Converts this `Character` object to a string. The result is a string whose length is `1`. The string's sole component is the primitive `char` value represented by this object.

Overrides: `toString` in class `Object`

Returns: a string representation of this object.

isLowerCase(char)

Declaration:
```
public static boolean isLowerCase(char ch)
```

Description:
Determines if the specified character is a lowercase character.

Note that by default CLDC only supports the ISO Latin-1 range of characters.

Of the ISO Latin-1 characters (character codes 0x0000 through 0x00FF), the following are lowercase:

a b c d e f g h i j k l m n o p q r s t u v w x y z \u00DF \u00E0 \u00E1 \u00E2 \u00E3 \u00E4 \u00E5 \u00E6 \u00E7 \u00E8 \u00E9 \u00EA \u00EB \u00EC \u00ED \u00EE \u00EF \u00F0 \u00F1 \u00F2 \u00F3 \u00F4 \u00F5 \u00F6 \u00F8 \u00F9 \u00FA \u00FB \u00FC \u00FD \u00FE \u00FF

Parameters:
`ch` - the character to be tested.

Returns: `true` if the character is lowercase; `false` otherwise.

Since: JDK1.0

isUpperCase(char)

Declaration:
```
public static boolean isUpperCase(char ch)
```

Description:
Determines if the specified character is an uppercase character.

Note that by default CLDC only supports the ISO Latin-1 range of characters.

Of the ISO Latin-1 characters (character codes 0x0000 through 0x00FF), the following are uppercase:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z \u00C0 \u00C1 \u00C2 \u00C3 \u00C4 \u00C5 \u00C6 \u00C7 \u00C8 \u00C9 \u00CA \u00CB \u00CC \u00CD \u00CE \u00CF \u00D0 \u00D1 \u00D2 \u00D3 \u00D4 \u00D5 \u00D6 \u00D8 \u00D9 \u00DA \u00DB \u00DC \u00DD \u00DE

Parameters:
`ch` - the character to be tested.

Returns: `true` if the character is uppercase; `false` otherwise.

Since: 1.0

See Also: `isLowerCase(char)`, `toUpperCase(char)`

isDigit(char)

Declaration:
```
public static boolean isDigit(char ch)
```

Description:
Determines if the specified character is a digit.

Parameters:
`ch` - the character to be tested.

Returns: `true` if the character is a digit; `false` otherwise.

Since: JDK1.0

toLowerCase(char)

Declaration:
```
public static char toLowerCase(char ch)
```

Description:
The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned.

Note that by default CLDC only supports the ISO Latin-1 range of characters.

Parameters:
`ch` - the character to be converted.

Returns: the lowercase equivalent of the character, if any; otherwise the character itself.

Since: JDK1.0

See Also: `isLowerCase(char)`, `isUpperCase(char)`, `toUpperCase(char)`

toUpperCase(char)

Declaration:
```
public static char toUpperCase(char ch)
```

Description:
Converts the character argument to uppercase; if the character has no uppercase equivalent, the character itself is returned.

Note that by default CLDC only supports the ISO Latin-1 range of characters.

Parameters:
`ch` - the character to be converted.

Returns: the uppercase equivalent of the character, if any; otherwise the character itself.

Since: JDK1.0

See Also: `isLowerCase(char)`, `isUpperCase(char)`, `toLowerCase(char)`

digit(char, int)

Declaration:
```
public static int digit(char ch, int radix)
```

Description:
Returns the numeric value of the character `ch` in the specified radix.

Parameters:
`ch` - the character to be converted.

`radix` - the radix.

Returns: the numeric value represented by the character in the specified radix.

Since: JDK1.0

See Also: `isDigit(char)`

java.lang
Class

Declaration

```
public final class Class
```

```
java.lang.Object
  |
  +--java.lang.Class
```

Description

Instances of the class `Class` represent classes and interfaces in a running Java application. Every array also belongs to a class that is reflected as a `Class` object that is shared by all arrays with the same element type and number of dimensions.

`Class` has no public constructor. Instead `Class` objects are constructed automatically by the Java Virtual Machine as classes are loaded.

The following example uses a `Class` object to print the class name of an object:

```
void printClassName(Object obj) {
    System.out.println("The class of " + obj +
                       " is " + obj.getClass().getName());
}
```

Since: JDK1.0, CLDC 1.0

Member Summary

Methods

static Class	forName(String className)
String	getName()
java.io.InputStream	getResourceAsStream(String name)
boolean	isArray()
boolean	isAssignableFrom(Class cls)
boolean	isInstance(Object obj)
boolean	isInterface()
Object	newInstance()
String	toString()

Inherited Member Summary

Methods inherited from class `Object`

equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Methods

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Converts the object to a string. The string representation is the string “class” or “interface”, followed by a space, and then by the fully qualified name of the class in the format returned by `getName`. If this `Class` object represents a primitive type, this method returns the name of the primitive type. If this `Class` object represents void this method returns “void”.

Overrides: `toString` in class `Object`

Returns: a string representation of this class object.

forName(String)

Declaration:
```
public static java.lang.Class forName(java.lang.String className)
            throws ClassNotFoundException
```

Description:
Returns the `Class` object associated with the class with the given string name. Given the fully-qualified name for a class or interface, this method attempts to locate, load and link the class.

For example, the following code fragment returns the runtime `Class` descriptor for the class named `java.lang.Thread`: `Class t = Class.forName(“java.lang.Thread”)`

Parameters:
`className` - the fully qualified name of the desired class.

Returns: the `Class` object for the class with the specified name.

Throws:
`ClassNotFoundException` - if the class could not be found.

`Error` - if the function fails for any other reason.

Since: JDK1.0

newInstance()

Declaration:
```
public java.lang.Object newInstance()
            throws InstantiationException, IllegalAccessException
```

Description:
Creates a new instance of a class.

Returns: a newly allocated instance of the class represented by this object. This is done exactly as if by a `new` expression with an empty argument list.

Throws:
`IllegalAccessException` - if the class or initializer is not accessible.

`InstantiationException` - if an application tries to instantiate an abstract class or an interface, or if the instantiation fails for some other reason.

Since: JDK1.0

isInstance(Object)

Declaration:
```
public boolean isInstance(java.lang.Object obj)
```

Description:
Determines if the specified `Object` is assignment-compatible with the object represented by this `Class`. This method is the dynamic equivalent of the Java language `instanceof` operator. The method returns `true` if the specified `Object` argument is non-null and can be cast to the reference type represented by this `Class` object without raising a `ClassCastException.` It returns `false` otherwise.

Specifically, if this `Class` object represents a declared class, this method returns `true` if the specified `Object` argument is an instance of the represented class (or of any of its subclasses); it returns `false` otherwise. If this `Class` object represents an array class, this method returns `true` if the specified `Object` argument can be converted to an object of the array class by an identity conversion or by a widening reference conversion; it returns `false` otherwise. If this `Class` object represents an interface, this method returns `true` if the class or any superclass of the specified `Object` argument implements this interface; it returns `false` otherwise. If this `Class` object represents a primitive type, this method returns `false`.

Parameters:
`obj` - the object to check

Returns: true if `obj` is an instance of this class

Since: JDK1.1

isAssignableFrom(Class)

Declaration:
```
public boolean isAssignableFrom(java.lang.Class cls)
```

Description:
Determines if the class or interface represented by this `Class` object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter. It returns `true` if so; otherwise it returns `false`. If this `Class` object represents a primitive type, this method returns `true` if the specified `Class` parameter is exactly this `Class` object; otherwise it returns `false`.

Specifically, this method tests whether the type represented by the specified `Class` parameter can be converted to the type represented by this `Class` object via an identity conversion or via a widening reference conversion. See *The Java Language Specification*, sections 5.1.1 and 5.1.4 , for details.

Parameters:
`cls` - the `Class` object to be checked

Returns: the `boolean` value indicating whether objects of the type `cls` can be assigned to objects of this class

Throws:
`NullPointerException` - if the specified Class parameter is null.

Since: JDK1.1

isInterface()

Declaration:
```
public boolean isInterface()
```

Description:
Determines if the specified `Class` object represents an interface type.

Returns: `true` if this object represents an interface; `false` otherwise.

isArray()

Declaration:
```
public boolean isArray()
```

Description:
Determines if this `Class` object represents an array class.

Returns: `true` if this object represents an array class; `false` otherwise.

Since: JDK1.1

getName()

Declaration:
```
public java.lang.String getName()
```

Description:
Returns the fully-qualified name of the entity (class, interface, array class, primitive type, or void) represented by this `Class` object, as a `String`.

If this `Class` object represents a class of arrays, then the internal form of the name consists of the name of the element type in Java signature format, preceded by one or more "`[`" characters representing the depth of array nesting. Thus:

```
(new Object[3]).getClass().getName()
```

returns "`[Ljava.lang.Object;`" and:

```
(new int[3][4][5][6][7][8][9]).getClass().getName()
```

returns "`[[[[[[[I`". The encoding of element type names is as follows:

```
B            byte
C            char
D            double
F            float
I            int
J            long
L<i>classname;</i>  class or interface
S            short
Z            boolean
```

The class or interface name `classname` is given in fully qualified form as shown in the example above.

Returns: the fully qualified name of the class or interface represented by this object.

getResourceAsStream(String)

Declaration:
```
public java.io.InputStream getResourceAsStream(java.lang.String name)
```

Description:
Finds a resource with a given name in the application's JAR file. This method returns `null` if no resource with this name is found in the application's JAR file.

The resource names can be represented in two different formats: absolute or relative.

Absolute format: `/packagePathName/resourceName`

Relative format: `resourceName`

In the absolute format, the programmer provides a fully qualified name that includes both the full path and the name of the resource inside the JAR file. In the path names, the character "/" is used as the separator.

In the relative format, the programmer provides only the name of the actual resource. Relative names are converted to absolute names by the system by prepending the resource name with the fully qualified package name of class upon which the `getResourceAsStream` method was called.

Parameters:
`name` - name of the desired resource

Returns: a `java.io.InputStream` object.

java.lang
ClassCastException

Declaration

```
public class ClassCastException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.ClassCastException
```

Description

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a `ClassCastException`:

```
Object x = new Integer(0);
System.out.println((String)x);
```

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`ClassCastException()`
	`ClassCastException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ClassCastException()

Declaration:
```
public ClassCastException()
```

Description:
Constructs a `ClassCastException` with no detail message.

ClassCastException(String)

Declaration:
```
public ClassCastException(java.lang.String s)
```

Description:
Constructs a `ClassCastException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
ClassNotFoundException

Declaration
```
public class ClassNotFoundException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.ClassNotFoundException
```

Description
Thrown when an application tries to load in a class through its string name using the `forName` method in class `Class` but no definition for the class with the specified name could be found.

Since: JDK1.0, CLDC 1.0

See Also: `Class.forName(String)`

Member Summary	
Constructors	
	`ClassNotFoundException()`
	`ClassNotFoundException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ClassNotFoundException()

Declaration:
```
public ClassNotFoundException()
```

Description:
Constructs a `ClassNotFoundException` with no detail message.

ClassNotFoundException(String)

Declaration:
```
public ClassNotFoundException(java.lang.String s)
```

Description:
Constructs a `ClassNotFoundException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Double

Declaration

```
public final class Double
```

```
java.lang.Object
  |
  +--java.lang.Double
```

Description

The Double class wraps a value of the primitive type `double` in an object. An object of type `Double` contains a single field whose type is `double`.

In addition, this class provides several methods for converting a `double` to a `String` and a `String` to a `double`, as well as other constants and methods useful when dealing with a `double`.

Since: JDK1.0, CLDC 1.1

Member Summary

Fields

static double	MAX_VALUE
static double	MIN_VALUE
static double	NaN
static double	NEGATIVE_INFINITY
static double	POSITIVE_INFINITY

Constructors

	Double(double value)

Methods

byte	byteValue()
static long	doubleToLongBits(double value)
double	doubleValue()
boolean	equals(Object obj)
float	floatValue()
int	hashCode()
int	intValue()
boolean	isInfinite()
static boolean	isInfinite(double v)
boolean	isNaN()
static boolean	isNaN(double v)
static double	longBitsToDouble(long bits)
long	longValue()
static double	parseDouble(String s)
short	shortValue()
String	toString()
static String	toString(double d)
static Double	valueOf(String s)

Inherited Member Summary
Methods inherited from class `Object` `getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

POSITIVE_INFINITY

Declaration:
`public static final double POSITIVE_INFINITY`

Description:
The positive infinity of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x7ff0000000000000L)`.

NEGATIVE_INFINITY

Declaration:
`public static final double NEGATIVE_INFINITY`

Description:
The negative infinity of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0xfff0000000000000L)`.

NaN

Declaration:
`public static final double NaN`

Description:
A Not-a-Number (NaN) value of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x7ff8000000000000L)`.

MAX_VALUE

Declaration:
`public static final double MAX_VALUE`

Description:
The largest positive finite value of type `double`. It is equal to the value returned by

```
Double.longBitsToDouble(0x7fefffffffffffL)
```

MIN_VALUE

Declaration:
`public static final double MIN_VALUE`

Double(double)

Description:
The smallest positive value of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x1L)`.

Constructors

Double(double)

Declaration:
```
public Double(double value)
```

Description:
Constructs a newly allocated `Double` object that represents the primitive `double` argument.

Parameters:
`value` - the value to be represented by the `Double`.

Methods

toString(double)

Declaration:
```
public static java.lang.String toString(double d)
```

Description:
Creates a string representation of the `double` argument. All characters mentioned below are ASCII characters.

- If the argument is NaN, the result is the string “NaN”.
- Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is `'-'` (`'-'`); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
- If *m* is infinity, it is represented by the characters `“Infinity”`; thus, positive infinity produces the result `“Infinity”` and negative infinity produces the result `“-Infinity”`.
- If *m* is zero, it is represented by the characters `“0.0”`; thus, negative zero produces the result `“-0.0”` and positive zero produces the result `“0.0”`.
- If *m* is greater than or equal to 10^{-3} but less than 10^{7}, then it is represented as the integer part of *m*, in decimal form with no leading zeroes, followed by `'.'` (`.`), followed by one or more decimal digits representing the fractional part of *m*.
- If *m* is less than 10^{-3} or not less than 10^{7}, then it is represented in so-called “computerized scientific notation.” Let *n* be the unique integer such that $10^{n} <= m < 10^{n+1}$; then let *a* be the mathematically exact quotient of *m* and 10^{n} so that $1 <= a < 10$. The magnitude is then represented as the integer part of *a*, as a single decimal digit, followed by `'.'` (`.`), followed by decimal digits representing the fractional part of *a*, followed by the letter `'E'` (`E`), followed by a representation of *n* as a decimal integer, as produced by the method `Integer.toString(int)`.

How many digits must be printed for the fractional part of *m* or *a*? There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to

uniquely distinguish the argument value from adjacent values of type `double`. That is, suppose that *x* is the exact mathematical value represented by the decimal representation produced by this method for a finite nonzero argument *d*. Then *d* must be the `double` value nearest to *x*; or if two `double` values are equally close to *x*, then *d* must be one of them and the least significant bit of the significand of *d* must be `0`.

Parameters:
`d` - the `double` to be converted.

Returns: a string representation of the argument.

valueOf(String)

Declaration:
```
public static java.lang.Double valueOf(java.lang.String s)
            throws NumberFormatException
```

Description:
Returns a new `Double` object initialized to the value represented by the specified string. The string `s` is interpreted as the representation of a floating-point value and a `Double` object representing that value is created and returned.

If `s` is `null`, then a `NullPointerException` is thrown.

Leading and trailing whitespace characters in s are ignored. The rest of `s` should constitute a *FloatValue* as described by the lexical rule:

```
FloatValue:
       Signopt FloatingPointLiteral
```

where *Sign* and *FloatingPointLiteral* are as defined in Section 3.10.2 of the Java Language Specification (http://java.sun.com/docs/books/jls/html/). If it does not have the form of a *FloatValue*, then a `NumberFormatException` is thrown. Otherwise, it is regarded as representing an exact decimal value in the usual "computerized scientific notation"; this exact decimal value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type `double` by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic. Finally, a new object of class `Double` is created to represent the `double` value.

Parameters:
`s` - the string to be parsed.

Returns: a newly constructed `Double` initialized to the value represented by the string argument.

Throws:
`NumberFormatException` - if the string does not contain a parsable number.

parseDouble(String)

Declaration:
```
public static double parseDouble(java.lang.String s)
            throws NumberFormatException
```

Description:
Returns a new double initialized to the value represented by the specified `String`, as performed by the `valueOf` method of class `Double`.

Parameters:
`s` - the string to be parsed.

Returns: the double value represented by the string argument.

Throws:
`NumberFormatException` - if the string does not contain a parsable double.

Since: JDK1.2

See Also: `valueOf(String)`

isNaN(double)

Declaration:
```
public static boolean isNaN(double v)
```

Description:
Returns true if the specified number is the special Not-a-Number (NaN) value.

Parameters:
`v` - the value to be tested.

Returns: `true` if the value of the argument is NaN; `false` otherwise.

isInfinite(double)

Declaration:
```
public static boolean isInfinite(double v)
```

Description:
Returns true if the specified number is infinitely large in magnitude.

Parameters:
`v` - the value to be tested.

Returns: `true` if the value of the argument is positive infinity or negative infinity; `false` otherwise.

isNaN()

Declaration:
```
public boolean isNaN()
```

Description:
Returns true if this Double value is the special Not-a-Number (NaN) value.

Returns: `true` if the value represented by this object is NaN; `false` otherwise.

isInfinite()

Declaration:
```
public boolean isInfinite()
```

Description:
Returns true if this Double value is infinitely large in magnitude.

Returns: `true` if the value represented by this object is positive infinity or negative infinity; `false` otherwise.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String representation of this Double object. The primitive `double` value represented by this object is converted to a string exactly as if by the method `toString` of one argument.

Overrides: `toString` in class `Object`

Returns: a `String` representation of this object.

See Also: `toString(double)`

byteValue()

Declaration:
```
public byte byteValue()
```

Description:
Returns the value of this Double as a byte (by casting to a byte).

Since: JDK1.1

shortValue()

Declaration:
```
public short shortValue()
```

Description:
Returns the value of this Double as a short (by casting to a short).

Since: JDK1.1

intValue()

Declaration:
```
public int intValue()
```

Description:
Returns the integer value of this Double (by casting to an int).

Returns: the `double` value represented by this object is converted to type `int` and the result of the conversion is returned.

longValue()

Declaration:
```
public long longValue()
```

Description:
Returns the long value of this Double (by casting to a long).

Returns: the `double` value represented by this object is converted to type `long` and the result of the conversion is returned.

floatValue()

Declaration:
```
public float floatValue()
```

Description:
Returns the float value of this Double.

doubleValue()

Returns: the `double` value represented by this object is converted to type `float` and the result of the conversion is returned.

Since: JDK1.0

doubleValue()

Declaration:
`public double doubleValue()`

Description:
Returns the double value of this Double.

Returns: the `double` value represented by this object.

hashCode()

Declaration:
`public int hashCode()`

Description:
Returns a hashcode for this `Double` object. The result is the exclusive OR of the two halves of the long integer bit representation, exactly as produced by the method `doubleToLongBits(double)`, of the primitive `double` value represented by this `Double` object. That is, the hashcode is the value of the expression:

```
(int)(v^(v>>>32))
```

where `v` is defined by:

```
long v = Double.doubleToLongBits(this.doubleValue());
```

Overrides: `hashCode` in class `Object`

Returns: a `hash code` value for this object.

equals(Object)

Declaration:
`public boolean equals(java.lang.Object obj)`

Description:
Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and is a `Double` object that represents a double that has the identical bit pattern to the bit pattern of the double represented by this object. For this purpose, two `double` values are considered to be the same if and only if the method `doubleToLongBits(double)` returns the same long value when applied to each.

Note that in most cases, for two instances of class `Double`, `d1` and `d2`, the value of `d1.equals(d2)` is `true` if and only if

```
d1.doubleValue() == d2.doubleValue()
```

also has the value `true`. However, there are two exceptions:

- If `d1` and `d2` both represent `Double.NaN`, then the `equals` method returns `true`, even though `Double.NaN==Double.NaN` has the value `false`.
- If `d1` represents `+0.0` while `d2` represents `-0.0`, or vice versa, the `equals` test has the value `false`, even though `+0.0==-0.0` has the value `true`. This allows hashtables to operate properly.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

doubleToLongBits(double)

Declaration:
```
public static long doubleToLongBits(double value)
```

Description:
Returns a representation of the specified floating-point value according to the IEEE 754 floating-point "double format" bit layout.

Bit 63 (the bit that is selected by the mask `0x8000000000000000L`) represents the sign of the floating-point number. Bits 62-52 (the bits that are selected by the mask `0x7ff0000000000000L`) represent the exponent. Bits 51-0 (the bits that are selected by the mask `0x000fffffffffffffL`) represent the significand (sometimes called the mantissa) of the floating-point number.

If the argument is positive infinity, the result is `0x7ff0000000000000L`.

If the argument is negative infinity, the result is `0xfff0000000000000L`.

If the argument is NaN, the result is `0x7ff8000000000000L`.

In all cases, the result is a `long` integer that, when given to the `longBitsToDouble(long)` method, will produce a floating-point value equal to the argument to `doubleToLongBits`.

Parameters:
`value` - a double precision floating-point number.

Returns: the bits that represent the floating-point number.

longBitsToDouble(long)

Declaration:
```
public static double longBitsToDouble(long bits)
```

Description:
Returns the double-float corresponding to a given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "double precision" bit layout. That floating-point value is returned as the result.

If the argument is `0x7ff0000000000000L`, the result is positive infinity.

If the argument is `0xfff0000000000000L`, the result is negative infinity.

If the argument is any value in the range `0x7ff0000000000001L` through `0x7fffffffffffffffL` or in the range `0xfff0000000000001L` through `0xffffffffffffffffL`, the result is NaN. All IEEE 754 NaN values of type `double` are, in effect, lumped together by the Java programming language into a single value called NaN.

In all other cases, let *s*, *e*, and *m* be three values that can be computed from the argument:

longBitsToDouble(long)

```
int s = ((bits >> 63) == 0) ? 1 : -1;
int e = (int)((bits >> 52) & 0x7ffL);
long m = (e == 0) ?
                (bits & 0xfffffffffffffL) << 1 :
                (bits & 0xfffffffffffffL) | 0x10000000000000L;
```

Then the floating-point result equals the value of the mathematical expression *s*·*m*·2^{e-1075}.

Parameters:

`bits` - any `long` integer.

Returns: the `double` floating-point value with the same bit pattern.

java.lang
Error

Declaration
```
public class Error extends Throwable
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
```

Direct Known Subclasses: `NoClassDefFoundError`, `VirtualMachineError`

Description
An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

A method is not required to declare in its `throws` clause any subclasses of `Error` that might be thrown during the execution of the method but not caught, since these errors are abnormal conditions that should never occur.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`Error()`
	`Error(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

Error()

Declaration:
```
public Error()
```

Error(String)

Description:
Constructs an `Error` with no specified detail message.

Error(String)

Declaration:
```
public Error(java.lang.String s)
```

Description:
Constructs an Error with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Exception

Declaration
```
public class Exception extends Throwable
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
```

Direct Known Subclasses: `ClassNotFoundException`, `IllegalAccessException`, `InstantiationException`, `InterruptedException`, `java.io.IOException`, `RuntimeException`

Description
The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

Since: JDK1.0, CLDC 1.0

See Also: `Error`

Member Summary	
Constructors	
	`Exception()`
	`Exception(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Exception()

Constructors

Exception()

Declaration:
```
public Exception()
```

Description:
Constructs an `Exception` with no specified detail message.

Exception(String)

Declaration:
```
public Exception(java.lang.String s)
```

Description:
Constructs an `Exception` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Float

Declaration

```
public final class Float
```

```
java.lang.Object
  |
  +--java.lang.Float
```

Description

The Float class wraps a value of primitive type `float` in an object. An object of type `Float` contains a single field whose type is `float`.

In addition, this class provides several methods for converting a `float` to a `String` and a `String` to a `float`, as well as other constants and methods useful when dealing with a `float`.

Since: JDK1.0, CLDC 1.1

Member Summary

Fields

static float	MAX_VALUE
static float	MIN_VALUE
static float	NaN
static float	NEGATIVE_INFINITY
static float	POSITIVE_INFINITY

Constructors

	Float(double value)
	Float(float value)

Methods

byte	byteValue()
double	doubleValue()
boolean	equals(Object obj)
static int	floatToIntBits(float value)
float	floatValue()
int	hashCode()
static float	intBitsToFloat(int bits)
int	intValue()
boolean	isInfinite()
static boolean	isInfinite(float v)
boolean	isNaN()
static boolean	isNaN(float v)
long	longValue()
static float	parseFloat(String s)
short	shortValue()
String	toString()
static String	toString(float f)

Member Summary
`static Float valueOf(String s)`

Inherited Member Summary
Methods inherited from class `Object` `getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

POSITIVE_INFINITY

Declaration:
`public static final float POSITIVE_INFINITY`

Description:
The positive infinity of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7f800000)`.

NEGATIVE_INFINITY

Declaration:
`public static final float NEGATIVE_INFINITY`

Description:
The negative infinity of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0xff800000)`.

NaN

Declaration:
`public static final float NaN`

Description:
The Not-a-Number (NaN) value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7fc00000)`.

MAX_VALUE

Declaration:
`public static final float MAX_VALUE`

Description:
The largest positive value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7f7fffff)`.

MIN_VALUE

Declaration:
`public static final float MIN_VALUE`

Description:
The smallest positive value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x1)`.

Constructors

Float(float)

Declaration:
```
public Float(float value)
```

Description:
Constructs a newly allocated `Float` object that represents the primitive `float` argument.

Parameters:
`value` - the value to be represented by the `Float`.

Float(double)

Declaration:
```
public Float(double value)
```

Description:
Constructs a newly allocated `Float`object that represents the argument converted to type `float`.

Parameters:
`value` - the value to be represented by the `Float`.

Methods

toString(float)

Declaration:
```
public static java.lang.String toString(float f)
```

Description:
Returns a String representation for the specified float value. The argument is converted to a readable string format as follows. All characters and characters in strings mentioned below are ASCII characters.

- If the argument is NaN, the result is the string `“NaN”`.
- Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is `'-'` (`'-'`); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
  - If *m* is infinity, it is represented by the characters `“Infinity”`; thus, positive infinity produces the result `“Infinity”` and negative infinity produces the result `“-Infinity”`.
  - If *m* is zero, it is represented by the characters `“0.0”`; thus, negative zero produces the result `“-0.0”` and positive zero produces the result `“0.0”`.
  - If *m* is greater than or equal to 10^{-3} but less than 10^{7}, then it is represented as the integer part of *m*, in decimal form with no leading zeroes, followed by `'.'` (`.`), followed by one or more decimal digits representing the fractional part of *m*.

- If *m* is less than 10^{-3} or not less than 10^{7}, then it is represented in so-called "computerized scientific notation." Let *n* be the unique integer such that $10^{n}<=m<1$; then let *a* be the mathematically exact quotient of *m* and 10^{n} so that 1<*a*&lt10. The magnitude is then represented as the integer part of *a*, as a single decimal digit, followed by `'.'` (`.`), followed by decimal digits representing the fractional part of *a*, followed by the letter `'E'` (`E`), followed by a representation of *n* as a decimal integer, as produced by the method `Integer.toString(int)` of one argument.

How many digits must be printed for the fractional part of *m* or *a*? There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to uniquely distinguish the argument value from adjacent values of type float. That is, suppose that *x* is the exact mathematical value represented by the decimal representation produced by this method for a finite nonzero argument *f*. Then *f* must be the float value nearest to *x*; or, if two float values are equally close to *x*then *f* must be one of them and the least significant bit of the significand of *f* must be `0`.

Parameters:
`f` - the float to be converted.

Returns: a string representation of the argument.

valueOf(String)

Declaration:
```
public static java.lang.Float valueOf(java.lang.String s)
            throws NumberFormatException
```

Description:
Returns the floating point value represented by the specified String. The string `s` is interpreted as the representation of a floating-point value and a `Float` object representing that value is created and returned.

If `s` is `null`, then a `NullPointerException` is thrown.

Leading and trailing whitespace characters in s are ignored. The rest of `s` should constitute a *FloatValue* as described by the lexical syntax rules:

```
FloatValue:
         Signopt FloatingPointLiteral
```

where *Sign*, *FloatingPointLiteral* are as defined in Section 3.10.2 of the Java Language Specification (http://java.sun.com/docs/books/jls/html/). If it does not have the form of a *FloatValue*, then a `NumberFormatException` is thrown. Otherwise, it is regarded as representing an exact decimal value in the usual "computerized scientific notation"; this exact decimal value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type float by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic.

Parameters:
`s` - the string to be parsed.

Returns: a newly constructed `Float` initialized to the value represented by the `String` argument.

Throws:
`NumberFormatException` - if the string does not contain a parsable number.

parseFloat(String)

Declaration:
```
public static float parseFloat(java.lang.String s)
            throws NumberFormatException
```

Description:
Returns a new float initialized to the value represented by the specified `String`.

Parameters:
`s` - the string to be parsed.

Returns: the float value represented by the string argument.

Throws:
`NumberFormatException` - if the string does not contain a parsable float.

Since: JDK1.2

isNaN(float)

Declaration:
```
public static boolean isNaN(float v)
```

Description:
Returns true if the specified number is the special Not-a-Number (NaN) value.

Parameters:
`v` - the value to be tested.

Returns: `true` if the argument is NaN; `false` otherwise.

isInfinite(float)

Declaration:
```
public static boolean isInfinite(float v)
```

Description:
Returns true if the specified number is infinitely large in magnitude.

Parameters:
`v` - the value to be tested.

Returns: `true` if the argument is positive infinity or negative infinity; `false` otherwise.

isNaN()

Declaration:
```
public boolean isNaN()
```

Description:
Returns true if this `Float` value is Not-a-Number (NaN).

Returns: `true` if the value represented by this object is NaN; `false` otherwise.

isInfinite()

Declaration:
```
public boolean isInfinite()
```

Description:
Returns true if this Float value is infinitely large in magnitude.

Returns: `true` if the value represented by this object is positive infinity or negative infinity; `false` otherwise.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String representation of this Float object. The primitive `float` value represented by this object is converted to a `String` exactly as if by the method `toString` of one argument.

Overrides: `toString` in class `Object`

Returns: a `String` representation of this object.

See Also: `toString(float)`

byteValue()

Declaration:
```
public byte byteValue()
```

Description:
Returns the value of this Float as a byte (by casting to a byte).

Since: JDK1.1

shortValue()

Declaration:
```
public short shortValue()
```

Description:
Returns the value of this Float as a short (by casting to a short).

Since: JDK1.1

intValue()

Declaration:
```
public int intValue()
```

Description:
Returns the integer value of this Float (by casting to an int).

Returns: the `float` value represented by this object converted to type `int` and the result of the conversion is returned.

longValue()

Declaration:
```
public long longValue()
```

Description:
Returns the long value of this Float (by casting to a long).

Returns: the `float` value represented by this object is converted to type `long` and the result of the conversion is returned.

floatValue()

Declaration:
```
public float floatValue()
```

Description:
Returns the float value of this `Float` object.

Returns: the `float` value represented by this object.

doubleValue()

Declaration:
```
public double doubleValue()
```

Description:
Returns the double value of this `Float` object.

Returns: the `float` value represented by this object is converted to type `double` and the result of the conversion is returned.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hashcode for this `Float` object. The result is the integer bit representation, exactly as produced by the method `floatToIntBits(float)`, of the primitive float value represented by this `Float` object.

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this object against some other object. The result is `true` if and only if the argument is not `null` and is a `Float` object that represents a `float` that has the identical bit pattern to the bit pattern of the `float` represented by this object. For this purpose, two float values are considered to be the same if and only if the method `floatToIntBits(float)` returns the same int value when applied to each.

Note that in most cases, for two instances of class `Float`, `f1` and `f2`, the value of `f1.equals(f2)` is `true` if and only if

```
f1.floatValue() == f2.floatValue()
```

also has the value `true`. However, there are two exceptions:

- If `f1` and `f2` both represent `Float.NaN`, then the `equals` method returns `true`, even though `Float.NaN==Float.NaN` has the value `false`.
- If `f1` represents `+0.0f` while `f2` represents `-0.0f`, or vice versa, the `equal` test has the value `false`, even though `0.0f==-0.0f` has the value `true`.

This definition allows hashtables to operate properly.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to be compared

Returns: `true` if the objects are the same; `false` otherwise.

See Also: `floatToIntBits(float)`

floatToIntBits(float)

Declaration:
```
public static int floatToIntBits(float value)
```

Description:
Returns the bit representation of a single-float value. The result is a representation of the floating-point argument according to the IEEE 754 floating-point “single precision” bit layout.

- Bit 31 (the bit that is selected by the mask `0x80000000`) represents the sign of the floating-point number.
- Bits 30-23 (the bits that are selected by the mask `0x7f800000`) represent the exponent.
- Bits 22-0 (the bits that are selected by the mask `0x007fffff`) represent the significand (sometimes called the mantissa) of the floating-point number.
- If the argument is positive infinity, the result is `0x7f800000`.
- If the argument is negative infinity, the result is `0xff800000`.
- If the argument is NaN, the result is `0x7fc00000`.

In all cases, the result is an integer that, when given to the `intBitsToFloat(int)` method, will produce a floating-point value equal to the argument to `floatToIntBits`.

Parameters:
`value` - a floating-point number.

Returns: the bits that represent the floating-point number.

intBitsToFloat(int)

Declaration:
```
public static float intBitsToFloat(int bits)
```

Description:
Returns the single-float corresponding to a given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point “single precision” bit layout.

If the argument is `0x7f800000`, the result is positive infinity.

If the argument is `0xff800000`, the result is negative infinity.

If the argument is any value in the range `0x7f800001` through `0x7fffffff` or in the range `0xff800001` through `0xffffffff`, the result is NaN. All IEEE 754 NaN values of type `float` are, in effect, lumped together by the Java programming language into a single `float` value called NaN.

In all other cases, let *s*, *e*, and *m* be three values that can be computed from the argument:

```
int s = ((bits >> 31) == 0) ? 1 : -1;
int e = ((bits >> 23) & 0xff);
int m = (e == 0) ?
                (bits & 0x7fffff) << 1 :
                (bits & 0x7fffff) | 0x800000;
```

Then the floating-point result equals the value of the mathematical expression *s·m·2e-150*.

Parameters:
`bits` - an integer.

Returns: the single-format floating-point value with the same bit pattern.

java.lang
IllegalAccessException

Declaration

```
public class IllegalAccessException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.IllegalAccessException
```

Description

Thrown when an application tries to load in a class, but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.

An instance of this class can also be thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the current method does not have access to the appropriate zero-argument constructor.

Since: JDK1.0, CLDC 1.0

See Also: `Class.forName(String)`, `Class.newInstance()`

Member Summary	
Constructors	
	`IllegalAccessException()`
	`IllegalAccessException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

IllegalAccessException()

Declaration:
```
public IllegalAccessException()
```

Description:
Constructs an `IllegalAccessException` without a detail message.

IllegalAccessException(String)

Declaration:
```
public IllegalAccessException(java.lang.String s)
```

Description:
Constructs an `IllegalAccessException` with a detail message.

Parameters:
`s` - the detail message.

java.lang
IllegalArgumentException

Declaration
`public class` **`IllegalArgumentException`** `extends` `RuntimeException`

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
```

Direct Known Subclasses: `IllegalThreadStateException`, `NumberFormatException`

Description
Thrown to indicate that a method has been passed an illegal or inappropriate argument.

Since: JDK1.0, CLDC 1.0

See Also: `Thread.setPriority(int)`

Member Summary	
Constructors	
	`IllegalArgumentException()`
	`IllegalArgumentException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

IllegalArgumentException()

Declaration:
```
public IllegalArgumentException()
```

Description:
Constructs an `IllegalArgumentException` with no detail message.

IllegalArgumentException(String)

Declaration:
```
public IllegalArgumentException(java.lang.String s)
```

Description:
Constructs an `IllegalArgumentException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
IllegalMonitorStateException

Declaration

```
public class IllegalMonitorStateException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalMonitorStateException
```

Description

Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

Since: JDK1.0, CLDC 1.0

See Also: `Object.notify()`, `Object.notifyAll()`, `Object.wait()`, `Object.wait(long)`, `Object.wait(long, int)`

Member Summary	
Constructors	
	`IllegalMonitorStateException()`
	`IllegalMonitorStateException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

IllegalMonitorStateException()

Declaration:
```
public IllegalMonitorStateException()
```

Description:
Constructs an `IllegalMonitorStateException` with no detail message.

IllegalMonitorStateException(String)

Declaration:
```
public IllegalMonitorStateException(java.lang.String s)
```

Description:
Constructs an `IllegalMonitorStateException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
IllegalThreadStateException

Declaration
```
public class IllegalThreadStateException extends IllegalArgumentException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
                          |
                          +--java.lang.IllegalThreadStateException
```

Description
Thrown to indicate that a thread is not in an appropriate state for the requested operation. See, for example, the `suspend` and `resume` methods in class `Thread`.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`IllegalThreadStateException()`
	`IllegalThreadStateException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

IllegalThreadStateException()

Declaration:
```
public IllegalThreadStateException()
```

Description:
Constructs an `IllegalThreadStateException` with no detail message.

IllegalThreadStateException(String)

Declaration:
```
public IllegalThreadStateException(java.lang.String s)
```

Description:
Constructs an `IllegalThreadStateException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
IndexOutOfBoundsException

Declaration
```
public class IndexOutOfBoundsException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
```

Direct Known Subclasses: ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

Description
Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Applications can subclass this class to indicate similar exceptions.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	IndexOutOfBoundsException()
	IndexOutOfBoundsException(String s)

Inherited Member Summary
Methods inherited from class Object
equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()
Methods inherited from class Throwable
getMessage(), printStackTrace(), toString()

Constructors

IndexOutOfBoundsException()

Declaration:
```
public IndexOutOfBoundsException()
```

Description:
Constructs an `IndexOutOfBoundsException` with no detail message.

IndexOutOfBoundsException(String)

Declaration:
```
public IndexOutOfBoundsException(java.lang.String s)
```

Description:
Constructs an `IndexOutOfBoundsException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
InstantiationException

Declaration
```
public class InstantiationException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.InstantiationException
```

Description
Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.

Since: JDK1.0, CLDC 1.0

See Also: `Class.newInstance()`

Member Summary	
Constructors	
	`InstantiationException()` `InstantiationException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

InstantiationException()

Declaration:
```
public InstantiationException()
```

Description:
Constructs an `InstantiationException` with no detail message.

InstantiationException(String)

Declaration:
```
public InstantiationException(java.lang.String s)
```

Description:
Constructs an `InstantiationException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Integer

Declaration

```
public final class Integer

java.lang.Object
  |
  +--java.lang.Integer
```

Description

The Integer class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

static int	MAX_VALUE
static int	MIN_VALUE

Constructors

	Integer(int value)

Methods

byte	byteValue()
double	doubleValue()
boolean	equals(Object obj)
float	floatValue()
int	hashCode()
int	intValue()
long	longValue()
static int	parseInt(String s)
static int	parseInt(String s, int radix)
short	shortValue()
static String	toBinaryString(int i)
static String	toHexString(int i)
static String	toOctalString(int i)
String	toString()
static String	toString(int i)
static String	toString(int i, int radix)
static Integer	valueOf(String s)
static Integer	valueOf(String s, int radix)

Inherited Member Summary
Methods inherited from class `Object` `getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

MIN_VALUE

Declaration:
```
public static final int MIN_VALUE
```

Description:
The smallest value of type `int`. The constant value of this field is `-2147483648`.

MAX_VALUE

Declaration:
```
public static final int MAX_VALUE
```

Description:
The largest value of type `int`. The constant value of this field is `2147483647`.

Constructors

Integer(int)

Declaration:
```
public Integer(int value)
```

Description:
Constructs a newly allocated `Integer` object that represents the primitive `int` argument.

Parameters:
`value` - the value to be represented by the `Integer`.

Methods

toString(int, int)

Declaration:
```
public static java.lang.String toString(int i, int radix)
```

Description:
Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`, then the radix `10` is used instead.

If the first argument is negative, the first element of the result is the ASCII minus character `'-'` (`'&#92;u002d'`). If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character `'0'` (`'&#92;u0030'`); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

```
0123456789abcdefghijklmnopqrstuvwxyz
```

These are `'&#92;u0030'` through `'&#92;u0039'` and `'&#92;u0061'` through `'&#92;u007a'`. If the `radix` is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are

```
0123456789abcdef.
```

Parameters:

`i` - an integer.

`radix` - the radix.

Returns: a string representation of the argument in the specified radix.

See Also: `Character.MAX_RADIX`, `Character.MIN_RADIX`

toHexString(int)

Declaration:

```
public static java.lang.String toHexString(int i)
```

Description:

Creates a string representation of the integer argument as an unsigned integer in base 16.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'&#92;u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as hexadecimal digits:

```
0123456789abcdef
```

These are the characters `'&#92;u0030'` through `'&#92;u0039'` and `'u\0039'` through `'&#92;u0066'`.

Parameters:

`i` - an integer.

Returns: the string representation of the unsigned integer value represented by the argument in hexadecimal (base 16).

Since: JDK1.0.2

toOctalString(int)

Declaration:

```
public static java.lang.String toOctalString(int i)
```

Description:

Creates a string representation of the integer argument as an unsigned integer in base 8.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading `0`s.

If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'&#92;u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The octal digits are:

```
01234567
```

These are the characters `'&#92;u0030'` through `'&#92;u0037'`.

Parameters:
`i` - an integer

Returns: the string representation of the unsigned integer value represented by the argument in octal (base 8).

Since: JDK1.0.2

toBinaryString(int)

Declaration:
```
public static java.lang.String toBinaryString(int i)
```

Description:
Creates a string representation of the integer argument as an unsigned integer in base 2.

The unsigned integer value is the argument plus 2^{32}if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading `0`s. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'&#92;u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters `'0'` (`'&#92;u0030'`) and `'1'` (`'&#92;u0031'`) are used as binary digits.

Parameters:
`i` - an integer.

Returns: the string representation of the unsigned integer value represented by the argument in binary (base 2).

Since: JDK1.0.2

toString(int)

Declaration:
```
public static java.lang.String toString(int i)
```

Description:
Returns a new String object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and radix `10` were given as arguments to the `toString(int, int)` method.

Parameters:
`i` - an integer to be converted.

Returns: a string representation of the argument in base 10.

parseInt(String, int)

Declaration:
```
public static int parseInt(java.lang.String s, int radix)
            throws NumberFormatException
```

Description:
Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether `Character.digit(char, int)` returns a nonnegative value), except that the first character may be an ASCII minus sign `'-'` (`'&#92;u002d'`) to indicate a negative value. The resulting integer value is returned.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The radix is either smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`.
- Any character of the string is not a digit of the specified radix, except that the first character may be a minus sign `'-'` (`'&#92;u002d'`) provided that the string is longer than length 1.
- The integer value represented by the string is not a value of type `int`.

Examples:

```
parseInt("0", 10) returns 0
parseInt("473", 10) returns 473
parseInt("-0", 10) returns 0
parseInt("-FF", 16) returns -255
parseInt("1100110", 2) returns 102
parseInt("2147483647", 10) returns 2147483647
parseInt("-2147483648", 10) returns -2147483648
parseInt("2147483648", 10) throws a NumberFormatException
parseInt("99", 8) throws a NumberFormatException
parseInt("Kona", 10) throws a NumberFormatException
parseInt("Kona", 27) returns 411787
```

Parameters:
`s` - the `String` containing the integer.

`radix` - the radix to be used.

Returns: the integer represented by the string argument in the specified radix.

Throws:
`NumberFormatException` - if the string does not contain a parsable integer.

parseInt(String)

Declaration:
```
public static int parseInt(java.lang.String s)
            throws NumberFormatException
```

Description:
Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'&#92;u002d'`) to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(String, int)` method.

Parameters:
`s` - a string.

Returns: the integer represented by the argument in decimal.

Throws:
`NumberFormatException` - if the string does not contain a parsable integer.

valueOf(String, int)

Declaration:

```
public static java.lang.Integer valueOf(java.lang.String s, int radix)
            throws NumberFormatException
```

Description:
Returns a new Integer object initialized to the value of the specified String. The first argument is interpreted as representing a signed integer in the radix specified by the second argument, exactly as if the arguments were given to the `parseInt(String, int)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s, radix))
```

Parameters:

`s` - the string to be parsed.

`radix` - the radix of the integer represented by string `s`

Returns: a newly constructed `Integer` initialized to the value represented by the string argument in the specified radix.

Throws:

`NumberFormatException` - if the String cannot be parsed as an `int`.

valueOf(String)

Declaration:

```
public static java.lang.Integer valueOf(java.lang.String s)
            throws NumberFormatException
```

Description:
Returns a new Integer object initialized to the value of the specified String. The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseInt(String)` method. The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s))
```

Parameters:

`s` - the string to be parsed.

Returns: a newly constructed `Integer` initialized to the value represented by the string argument.

Throws:

`NumberFormatException` - if the string cannot be parsed as an integer.

byteValue()

Declaration:

```
public byte byteValue()
```

Description:
Returns the value of this Integer as a byte.

Returns: the value of this Integer as a byte.

Since: JDK1.1

shortValue()

Declaration:
```
public short shortValue()
```

Description:
Returns the value of this Integer as a short.

Returns: the value of this Integer as a short.

Since: JDK1.1

intValue()

Declaration:
```
public int intValue()
```

Description:
Returns the value of this Integer as an int.

Returns: the `int` value represented by this object.

longValue()

Declaration:
```
public long longValue()
```

Description:
Returns the value of this Integer as a `long`.

Returns: the `int` value represented by this object that is converted to type `long` and the result of the conversion is returned.

floatValue()

Declaration:
```
public float floatValue()
```

Description:
Returns the value of this Integer as a `float`.

Returns: the `int` value represented by this object is converted to type `float` and the result of the conversion is returned.

Since: CLDC 1.1

doubleValue()

Declaration:
```
public double doubleValue()
```

Description:
Returns the value of this Integer as a `double`.

Returns: the `int` value represented by this object is converted to type `double` and the result of the conversion is returned.

Since: CLDC 1.1

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String object representing this Integer's value. The value is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString(int)` method.

Overrides: `toString` in class `Object`

Returns: a string representation of the value of this object in base 10.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hashcode for this Integer.

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object, equal to the primitive `int` value represented by this `Integer` object.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this object to the specified object. The result is `true` if and only if the argument is not `null` and is an `Integer` object that contains the same `int` value as this object.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

java.lang
InterruptedException

Declaration
```
public class InterruptedException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.InterruptedException
```

Description
Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it.

Since: JDK1.0, CLDC 1.0

See Also: `Object.wait()`, `Object.wait(long)`, `Object.wait(long, int)`, `Thread.sleep(long)`

Member Summary	
Constructors	
	`InterruptedException()`
	`InterruptedException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

InterruptedException()

Declaration:
```
public InterruptedException()
```

Description:
Constructs an `InterruptedException` with no detail message.

InterruptedException(String)

Declaration:
```
public InterruptedException(java.lang.String s)
```

Description:
Constructs an `InterruptedException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Long

Declaration

```
public final class Long
```

```
java.lang.Object
  |
  +--java.lang.Long
```

Description

The Long class wraps a value of the primitive type `long` in an object. An object of type `Long` contains a single field whose type is `long`.

In addition, this class provides several methods for converting a `long` to a `String` and a `String` to a `long`, as well as other constants and methods useful when dealing with a `long`.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

```
            static long  MAX_VALUE
            static long  MIN_VALUE
```

Constructors

```
                         Long(long value)
```

Methods

```
                  double  doubleValue()
                 boolean  equals(Object obj)
                   float  floatValue()
                     int  hashCode()
                    long  longValue()
            static long  parseLong(String s)
            static long  parseLong(String s, int radix)
                  String  toString()
          static String  toString(long i)
          static String  toString(long i, int radix)
```

Inherited Member Summary

Methods inherited from class `Object`

`getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

MIN_VALUE

Declaration:
```
public static final long MIN_VALUE
```

Description:
The smallest value of type `long`.

MAX_VALUE

Declaration:
```
public static final long MAX_VALUE
```

Description:
The largest value of type `long`.

Constructors

Long(long)

Declaration:
```
public Long(long value)
```

Description:
Constructs a newly allocated `Long` object that represents the primitive `long` argument.

Parameters:
`value` - the value to be represented by the `Long` object.

Methods

toString(long, int)

Declaration:
```
public static java.lang.String toString(long i, int radix)
```

Description:
Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`, then the radix `10` is used instead.

If the first argument is negative, the first element of the result is the ASCII minus sign `'-'` (`'&#92;u002d'`. If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character `'0'` (`'&#92;u0030'`); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

```
0123456789abcdefghijklmnopqrstuvwxyz
```

These are `'&#92;u0030'` through `'&#92;u0039'` and `'&#92;u0061'` through `'&#92;u007a'`. If the radix is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are

```
0123456789abcdef.
```

Parameters:

`i` - a long.

`radix` - the radix.

Returns: a string representation of the argument in the specified radix.

See Also: `Character.MAX_RADIX`, `Character.MIN_RADIX`

toString(long)

Declaration:

```
public static java.lang.String toString(long i)
```

Description:

Returns a new String object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and the radix 10 were given as arguments to the `toString(long, int)` method that takes two arguments.

Parameters:

`i` - a `long` to be converted.

Returns: a string representation of the argument in base 10.

parseLong(String, int)

Declaration:

```
public static long parseLong(java.lang.String s, int radix)
            throws NumberFormatException
```

Description:

Parses the string argument as a signed `long` in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether `Character.digit` returns a nonnegative value), except that the first character may be an ASCII minus sign `'-'` (`'&#92;u002d'` to indicate a negative value. The resulting `long` value is returned.

Note that neither `L` nor `l` is permitted to appear at the end of the string as a type indicator, as would be permitted in Java programming language source code - except that either `L` or `l` may appear as a digit for a radix greater than 22.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The `radix` is either smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`.
- The first character of the string is not a digit of the specified `radix` and is not a minus sign `'-'` (`'&#92;u002d'`).
- The first character of the string is a minus sign and the string is of length 1.
- Any character of the string after the first is not a digit of the specified `radix`.

- The integer value represented by the string cannot be represented as a value of type `long`.

Examples:

```
parseLong(“0”, 10) returns 0L
parseLong(“473”, 10) returns 473L
parseLong(“-0”, 10) returns 0L
parseLong(“-FF”, 16) returns -255L
parseLong(“1100110”, 2) returns 102L
parseLong(“99”, 8) throws a NumberFormatException
parseLong(“Hazelnut”, 10) throws a NumberFormatException
parseLong(“Hazelnut”, 36) returns 1356099454469L
```

Parameters:

`s` - the `String` containing the `long`.

`radix` - the radix to be used.

Returns: the `long` represented by the string argument in the specified radix.

Throws:

`NumberFormatException` - if the string does not contain a parsable integer.

parseLong(String)

Declaration:

```
public static long parseLong(java.lang.String s)
            throws NumberFormatException
```

Description:

Parses the string argument as a signed decimal `long`. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`&#92;u002d'`) to indicate a negative value. The resulting long value is returned, exactly as if the argument and the radix `10` were given as arguments to the `parseLong(String, int)` method that takes two arguments.

Note that neither `L` nor `l` is permitted to appear at the end of the string as a type indicator, as would be permitted in Java programming language source code.

Parameters:

`s` - a string.

Returns: the `long` represented by the argument in decimal.

Throws:

`NumberFormatException` - if the string does not contain a parsable `long`.

longValue()

Declaration:

```
public long longValue()
```

Description:

Returns the value of this Long as a long value.

Returns: the `long` value represented by this object.

floatValue()

Declaration:

```
public float floatValue()
```

doubleValue()

Description:
Returns the value of this Long as a float.

Returns: the `long` value represented by this object is converted to type `float` and the result of the conversion is returned.

Since: CLDC 1.1

doubleValue()

Declaration:
```
public double doubleValue()
```

Description:
Returns the value of this Long as a double.

Returns: the `long` value represented by this object that is converted to type `double` and the result of the conversion is returned.

Since: CLDC 1.1

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a String object representing this Long's value. The long integer value represented by this Long object is converted to signed decimal representation and returned as a string, exactly as if the long value were given as an argument to the `toString(long)` method that takes one argument.

Overrides: `toString` in class `Object`

Returns: a string representation of this object in base 10.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Computes a hashcode for this Long. The result is the exclusive OR of the two halves of the primitive `long` value represented by this `Long` object. That is, the hashcode is the value of the expression:

```
(int)(this.longValue()^(this.longValue()>>>32))
```

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and is a `Long` object that contains the same `long` value as this object.

Overrides: `equals` in class `Object`

Parameters:

`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

java.lang
Math

Declaration

```
public final class Math
```

```
java.lang.Object
  |
  +--java.lang.Math
```

Description

The class `Math` contains methods for performing basic numeric operations.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

```
          static double  E
          static double  PI
```

Methods

```
          static double  abs(double a)
           static float  abs(float a)
             static int  abs(int a)
            static long  abs(long a)
          static double  ceil(double a)
          static double  cos(double a)
          static double  floor(double a)
          static double  max(double a, double b)
           static float  max(float a, float b)
             static int  max(int a, int b)
            static long  max(long a, long b)
          static double  min(double a, double b)
           static float  min(float a, float b)
             static int  min(int a, int b)
            static long  min(long a, long b)
          static double  sin(double a)
          static double  sqrt(double a)
          static double  tan(double a)
          static double  toDegrees(double angrad)
          static double  toRadians(double angdeg)
```

Inherited Member Summary

Methods inherited from class `Object`

Inherited Member Summary
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

E

Declaration:
```
public static final double E
```

Description:
The `double` value that is closer than any other to `e`, the base of the natural logarithms.

Since: CLDC 1.1

PI

Declaration:
```
public static final double PI
```

Description:
The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Since: CLDC 1.1

Methods

sin(double)

Declaration:
```
public static double sin(double a)
```

Description:
Returns the trigonometric sine of an angle. Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero.

Parameters:
`a` - an angle, in radians.

Returns: the sine of the argument.

Since: CLDC 1.1

cos(double)

Declaration:
```
public static double cos(double a)
```

Description:
Returns the trigonometric cosine of an angle. Special case:

- If the argument is NaN or an infinity, then the result is NaN.

Parameters:
`a` - an angle, in radians.

Returns: the cosine of the argument.

Since: CLDC 1.1

tan(double)

Declaration:
```
public static double tan(double a)
```

Description:
Returns the trigonometric tangent of an angle. Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero

Parameters:
`a` - an angle, in radians.

Returns: the tangent of the argument.

Since: CLDC 1.1

toRadians(double)

Declaration:
```
public static double toRadians(double angdeg)
```

Description:
Converts an angle measured in degrees to the equivalent angle measured in radians.

Parameters:
`angdeg` - an angle, in degrees

Returns: the measurement of the angle `angdeg` in radians.

Since: CLDC 1.1

toDegrees(double)

Declaration:
```
public static double toDegrees(double angrad)
```

Description:
Converts an angle measured in radians to the equivalent angle measured in degrees.

Parameters:
`angrad` - an angle, in radians

Returns: the measurement of the angle `angrad` in degrees.

Since: CLDC 1.1

sqrt(double)

Declaration:
```
public static double sqrt(double a)
```

Description:
Returns the correctly rounded positive square root of a `double` value. Special cases:

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Parameters:
`a` - a `double` value.

Returns: the positive square root of `a`. If the argument is NaN or less than zero, the result is NaN.

Since: CLDC 1.1

ceil(double)

Declaration:
```
public static double ceil(double a)
```

Description:
Returns the smallest (closest to negative infinity) `double` value that is not less than the argument and is equal to a mathematical integer. Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.
- If the argument value is less than zero but greater than -1.0, then the result is negative zero.

Note that the value of `Math.ceil(x)` is exactly the value of `-Math.floor(-x)`.

Parameters:
`a` - a `double` value.

Returns: the smallest (closest to negative infinity) `double` value that is not less than the argument and is equal to a mathematical integer.

Since: CLDC 1.1

floor(double)

Declaration:
```
public static double floor(double a)
```

Description:
Returns the largest (closest to positive infinity) `double` value that is not greater than the argument and is equal to a mathematical integer. Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

Parameters:
`a` - a `double` value.

Returns: the largest (closest to positive infinity) `double` value that is not greater than the argument and is equal to a mathematical integer.

Since: CLDC 1.1

abs(int)

Declaration:
```
public static int abs(int a)
```

Description:
Returns the absolute value of an `int` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Integer.MIN_VALUE`, the most negative representable `int` value, the result is that same value, which is negative.

Parameters:
`a` - an `int` value.

Returns: the absolute value of the argument.

See Also: `Integer.MIN_VALUE`

abs(long)

Declaration:
```
public static long abs(long a)
```

Description:
Returns the absolute value of a `long` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Long.MIN_VALUE`, the most negative representable `long` value, the result is that same value, which is negative.

Parameters:
`a` - a `long` value.

Returns: the absolute value of the argument.

See Also: `Long.MIN_VALUE`

abs(float)

Declaration:
```
public static float abs(float a)
```

Description:
Returns the absolute value of a `float` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is equal to the value of the expression:

`Float.intBitsToFloat(0x7fffffff & Float.floatToIntBits(a))`

Parameters:
`a` - a `float` value.

Returns: the absolute value of the argument.

Since: CLDC 1.1

abs(double)

Declaration:
`public static double abs(double a)`

Description:
Returns the absolute value of a `double` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is equal to the value of the expression:

`Double.longBitsToDouble((Double.doubleToLongBits(a)<<1)>>>1)`

Parameters:
`a` - a `double` value.

Returns: the absolute value of the argument.

Since: CLDC 1.1

max(int, int)

Declaration:
`public static int max(int a, int b)`

Description:
Returns the greater of two `int` values. That is, the result is the argument closer to the value of `Integer.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:
`a` - an `int` value.

`b` - an `int` value.

Returns: the larger of `a` and `b`.

See Also: `Long.MAX_VALUE`

max(long, long)

Declaration:
`public static long max(long a, long b)`

Description:
Returns the greater of two `long` values. That is, the result is the argument closer to the value of `Long.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:
`a` - a `long` value.

b - a `long` value.

Returns: the larger of `a` and `b`.

See Also: `Long.MAX_VALUE`

max(float, float)

Declaration:
```
public static float max(float a, float b)
```

Description:
Returns the greater of two `float` values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is `NaN`, then the result is `NaN`. Unlike the the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

Parameters:

a - a `float` value.

b - a `float` value.

Returns: the larger of `a` and `b`.

max(double, double)

Declaration:
```
public static double max(double a, double b)
```

Description:
Returns the greater of two `double` values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is `NaN`, then the result is `NaN`. Unlike the the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

Parameters:

a - a `double` value.

b - a `double` value.

Returns: the larger of `a` and `b`.

min(int, int)

Declaration:
```
public static int min(int a, int b)
```

Description:
Returns the smaller of two `int` values. That is, the result the argument closer to the value of `Integer.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

a - an `int` value.

b - an `int` value.

Returns: the smaller of `a` and `b`.

See Also: `Long.MIN_VALUE`

min(long, long)

Declaration:
```
public static long min(long a, long b)
```

Description:
Returns the smaller of two `long` values. That is, the result is the argument closer to the value of `Long.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:
`a` - a `long` value.

`b` - a `long` value.

Returns: the smaller of `a` and `b`.

See Also: `Long.MIN_VALUE`

min(float, float)

Declaration:
```
public static float min(float a, float b)
```

Description:
Returns the smaller of two `float` values. That is, the result is the value closer to negative infinity. If the arguments have the same value, the result is that same value. If either value is `NaN`, then the result is `NaN`. Unlike the the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other is negative zero, the result is negative zero.

Parameters:
`a` - a `float` value.

`b` - a `float` value.

Returns: the smaller of `a` and `b.`

Since: CLDC 1.1

min(double, double)

Declaration:
```
public static double min(double a, double b)
```

Description:
Returns the smaller of two `double` values. That is, the result is the value closer to negative infinity. If the arguments have the same value, the result is that same value. If either value is `NaN`, then the result is `NaN`. Unlike the the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other is negative zero, the result is negative zero.

Parameters:
`a` - a `double` value.

`b` - a `double` value.

Returns: the smaller of `a` and `b`.

Since: CLDC 1.1

java.lang
NegativeArraySizeException

Declaration
`public class` **`NegativeArraySizeException`** `extends` `RuntimeException`

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NegativeArraySizeException
```

Description
Thrown if an application tries to create an array with negative size.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`NegativeArraySizeException()` `NegativeArraySizeException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

NegativeArraySizeException()

Declaration:
`public` **`NegativeArraySizeException`**`()`

Description:
Constructs a `NegativeArraySizeException` with no detail message.

NegativeArraySizeException(String)

Declaration:
```
public NegativeArraySizeException(java.lang.String s)
```

Description:
Constructs a `NegativeArraySizeException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
NoClassDefFoundError

Declaration
```
public class NoClassDefFoundError extends Error
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
              |
              +--java.lang.NoClassDefFoundError
```

Description
Thrown if the Java Virtual Machine tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the `new` expression) and no definition of the class could be found.

The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

Since: JDK1.0, CLDC 1.1

Member Summary	
Constructors	
	`NoClassDefFoundError()` `NoClassDefFoundError(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

NoClassDefFoundError()

Declaration:
```
public NoClassDefFoundError()
```

Description:
Constructs a `NoClassDefFoundError` with no detail message.

NoClassDefFoundError(String)

Declaration:
```
public NoClassDefFoundError(java.lang.String s)
```

Description:
Constructs a `NoClassDefFoundError` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
NullPointerException

Declaration

```
public class NullPointerException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.NullPointerException
```

Description

Thrown when an application attempts to use `null` in a case where an object is required. These include:

- Calling the instance method of a `null` object.
- Accessing or modifying the field of a `null` object.
- Taking the length of `null` as if it were an array.
- Accessing or modifying the slots of `null` as if it were an array.
- Throwing `null` as if it were a `Throwable` value.

Applications should throw instances of this class to indicate other illegal uses of the `null` object.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`NullPointerException()` `NullPointerException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

NullPointerException()

Declaration:
```
public NullPointerException()
```

Description:
Constructs a `NullPointerException` with no detail message.

NullPointerException(String)

Declaration:
```
public NullPointerException(java.lang.String s)
```

Description:
Constructs a `NullPointerException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
NumberFormatException

Declaration

```
public class NumberFormatException extends IllegalArgumentException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IllegalArgumentException
                          |
                          +--java.lang.NumberFormatException
```

Description

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

Since: JDK1.0, CLDC 1.0

See Also: `Integer.toString()`

Member Summary	
Constructors	
	`NumberFormatException()`
	`NumberFormatException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

NumberFormatException()

Declaration:
```
public NumberFormatException()
```

Description:
Constructs a `NumberFormatException` with no detail message.

NumberFormatException(String)

Declaration:
```
public NumberFormatException(java.lang.String s)
```

Description:
Constructs a `NumberFormatException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Object

Declaration
`public class` **`Object`**

`java.lang.Object`

Description
Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since: JDK1.0, CLDC 1.0

See Also: `Class`

Member Summary	
Constructors	
	`Object()`
Methods	
`boolean`	`equals(Object obj)`
`Class`	`getClass()`
`int`	`hashCode()`
`void`	`notify()`
`void`	`notifyAll()`
`String`	`toString()`
`void`	`wait()`
`void`	`wait(long timeout)`
`void`	`wait(long timeout, int nanos)`

Constructors

Object()

Declaration:
`public` **`Object`**`()`

Methods

getClass()

Declaration:
`public final java.lang.Class` **`getClass`**`()`

Description:
Returns the runtime class of an object. That `Class` object is the object that is locked by `static synchronized` methods of the represented class.

Returns: the object of type `Class` that represents the runtime class of the object.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by `java.util.Hashtable`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the JavaTM programming language.)

Returns: a hash code value for this object.

See Also: `equals(Object)`, `java.util.Hashtable`

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Indicates whether some other object is “equal to” this one.

The `equals` method implements an equivalence relation:

- It is *reflexive*: for any reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the object is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x==y` has the value `true`).

Parameters:
`obj` - the reference object with which to compare.

Returns: `true` if this object is the same as the obj argument; `false` otherwise.

See Also: `Boolean.hashCode()`, `java.util.Hashtable`

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns: a string representation of the object.

notify()

Declaration:
```
public final void notify()
```

Description:
Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the `wait` methods.

The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object. The awakened thread will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened thread enjoys no reliable privilege or disadvantage in being the next thread to lock this object.

This method should only be called by a thread that is the owner of this object's monitor. A thread becomes the owner of the object's monitor in one of three ways:

- By executing a synchronized instance method of that object.
- By executing the body of a `synchronized` statement that synchronizes on the object.
- For objects of type `Class,` by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.

Throws:
`IllegalMonitorStateException` - if the current thread is not the owner of this object's monitor.

See Also: `notifyAll()`, `wait()`

notifyAll()

Declaration:
```
public final void notifyAll()
```

Description:
Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the `wait` methods.

The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object. The awakened threads will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened threads enjoy no reliable privilege or disadvantage in being the next thread to lock this object.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Throws:
`IllegalMonitorStateException` - if the current thread is not the owner of this object's monitor.

See Also: `notify()`, `wait()`

wait(long)

Declaration:
```
public final void wait(long timeout)
            throws InterruptedException
```

Description:
Causes current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor.

This method causes the current thread (call it *T*) to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object. Thread *T* becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

- Some other thread invokes the `notify` method for this object and thread *T* happens to be arbitrarily chosen as the thread to be awakened.
- Some other thread invokes the `notifyAll` method for this object.
- Some other thread `interrupts` thread *T*.
- The specified amount of real time has elapsed, more or less. If `timeout` is zero, however, then real time is not taken into consideration and the thread simply waits until notified.

The thread *T* is then removed from the wait set for this object and re-enabled for thread scheduling. It then competes in the usual manner with other threads for the right to synchronize on the object; once it has gained control of the object, all its synchronization claims on the object are restored to the status quo ante - that is, to the situation as of the time that the `wait` method was invoked. Thread *T* then returns from the invocation of the `wait` method. Thus, on return from the `wait` method, the synchronization state of the object and of thread *T* is exactly as it was when the `wait` method was invoked.

If the current thread is `interrupted` by another thread while it is waiting, then an `InterruptedException` is thrown. This exception is not thrown until the lock status of this object has been restored as described above.

Note that the `wait` method, as it places the current thread into the wait set for this object, unlocks only this object; any other objects on which the current thread may be synchronized remain locked while the thread waits.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout` - the maximum time to wait in milliseconds.

Throws:

`IllegalArgumentException` - if the value of timeout is negative.

`IllegalMonitorStateException` - if the current thread is not the owner of the object's monitor.

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also: `notify()`, `notifyAll()`

wait(long, int)

Declaration:

```
public final void wait(long timeout, int nanos)
            throws InterruptedException
```

Description:

Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

This method is similar to the `wait` method of one argument, but it allows finer control over the amount of time to wait for a notification before giving up. The amount of real time, measured in nanoseconds, is given by:

```
1000000*timeout+nanos
```

In all other respects, this method does the same thing as the method `wait(long)` of one argument. In particular, `wait(0, 0)` means the same thing as `wait(0)`.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until either of the following two conditions has occurred:

- Another thread notifies threads waiting on this object's monitor to wake up either through a call to the `notify` method or the `notifyAll` method.
- The timeout period, specified by `timeout` milliseconds plus `nanos` nanoseconds arguments, has elapsed.

The thread then waits until it can re-obtain ownership of the monitor and resumes execution

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout` - the maximum time to wait in milliseconds.

`nanos` - additional time, in nanoseconds range 0-999999.

Throws:

`IllegalArgumentException` - if the value of timeout is negative or the value of nanos is not in the range 0-999999.

`IllegalMonitorStateException` - if the current thread is not the owner of this object's monitor.

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

wait()

Declaration:

```
public final void wait()
            throws InterruptedException
```

Description:

Causes current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object. In other word's this method behaves exactly as if it simply performs the call `wait(0)`.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until another thread notifies threads waiting on this object's monitor to wake up either through a call to the `notify` method or the `notifyAll` method. The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

This method should only be called by a thread that is the owner of this object's monitor. See the `notify` method for a description of the ways in which a thread can become the owner of a monitor.

Throws:

`IllegalMonitorStateException` - if the current thread is not the owner of the object's monitor.

`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also: `notify()`, `notifyAll()`

java.lang
OutOfMemoryError

Declaration

```
public class OutOfMemoryError extends VirtualMachineError
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
              |
              +--java.lang.VirtualMachineError
                    |
                    +--java.lang.OutOfMemoryError
```

Description

Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`OutOfMemoryError()`
	`OutOfMemoryError(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

OutOfMemoryError()

Declaration:

```
public OutOfMemoryError()
```

Description:
Constructs an `OutOfMemoryError` with no detail message.

OutOfMemoryError(String)

Declaration:
```
public OutOfMemoryError(java.lang.String s)
```

Description:
Constructs an `OutOfMemoryError` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
Runnable

Declaration
```
public interface Runnable
```

All Known Implementing Classes: Thread

Description
The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

In addition, `Runnable` provides the means for a class to be active while not subclassing `Thread`. A class that implements `Runnable` can run without subclassing `Thread` by instantiating a `Thread` instance and passing itself in as the target. In most cases, the `Runnable` interface should be used if you are only planning to override the `run()` method and no other `Thread` methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

Since: JDK1.0, CLDC 1.0

See Also: Thread

Member Summary	
Methods	
void	run()

Methods

run()

Declaration:
```
public void run()
```

Description:
When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

The general contract of the method `run` is that it may take any action whatsoever.

See Also: Thread.run()

java.lang
Runtime

Declaration
```
public class Runtime
```

```
java.lang.Object
  |
  +--java.lang.Runtime
```

Description
Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the `getRuntime` method.

An application cannot create its own instance of this class.

Since: JDK1.0, CLDC 1.0

See Also: `getRuntime()`

Member Summary	
Methods	
void	exit(int status)
long	freeMemory()
void	gc()
static Runtime	getRuntime()
long	totalMemory()

Inherited Member Summary
Methods inherited from class `Object`
equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods

getRuntime()

Declaration:
```
public static java.lang.Runtime getRuntime()
```

Description:
Returns the runtime object associated with the current Java application. Most of the methods of class `Runtime` are instance methods and must be invoked with respect to the current runtime object.

Returns: the `Runtime` object associated with the current Java application.

exit(int)

Declaration:
```
public void exit(int status)
```

Description:
Terminates the currently running Java application. This method never returns normally.

The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

Parameters:
`status` - exit status.

Since: JDK1.0

freeMemory()

Declaration:
```
public long freeMemory()
```

Description:
Returns the amount of free memory in the system. Calling the `gc` method may result in increasing the value returned by `freeMemory.`

Returns: an approximation to the total amount of memory currently available for future allocated objects, measured in bytes.

totalMemory()

Declaration:
```
public long totalMemory()
```

Description:
Returns the total amount of memory in the Java Virtual Machine. The value returned by this method may vary over time, depending on the host environment.

Note that the amount of memory required to hold an object of any given type may be implementation-dependent.

Returns: the total amount of memory currently available for current and future objects, measured in bytes.

gc()

Declaration:
```
public void gc()
```

Description:
Runs the garbage collector. Calling this method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made its best effort to recycle all discarded objects.

The name `gc` stands for “garbage collector”. The Java Virtual Machine performs this recycling process automatically as needed even if the `gc` method is not invoked explicitly.

The method `System.gc()` is the conventional and convenient means of invoking this method.

java.lang
RuntimeException

Declaration

```
public class RuntimeException extends Exception
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
```

Direct Known Subclasses: ArithmeticException, ArrayStoreException, ClassCastException, java.util.EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IndexOutOfBoundsException, NegativeArraySizeException, java.util.NoSuchElementException, NullPointerException, SecurityException

Description

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

A method is not required to declare in its `throws` clause any subclasses of `RuntimeException` that might be thrown during the execution of the method but not caught.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`RuntimeException()`
	`RuntimeException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

RuntimeException()

Declaration:
```
public RuntimeException()
```

Description:
Constructs a `RuntimeException` with no detail message.

RuntimeException(String)

Declaration:
```
public RuntimeException(java.lang.String s)
```

Description:
Constructs a `RuntimeException` with the specified detail message.

Parameters:
`s` - the detail message.

java.lang
SecurityException

Declaration
```
public class SecurityException extends RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.SecurityException
```

Description
Thrown by the system to indicate a security violation.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`SecurityException()`
	`SecurityException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

SecurityException()

Declaration:
```
public SecurityException()
```

Description:
Constructs a `SecurityException` with no detail message.

SecurityException(String)

Declaration:

```
public SecurityException(java.lang.String s)
```

Description:

Constructs a `SecurityException` with the specified detail message.

Parameters:

`s` - the detail message.

java.lang
Short

Declaration
```
public final class Short
```

```
java.lang.Object
  |
  +--java.lang.Short
```

Description
The Short class is the standard wrapper for short values.

Since: JDK1.1, CLDC 1.0

Member Summary	
Fields	
static short	MAX_VALUE
static short	MIN_VALUE
Constructors	
	Short(short value)
Methods	
boolean	equals(Object obj)
int	hashCode()
static short	parseShort(String s)
static short	parseShort(String s, int radix)
short	shortValue()
String	toString()

Inherited Member Summary
Methods inherited from class Object
getClass(), notify(), notifyAll(), wait(), wait(), wait()

Fields

MIN_VALUE

Declaration:
```
public static final short MIN_VALUE
```

Description:
The minimum value a Short can have.

MAX_VALUE

Declaration:
```
public static final short MAX_VALUE
```

Description:
The maximum value a Short can have.

Constructors

Short(short)

Declaration:
```
public Short(short value)
```

Description:
Constructs a Short object initialized to the specified short value.

Parameters:
`value` - the initial value of the Short

Methods

parseShort(String)

Declaration:
```
public static short parseShort(java.lang.String s)
            throws NumberFormatException
```

Description:
Assuming the specified String represents a short, returns that short's value. Throws an exception if the String cannot be parsed as a short. The radix is assumed to be 10.

Parameters:
`s` - the String containing the short

Returns: The short value represented by the specified string

Throws:
`NumberFormatException` - If the string does not contain a parsable short.

parseShort(String, int)

Declaration:
```
public static short parseShort(java.lang.String s, int radix)
            throws NumberFormatException
```

Description:
Assuming the specified String represents a short, returns that short's value in the radix specified by the second argument. Throws an exception if the String cannot be parsed as a short.

shortValue()

Parameters:

`s` - the String containing the short

`radix` - the radix to be used

Returns: The short value represented by the specified string in the specified radix.

Throws:

`NumberFormatException` - If the String does not contain a parsable short.

shortValue()

Declaration:

```
public short shortValue()
```

Description:

Returns the value of this Short as a short.

Returns: the value of this Short as a short.

toString()

Declaration:

```
public java.lang.String toString()
```

Description:

Returns a String object representing this Short's value.

Overrides: `toString` in class `Object`

Returns: a string representation of the object.

hashCode()

Declaration:

```
public int hashCode()
```

Description:

Returns a hashcode for this Short.

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

equals(Object)

Declaration:

```
public boolean equals(java.lang.Object obj)
```

Description:

Compares this object to the specified object.

Overrides: `equals` in class `Object`

Parameters:

`obj` - the object to compare with

Returns: true if the objects are the same; false otherwise.

java.lang
String

Declaration

```
public final class String
```

```
java.lang.Object
  |
  +--java.lang.String
```

Description

The `String` class represents character strings. All string literals in Java programs, such as `“abc”`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = “abc”;
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println(“abc”);
String cde = “cde”;
System.out.println(“abc” + cde);
String c = “abc”.substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuffer` class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Since: JDK1.0, CLDC 1.0

See Also: `Object.toString()`, `StringBuffer`, `StringBuffer.append(boolean)`, `StringBuffer.append(char)`, `StringBuffer.append(char[])`, `StringBuffer.append(char[], int, int)`, `StringBuffer.append(int)`, `StringBuffer.append(long)`, `StringBuffer.append(Object)`, `StringBuffer.append(String)`

Member Summary

Constructors

```
                        String()
                        String(byte[] bytes)
                        String(byte[] bytes, int off, int len)
                        String(byte[] bytes, int off, int len, String enc)
                        String(byte[] bytes, String enc)
                        String(char[] value)
                        String(char[] value, int offset, int count)
                        String(String value)
                        String(StringBuffer buffer)
```

Methods

```
                  char  charAt(int index)
                   int  compareTo(String anotherString)
                String  concat(String str)
               boolean  endsWith(String suffix)
               boolean  equals(Object anObject)
               boolean  equalsIgnoreCase(String anotherString)
                byte[]  getBytes()
                byte[]  getBytes(String enc)
                  void  getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
                   int  hashCode()
                   int  indexOf(int ch)
                   int  indexOf(int ch, int fromIndex)
                   int  indexOf(String str)
                   int  indexOf(String str, int fromIndex)
                String  intern()
                   int  lastIndexOf(int ch)
                   int  lastIndexOf(int ch, int fromIndex)
                   int  length()
               boolean  regionMatches(boolean ignoreCase, int toffset, String other,
                                      int ooffset, int len)
                String  replace(char oldChar, char newChar)
               boolean  startsWith(String prefix)
               boolean  startsWith(String prefix, int toffset)
                String  substring(int beginIndex)
                String  substring(int beginIndex, int endIndex)
                char[]  toCharArray()
                String  toLowerCase()
                String  toString()
                String  toUpperCase()
                String  trim()
        static String  valueOf(boolean b)
        static String  valueOf(char c)
        static String  valueOf(char[] data)
        static String  valueOf(char[] data, int offset, int count)
        static String  valueOf(double d)
        static String  valueOf(float f)
        static String  valueOf(int i)
        static String  valueOf(long l)
        static String  valueOf(Object obj)
```

Inherited Member Summary
Methods inherited from class `Object`
`getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

String()

Declaration:
```
public String()
```

Description:
Initializes a newly created `String` object so that it represents an empty character sequence.

String(String)

Declaration:
```
public String(java.lang.String value)
```

Description:
Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

Parameters:
`value` - a `String`.

String(char[])

Declaration:
```
public String(char[] value)
```

Description:
Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:
`value` - the initial value of the string.

Throws:
`NullPointerException` - if `value` is `null`.

String(char[], int, int)

Declaration:
```
public String(char[] value, int offset, int count)
```

Description:
Allocates a new `String` that contains characters from a subarray of the character array argument. The `offset` argument is the index of the first character of the subarray and the `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

String(byte[], int, int, String)

Parameters:

`value` - array that is the source of characters.

`offset` - the initial offset.

`count` - the length.

Throws:

`IndexOutOfBoundsException` - if the `offset` and `count` arguments index characters outside the bounds of the `value` array.

`NullPointerException` - if `value` is `null`.

String(byte[], int, int, String)

Declaration:

```
public String(byte[] bytes, int off, int len, java.lang.String enc)
            throws UnsupportedEncodingException
```

Description:

Construct a new `String` by converting the specified subarray of bytes using the specified character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the subarray.

Parameters:

`bytes` - The bytes to be converted into characters

`off` - Index of the first byte to convert

`len` - Number of bytes to convert

`enc` - The name of a character encoding

Throws:

`java.io.UnsupportedEncodingException` - If the named encoding is not supported

Since: JDK1.1

String(byte[], String)

Declaration:

```
public String(byte[] bytes, java.lang.String enc)
            throws UnsupportedEncodingException
```

Description:

Construct a new `String` by converting the specified array of bytes using the specified character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the byte array.

Parameters:

`bytes` - The bytes to be converted into characters

`enc` - The name of a supported character encoding

Throws:

`java.io.UnsupportedEncodingException` - If the named encoding is not supported

Since: JDK1.1

String(byte[], int, int)

Declaration:
```
public String(byte[] bytes, int off, int len)
```

Description:
Construct a new `String` by converting the specified subarray of bytes using the platform's default character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the subarray.

Parameters:
`bytes` - The bytes to be converted into characters

`off` - Index of the first byte to convert

`len` - Number of bytes to convert

Since: JDK1.1

String(byte[])

Declaration:
```
public String(byte[] bytes)
```

Description:
Construct a new `String` by converting the specified array of bytes using the platform's default character encoding. The length of the new `String` is a function of the encoding, and hence may not be equal to the length of the byte array.

Parameters:
`bytes` - The bytes to be converted into characters

Since: JDK1.1

String(StringBuffer)

Declaration:
```
public String(java.lang.StringBuffer buffer)
```

Description:
Allocates a new string that contains the sequence of characters currently contained in the string buffer argument. The contents of the string buffer are copied; subsequent modification of the string buffer does not affect the newly created string.

Parameters:
`buffer` - a `StringBuffer`.

Throws:
`NullPointerException` - If `buffer` is `null`.

Methods

length()

Declaration:
```
public int length()
```

Description:
Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Returns: the length of the sequence of characters represented by this object.

charAt(int)

Declaration:
```
public char charAt(int index)
```

Description:
Returns the character at the specified index. An index ranges from `0` to `length() - 1`. The first character of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Parameters:
`index` - the index of the character.

Returns: the character at the specified index of this string. The first character is at index `0`.

Throws:
`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

getChars(int, int, char[], int)

Declaration:
```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

Description:
Copies characters from this string into the destination character array.

The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus the total number of characters to be copied is `srcEnd-srcBegin`). The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:
`srcBegin` - index of the first character in the string to copy.

`srcEnd` - index after the last character in the string to copy.

`dst` - the destination array.

`dstBegin` - the start offset in the destination array.

Throws:
`IndexOutOfBoundsException` - If any of the following is true:

- `srcBegin` is negative.
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this string
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

`NullPointerException` - if `dst` is `null`

getBytes(String)

Declaration:
```
public byte[] getBytes(java.lang.String enc)
            throws UnsupportedEncodingException
```

Description:
Convert this `String` into bytes according to the specified character encoding, storing the result into a new byte array.

Parameters:
`enc` - A character-encoding name

Returns: The resultant byte array

Throws:
`java.io.UnsupportedEncodingException` - If the named encoding is not supported

Since: JDK1.1

getBytes()

Declaration:
```
public byte[] getBytes()
```

Description:
Convert this `String` into bytes according to the platform's default character encoding, storing the result into a new byte array.

Returns: the resultant byte array.

Since: JDK1.1

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object anObject)
```

Description:
Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Overrides: `equals` in class `Object`

Parameters:
`anObject` - the object to compare this `String` against.

Returns: `true` if the `String` are equal; `false` otherwise.

See Also: `compareTo(String)`, `equalsIgnoreCase(String)`

equalsIgnoreCase(String)

Declaration:
```
public boolean equalsIgnoreCase(java.lang.String anotherString)
```

Description:
Compares this `String` to another `String`, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Two characters `c1` and `c2` are considered the same, ignoring case if at least one of the following is true:

- The two characters are the same (as compared by the `==` operator).
- Applying the method `Character.toUpperCase(char)` to each character produces the same result.
- Applying the method `Character.toLowerCase(char)` to each character produces the same result.

Parameters:
`anotherString` - the `String` to compare this `String` against.

Returns: `true` if the argument is not `null` and the `String`s are equal, ignoring case; `false` otherwise.

See Also: `equals(Object)`, `Character.toLowerCase(char)`, `Character.toUpperCase(char)`

compareTo(String)

Declaration:
```
public int compareTo(java.lang.String anotherString)
```

Description:
Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position *k* in the two string —- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings —- that is, the value:

```
this.length()-anotherString.length()
```

Parameters:
`anotherString` - the `String` to be compared.

Returns: the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

Throws:
`NullPointerException` - if `anotherString` is `null`.

regionMatches(boolean, int, String, int, int)

Declaration:

```
public boolean regionMatches(boolean ignoreCase, int toffset, java.lang.String other,
            int ooffset, int len)
```

Description:

Tests if two string regions are equal.

A substring of this `String` object is compared to a substring of the argument `other`. The result is `true` if these substrings represent character sequences that are the same, ignoring case if and only if `ignoreCase` is true. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of `other` to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the other argument.
- There is some nonnegative integer *k* less than `len` such that:

```
this.charAt(toffset+k) != other.charAt(ooffset+k)
```

- `ignoreCase` is `true` and there is some nonnegative integer *k* less than `len` such that:

```
Character.toLowerCase(this.charAt(toffset+k)) !=
              Character.toLowerCase(other.charAt(ooffset+k))
```

and:

```
Character.toUpperCase(this.charAt(toffset+k)) !=
        Character.toUpperCase(other.charAt(ooffset+k))
```

Parameters:

`ignoreCase` - if `true`, ignore case when comparing characters.

`toffset` - the starting offset of the subregion in this string.

`other` - the string argument.

`ooffset` - the starting offset of the subregion in the string argument.

`len` - the number of characters to compare.

Returns: `true` if the specified subregion of this string matches the specified subregion of the string argument; `false` otherwise. Whether the matching is exact or case insensitive depends on the `ignoreCase` argument.

startsWith(String, int)

Declaration:

```
public boolean startsWith(java.lang.String prefix, int toffset)
```

Description:

Tests if this string starts with the specified prefix beginning at the specified index.

Parameters:

`prefix` - the prefix.

`toffset` - where to begin looking in the string.

Returns: `true` if the character sequence represented by the argument is a prefix of the substring of this object starting at index `toffset`; `false` otherwise. The result is `false` if `toffset` is negative or greater than the length of this `String` object; otherwise the result is the same as the result of the expression

```
this.subString(toffset).startsWith(prefix)
```

Throws:
`NullPointerException` - if `prefix` is `null`.

startsWith(String)

Declaration:
```
public boolean startsWith(java.lang.String prefix)
```

Description:
Tests if this string starts with the specified prefix.

Parameters:
`prefix` - the prefix.

Returns: `true` if the character sequence represented by the argument is a prefix of the character sequence represented by this string; `false` otherwise. Note also that `true` will be returned if the argument is an empty string or is equal to this `String` object as determined by the `equals(Object)` method.

Throws:
`NullPointerException` - if `prefix` is `null`.

Since: JDK1.0

endsWith(String)

Declaration:
```
public boolean endsWith(java.lang.String suffix)
```

Description:
Tests if this string ends with the specified suffix.

Parameters:
`suffix` - the suffix.

Returns: `true` if the character sequence represented by the argument is a suffix of the character sequence represented by this object; `false` otherwise. Note that the result will be `true` if the argument is the empty string or is equal to this `String` object as determined by the `equals(Object)` method.

Throws:
`NullPointerException` - if `suffix` is `null`.

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hashcode for this string. The hashcode for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and `^` indicates exponentiation. (The hash value of the empty string is zero.)

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

indexOf(int)

Declaration:
```
public int indexOf(int ch)
```

Description:
Returns the index within this string of the first occurrence of the specified character. If a character with value `ch` occurs in the character sequence represented by this `String` object, then the index of the first such occurrence is returned —- that is, the smallest value *k* such that:

```
this.charAt(k) == ch
```

is `true`. If no such character occurs in this string, then `-1` is returned.

Parameters:
`ch` - a character.

Returns: the index of the first occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

indexOf(int, int)

Declaration:
```
public int indexOf(int ch, int fromIndex)
```

Description:
Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

If a character with value `ch` occurs in the character sequence represented by this `String` object at an index no smaller than `fromIndex`, then the index of the first such occurrence is returned—-that is, the smallest value *k* such that:

```
(this.charAt(k) == ch) && (k >= fromIndex)
```

is true. If no such character occurs in this string at or after position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: `-1` is returned.

Parameters:
`ch` - a character.

`fromIndex` - the index to start the search from.

Returns: the index of the first occurrence of the character in the character sequence represented by this object that is greater than or equal to `fromIndex`, or `-1` if the character does not occur.

lastIndexOf(int)

Declaration:
```
public int lastIndexOf(int ch)
```

Description:
Returns the index within this string of the last occurrence of the specified character. That is, the index returned is the largest value *k* such that:

```
this.charAt(k) == ch
```

is true. The String is searched backwards starting at the last character.

Parameters:
`ch` - a character.

Returns: the index of the last occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

lastIndexOf(int, int)

Declaration:
```
public int lastIndexOf(int ch, int fromIndex)
```

Description:
Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index. That is, the index returned is the largest value *k* such that:

```
(this.charAt(k) == ch) && (k <= fromIndex)
```

is true.

Parameters:
`ch` - a character.

`fromIndex` - the index to start the search from. There is no restriction on the value of `fromIndex`. If it is greater than or equal to the length of this string, it has the same effect as if it were equal to one less than the length of this string: this entire string may be searched. If it is negative, it has the same effect as if it were -1: -1 is returned.

Returns: the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to `fromIndex`, or `-1` if the character does not occur before that point.

indexOf(String)

Declaration:
```
public int indexOf(java.lang.String str)
```

Description:
Returns the index within this string of the first occurrence of the specified substring. The integer returned is the smallest value *k* such that:

```
this.startsWith(str, k)
```

is `true`.

Parameters:
`str` - any string.

Returns: if the string argument occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring, `-1` is returned.

Throws:
`NullPointerException` - if `str` is `null`.

indexOf(String, int)

Declaration:
```
public int indexOf(java.lang.String str, int fromIndex)
```

Description:
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. The integer returned is the smallest value *k* such that:

```
this.startsWith(str, k) && (k >= fromIndex)
```

is `true`.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: `-1` is returned.

Parameters:
`str` - the substring to search for.

`fromIndex` - the index to start the search from.

Returns: If the string argument occurs as a substring within this object at a starting index no smaller than `fromIndex`, then the index of the first character of the first such substring is returned. If it does not occur as a substring starting at `fromIndex` or beyond, `-1` is returned.

Throws:
`NullPointerException` - if `str` is `null`

substring(int)

Declaration:
```
public java.lang.String substring(int beginIndex)
```

Description:
Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
“unhappy”.substring(2) returns “happy”
“Harbison”.substring(3) returns “bison”
“emptiness”.substring(9) returns “” (an empty string)
```

Parameters:
`beginIndex` - the beginning index, inclusive.

Returns: the specified substring.

Throws:
`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

substring(int, int)

Declaration:
```
public java.lang.String substring(int beginIndex, int endIndex)
```

Description:
Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:
`beginIndex` - the beginning index, inclusive.

`endIndex` - the ending index, exclusive.

Returns: the specified substring.

Throws:
`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

concat(String)

Declaration:
```
public java.lang.String concat(java.lang.String str)
```

Description:
Concatenates the specified string to the end of this string.

If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a new `String` object is created, representing a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

Parameters:
`str` - the `String` that is concatenated to the end of this `String`.

Returns: a string that represents the concatenation of this object's characters followed by the string argument's characters.

Throws:
`NullPointerException` - if `str` is `null`.

replace(char, char)

Declaration:
```
public java.lang.String replace(char oldChar, char newChar)
```

Description:
Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

If the character `oldChar` does not occur in the character sequence represented by this `String` object, then a reference to this `String` object is returned. Otherwise, a new `String` object is created that represents a character sequence identical to the character sequence represented by this `String` object, except that every occurrence of `oldChar` is replaced by an occurrence of `newChar`.

Examples:

```
"mesquite in your cellar".replace('e', 'o')
        returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
        returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
        returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```

Parameters:

`oldChar` - the old character.

`newChar` - the new character.

Returns: a string derived from this string by replacing every occurrence of `oldChar` with `newChar`.

toLowerCase()

Declaration:
`public java.lang.String toLowerCase()`

Description:
Converts all of the characters in this `String` to lower case.

Returns: the String, converted to lowercase.

See Also: `Character.toLowerCase(char)`, `toUpperCase()`

toUpperCase()

Declaration:
`public java.lang.String toUpperCase()`

Description:
Converts all of the characters in this `String` to upper case.

Returns: the String, converted to uppercase.

See Also: `Character.toLowerCase(char)`, `toUpperCase()`

trim()

Declaration:
`public java.lang.String trim()`

Description:
Removes white space from both ends of this string.

If this `String` object represents an empty character sequence, or the first and last characters of character sequence represented by this `String` object both have codes greater than `'&#92;u0020'` (the space character), then a reference to this `String` object is returned.

Otherwise, if there is no character with a code greater than `'&#92;u0020'` in the string, then a new `String` object representing an empty string is created and returned.

Otherwise, let *k* be the index of the first character in the string whose code is greater than `'&#92;u0020'`, and let *m* be the index of the last character in the string whose code is greater than `'&#92;u0020'`. A new `String` object is created, representing the substring of this string that begins with the character at index *k* and ends with the character at index *m*-that is, the result of `this.substring(k, m+1)`.

This method may be used to trim whitespace from the beginning and end of a string; in fact, it trims all ASCII control characters as well.

Returns: this string, with white space removed from the front and end.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
This object (which is already a string!) is itself returned.

Overrides: `toString` in class `Object`

Returns: the string itself.

toCharArray()

Declaration:
```
public char[] toCharArray()
```

Description:
Converts this string to a new character array.

Returns: a newly allocated character array whose length is the length of this string and whose contents are initialized to contain the character sequence represented by this string.

valueOf(Object)

Declaration:
```
public static java.lang.String valueOf(java.lang.Object obj)
```

Description:
Returns the string representation of the `Object` argument.

Parameters:
`obj` - an `Object`.

Returns: if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj.toString()` is returned.

See Also: `Object.toString()`

valueOf(char[])

Declaration:
```
public static java.lang.String valueOf(char[] data)
```

Description:
Returns the string representation of the `char` array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:
`data` - a `char` array.

Returns: a newly allocated string representing the same sequence of characters contained in the character array argument.

valueOf(char[], int, int)

Declaration:
```
public static java.lang.String valueOf(char[] data, int offset, int count)
```

Description:
Returns the string representation of a specific subarray of the `char` array argument.

The `offset` argument is the index of the first character of the subarray. The `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:
`data` - the character array.

`offset` - the initial offset into the value of the `String`.

`count` - the length of the value of the `String`.

Returns: a newly allocated string representing the sequence of characters contained in the subarray of the character array argument.

Throws:
`NullPointerException` - if `data` is `null`.

`IndexOutOfBoundsException` - if `offset` is negative, or `count` is negative, or `offset+count` is larger than `data.length`.

valueOf(boolean)

Declaration:
```
public static java.lang.String valueOf(boolean b)
```

Description:
Returns the string representation of the `boolean` argument.

Parameters:
`b` - a `boolean`.

Returns: if the argument is `true`, a string equal to `"true"` is returned; otherwise, a string equal to `"false"` is returned.

valueOf(char)

Declaration:
```
public static java.lang.String valueOf(char c)
```

Description:
Returns the string representation of the `char` argument.

Parameters:
`c` - a `char`.

Returns: a newly allocated string of length `1` containing as its single character the argument `c`.

valueOf(int)

Declaration:
```
public static java.lang.String valueOf(int i)
```

Description:
Returns the string representation of the `int` argument.

The representation is exactly the one returned by the `Integer.toString` method of one argument.

Parameters:
`i` - an `int`.

Returns: a newly allocated string containing a string representation of the `int` argument.

See Also: `Integer.toString(int, int)`

valueOf(long)

Declaration:
```
public static java.lang.String valueOf(long l)
```

Description:
Returns the string representation of the `long` argument.

The representation is exactly the one returned by the `Long.toString` method of one argument.

Parameters:
`l` - a `long`.

Returns: a newly allocated string containing a string representation of the `long` argument.

See Also: `Long.toString(long)`

valueOf(float)

Declaration:
```
public static java.lang.String valueOf(float f)
```

Description:
Returns the string representation of the `float` argument.

The representation is exactly the one returned by the `Float.toString` method of one argument.

Parameters:
`f` - a `float`.

Returns: a newly allocated string containing a string representation of the `float` argument.

Since: CLDC 1.1

See Also: `Float.toString(float)`

valueOf(double)

Declaration:
```
public static java.lang.String valueOf(double d)
```

Description:
Returns the string representation of the `double` argument.

The representation is exactly the one returned by the `Double.toString` method of one argument.

Parameters:
`d` - a `double`.

Returns: a newly allocated string containing a string representation of the `double` argument.

Since: CLDC 1.1

See Also: `Double.toString(double)`

intern()

Declaration:
`public java.lang.String intern()`

Description:
Returns a canonical representation for the string object.

A pool of strings, initially empty, is maintained privately by the class `String`.

When the intern method is invoked, if the pool already contains a string equal to this `String` object as determined by the `equals(Object)` method, then the string from the pool is returned. Otherwise, this `String` object is added to the pool and a reference to this `String` object is returned.

It follows that for any two strings `s` and `t`, `s.intern() == t.intern()` is `true` if and only if `s.equals(t)` is `true`.

All literal strings and string-valued constant expressions are interned. String literals are defined in Section 3.10.5 of the Java Language Specification (http://java.sun.com/docs/books/jls/html/)

Returns: a string that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

Since: CLDC 1.1

java.lang
StringBuffer

Declaration

```
public final class StringBuffer

java.lang.Object
  |
  +--java.lang.StringBuffer
```

Description

A string buffer implements a mutable sequence of characters. A string buffer is like a `String`, but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

String buffers are used by the compiler to implement the binary string concatenation operator +. For example, the code:

```
x = "a" + 4 + "c"
```

is compiled to the equivalent of:

```
x = new StringBuffer().append("a").append(4).append("c")
                      .toString()
```

which creates a new string buffer (initially empty), appends the string representation of each operand to the string buffer in turn, and then converts the contents of the string buffer to a string. Overall, this avoids creating many temporary strings.

The principal operations on a `StringBuffer` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string buffer. The `append` method always adds these characters at the end of the buffer; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string buffer object whose current contents are "`start`", then the method call `z.append("le")` would cause the string buffer to contain "`startle`", whereas `z.insert(4, "le")` would alter the string buffer to contain "`starlet`".

In general, if sb refers to an instance of a `StringBuffer`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`.

Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

Since: JDK1.0, CLDC 1.0

See Also: `java.io.ByteArrayOutputStream`, `String`

Member Summary

Constructors

```
                      StringBuffer()
                      StringBuffer(int length)
                      StringBuffer(String str)
```

Methods

```
           StringBuffer  append(boolean b)
           StringBuffer  append(char c)
           StringBuffer  append(char[] str)
           StringBuffer  append(char[] str, int offset, int len)
           StringBuffer  append(double d)
           StringBuffer  append(float f)
           StringBuffer  append(int i)
           StringBuffer  append(long l)
           StringBuffer  append(Object obj)
           StringBuffer  append(String str)
                    int  capacity()
                   char  charAt(int index)
           StringBuffer  delete(int start, int end)
           StringBuffer  deleteCharAt(int index)
                   void  ensureCapacity(int minimumCapacity)
                   void  getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
           StringBuffer  insert(int offset, boolean b)
           StringBuffer  insert(int offset, char c)
           StringBuffer  insert(int offset, char[] str)
           StringBuffer  insert(int offset, double d)
           StringBuffer  insert(int offset, float f)
           StringBuffer  insert(int offset, int i)
           StringBuffer  insert(int offset, long l)
           StringBuffer  insert(int offset, Object obj)
           StringBuffer  insert(int offset, String str)
                    int  length()
           StringBuffer  reverse()
                   void  setCharAt(int index, char ch)
                   void  setLength(int newLength)
                 String  toString()
```

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

StringBuffer()

Declaration:
```
public StringBuffer()
```

Description:
Constructs a string buffer with no characters in it and an initial capacity of 16 characters.

StringBuffer(int)

Declaration:
```
public StringBuffer(int length)
```

Description:
Constructs a string buffer with no characters in it and an initial capacity specified by the `length` argument.

Parameters:
`length` - the initial capacity.

Throws:
`NegativeArraySizeException` - if the `length` argument is less than `0`.

StringBuffer(String)

Declaration:
```
public StringBuffer(java.lang.String str)
```

Description:
Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string. The initial capacity of the string buffer is `16` plus the length of the string argument.

Parameters:
`str` - the initial contents of the buffer.

Methods

length()

Declaration:
```
public int length()
```

Description:
Returns the length (character count) of this string buffer.

Returns: the length of the sequence of characters currently represented by this string buffer.

capacity()

Declaration:
```
public int capacity()
```

Description:
Returns the current capacity of the String buffer. The capacity is the amount of storage available for newly inserted characters; beyond which an allocation will occur.

Returns: the current capacity of this string buffer.

ensureCapacity(int)

Declaration:
```
public void ensureCapacity(int minimumCapacity)
```

Description:
Ensures that the capacity of the buffer is at least equal to the specified minimum. If the current capacity of this string buffer is less than the argument, then a new internal buffer is allocated with greater capacity. The new capacity is the larger of:

- The `minimumCapacity` argument.
- Twice the old capacity, plus `2`.

If the `minimumCapacity` argument is nonpositive, this method takes no action and simply returns.

Parameters:
`minimumCapacity` - the minimum desired capacity.

setLength(int)

Declaration:
```
public void setLength(int newLength)
```

Description:
Sets the length of this string buffer. This string buffer is altered to represent a new character sequence whose length is specified by the argument. For every nonnegative index *k* less than `newLength`, the character at index *k* in the new character sequence is the same as the character at index *k* in the old sequence if *k* is less than the length of the old character sequence; otherwise, it is the null character `'&#92;u0000'`. In other words, if the `newLength` argument is less than the current length of the string buffer, the string buffer is truncated to contain exactly the number of characters given by the `newLength` argument.

If the `newLength` argument is greater than or equal to the current length, sufficient null characters (`'&#92;u0000'`) are appended to the string buffer so that length becomes the `newLength` argument.

The `newLength` argument must be greater than or equal to `0`.

Parameters:
`newLength` - the new length of the buffer.

Throws:
`IndexOutOfBoundsException` - if the `newLength` argument is negative.

See Also: `length()`

charAt(int)

Declaration:
```
public char charAt(int index)
```

Description:
The specified character of the sequence currently represented by the string buffer, as indicated by the `index` argument, is returned. The first character of a string buffer is at index `0`, the next at index `1`, and so on, for array indexing.

The index argument must be greater than or equal to `0`, and less than the length of this string buffer.

Parameters:
`index` - the index of the desired character.

Returns: the character at the specified index of this string buffer.

Throws:
`IndexOutOfBoundsException` - if `index` is negative or greater than or equal to `length()`.

See Also: `length()`

getChars(int, int, char[], int)

Declaration:
```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

Description:
Characters are copied from this string buffer into the destination character array `dst`. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1`. The total number of characters to be copied is `srcEnd-srcBegin`. The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:
`srcBegin` - start copying at this offset in the string buffer.

`srcEnd` - stop copying at this offset in the string buffer.

`dst` - the array to copy the data into.

`dstBegin` - offset into `dst`.

Throws:
`NullPointerException` - if `dst` is `null`.

`IndexOutOfBoundsException` - if any of the following is true:

- `srcBegin` is negative
- `dstBegin` is negative
- the `srcBegin` argument is greater than the `srcEnd` argument.
- `srcEnd` is greater than `this.length()`, the current length of this string buffer.
- `dstBegin+srcEnd-srcBegin` is greater than `dst.length`

setCharAt(int, char)

Declaration:
```
public void setCharAt(int index, char ch)
```

Description:
The character at the specified index of this string buffer is set to `ch`. The string buffer is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character `ch` at position `index`.

The offset argument must be greater than or equal to `0`, and less than the length of this string buffer.

Parameters:
`index` - the index of the character to modify.

`ch` - the new character.

Throws:
`IndexOutOfBoundsException` - if `index` is negative or greater than or equal to `length()`.

See Also: `length()`

append(Object)

Declaration:
```
public java.lang.StringBuffer append(java.lang.Object obj)
```

Description:
Appends the string representation of the `Object` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`obj` - an `Object`.

Returns: a reference to this `StringBuffer` object.

See Also: `String.valueOf(Object)`, `append(String)`

append(String)

Declaration:
```
public java.lang.StringBuffer append(java.lang.String str)
```

Description:
Appends the string to this string buffer.

The characters of the `String` argument are appended, in order, to the contents of this string buffer, increasing the length of this string buffer by the length of the argument. If `str` is `null`, then the four characters `"null"` are appended to this string buffer.

Let *n* be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `append` method. Then the character at index *k* in the new character sequence is equal to the character at index *k* in the old character sequence, if *k* is less than *n*; otherwise, it is equal to the character at index *k*-*n* in the argument `str`.

Parameters:
`str` - a string.

Returns: a reference to this `StringBuffer`.

append(char[])

Declaration:
```
public java.lang.StringBuffer append(char[] str)
```

Description:
Appends the string representation of the `char` array argument to this string buffer.

The characters of the array argument are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the length of the argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char[])` and the characters of that string were then `appended` to this `StringBuffer` object.

Parameters:
`str` - the characters to be appended.

Returns: a reference to this `StringBuffer` object.

append(char[], int, int)

Declaration:
```
public java.lang.StringBuffer append(char[] str, int offset, int len)
```

Description:
Appends the string representation of a subarray of the `char` array argument to this string buffer.

Characters of the character array `str`, starting at index `offset`, are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the value of `len`.

The overall effect is exactly as if the arguments were converted to a string by the method `String.valueOf(char[], int, int)` and the characters of that string were then `appended` to this `StringBuffer` object.

Parameters:
`str` - the characters to be appended.

`offset` - the index of the first character to append.

`len` - the number of characters to append.

Returns: a reference to this `StringBuffer` object.

append(boolean)

Declaration:
```
public java.lang.StringBuffer append(boolean b)
```

Description:
Appends the string representation of the `boolean` argument to the string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`b` - a `boolean`.

Returns: a reference to this `StringBuffer`.

See Also: `String.valueOf(boolean)`, `append(String)`

append(char)

Declaration:
```
public java.lang.StringBuffer append(char c)
```

Description:
Appends the string representation of the `char` argument to this string buffer.

The argument is appended to the contents of this string buffer. The length of this string buffer increases by `1`.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char)` and the character in that string were then `appended` to this `StringBuffer` object.

Parameters:
`c` - a `char`.

Returns: a reference to this `StringBuffer` object.

append(int)

Declaration:
```
public java.lang.StringBuffer append(int i)
```

Description:
Appends the string representation of the `int` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`i` - an `int`.

Returns: a reference to this `StringBuffer` object.

See Also: `String.valueOf(int)`, `append(String)`

append(long)

Declaration:
```
public java.lang.StringBuffer append(long l)
```

Description:
Appends the string representation of the `long` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`l` - a `long`.

Returns: a reference to this `StringBuffer` object.

See Also: `String.valueOf(long)`, `append(String)`

append(float)

Declaration:
```
public java.lang.StringBuffer append(float f)
```

Description:
Appends the string representation of the `float` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

append(double)

Parameters:
`f` - a `float`.

Returns: a reference to this `StringBuffer` object.

Since: CLDC 1.1

See Also: `String.valueOf(float)`, `append(String)`

append(double)

Declaration:
```
public java.lang.StringBuffer append(double d)
```

Description:
Appends the string representation of the `double` argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then appended to this string buffer.

Parameters:
`d` - a `double`.

Returns: a reference to this `StringBuffer` object.

Since: CLDC 1.1

See Also: `String.valueOf(double)`, `append(String)`

delete(int, int)

Declaration:
```
public java.lang.StringBuffer delete(int start, int end)
```

Description:
Removes the characters in a substring of this `StringBuffer`. The substring begins at the specified `start` and extends to the character at index `end - 1` or to the end of the `StringBuffer` if no such character exists. If `start` is equal to `end`, no changes are made.

Parameters:
`start` - The beginning index, inclusive.

`end` - The ending index, exclusive.

Returns: This string buffer.

Throws:
`StringIndexOutOfBoundsException` - if `start` is negative, greater than `length()`, or greater than `end`.

Since: JDK1.2

deleteCharAt(int)

Declaration:
```
public java.lang.StringBuffer deleteCharAt(int index)
```

Description:
Removes the character at the specified position in this `StringBuffer` (shortening the `StringBuffer` by one character).

Parameters:
`index` - Index of character to remove

Returns: This string buffer.

Throws:
`StringIndexOutOfBoundsException` - if the `index` is negative or greater than or equal to `length()`.

Since: JDK1.2

insert(int, Object)

Declaration:
```
public java.lang.StringBuffer insert(int offset, java.lang.Object obj)
```

Description:
Inserts the string representation of the `Object` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`obj` - an `Object`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

See Also: `String.valueOf(Object)`, `insert(int, String)`, `length()`

insert(int, String)

Declaration:
```
public java.lang.StringBuffer insert(int offset, java.lang.String str)
```

Description:
Inserts the string into this string buffer.

The characters of the `String` argument are inserted, in order, into this string buffer at the indicated offset, moving up any characters originally above that position and increasing the length of this string buffer by the length of the argument. If `str` is `null`, then the four characters `"null"` are inserted into this string buffer.

The character at index *k* in the new character sequence is equal to:

- the character at index *k* in the old character sequence, if *k* is less than `offset`
- the character at index *k*`-offset` in the argument `str`, if *k* is not less than `offset` but is less than `offset+str.length()`
- the character at index *k*`-str.length()` in the old character sequence, if *k* is not less than `offset+str.length()`

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`str` - a string.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

See Also: `length()`

insert(int, char[])

Declaration:
```
public java.lang.StringBuffer insert(int offset, char[] str)
```

Description:
Inserts the string representation of the `char` array argument into this string buffer.

The characters of the array argument are inserted into the contents of this string buffer at the position indicated by `offset`. The length of this string buffer increases by the length of the argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char[])` and the characters of that string were then `inserted` into this `StringBuffer` object at the position indicated by `offset`.

Parameters:
`offset` - the offset.

`str` - a character array.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

insert(int, boolean)

Declaration:
```
public java.lang.StringBuffer insert(int offset, boolean b)
```

Description:
Inserts the string representation of the `boolean` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`b` - a `boolean`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

See Also: `String.valueOf(boolean)`, `insert(int, String)`, `length()`

insert(int, char)

Declaration:
```
public java.lang.StringBuffer insert(int offset, char c)
```

Description:
Inserts the string representation of the `char` argument into this string buffer.

The second argument is inserted into the contents of this string buffer at the position indicated by `offset`. The length of this string buffer increases by one.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char)` and the character in that string were then `inserted` into this `StringBuffer` object at the position indicated by `offset`.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`c` - a `char`.

Returns: a reference to this `StringBuffer` object.

Throws:
`IndexOutOfBoundsException` - if the offset is invalid.

See Also: `length()`

insert(int, int)

Declaration:
```
public java.lang.StringBuffer insert(int offset, int i)
```

Description:
Inserts the string representation of the second `int` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`i` - an `int`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

See Also: `String.valueOf(int)`, `insert(int, String)`, `length()`

insert(int, long)

Declaration:
```
public java.lang.StringBuffer insert(int offset, long l)
```

Description:
Inserts the string representation of the `long` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the position indicated by `offset`.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`l` - a `long`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

See Also: `String.valueOf(long)`, `insert(int, String)`, `length()`

insert(int, float)

Declaration:
```
public java.lang.StringBuffer insert(int offset, float f)
```

Description:
Inserts the string representation of the `float` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`f` - a `float`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

Since: CLDC 1.1

See Also: `String.valueOf(float)`, `insert(int, String)`, `length()`

insert(int, double)

Declaration:
```
public java.lang.StringBuffer insert(int offset, double d)
```

Description:
Inserts the string representation of the `double` argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf`, and the characters of that string are then inserted into this string buffer at the indicated offset.

The offset argument must be greater than or equal to `0`, and less than or equal to the length of this string buffer.

Parameters:
`offset` - the offset.

`d` - a `double`.

Returns: a reference to this `StringBuffer` object.

Throws:
`StringIndexOutOfBoundsException` - if the offset is invalid.

Since: CLDC 1.1

See Also: `String.valueOf(double)`, `insert(int, String)`, `length()`

reverse()

Declaration:
`public java.lang.StringBuffer reverse()`

Description:
The character sequence contained in this string buffer is replaced by the reverse of the sequence.

Let *n* be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `reverse` method. Then the character at index *k* in the new character sequence is equal to the character at index *n-k-1* in the old character sequence.

Returns: a reference to this `StringBuffer` object..

Since: JDK1.0.2

toString()

Declaration:
`public java.lang.String toString()`

Description:
Converts to a string representing the data in this string buffer. A new `String` object is allocated and initialized to contain the character sequence currently represented by this string buffer. This `String` is then returned. Subsequent changes to the string buffer do not affect the contents of the `String`.

Implementation advice: This method can be coded so as to create a new `String` object without allocating new memory to hold a copy of the character sequence. Instead, the string can share the memory used by the string buffer. Any subsequent operation that alters the content or capacity of the string buffer must then make a copy of the internal buffer at that time. This strategy is effective for reducing the amount of memory allocated by a string concatenation operation when it is implemented using a string buffer.

Overrides: `toString` in class `Object`

Returns: a string representation of the string buffer.

java.lang
StringIndexOutOfBoundsException

Declaration

```
public class StringIndexOutOfBoundsException extends IndexOutOfBoundsException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.lang.IndexOutOfBoundsException
                          |
                          +--java.lang.StringIndexOutOfBoundsException
```

Description

Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.

Since: JDK1.0, CLDC 1.0

See Also: `String.charAt(int)`

Member Summary	
Constructors	
	`StringIndexOutOfBoundsException()`
	`StringIndexOutOfBoundsException(int index)`
	`StringIndexOutOfBoundsException(String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

StringIndexOutOfBoundsException()

Declaration:
```
public StringIndexOutOfBoundsException()
```

Description:
Constructs a `StringIndexOutOfBoundsException` with no detail message.

Since: JDK1.0.

StringIndexOutOfBoundsException(String)

Declaration:
```
public StringIndexOutOfBoundsException(java.lang.String s)
```

Description:
Constructs a `StringIndexOutOfBoundsException` with the specified detail message.

Parameters:
`s` - the detail message.

StringIndexOutOfBoundsException(int)

Declaration:
```
public StringIndexOutOfBoundsException(int index)
```

Description:
Constructs a new `StringIndexOutOfBoundsException` class with an argument indicating the illegal index.

Parameters:
`index` - the illegal index.

java.lang
System

Declaration
```
public final class System
```

```
java.lang.Object
  |
  +--java.lang.System
```

Description
The `System` class contains several useful class fields and methods. It cannot be instantiated.

Since: JDK1.0, CLDC 1.0

Member Summary	
Fields	
static	err
java.io.PrintStream	
static	out
java.io.PrintStream	
Methods	
static void	arraycopy(Object src, int srcOffset, Object dst, int dstOffset, int length)
static long	currentTimeMillis()
static void	exit(int status)
static void	gc()
static String	getProperty(String key)
static int	identityHashCode(Object x)

Inherited Member Summary
Methods inherited from class `Object`
equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

out

Declaration:
```
public static final java.io.PrintStream out
```

Description:
The “standard” output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

For simple stand-alone Java applications, a typical way to write a line of output data is:

```
System.out.println(data)
```

See the `println` methods in class `PrintStream`.

See Also: `java.io.PrintStream.println()`, `java.io.PrintStream.println(boolean)`, `java.io.PrintStream.println(char)`, `java.io.PrintStream.println(char[])`, `java.io.PrintStream.println(int)`, `java.io.PrintStream.println(long)`, `java.io.PrintStream.println(Object)`, `java.io.PrintStream.println(String)`

err

Declaration:
```
public static final java.io.PrintStream err
```

Description:
The “standard” error output stream. This stream is already open and ready to accept output data.

Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable `out`, has been redirected to a file or other destination that is typically not continuously monitored.

Methods

currentTimeMillis()

Declaration:
```
public static long currentTimeMillis()
```

Description:
Returns the current time in milliseconds.

Returns: the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

arraycopy(Object, int, Object, int, int)

Declaration:
```
public static void arraycopy(java.lang.Object src, int srcOffset, java.lang.Object dst,
            int dstOffset, int length)
```

Description:
Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. A subsequence of array components are copied from the source array referenced by `src` to the destination array referenced by `dst`. The number of components copied is equal to the `length`

argument. The components at positions `srcOffset` through `srcOffset+length-1` in the source array are copied into positions `dstOffset` through `dstOffset+length-1`, respectively, of the destination array.

If the `src` and `dst` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOffset` through `srcOffset+length-1` were first copied to a temporary array with `length` components and then the contents of the temporary array were copied into positions `dstOffset` through `dstOffset+length-1` of the destination array.

If `dst` is `null`, then a `NullPointerException` is thrown.

If `src` is `null`, then a `NullPointerException` is thrown and the destination array is not modified.

Otherwise, if any of the following is true, an `ArrayStoreException` is thrown and the destination is not modified:

- The `src` argument refers to an object that is not an array.
- The `dst` argument refers to an object that is not an array.
- The `src` argument and `dst` argument refer to arrays whose component types are different primitive types.
- The `src` argument refers to an array with a primitive component type and the `dst` argument refers to an array with a reference component type.
- The `src` argument refers to an array with a reference component type and the `dst` argument refers to an array with a primitive component type.

Otherwise, if any of the following is true, an `IndexOutOfBoundsException` is thrown and the destination is not modified:

- The `srcOffset` argument is negative.
- The `dstOffset` argument is negative.
- The `length` argument is negative.
- `srcOffset+length` is greater than `src.length`, the length of the source array.
- `dstOffset+length` is greater than `dst.length`, the length of the destination array.

Otherwise, if any actual component of the source array from position `srcOffset` through `srcOffset+length-1` cannot be converted to the component type of the destination array by assignment conversion, an `ArrayStoreException` is thrown. In this case, let *k* be the smallest nonnegative integer less than length such that `src[srcOffset+`*k*`]` cannot be converted to the component type of the destination array; when the exception is thrown, source array components from positions `srcOffset` through `srcOffset+`*k*`-1` will already have been copied to destination array positions `dstOffset` through `dstOffset+`*k*`-1` and no other positions of the destination array will have been modified. (Because of the restrictions already itemized, this paragraph effectively applies only to the situation where both arrays have component types that are reference types.)

Parameters:

`src` - the source array.

`srcOffset` - start position in the source array.

`dst` - the destination array.

`dstOffset` - start position in the destination data.

`length` - the number of array elements to be copied.

Throws:

`IndexOutOfBoundsException` - if copying would cause access of data outside array bounds.

`ArrayStoreException` - if an element in the `src` array could not be stored into the `dest` array because of a type mismatch.

`NullPointerException` - if either `src` or `dst` is `null`.

identityHashCode(Object)

Declaration:
```
public static int identityHashCode(java.lang.Object x)
```

Description:
Returns the same hashcode for the given object as would be returned by the default method hashCode(), whether or not the given object's class overrides hashCode(). The hashcode for the null reference is zero.

Parameters:

`x` - object for which the hashCode is to be calculated

Returns: the hashCode

Since: JDK1.1

getProperty(String)

Declaration:
```
public static java.lang.String getProperty(java.lang.String key)
```

Description:
Gets the system property indicated by the specified key.

Parameters:

`key` - the name of the system property.

Returns: the string value of the system property, or `null` if there is no property with that key.

Throws:

`NullPointerException` - if `key` is `null`.

`IllegalArgumentException` - if `key` is empty.

exit(int)

Declaration:
```
public static void exit(int status)
```

Description:
Terminates the currently running Java application. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

This method calls the `exit` method in class `Runtime`. This method never returns normally.

The call `System.exit(n)` is effectively equivalent to the call:

```
Runtime.getRuntime().exit(n)
```

Parameters:

`status` - exit status.

See Also: `Runtime.exit(int)`

gc()

Declaration:
```
public static void gc()
```

Description:
Runs the garbage collector.

Calling the `gc` method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all discarded objects.

The call `System.gc()` is effectively equivalent to the call:

```
Runtime.getRuntime().gc()
```

See Also: `Runtime.gc()`

java.lang
Thread

Declaration

```
public class Thread implements Runnable

java.lang.Object
  |
  +--java.lang.Thread
```

All Implemented Interfaces: Runnable

Description

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of `Thread`. This subclass should override the `run` method of class `Thread`. An instance of the subclass can then be allocated and started. For example, a thread that computes primes larger than a stated value could be written as follows:

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
p.start();
```

The other way to create a thread is to declare a class that implements the `Runnable` interface. That class then implements the `run` method. An instance of the class can then be allocated, passed as an argument when creating `Thread`, and started. The same example in this other style looks like the following:

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Since: JDK1.0, CLDC 1.0

See Also: `Runnable`, `Runtime.exit(int)`, `run()`

Member Summary

Fields

static int	MAX_PRIORITY
static int	MIN_PRIORITY
static int	NORM_PRIORITY

Constructors

	Thread()
	Thread(Runnable target)
	Thread(Runnable target, String name)
	Thread(String name)

Methods

static int	activeCount()
static Thread	currentThread()
String	getName()
int	getPriority()
void	interrupt()
boolean	isAlive()
void	join()
void	run()
void	setPriority(int newPriority)
static void	sleep(long millis)
void	start()
String	toString()
static void	yield()

Inherited Member Summary

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Fields

MIN_PRIORITY

Declaration:
`public static final int` **`MIN_PRIORITY`**

Description:
The minimum priority that a thread can have.

NORM_PRIORITY

Declaration:
```
public static final int NORM_PRIORITY
```

Description:
The default priority that is assigned to a thread.

MAX_PRIORITY

Declaration:
```
public static final int MAX_PRIORITY
```

Description:
The maximum priority that a thread can have.

Constructors

Thread()

Declaration:
```
public Thread()
```

Description:
Allocates a new `Thread` object.

Threads created this way must have overridden their `run()` method to actually do anything.

See Also: `Runnable`

Thread(String)

Declaration:
```
public Thread(java.lang.String name)
```

Description:
Allocates a new `Thread` object with the given name. Threads created this way must have overridden their `run()` method to actually do anything.

Parameters:
`name` - the name of the new thread.

Thread(Runnable)

Declaration:
```
public Thread(java.lang.Runnable target)
```

Description:
Allocates a new `Thread` object with a specific target object whose `run` method is called.

Parameters:
`target` - the object whose `run` method is called.

Thread(Runnable, String)

Declaration:
```
public Thread(java.lang.Runnable target, java.lang.String name)
```

Description:
Allocates a new `Thread` object with the given target and name.

Parameters:
`target` - the object whose `run` method is called.

`name` - the name of the new thread.

Methods

currentThread()

Declaration:
```
public static java.lang.Thread currentThread()
```

Description:
Returns a reference to the currently executing `Thread` object.

Returns: the currently executing thread.

yield()

Declaration:
```
public static void yield()
```

Description:
Causes the currently executing thread object to temporarily pause and allow other threads to execute.

sleep(long)

Declaration:
```
public static void sleep(long millis)
            throws InterruptedException
```

Description:
Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.

Parameters:
`millis` - the length of time to sleep in milliseconds.

Throws:
`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also: `Object.notify()`

start()

Declaration:
```
public void start()
```

Description:
Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the other thread (which executes its `run` method).

Throws:
`IllegalThreadStateException` - if the thread was already started.

See Also: `run()`

run()

Declaration:
```
public void run()
```

Description:
If this thread was constructed using a separate `Runnable` run object, then that `Runnable` object's `run` method is called; otherwise, this method does nothing and returns.

Subclasses of `Thread` should override this method.

Specified By: `run` in interface `Runnable`

See Also: `start()`, `Runnable.run()`

interrupt()

Declaration:
```
public void interrupt()
```

Description:
Interrupts this thread. In an implementation conforming to the CLDC Specification, this operation is not required to cancel or clean up any pending I/O operations that the thread may be waiting for.

Since: JDK 1.0, CLDC 1.1

isAlive()

Declaration:
```
public final boolean isAlive()
```

Description:
Tests if this thread is alive. A thread is alive if it has been started and has not yet died.

Returns: `true` if this thread is alive; `false` otherwise.

setPriority(int)

Declaration:
```
public final void setPriority(int newPriority)
```

Description:
Changes the priority of this thread.

Parameters:
`newPriority` - priority to set this thread to

Throws:
`IllegalArgumentException` - If the priority is not in the range `MIN_PRIORITY` to `MAX_PRIORITY`.

See Also: `getPriority()`, `MAX_PRIORITY`, `MIN_PRIORITY`

getPriority()

Declaration:
```
public final int getPriority()
```

Description:
Returns this thread's priority.

Returns: this thread's priority.

See Also: `setPriority(int)`

getName()

Declaration:
```
public final java.lang.String getName()
```

Description:
Returns this thread's name. Note that in CLDC the name of the thread can only be set when creating the thread.

Returns: this thread's name.

activeCount()

Declaration:
```
public static int activeCount()
```

Description:
Returns the current number of active threads in the virtual machine.

Returns: the current number of active threads.

join()

Declaration:
```
public final void join()
            throws InterruptedException
```

Description:
Waits for this thread to die.

Throws:
`InterruptedException` - if another thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a string representation of this thread, including the thread's name and priority.

Overrides: `toString` in class `Object`

Returns: a string representation of this thread.

java.lang
Throwable

Declaration

```
public class Throwable

java.lang.Object
  |
  +--java.lang.Throwable
```

Direct Known Subclasses: `Error`, `Exception`

Description

The `Throwable` class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java `throw` statement. Similarly, only this class or one of its subclasses can be the argument type in a `catch` clause.

Instances of two subclasses, `Error` and `Exception`, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

By convention, class `Throwable` and its subclasses have two constructors, one that takes no arguments and one that takes a `String` argument that can be used to produce an error message.

A `Throwable` class contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error.

Here is one example of catching an exception:

```
try {
    int a[] = new int[2];
    int b = a[4];
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("exception: " + e.getMessage());
    e.printStackTrace();
}
```

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	`Throwable()`
	`Throwable(String message)`
Methods	
`String`	`getMessage()`
`void`	`printStackTrace()`
`String`	`toString()`

Throwable()

Inherited Member Summary
Methods inherited from class `Object` `equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

Throwable()

Declaration:
`public Throwable()`

Description:
Constructs a new `Throwable` with `null` as its error message string.

Throwable(String)

Declaration:
`public Throwable(java.lang.String message)`

Description:
Constructs a new `Throwable` with the specified error message.

Parameters:
`message` - the error message. The error message is saved for later retrieval by the `getMessage()` method.

Methods

getMessage()

Declaration:
`public java.lang.String getMessage()`

Description:
Returns the error message string of this `Throwable` object.

Returns: the error message string of this `Throwable` object if it was `created` with an error message string; or `null` if it was `created` with no error message.

toString()

Declaration:
`public java.lang.String toString()`

Description:
Returns a short description of this `Throwable` object. If this `Throwable` object was `created` with an error message string, then the result is the concatenation of three strings:

- The name of the actual class of this object
- ": " (a colon and a space)

- The result of the `getMessage()` method for this object

If this `Throwable` object was `created` with no error message string, then the name of the actual class of this object is returned.

Overrides: `toString` in class `Object`

Returns: a string representation of this `Throwable`.

printStackTrace()

Declaration:
```
public void printStackTrace()
```

Description:
Prints this `Throwable` and its backtrace to the standard error stream. This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object.

The format of the backtrace information depends on the implementation.

java.lang
VirtualMachineError

Declaration
```
public abstract class VirtualMachineError extends Error
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Error
              |
              +--java.lang.VirtualMachineError
```

Direct Known Subclasses: OutOfMemoryError

Description
Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

Since: JDK1.0, CLDC 1.0

Member Summary	
Constructors	
	VirtualMachineError() VirtualMachineError(String s)

Inherited Member Summary
Methods inherited from class Object equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait() **Methods inherited from class Throwable** getMessage(), printStackTrace(), toString()

Constructors

VirtualMachineError()

Declaration:
```
public VirtualMachineError()
```

Description:
Constructs a `VirtualMachineError` with no detail message.

VirtualMachineError(String)

Declaration:
```
public VirtualMachineError(java.lang.String s)
```

Description:
Constructs a `VirtualMachineError` with the specified detail message.

Parameters:
`s` - the detail message.

VirtualMachineError java.lang

VirtualMachineError(String)

CHAPTER 3

Package java.util

Description

Contains the collection classes, and the date and time facilities.

Since: CLDC 1.0

Class Summary	
Interfaces	
`Enumeration`	An object that implements the Enumeration interface generates a series of elements, one at a time.
Classes	
`Calendar`	`Calendar` is an abstract base class for converting between a `Date` object and a set of integer fields such as `YEAR`, `MONTH`, `DAY`, `HOUR`, and so on.
`Date`	The class Date represents a specific instant in time, with millisecond precision.
`Hashtable`	This class implements a hashtable, which maps keys to values.
`Random`	An instance of this class is used to generate a stream of pseudorandom numbers.
`Stack`	The `Stack` class represents a last-in-first-out (LIFO) stack of objects.
`TimeZone`	`TimeZone` represents a time zone offset, and also figures out daylight savings.
`Vector`	The `Vector` class implements a growable array of objects.
Exceptions	
`EmptyStackException`	Thrown by methods in the `Stack` class to indicate that the stack is empty.
`NoSuchElementException`	Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.

java.util
Calendar

Declaration

```
public abstract class Calendar
```

```
java.lang.Object
  |
  +--java.util.Calendar
```

Description

`Calendar` is an abstract base class for converting between a `Date` object and a set of integer fields such as `YEAR`, `MONTH`, `DAY`, `HOUR`, and so on. (A `Date` object represents a specific instant in time with millisecond precision. See `Date` for information about the `Date` class.)

Subclasses of `Calendar` interpret a `Date` according to the rules of a specific calendar system.

Like other locale-sensitive classes, `Calendar` provides a class method, `getInstance`, for getting a generally useful object of this type.

```
Calendar rightNow = Calendar.getInstance();
```

A `Calendar` object can produce all the time field values needed to implement the date-time formatting for a particular language and calendar style (for example, Japanese-Gregorian, Japanese-Traditional).

When computing a `Date` from time fields, there may be insufficient information to compute the `Date` (such as only year and month but no day in the month).

Insufficient information. The calendar will use default information to specify the missing fields. This may vary by calendar; for the Gregorian calendar, the default for a field is the same as that of the start of the epoch: i.e., YEAR = 1970, MONTH = JANUARY, DATE = 1, etc. **Note:** The ambiguity in interpretation of what day midnight belongs to, is resolved as so: midnight “belongs” to the following day.
23:59 on Dec 31, 1969 < 00:00 on Jan 1, 1970.
12:00 PM is midday, and 12:00 AM is midnight.
11:59 PM on Jan 1 < 12:00 AM on Jan 2 < 12:01 AM on Jan 2.
11:59 AM on Mar 10 < 12:00 PM on Mar 10 < 12:01 PM on Mar 10.
24:00 or greater are invalid. Hours greater than 12 are invalid in AM/PM mode. Setting the time will never change the date.

If equivalent times are entered in AM/PM or 24 hour mode, equality will be determined by the actual time rather than the entered time.

This class has been subset for J2ME based on the JDK 1.3 Calendar class. Many methods and variables have been pruned, and other methods simplified, in an effort to reduce the size of this class.

See Also: `Date`, `TimeZone`

Member Summary	
Fields	
`static int`	`AM`
`static int`	`AM_PM`

Member Summary

static int	APRIL
static int	AUGUST
static int	DATE
static int	DAY_OF_MONTH
static int	DAY_OF_WEEK
static int	DECEMBER
static int	FEBRUARY
protected int[]	fields
static int	FRIDAY
static int	HOUR
static int	HOUR_OF_DAY
protected boolean[]	isSet
static int	JANUARY
static int	JULY
static int	JUNE
static int	MARCH
static int	MAY
static int	MILLISECOND
static int	MINUTE
static int	MONDAY
static int	MONTH
static int	NOVEMBER
static int	OCTOBER
static int	PM
static int	SATURDAY
static int	SECOND
static int	SEPTEMBER
static int	SUNDAY
static int	THURSDAY
protected long	time
static int	TUESDAY
static int	WEDNESDAY
static int	YEAR

Constructors

protected	Calendar()

Methods

boolean	after(java.lang.Object when)
boolean	before(java.lang.Object when)
protected abstract void	computeFields()
protected abstract void	computeTime()
boolean	equals(java.lang.Object obj)
int	get(int field)
static Calendar	getInstance()
static Calendar	getInstance(TimeZone zone)
Date	getTime()
protected long	getTimeInMillis()
TimeZone	getTimeZone()
void	set(int field, int value)
void	setTime(Date date)
protected void	setTimeInMillis(long millis)

Member Summary
void setTimeZone(TimeZone value)

Inherited Member Summary
Methods inherited from class `Object` `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Fields

YEAR

Declaration:
`public static final int YEAR`

Description:
Field number for `get` and `set` indicating the year. This is a calendar-specific value.

MONTH

Declaration:
`public static final int MONTH`

Description:
Field number for `get` and `set` indicating the month. This is a calendar-specific value.

DATE

Declaration:
`public static final int DATE`

Description:
Field number for `get` and `set` indicating the day of the month. This is a synonym for `DAY_OF_MONTH`.

See Also: `DAY_OF_MONTH`

DAY_OF_MONTH

Declaration:
`public static final int DAY_OF_MONTH`

Description:
Field number for `get` and `set` indicating the day of the month. This is a synonym for `DATE`.

See Also: `DATE`

DAY_OF_WEEK

Declaration:
`public static final int DAY_OF_WEEK`

Description:
Field number for `get` and `set` indicating the day of the week.

AM_PM

Declaration:
```
public static final int AM_PM
```

Description:
Field number for `get` and `set` indicating whether the `HOUR` is before or after noon. E.g., at 10:04:15.250 PM the `AM_PM` is `PM`.

See Also: `AM`, `PM`, `HOUR`

HOUR

Declaration:
```
public static final int HOUR
```

Description:
Field number for `get` and `set` indicating the hour of the morning or afternoon. `HOUR` is used for the 12-hour clock. E.g., at 10:04:15.250 PM the `HOUR` is 10.

See Also: `AM_PM`, `HOUR_OF_DAY`

HOUR_OF_DAY

Declaration:
```
public static final int HOUR_OF_DAY
```

Description:
Field number for `get` and `set` indicating the hour of the day. `HOUR_OF_DAY` is used for the 24-hour clock. E.g., at 10:04:15.250 PM the `HOUR_OF_DAY` is 22.

MINUTE

Declaration:
```
public static final int MINUTE
```

Description:
Field number for `get` and `set` indicating the minute within the hour. E.g., at 10:04:15.250 PM the `MINUTE` is 4.

SECOND

Declaration:
```
public static final int SECOND
```

Description:
Field number for `get` and `set` indicating the second within the minute. E.g., at 10:04:15.250 PM the `SECOND` is 15.

MILLISECOND

Declaration:
```
public static final int MILLISECOND
```

Description:
Field number for `get` and `set` indicating the millisecond within the second. E.g., at 10:04:15.250 PM the `MILLISECOND` is 250.

SUNDAY

Declaration:
```
public static final int SUNDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Sunday.

MONDAY

Declaration:
```
public static final int MONDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Monday.

TUESDAY

Declaration:
```
public static final int TUESDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Tuesday.

WEDNESDAY

Declaration:
```
public static final int WEDNESDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Wednesday.

THURSDAY

Declaration:
```
public static final int THURSDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Thursday.

FRIDAY

Declaration:
```
public static final int FRIDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Friday.

SATURDAY

Declaration:
```
public static final int SATURDAY
```

Description:
Value of the `DAY_OF_WEEK` field indicating Saturday.

JANUARY

Declaration:
```
public static final int JANUARY
```

Description:
Value of the `MONTH` field indicating the first month of the year.

FEBRUARY

Declaration:
```
public static final int FEBRUARY
```

Description:
Value of the `MONTH` field indicating the second month of the year.

MARCH

Declaration:
```
public static final int MARCH
```

Description:
Value of the `MONTH` field indicating the third month of the year.

APRIL

Declaration:
```
public static final int APRIL
```

Description:
Value of the `MONTH` field indicating the fourth month of the year.

MAY

Declaration:
```
public static final int MAY
```

Description:
Value of the `MONTH` field indicating the fifth month of the year.

JUNE

Declaration:
```
public static final int JUNE
```

Description:
Value of the `MONTH` field indicating the sixth month of the year.

JULY

Declaration:
```
public static final int JULY
```

Description:
Value of the `MONTH` field indicating the seventh month of the year.

AUGUST

Declaration:
```
public static final int AUGUST
```

Description:
Value of the MONTH field indicating the eighth month of the year.

SEPTEMBER

Declaration:
```
public static final int SEPTEMBER
```

Description:
Value of the MONTH field indicating the ninth month of the year.

OCTOBER

Declaration:
```
public static final int OCTOBER
```

Description:
Value of the MONTH field indicating the tenth month of the year.

NOVEMBER

Declaration:
```
public static final int NOVEMBER
```

Description:
Value of the MONTH field indicating the eleventh month of the year.

DECEMBER

Declaration:
```
public static final int DECEMBER
```

Description:
Value of the MONTH field indicating the twelfth month of the year.

AM

Declaration:
```
public static final int AM
```

Description:
Value of the AM_PM field indicating the period of the day from midnight to just before noon.

PM

Declaration:
```
public static final int PM
```

Description:
Value of the AM_PM field indicating the period of the day from noon to just before midnight.

fields

Declaration:
```
protected int[] fields
```

Description:
The field values for the currently set time for this calendar.

isSet

Declaration:
```
protected boolean[] isSet
```

Description:
The flags which tell if a specified time field for the calendar is set. This is an array of `FIELD_COUNT` booleans,

time

Declaration:
```
protected long time
```

Description:
The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT.

Constructors

Calendar()

Declaration:
```
protected Calendar()
```

Description:
Constructs a Calendar with the default time zone.

See Also: `TimeZone.getDefault()`

Methods

getTime()

Declaration:
```
public final java.util.Date getTime()
```

Description:
Gets this Calendar's current time.

Returns: the current time.

See Also: `setTime(Date)`

setTime(Date)

Declaration:
```
public final void setTime(java.util.Date date)
```

Description:
Sets this Calendar's current time with the given Date.

Note: Calling `setTime()` with `Date(Long.MAX_VALUE)` or `Date(Long.MIN_VALUE)` may yield incorrect field values from `get()`.

Parameters:
`date` - the given Date.

See Also: `getTime()`

getInstance()

Declaration:
```
public static java.util.Calendar getInstance()
```

Description:
Gets a calendar using the default time zone.

Returns: a Calendar.

getInstance(TimeZone)

Declaration:
```
public static java.util.Calendar getInstance(java.util.TimeZone zone)
```

Description:
Gets a calendar using the specified time zone.

Parameters:
`zone` - the time zone to use

Returns: a Calendar.

getTimeInMillis()

Declaration:
```
protected long getTimeInMillis()
```

Description:
Gets this Calendar's current time as a long expressed in milliseconds after January 1, 1970, 0:00:00 GMT (the epoch).

Returns: the current time as UTC milliseconds from the epoch.

See Also: `setTimeInMillis(long)`

setTimeInMillis(long)

Declaration:
```
protected void setTimeInMillis(long millis)
```

Description:
Sets this Calendar's current time from the given long value.

Parameters:
`millis` - the new time in UTC milliseconds from the epoch.

See Also: `getTimeInMillis()`

get(int)

Declaration:
```
public final int get(int field)
```

Description:
Gets the value for a given time field.

Parameters:
`field` - the given time field (either YEAR, MONTH, DATE, DAY_OF_WEEK, HOUR_OF_DAY, HOUR, AM_PM, MINUTE, SECOND, or MILLISECOND

Returns: the value for the given time field.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if the parameter is not one of the above.

set(int, int)

Declaration:
```
public final void set(int field, int value)
```

Description:
Sets the time field with the given value.

Parameters:
`field` - the given time field.

`value` - the value to be set for the given time field.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if an illegal field parameter is received.

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares this calendar to the specified object. The result is `true` if and only if the argument is not `null` and is a `Calendar` object that represents the same calendar as this object.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

before(Object)

Declaration:
```
public boolean before(java.lang.Object when)
```

Description:
Compares the time field records. Equivalent to comparing result of conversion to UTC.

Parameters:
`when` - the Calendar to be compared with this Calendar.

Returns: true if the current time of this Calendar is before the time of Calendar when; false otherwise.

after(Object)

Declaration:
```
public boolean after(java.lang.Object when)
```

Description:
Compares the time field records. Equivalent to comparing result of conversion to UTC.

Parameters:
`when` - the Calendar to be compared with this Calendar.

Returns: true if the current time of this Calendar is after the time of Calendar when; false otherwise.

setTimeZone(TimeZone)

Declaration:
```
public void setTimeZone(java.util.TimeZone value)
```

Description:
Sets the time zone with the given time zone value.

Parameters:
`value` - the given time zone.

See Also: `getTimeZone()`

getTimeZone()

Declaration:
```
public java.util.TimeZone getTimeZone()
```

Description:
Gets the time zone.

Returns: the time zone object associated with this calendar.

See Also: `setTimeZone(TimeZone)`

computeFields()

Declaration:
```
protected abstract void computeFields()
```

Description:
Converts the current millisecond time value `time` to field values in `fields[]`. This allows you to sync up the time field values with a new time that is set for the calendar.

computeTime()

Declaration:
```
protected abstract void computeTime()
```

Description:
Converts the current field values in `fields[]` to the millisecond time value `time`.

java.util
Date

Declaration

```
public class Date

java.lang.Object
  |
  +--java.util.Date
```

Description

The class Date represents a specific instant in time, with millisecond precision.

This class has been subset for the J2ME based on the JDK 1.3 Date class. Many methods and variables have been pruned, and other methods simplified, in an effort to reduce the size of this class.

Although the Date class is intended to reflect coordinated universal time (UTC), it may not do so exactly, depending on the host environment of the Java Virtual Machine. Nearly all modern operating systems assume that 1 day = 24x60x60 = 86400 seconds in all cases. In UTC, however, about once every year or two there is an extra second, called a "leap second." The leap second is always added as the last second of the day, and always on December 31 or June 30. For example, the last minute of the year 1995 was 61 seconds long, thanks to an added leap second. Most computer clocks are not accurate enough to be able to reflect the leap-second distinction.

See Also: `TimeZone`, `Calendar`

Member Summary

Constructors

	`Date()`
	`Date(long date)`

Methods

`boolean`	`equals(java.lang.Object obj)`
`long`	`getTime()`
`int`	`hashCode()`
`void`	`setTime(long time)`
`java.lang.String`	`toString()`

Inherited Member Summary

Methods inherited from class `Object`

`getClass()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

Date()

Declaration:
```
public Date()
```

Description:
Allocates a `Date` object and initializes it to represent the current time specified number of milliseconds since the standard base time known as “the epoch”, namely January 1, 1970, 00:00:00 GMT.

See Also: `java.lang.System.currentTimeMillis()`

Date(long)

Declaration:
```
public Date(long date)
```

Description:
Allocates a `Date` object and initializes it to represent the specified number of milliseconds since the standard base time known as “the epoch”, namely January 1, 1970, 00:00:00 GMT.

Parameters:
`date` - the milliseconds since January 1, 1970, 00:00:00 GMT.

See Also: `java.lang.System.currentTimeMillis()`

Methods

getTime()

Declaration:
```
public long getTime()
```

Description:
Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this `Date` object.

Returns: the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this date.

See Also: `setTime(long)`

setTime(long)

Declaration:
```
public void setTime(long time)
```

Description:
Sets this `Date` object to represent a point in time that is `time` milliseconds after January 1, 1970 00:00:00 GMT.

Parameters:
`time` - the number of milliseconds.

See Also: `getTime()`

equals(Object)

Declaration:
```
public boolean equals(java.lang.Object obj)
```

Description:
Compares two dates for equality. The result is `true` if and only if the argument is not `null` and is a `Date` object that represents the same point in time, to the millisecond, as this object.

Thus, two `Date` objects are equal if and only if the `getTime` method returns the same `long` value for both.

Overrides: `equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns: `true` if the objects are the same; `false` otherwise.

See Also: `getTime()`

hashCode()

Declaration:
```
public int hashCode()
```

Description:
Returns a hash code value for this object. The result is the exclusive OR of the two halves of the primitive `long` value returned by the `getTime()` method. That is, the hash code is the value of the expression:

```
(int)(this.getTime()^(this.getTime() >>> 32))
```

Overrides: `hashCode` in class `Object`

Returns: a hash code value for this object.

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Converts this `Date` object to a `String` of the form:

```
dow mon dd hh:mm:ss zzz yyyy
```
where:

- `dow` is the day of the week (`Sun, Mon, Tue, Wed, Thu, Fri, Sat`).
- `mon` is the month (`Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec`).
- `dd` is the day of the month (`01` through `31`), as two decimal digits.
- `hh` is the hour of the day (`00` through `23`), as two decimal digits.
- `mm` is the minute within the hour (`00` through `59`), as two decimal digits.
- `ss` is the second within the minute (`00` through `61`, as two decimal digits.
- `zzz` is the time zone (and may reflect daylight savings time). If time zone information is not available, then `zzz` is empty - that is, it consists of no characters at all.
- `yyyy` is the year, as four decimal digits.

toString()

Overrides: `toString` in class `Object`

Returns: a string representation of this date.

Since: CLDC 1.1

java.util
EmptyStackException

Declaration
```
public class EmptyStackException extends java.lang.RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.util.EmptyStackException
```

Description
Thrown by methods in the `Stack` class to indicate that the stack is empty.

Since: JDK1.0, CLDC 1.0

See Also: `Stack`

Member Summary
Constructors
`EmptyStackException()`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

EmptyStackException()

Declaration:
```
public EmptyStackException()
```

EmptyStackException()

Description:
Constructs a new `EmptyStackException` with `null` as its error message string.

java.util
Enumeration

Declaration
```
public interface Enumeration
```

Description
An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the `nextElement` method return successive elements of the series.

For example, to print all elements of a vector *v*:

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
    System.out.println(e.nextElement());

}
```

Methods are provided to enumerate through the elements of a vector, the keys of a hashtable, and the values in a hashtable.

Since: JDK1.0, CLDC 1.0

See Also: `nextElement()`, `Hashtable`, `Hashtable.elements()`, `Hashtable.keys()`, `Vector`, `Vector.elements()`

Member Summary	
Methods	
`boolean`	`hasMoreElements()`
`java.lang.Object`	`nextElement()`

Methods

hasMoreElements()

Declaration:
```
public boolean hasMoreElements()
```

Description:
Tests if this enumeration contains more elements.

Returns: `true` if and only if this enumeration object contains at least one more element to provide; `false` otherwise.

nextElement()

Declaration:
```
public java.lang.Object nextElement()
```

nextElement()

Description:
Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Returns: the next element of this enumeration.

Throws:
`NoSuchElementException` - if no more elements exist.

java.util
Hashtable

Declaration

```
public class Hashtable

java.lang.Object
  |
  +--java.util.Hashtable
```

Description

This class implements a hashtable, which maps keys to values. Any non-`null` object can be used as a key or as a value.

To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the `hashCode` method and the `equals` method.

An instance of `Hashtable` has two parameters that affect its efficiency: its *capacity* and its *load factor*. The load factor in the CLDC implementation of the hashtable class is always 75 percent. When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling the `rehash` method.

If many entries are to be made into a `Hashtable`, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

This example creates a hashtable of numbers. It uses the names of the numbers as keys:

```
Hashtable numbers = new Hashtable();
numbers.put(“one”, new Integer(1));
numbers.put(“two”, new Integer(2));
numbers.put(“three”, new Integer(3));
```

To retrieve a number, use the following code:

```
Integer n = (Integer)numbers.get(“two”);
if (n != null) {
    System.out.println(“two = ” + n);
}
```

Since: JDK1.0, CLDC 1.0

See Also: `java.lang.Object.equals(Object)`, `java.lang.Object.hashCode()`, `rehash()`

Member Summary	
Constructors	
	`Hashtable()`
	`Hashtable(int initialCapacity)`
Methods	
`void`	`clear()`
`boolean`	`contains(java.lang.Object value)`

Member Summary	
boolean	containsKey(java.lang.Object key)
Enumeration	elements()
java.lang.Object	get(java.lang.Object key)
boolean	isEmpty()
Enumeration	keys()
java.lang.Object	put(java.lang.Object key, java.lang.Object value)
protected void	rehash()
java.lang.Object	remove(java.lang.Object key)
int	size()
java.lang.String	toString()

Inherited Member Summary
Methods inherited from class Object
equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Constructors

Hashtable(int)

Declaration:
```
public Hashtable(int initialCapacity)
```

Description:
Constructs a new, empty hashtable with the specified initial capacity.

Parameters:
`initialCapacity` - the initial capacity of the hashtable.

Throws:
`java.lang.IllegalArgumentException` - if the initial capacity is less than zero

Since: JDK1.0

Hashtable()

Declaration:
```
public Hashtable()
```

Description:
Constructs a new, empty hashtable with a default capacity and load factor.

Since: JDK1.0

Methods

size()

Declaration:
```
public int size()
```

Description:
Returns the number of keys in this hashtable.

Returns: the number of keys in this hashtable.

Since: JDK1.0

isEmpty()

Declaration:
```
public boolean isEmpty()
```

Description:
Tests if this hashtable maps no keys to values.

Returns: `true` if this hashtable maps no keys to values; `false` otherwise.

Since: JDK1.0

keys()

Declaration:
```
public java.util.Enumeration keys()
```

Description:
Returns an enumeration of the keys in this hashtable.

Returns: an enumeration of the keys in this hashtable.

Since: JDK1.0

See Also: `Enumeration`, `elements()`

elements()

Declaration:
```
public java.util.Enumeration elements()
```

Description:
Returns an enumeration of the values in this hashtable. Use the Enumeration methods on the returned object to fetch the elements sequentially.

Returns: an enumeration of the values in this hashtable.

Since: JDK1.0

See Also: `Enumeration`, `keys()`

contains(Object)

Declaration:
```
public boolean contains(java.lang.Object value)
```

Description:
Tests if some key maps into the specified value in this hashtable. This operation is more expensive than the `containsKey` method.

Parameters:
`value` - a value to search for.

Returns: `true` if some key maps to the `value` argument in this hashtable; `false` otherwise.

Throws:
`java.lang.NullPointerException` - if the value is `null`.

Since: JDK1.0

See Also: `containsKey(Object)`

containsKey(Object)

Declaration:
```
public boolean containsKey(java.lang.Object key)
```

Description:
Tests if the specified object is a key in this hashtable.

Parameters:
`key` - possible key.

Returns: `true` if the specified object is a key in this hashtable; `false` otherwise.

Since: JDK1.0

See Also: `contains(Object)`

get(Object)

Declaration:
```
public java.lang.Object get(java.lang.Object key)
```

Description:
Returns the value to which the specified key is mapped in this hashtable.

Parameters:
`key` - a key in the hashtable.

Returns: the value to which the key is mapped in this hashtable; `null` if the key is not mapped to any value in this hashtable.

Since: JDK1.0

See Also: `put(Object, Object)`

rehash()

Declaration:
```
protected void rehash()
```

Description:
Rehashes the contents of the hashtable into a hashtable with a larger capacity. This method is called automatically when the number of keys in the hashtable exceeds this hashtable's capacity and load factor.

Since: JDK1.0

put(Object, Object)

Declaration:
```
public java.lang.Object put(java.lang.Object key, java.lang.Object value)
```

Description:
Maps the specified `key` to the specified `value` in this hashtable. Neither the key nor the value can be `null`.

The value can be retrieved by calling the `get` method with a key that is equal to the original key.

Parameters:
`key` - the hashtable key.

`value` - the value.

Returns: the previous value of the specified key in this hashtable, or `null` if it did not have one.

Throws:
`java.lang.NullPointerException` - if the key or value is `null`.

Since: JDK1.0

See Also: `java.lang.Object.equals(Object)`, `get(Object)`

remove(Object)

Declaration:
```
public java.lang.Object remove(java.lang.Object key)
```

Description:
Removes the key (and its corresponding value) from this hashtable. This method does nothing if the key is not in the hashtable.

Parameters:
`key` - the key that needs to be removed.

Returns: the value to which the key had been mapped in this hashtable, or `null` if the key did not have a mapping.

Since: JDK1.0

clear()

Declaration:
```
public void clear()
```

Description:
Clears this hashtable so that it contains no keys.

Since: JDK1.0

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a rather long string representation of this hashtable.

Overrides: `toString` in class `Object`

Returns: a string representation of this hashtable.

toString()

Since: JDK1.0

java.util
NoSuchElementException

Declaration
```
public class NoSuchElementException extends java.lang.RuntimeException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--java.util.NoSuchElementException
```

Description
Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.

Since: JDK1.0, CLDC 1.0

See Also: `Enumeration`, `Enumeration.nextElement()`

Member Summary	
Constructors	
	`NoSuchElementException()`
	`NoSuchElementException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

NoSuchElementException()

Declaration:
```
public NoSuchElementException()
```

NoSuchElementException(String)

Description:
Constructs a `NoSuchElementException` with `null` as its error message string.

NoSuchElementException(String)

Declaration:
```
public NoSuchElementException(java.lang.String s)
```

Description:
Constructs a `NoSuchElementException`, saving a reference to the error message string `s` for later retrieval by the `getMessage` method.

Parameters:
`s` - the detail message.

java.util
Random

Declaration

```
public class Random
```

```
java.lang.Object
  |
  +--java.util.Random
```

Description

An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.)

If two instances of `Random` are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers. In order to guarantee this property, particular algorithms are specified for the class `Random`. Java implementations must use all the algorithms shown here for the class `Random`, for the sake of absolute portability of Java code. However, subclasses of class `Random` are permitted to use other algorithms, so long as they adhere to the general contracts for all the methods.

The algorithms implemented by class `Random` use a `protected` utility method that on each invocation can supply up to 32 pseudorandomly generated bits.

Since: JDK1.0, CLDC 1.0

Member Summary

Constructors

	`Random()`
	`Random(long seed)`

Methods

`protected int`	`next(int bits)`
`double`	`nextDouble()`
`float`	`nextFloat()`
`int`	`nextInt()`
`int`	`nextInt(int n)`
`long`	`nextLong()`
`void`	`setSeed(long seed)`

Inherited Member Summary

Methods inherited from class `Object`

Inherited Member Summary
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

Random()

Declaration:
`public Random()`

Description:
Creates a new random number generator. Its seed is initialized to a value based on the current time:

```
public Random() { this(System.currentTimeMillis()); }
```

See Also: `java.lang.System.currentTimeMillis()`

Random(long)

Declaration:
`public Random(long seed)`

Description:
Creates a new random number generator using a single `long` seed:

```
public Random(long seed) { setSeed(seed); }
```

Used by method `next` to hold the state of the pseudorandom number generator.

Parameters:
`seed` - the initial seed.

See Also: `setSeed(long)`

Methods

setSeed(long)

Declaration:
`public void setSeed(long seed)`

Description:
Sets the seed of this random number generator using a single `long` seed. The general contract of `setSeed` is that it alters the state of this random number generator object so as to be in exactly the same state as if it had just been created with the argument `seed` as a seed. The method `setSeed` is implemented by class Random as follows:

```
synchronized public void setSeed(long seed) {
      this.seed = (seed ^ 0x5DEECE66DL) & ((1L << 48) - 1);
}
```

The implementation of `setSeed` by class `Random` happens to use only 48 bits of the given seed. In general, however, an overriding method may use all 64 bits of the long argument as a seed value.

Parameters:
`seed` - the initial seed.

next(int)

Declaration:
```
protected int next(int bits)
```

Description:
Generates the next pseudorandom number. Subclass should override this, as this is used by all other methods.

The general contract of `next` is that it returns an `int` value and if the argument bits is between `1` and `32` (inclusive), then that many low-order bits of the returned value will be (approximately) independently chosen bit values, each of which is (approximately) equally likely to be `0` or `1`. The method `next` is implemented by class `Random` as follows:

```
synchronized protected int next(int bits) {
      seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
      return (int)(seed >>> (48 - bits));
}
```

This is a linear congruential pseudorandom number generator, as defined by D. H. Lehmer and described by Donald E. Knuth in *The Art of Computer Programming*, Volume 2: *Seminumerical Algorithms*, section 3.2.1.

Parameters:
`bits` - random bits

Returns: the next pseudorandom value from this random number generator's sequence.

nextInt()

Declaration:
```
public int nextInt()
```

Description:
Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence. The general contract of `nextInt` is that one `int` value is pseudorandomly generated and returned. All 232 possible `int` values are produced with (approximately) equal probability. The method `nextInt` is implemented by class `Random` as follows:

```
public int nextInt() {  return next(32); }
```

Returns: the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.

nextInt(int)

Declaration:
```
public int nextInt(int n)
```

Description:
Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. The general contract of `nextInt` is that one `int` value in the specified range is pseudorandomly generated and returned. All `n` possible `int` values are produced with (approximately) equal probability. The method `nextInt(int n)` is implemented by class `Random` as follows:

nextLong()

```
public int nextInt(int n) {
    if (n<=0)
        throw new IllegalArgumentException("n must be positive");
    if ((n & -n) == n)  // i.e., n is a power of 2
        return (int)((n * (long)next(31)) >> 31);
    int bits, val;
    do {
        bits = next(31);
        val = bits % n;
    } while(bits - val + (n-1) < 0);
    return val;
}
```

The hedge "approximately" is used in the foregoing description only because the next method is only approximately an unbiased source of independently chosen bits. If it were a perfect source of randomly chosen bits, then the algorithm shown would choose `int` values from the stated range with perfect uniformity.

The algorithm is slightly tricky. It rejects values that would result in an uneven distribution (due to the fact that 2^31 is not divisible by n). The probability of a value being rejected depends on n. The worst case is n=2^30+1, for which the probability of a reject is 1/2, and the expected number of iterations before the loop terminates is 2.

The algorithm treats the case where n is a power of two specially: it returns the correct number of high-order bits from the underlying pseudo-random number generator. In the absence of special treatment, the correct number of *low-order* bits would be returned. Linear congruential pseudo-random number generators such as the one implemented by this class are known to have short periods in the sequence of values of their low-order bits. Thus, this special case greatly increases the length of the sequence of values returned by successive calls to this method if n is a small power of two.

Parameters:

`n` - the bound on the random number to be returned. Must be positive.

Returns: a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and n (exclusive).

Throws:

`java.lang.IllegalArgumentException` - n is not positive.

Since: CLDC 1.1

nextLong()

Declaration:

```
public long nextLong()
```

Description:

Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence. The general contract of `nextLong` is that one long value is pseudorandomly generated and returned. All 264 possible `long` values are produced with (approximately) equal probability. The method `nextLong` is implemented by class `Random` as follows:

```
public long nextLong() {
      return ((long)next(32) << 32) + next(32);
}
```

Returns: the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.

nextFloat()

Declaration:
```
public float nextFloat()
```

Description:
Returns the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence.

The general contract of `nextFloat` is that one `float` value, chosen (approximately) uniformly from the range `0.0f` (inclusive) to `1.0f` (exclusive), is pseudorandomly generated and returned. All 224 possible `float` values of the form *m x* 2-24, where *m* is a positive integer less than 224 , are produced with (approximately) equal probability. The method `nextFloat` is implemented by class `Random` as follows:

```
public float nextFloat() {
     return next(24) / ((float)(1 << 24));
}
```

The hedge "approximately" is used in the foregoing description only because the next method is only approximately an unbiased source of independently chosen bits. If it were a perfect source or randomly chosen bits, then the algorithm shown would choose `float` values from the stated range with perfect uniformity.

[In early versions of Java, the result was incorrectly calculated as:

```
return next(30) / ((float)(1 << 30));
```

This might seem to be equivalent, if not better, but in fact it introduced a slight nonuniformity because of the bias in the rounding of floating-point numbers: it was slightly more likely that the low-order bit of the significand would be 0 than that it would be 1.]

Returns: the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence.

Since: CLDC 1.1

nextDouble()

Declaration:
```
public double nextDouble()
```

Description:
Returns the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence.

The general contract of `nextDouble` is that one `double` value, chosen (approximately) uniformly from the range `0.0d` (inclusive) to `1.0d` (exclusive), is pseudorandomly generated and returned. All 253 possible `float` values of the form *m x* 2-53 , where *m* is a positive integer less than 253, are produced with (approximately) equal probability. The method `nextDouble` is implemented by class `Random` as follows:

```
public double nextDouble() {
      return (((long)next(26) << 27) + next(27))
          / (double)(1L << 53);
}
```

The hedge "approximately" is used in the foregoing description only because the `next` method is only approximately an unbiased source of independently chosen bits. If it were a perfect source or randomly chosen bits, then the algorithm shown would choose `double` values from the stated range with perfect uniformity.

[In early versions of Java, the result was incorrectly calculated as:

nextDouble()

```
return (((long)next(27) << 27) + next(27))
    / (double)(1L << 54);
```

This might seem to be equivalent, if not better, but in fact it introduced a large nonuniformity because of the bias in the rounding of floating-point numbers: it was three times as likely that the low-order bit of the significand would be 0 than that it would be 1! This nonuniformity probably doesn't matter much in practice, but we strive for perfection.]

Returns: the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence.

Since: CLDC 1.1

java.util
Stack

Declaration

```
public class Stack extends Vector
```

```
java.lang.Object
  |
  +--java.util.Vector
        |
        +--java.util.Stack
```

Description

The `Stack` class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

Since: JDK1.0, CLDC 1.0

Member Summary

Constructors

	`Stack()`

Methods

`boolean`	`empty()`
`java.lang.Object`	`peek()`
`java.lang.Object`	`pop()`
`java.lang.Object`	`push(java.lang.Object item)`
`int`	`search(java.lang.Object o)`

Inherited Member Summary

Fields inherited from class `Vector`

`capacityIncrement`, `elementCount`, `elementData`

Methods inherited from class `Object`

`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `Vector`

Stack()

Inherited Member Summary
addElement(Object), capacity(), contains(Object), copyInto(Object[]), elementAt(int), elements(), ensureCapacity(int), firstElement(), indexOf(Object, int), indexOf(Object, int), insertElementAt(Object, int), isEmpty(), lastElement(), lastIndexOf(Object, int), lastIndexOf(Object, int), removeAllElements(), removeElement(Object), removeElementAt(int), setElementAt(Object, int), setSize(int), size(), toString(), trimToSize()

Constructors

Stack()

Declaration:
`public Stack()`

Description:
Creates an empty Stack.

Methods

push(Object)

Declaration:
`public java.lang.Object push(java.lang.Object item)`

Description:
Pushes an item onto the top of this stack. This has exactly the same effect as:

```
addElement(item)
```

Parameters:
`item` - the item to be pushed onto this stack.

Returns: the `item` argument.

See Also: `Vector.addElement(Object)`

pop()

Declaration:
`public java.lang.Object pop()`

Description:
Removes the object at the top of this stack and returns that object as the value of this function.

Returns: The object at the top of this stack (the last item of the `Vector` object).

Throws:
`EmptyStackException` - if this stack is empty.

peek()

Declaration:
`public java.lang.Object peek()`

Description:
Looks at the object at the top of this stack without removing it from the stack.

Returns: the object at the top of this stack (the last item of the `Vector` object).

Throws:
`EmptyStackException` - if this stack is empty.

empty()

Declaration:
```
public boolean empty()
```

Description:
Tests if this stack is empty.

Returns: `true` if and only if this stack contains no items; `false` otherwise.

search(Object)

Declaration:
```
public int search(java.lang.Object o)
```

Description:
Returns the 1-based position where an object is on this stack. If the object `o` occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance `1`. The `equals` method is used to compare `o` to the items in this stack.

Parameters:
`o` - the desired object.

Returns: the 1-based position from the top of the stack where the object is located; the return value `-1` indicates that the object is not on the stack.

java.util
TimeZone

Declaration

```
public abstract class TimeZone
```

```
java.lang.Object
  |
  +--java.util.TimeZone
```

Description

`TimeZone` represents a time zone offset, and also figures out daylight savings.

Typically, you get a `TimeZone` using `getDefault` which creates a `TimeZone` based on the time zone where the program is running. For example, for a program running in Japan, `getDefault` creates a `TimeZone` object based on Japanese Standard Time.

You can also get a `TimeZone` using `getTimeZone` along with a time zone ID. For instance, the time zone ID for the Pacific Standard Time zone is “PST”. So, you can get a PST `TimeZone` object with:

```
TimeZone tz = TimeZone.getTimeZone(“PST”);
```

This class is a pure subset of the java.util.TimeZone class in JDK 1.3.

The only time zone ID that is required to be supported is “GMT”.

Apart from the methods and variables being subset, the semantics of the getTimeZone() method may also be subset: custom IDs such as “GMT-8:00” are not required to be supported.

See Also: `Calendar`, `Date`

Member Summary

Constructors

	`TimeZone()`

Methods

static java.lang.String	`getAvailableIDs()`
static TimeZone	`getDefault()`
java.lang.String	`getID()`
abstract int	`getOffset(int era, int year, int month, int day, int dayOfWeek, int millis)`
abstract int	`getRawOffset()`
static TimeZone	`getTimeZone(java.lang.String ID)`
abstract boolean	`useDaylightTime()`

Inherited Member Summary
Methods inherited from class `Object` `equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Constructors

TimeZone()

Declaration:
```
public TimeZone()
```

Methods

getOffset(int, int, int, int, int, int)

Declaration:
```
public abstract int getOffset(int era, int year, int month, int day, int dayOfWeek,
            int millis)
```

Description:
Gets offset, for current date, modified in case of daylight savings. This is the offset to add *to* GMT to get local time. Gets the time zone offset, for current date, modified in case of daylight savings. This is the offset to add *to* GMT to get local time. Assume that the start and end month are distinct. This method may return incorrect results for rules that start at the end of February (e.g., last Sunday in February) or the beginning of March (e.g., March 1).

Parameters:

`era` - The era of the given date (0 = BC, 1 = AD).

`year` - The year in the given date.

`month` - The month in the given date. Month is 0-based. e.g., 0 for January.

`day` - The day-in-month of the given date.

`dayOfWeek` - The day-of-week of the given date.

`millis` - The milliseconds in day in *standard* local time.

Returns: The offset to add *to* GMT to get local time.

Throws:

`java.lang.IllegalArgumentException` - the era, month, day, dayOfWeek, or millis parameters are out of range

getRawOffset()

Declaration:
```
public abstract int getRawOffset()
```

Description:
Gets the GMT offset for this time zone.

Returns: the GMT offset for this time zone.

useDaylightTime()

Declaration:
```
public abstract boolean useDaylightTime()
```

Description:
Queries if this time zone uses Daylight Savings Time.

Returns: if this time zone uses Daylight Savings Time.

getID()

Declaration:
```
public java.lang.String getID()
```

Description:
Gets the ID of this time zone.

Returns: the ID of this time zone.

getTimeZone(String)

Declaration:
```
public static java.util.TimeZone getTimeZone(java.lang.String ID)
```

Description:
Gets the `TimeZone` for the given ID.

Parameters:
`ID` - the ID for a `TimeZone`, either an abbreviation such as “GMT”, or a full name such as “America/Los_Angeles”.

The only time zone ID that is required to be supported is “GMT”.

Returns: the specified TimeZone, or the GMT zone if the given ID cannot be understood.

getDefault()

Declaration:
```
public static java.util.TimeZone getDefault()
```

Description:
Gets the default `TimeZone` for this host. The source of the default `TimeZone` may vary with implementation.

Returns: a default `TimeZone`.

getAvailableIDs()

Declaration:
```
public static java.lang.String[] getAvailableIDs()
```

Description:
Gets all the available IDs supported.

Returns: an array of IDs.

java.util
Vector

Declaration

```
public class Vector
```

```
java.lang.Object
  |
  +--java.util.Vector
```

Direct Known Subclasses: Stack

Description

The `Vector` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Vector` can grow or shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Each vector tries to optimize storage management by maintaining a `capacity` and a `capacityIncrement`. The `capacity` is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of `capacityIncrement`. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

Since: JDK1.0, CLDC 1.0

Member Summary

Fields

protected int	capacityIncrement
protected int	elementCount
protected java.lang.Object	elementData

Constructors

Vector()
Vector(int initialCapacity)
Vector(int initialCapacity, int capacityIncrement)

Methods

void	addElement(java.lang.Object obj)
int	capacity()
boolean	contains(java.lang.Object elem)
void	copyInto(java.lang.Object anArray)
java.lang.Object	elementAt(int index)
Enumeration	elements()
void	ensureCapacity(int minCapacity)
java.lang.Object	firstElement()
int	indexOf(java.lang.Object elem)
int	indexOf(java.lang.Object elem, int index)
void	insertElementAt(java.lang.Object obj, int index)

Member Summary	
boolean	isEmpty()
java.lang.Object	lastElement()
int	lastIndexOf(java.lang.Object elem)
int	lastIndexOf(java.lang.Object elem, int index)
void	removeAllElements()
boolean	removeElement(java.lang.Object obj)
void	removeElementAt(int index)
void	setElementAt(java.lang.Object obj, int index)
void	setSize(int newSize)
int	size()
java.lang.String	toString()
void	trimToSize()

Inherited Member Summary
Methods inherited from class `Object`
equals(Object), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()

Fields

elementData

Declaration:
`protected java.lang.Object[] elementData`

Description:
The array buffer into which the components of the vector are stored. The capacity of the vector is the length of this array buffer.

Since: JDK1.0

elementCount

Declaration:
`protected int elementCount`

Description:
The number of valid components in the vector.

Since: JDK1.0

capacityIncrement

Declaration:
`protected int capacityIncrement`

Description:
The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity. If the capacity increment is `0`, the capacity of the vector is doubled each time it needs to grow.

Since: JDK1.0

Constructors

Vector(int, int)

Declaration:
```
public Vector(int initialCapacity, int capacityIncrement)
```

Description:
Constructs an empty vector with the specified initial capacity and capacity increment.

Parameters:
`initialCapacity` - the initial capacity of the vector.

`capacityIncrement` - the amount by which the capacity is increased when the vector overflows.

Throws:
`java.lang.IllegalArgumentException` - if the specified initial capacity is negative

Vector(int)

Declaration:
```
public Vector(int initialCapacity)
```

Description:
Constructs an empty vector with the specified initial capacity.

Parameters:
`initialCapacity` - the initial capacity of the vector.

Since: JDK1.0

Vector()

Declaration:
```
public Vector()
```

Description:
Constructs an empty vector.

Since: JDK1.0

Methods

copyInto(Object[])

Declaration:
```
public void copyInto(java.lang.Object[] anArray)
```

Description:
Copies the components of this vector into the specified array. The array must be big enough to hold all the objects in this vector.

Parameters:
`anArray` - the array into which the components get copied.

Since: JDK1.0

trimToSize()

Declaration:
```
public void trimToSize()
```

Description:
Trims the capacity of this vector to be the vector's current size. An application can use this operation to minimize the storage of a vector.

Since: JDK1.0

ensureCapacity(int)

Declaration:
```
public void ensureCapacity(int minCapacity)
```

Description:
Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

Parameters:
`minCapacity` - the desired minimum capacity.

Since: JDK1.0

setSize(int)

Declaration:
```
public void setSize(int newSize)
```

Description:
Sets the size of this vector. If the new size is greater than the current size, new `null` items are added to the end of the vector. If the new size is less than the current size, all components at index `newSize` and greater are discarded.

Parameters:
`newSize` - the new size of this vector.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if new size is negative.

Since: JDK1.0

capacity()

Declaration:
```
public int capacity()
```

Description:
Returns the current capacity of this vector.

Returns: the current capacity of this vector.

Since: JDK1.0

size()

Declaration:
```
public int size()
```

Description:
Returns the number of components in this vector.

Returns: the number of components in this vector.

Since: JDK1.0

isEmpty()

Declaration:
```
public boolean isEmpty()
```

Description:
Tests if this vector has no components.

Returns: `true` if this vector has no components; `false` otherwise.

Since: JDK1.0

elements()

Declaration:
```
public java.util.Enumeration elements()
```

Description:
Returns an enumeration of the components of this vector.

Returns: an enumeration of the components of this vector.

Since: JDK1.0

See Also: `Enumeration`

contains(Object)

Declaration:
```
public boolean contains(java.lang.Object elem)
```

Description:
Tests if the specified object is a component in this vector.

Parameters:
`elem` - an object.

Returns: `true` if the specified object is a component in this vector; `false` otherwise.

Since: JDK1.0

indexOf(Object)

Declaration:
```
public int indexOf(java.lang.Object elem)
```

Description:
Searches for the first occurence of the given argument, testing for equality using the `equals` method.

Parameters:
`elem` - an object.

Returns: the index of the first occurrence of the argument in this vector; returns `-1` if the object is not found.

Since: JDK1.0

See Also: `java.lang.Object.equals(Object)`

indexOf(Object, int)

Declaration:
```
public int indexOf(java.lang.Object elem, int index)
```

Description:
Searches for the first occurence of the given argument, beginning the search at `index`, and testing for equality using the `equals` method.

Parameters:
`elem` - an object.

`index` - the index to start searching from.

Returns: the index of the first occurrence of the object argument in this vector at position `index` or later in the vector; returns `-1` if the object is not found.

Since: JDK1.0

See Also: `java.lang.Object.equals(Object)`

lastIndexOf(Object)

Declaration:
```
public int lastIndexOf(java.lang.Object elem)
```

Description:
Returns the index of the last occurrence of the specified object in this vector.

Parameters:
`elem` - the desired component.

Returns: the index of the last occurrence of the specified object in this vector; returns `-1` if the object is not found.

Since: JDK1.0

lastIndexOf(Object, int)

Declaration:
```
public int lastIndexOf(java.lang.Object elem, int index)
```

Description:
Searches backwards for the specified object, starting from the specified index, and returns an index to it.

Parameters:
`elem` - the desired component.

`index` - the index to start searching from.

Returns: the index of the last occurrence of the specified object in this vector at position less than `index` in the vector; `-1` if the object is not found.

Throws:
`java.lang.IndexOutOfBoundsException` - if `index` is greater than or equal to the current size of this vector.

Since: JDK1.0

elementAt(int)

Declaration:
```
public java.lang.Object elementAt(int index)
```

Description:
Returns the component at the specified index.

Parameters:
`index` - an index into this vector.

Returns: the component at the specified index.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if an invalid index was given.

Since: JDK1.0

firstElement()

Declaration:
```
public java.lang.Object firstElement()
```

Description:
Returns the first component of this vector.

Returns: the first component of this vector.

Throws:
`NoSuchElementException` - if this vector has no components.

Since: JDK1.0

lastElement()

Declaration:
```
public java.lang.Object lastElement()
```

Description:
Returns the last component of the vector.

Returns: the last component of the vector, i.e., the component at index `size() - 1`.

Throws:
`NoSuchElementException` - if this vector is empty.

Since: JDK1.0

setElementAt(Object, int)

Declaration:
```
public void setElementAt(java.lang.Object obj, int index)
```

Description:
Sets the component at the specified `index` of this vector to be the specified object. The previous component at that position is discarded.

The index must be a value greater than or equal to `0` and less than the current size of the vector.

Parameters:
`obj` - what the component is to be set to.

`index` - the specified index.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if the index was invalid.

Since: JDK1.0

See Also: `size()`

removeElementAt(int)

Declaration:
```
public void removeElementAt(int index)
```

Description:
Deletes the component at the specified index. Each component in this vector with an index greater or equal to the specified `index` is shifted downward to have an index one smaller than the value it had previously.

The index must be a value greater than or equal to `0` and less than the current size of the vector.

Parameters:
`index` - the index of the object to remove.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if the index was invalid.

Since: JDK1.0

See Also: `size()`

insertElementAt(Object, int)

Declaration:
```
public void insertElementAt(java.lang.Object obj, int index)
```

Description:
Inserts the specified object as a component in this vector at the specified `index`. Each component in this vector with an index greater or equal to the specified `index` is shifted upward to have an index one greater than the value it had previously.

The index must be a value greater than or equal to `0` and less than or equal to the current size of the vector.

Parameters:
`obj` - the component to insert.

`index` - where to insert the new component.

Throws:
`java.lang.ArrayIndexOutOfBoundsException` - if the index was invalid.

Since: JDK1.0

See Also: `size()`

addElement(Object)

Declaration:
```
public void addElement(java.lang.Object obj)
```

Description:
Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

Parameters:
`obj` - the component to be added.

Since: JDK1.0

removeElement(Object)

Declaration:
```
public boolean removeElement(java.lang.Object obj)
```

Description:
Removes the first occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.

Parameters:
`obj` - the component to be removed.

Returns: `true` if the argument was a component of this vector; `false` otherwise.

Since: JDK1.0

removeAllElements()

Declaration:
```
public void removeAllElements()
```

Description:
Removes all components from this vector and sets its size to zero.

Since: JDK1.0

toString()

Declaration:
```
public java.lang.String toString()
```

Description:
Returns a string representation of this vector.

Overrides: `toString` in class `Object`

Returns: a string representation of this vector.

Since: JDK1.0

toString()

CHAPTER 4

Package javax.microedition.io

Description

Classes for the Generic Connection framework.

Since: CLDC 1.0

Class Summary	
Interfaces	
Connection	This is the most basic type of generic connection.
ContentConnection	This interface defines the stream connection over which content is passed.
Datagram	This class defines an abstract interface for datagram packets.
DatagramConnection	This interface defines the capabilities that a datagram connection must have.
InputConnection	This interface defines the capabilities that an input stream connection must have.
OutputConnection	This interface defines the capabilities that an output stream connection must have.
StreamConnection	This interface defines the capabilities that a stream connection must have.
StreamConnectionNotifier	This interface defines the capabilities that a connection notifier must have.
Classes	
Connector	This class is factory for creating new Connection objects.
Exceptions	
ConnectionNotFoundException	This class is used to signal that a connection target cannot be found, or the protocol type is not supported.

javax.microedition.io
Connection

Declaration
```
public interface Connection
```

All Known Subinterfaces: `ContentConnection`, `DatagramConnection`, `InputConnection`, `OutputConnection`, `StreamConnection`, `StreamConnectionNotifier`

Description
This is the most basic type of generic connection. Only the `close` method is defined. No open method is defined here because opening is always done using the `Connector.open()` methods.

Since: CLDC 1.0

Member Summary	
Methods	
`void`	`close()`

Methods

close()

Declaration:
```
public void close()
            throws IOException
```

Description:
Close the connection.

When a connection has been closed, access to any of its methods that involve an I/O operation will cause an `IOException` to be thrown. Closing an already closed connection has no effect. Streams derived from the connection may be open when method is called. Any open streams will cause the connection to be held open until they themselves are closed. In this latter case access to the open streams is permitted, but access to the connection is not.

Throws:
`java.io.IOException` - If an I/O error occurs

javax.microedition.io
ConnectionNotFoundException

Declaration
```
public class ConnectionNotFoundException extends java.io.IOException
```

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.io.IOException
                    |
                    +--javax.microedition.io.ConnectionNotFoundException
```

Description
This class is used to signal that a connection target cannot be found, or the protocol type is not supported.

Since: CLDC 1.0

Member Summary	
Constructors	
	`ConnectionNotFoundException()`
	`ConnectionNotFoundException(java.lang.String s)`

Inherited Member Summary
Methods inherited from class `Object`
`equals(Object)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`
Methods inherited from class `Throwable`
`getMessage()`, `printStackTrace()`, `toString()`

Constructors

ConnectionNotFoundException()

Declaration:
```
public ConnectionNotFoundException()
```

Description:
Constructs a ConnectionNotFoundException with no detail message.

ConnectionNotFoundException(String)

Declaration:
```
public ConnectionNotFoundException(java.lang.String s)
```

Description:
Constructs a ConnectionNotFoundException with the specified detail message. A detail message is a String that describes this particular exception.

Parameters:
`s` - the detail message

javax.microedition.io
Connector

Declaration

```
public class Connector
```

```
java.lang.Object
  |
  +--javax.microedition.io.Connector
```

Description

This class is factory for creating new Connection objects.

The creation of Connections is performed dynamically by looking up a protocol implementation class whose name is formed from the platform name (read from a system property) and the protocol name of the requested connection (extracted from the parameter string supplied by the application programmer.) The parameter string that describes the target should conform to the URL format as described in RFC 2396. This takes the general form:

`{scheme}:[{target}][{params}]`

where `{scheme}` is the name of a protocol such as *http*}.

The `{target}` is normally some kind of network address.

Any `{params}` are formed as a series of equates of the form ";x=y". Example: ";type=a".

An optional second parameter may be specified to the open function. This is a mode flag that indicates to the protocol handler the intentions of the calling code. The options here specify if the connection is going to be read (READ), written (WRITE), or both (READ_WRITE). The validity of these flag settings is protocol dependent. For instance, a connection for a printer would not allow read access, and would throw an IllegalArgumentException. If the mode parameter is not specified, READ_WRITE is used by default.

An optional third parameter is a boolean flag that indicates if the calling code can handle timeout exceptions. If this flag is set, the protocol implementation may throw an InterruptedIOException when it detects a timeout condition. This flag is only a hint to the protocol handler, and it does not guarantee that such exceptions will actually be thrown. If this parameter is not set, no timeout exceptions will be thrown.

Because connections are frequently opened just to gain access to a specific input or output stream, four convenience functions are provided for this purpose. See also: `DatagramConnection` for information relating to datagram addressing

Since: CLDC 1.0

Member Summary

Fields

```
            static int  READ
            static int  READ_WRITE
            static int  WRITE
```

Methods

```
     static Connection  open(java.lang.String name)
```

READ

Member Summary	
static Connection	open(java.lang.String name, int mode)
static Connection	open(java.lang.String name, int mode, boolean timeouts)
static java.io.DataInputStream	openDataInputStream(java.lang.String name)
static java.io.DataOutputStream	openDataOutputStream(java.lang.String name)
static java.io.InputStream	openInputStream(java.lang.String name)
static java.io.OutputStream	openOutputStream(java.lang.String name)

Inherited Member Summary
Methods inherited from class `Object`
equals(Object), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Fields

READ

Declaration:
`public static final int READ`

Description:
Access mode READ.

WRITE

Declaration:
`public static final int WRITE`

Description:
Access mode WRITE.

READ_WRITE

Declaration:
`public static final int READ_WRITE`

Description:
Access mode READ_WRITE.

Methods

open(String)

Declaration:
```
public static javax.microedition.io.Connection open(java.lang.String name)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:
`name` - The URL for the connection.

Returns: A new Connection object.

Throws:
`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

open(String, int)

Declaration:
```
public static javax.microedition.io.Connection open(java.lang.String name, int mode)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:
`name` - The URL for the connection.

`mode` - The access mode.

Returns: A new Connection object.

Throws:
`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

open(String, int, boolean)

Declaration:
```
public static javax.microedition.io.Connection open(java.lang.String name, int mode,
            boolean timeouts)
            throws IOException
```

Description:
Create and open a Connection.

Parameters:
`name` - The URL for the connection

`mode` - The access mode

`timeouts` - A flag to indicate that the caller wants timeout exceptions

Returns: A new Connection object

Throws:
`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

openDataInputStream(String)

Declaration:
```
public static java.io.DataInputStream openDataInputStream(java.lang.String name)
            throws IOException
```

Description:
Create and open a connection input stream.

Parameters:
`name` - The URL for the connection.

Returns: A DataInputStream.

Throws:
`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

openDataOutputStream(String)

Declaration:
```
public static java.io.DataOutputStream openDataOutputStream(java.lang.String name)
            throws IOException
```

Description:
Create and open a connection output stream.

Parameters:
`name` - The URL for the connection.

Returns: A DataOutputStream.

Throws:

`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

openInputStream(String)

Declaration:

```
public static java.io.InputStream openInputStream(java.lang.String name)
            throws IOException
```

Description:
Create and open a connection input stream.

Parameters:

`name` - The URL for the connection.

Returns: An InputStream.

Throws:

`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

openOutputStream(String)

Declaration:

```
public static java.io.OutputStream openOutputStream(java.lang.String name)
            throws IOException
```

Description:
Create and open a connection output stream.

Parameters:

`name` - The URL for the connection.

Returns: An OutputStream.

Throws:

`java.lang.IllegalArgumentException` - If a parameter is invalid.

`ConnectionNotFoundException` - If the target of the name cannot be found, or if the requested protocol type is not supported.

`java.io.IOException` - If some other kind of I/O error occurs.

`java.lang.SecurityException` - May be thrown if access to the protocol handler is prohibited.

javax.microedition.io
ContentConnection

Declaration
```
public interface ContentConnection extends StreamConnection
```

All Superinterfaces: Connection, InputConnection, OutputConnection, StreamConnection

Description
This interface defines the stream connection over which content is passed.

Since: CLDC 1.0

Member Summary
Methods
java.lang.String getEncoding()
long getLength()
java.lang.String getType()

Inherited Member Summary
Methods inherited from interface Connection
close()
Methods inherited from interface InputConnection
openDataInputStream(), openInputStream()
Methods inherited from interface OutputConnection
openDataOutputStream(), openOutputStream()

Methods

getType()

Declaration:
```
public java.lang.String getType()
```

Description:
Returns the type of content that the resource connected to is providing. For instance, if the connection is via HTTP, then the value of the `content-type` header field is returned.

Returns: the content type of the resource that the URL references, or `null` if not known.

getEncoding()

Declaration:
```
public java.lang.String getEncoding()
```

Description:
Returns a string describing the encoding of the content which the resource connected to is providing. E.g. if the connection is via HTTP, the value of the `content-encoding` header field is returned.

Returns: the content encoding of the resource that the URL references, or `null` if not known.

getLength()

Declaration:
```
public long getLength()
```

Description:
Returns the length of the content which is being provided. E.g. if the connection is via HTTP, then the value of the `content-length` header field is returned.

Returns: the content length of the resource that this connection's URL references, or `-1` if the content length is not known.

javax.microedition.io
Datagram

Declaration
`public interface` **`Datagram extends`** **`java.io.DataInput`**, `java.io.DataOutput`

All Superinterfaces: `java.io.DataInput`, `java.io.DataOutput`

Description
This class defines an abstract interface for datagram packets. The implementations of this interface hold data to be sent or received from a `DatagramConnection` object.

Since this is an interface class, the internal structure of the datagram packets is not defined here. However, it is assumed that each implementation of this interface will provide the following fields / state variables (the actual implementation and the names of these fields may vary):

- *buffer*: the internal buffer in which data is stored
- *offset*: the read/write offset for the internal buffer
- *length*: the length of the data in datagram packet
- *address*: the destination or source address
- *read/write pointer*: a pointer that is added to the *offset* to point to the current data location during a read or write operation

Reading and Writing

The `Datagram` interface extends interfaces `DataInput` and `DataOutput` in order to provide a simple way to read and write binary data in and out of the datagram buffer instead of using `getData` and `setData` methods. Writing automatically increments *length* and reading will continue while the *read/write pointer* is less than *length*. Before any writing is done reset must be called. If `setData()` is to be used when reading or writing, any value for the `offset` parameter other than 0 is not supported.

For example to write to datagram:

```
datagram = connection.newDatagram(max);
// Reset prepares the datagram for writing new message.
datagram.reset();
// writeUTF automatically increases the datagram length.
datagram.writeUTF(“hello world”);
connection.send(datagram);
```

For example to read from a datagram (single use only):

```
datagram = connection.newDatagram(max);
connection.receive(datagram);
message = datagram.readUTF();
```

Reusing Datagrams

It should be noted the *length* above is returned from `getLength` and can have different meanings at different times. When sending *length* is the number of bytes to send. Before receiving *length* is the maximum number of bytes to receive. After receiving *length* is the number of bytes that were received. So when reusing a datagram to receive after sending or receiving, length must be set back to the maximum using `setLength`.

```
datagram = connection.newDatagram(max);
while (notDone) {
    // The last receive in the loop changed the length
    // so put it back to the maximum length.
    datagram.setLength(max);
    connection.receive(datagram);
    data = datagram.getData();
    bytesReceived = datagram.getLength();
    // process datagram ...
}
```

When reading instead of using `getData` the `reset` method must be used.

```
datagram = connection.newDatagram(max);
while (notDone) {
    // The last read in the loop changed the read pointer
    // so reset the pointer.
    datagram.reset();
    datagram.setLength(max);
    connection.receive(datagram);
    message = datagram.readUTF(message);
    // process message ...
}
```

For example to reread a datagram:

```
connection.receive(datagram);
message = datagram.readUTF(message);
len = datagram.getLength();
datagram.reset();
datagram.setLength(len);
copy = datagram.readUTF(message);
```

Since: CLDC 1.0

Member Summary

Methods

java.lang.String	getAddress()
byte[]	getData()
int	getLength()
int	getOffset()
void	reset()
void	setAddress(Datagram reference)
void	setAddress(java.lang.String addr)
void	setData(byte[] buffer, int offset, int len)
void	setLength(int len)

Inherited Member Summary

Methods inherited from interface DataInput

readBoolean(), readByte(), readChar(), readDouble(), readFloat(), readFully(byte[], int, int), readFully(byte[], int, int), readInt(), readLong(), readShort(), readUTF(), readUnsignedByte(), readUnsignedShort(), skipBytes(int)

Inherited Member Summary
Methods inherited from interface `DataOutput` `write(byte[], int, int)`, `write(byte[], int, int)`, `write(byte[], int, int)`, `writeBoolean(boolean)`, `writeByte(int)`, `writeChar(int)`, `writeChars(String)`, `writeDouble(double)`, `writeFloat(float)`, `writeInt(int)`, `writeLong(long)`, `writeShort(int)`, `writeUTF(String)`

Methods

getAddress()

Declaration:
`public java.lang.String getAddress()`

Description:
Get the address of the datagram.

Returns: the address in string form, or null if no address was set

See Also: `setAddress(String)`

getData()

Declaration:
`public byte[] getData()`

Description:
Get the contents of the data buffer.

Depending on the implementation, this operation may return the internal buffer or a copy of it. However, the user must not assume that the contents of the internal data buffer can be manipulated by modifying the data returned by this operation. Rather, the `setData` operation should be used for changing the contents of the internal buffer.

Returns: the data buffer as a byte array

See Also: `setData(byte[], int, int)`

getLength()

Declaration:
`public int getLength()`

Description:
Get the length of the datagram.

Returns: the length state variable

See Also: `setLength(int)`

getOffset()

Declaration:
`public int getOffset()`

Description:
Get the offset.

Returns: the offset state variable

setAddress(String)

Declaration:
```
public void setAddress(java.lang.String addr)
            throws IOException
```

Description:
Set datagram address.

The actual addressing scheme is implementation-dependent. Please read the general comments on datagram addressing in `DatagramConnection.java`.

Note that if the address of a datagram is not specified, then it defaults to that of the connection.

Parameters:
`addr` - the new target address as a URL

Throws:
`java.lang.IllegalArgumentException` - if the address is not valid

`java.io.IOException` - if a some kind of I/O error occurs

See Also: `getAddress()`

setAddress(Datagram)

Declaration:
```
public void setAddress(javax.microedition.io.Datagram reference)
```

Description:
Set datagram address, copying the address from another datagram.

Parameters:
`reference` - to the datagram whose address will be copied as the new target address for this datagram.

Throws:
`java.lang.IllegalArgumentException` - if the address is not valid

See Also: `getAddress()`

setLength(int)

Declaration:
```
public void setLength(int len)
```

Description:
Set the `length` state variable.

Parameters:
`len` - the new length of the datagram

Throws:
`java.lang.IllegalArgumentException` - if the length or length plus offset fall outside the buffer

See Also: `getLength()`

setData(byte[], int, int)

Declaration:
```
public void setData(byte[] buffer, int offset, int len)
```

Description:
Set the `buffer`, `offset` and `length` state variables. Depending on the implementation, this operation may copy the buffer or just set the state variable `buffer` to the value of the `buffer` argument. However, the user must not assume that the contents of the internal data buffer can be manipulated by modifying the buffer passed on to this operation.

Parameters:

`buffer` - the data buffer

`offset` - the offset into the data buffer

`len` - the length of the data in the buffer

Throws:

`java.lang.IllegalArgumentException` - if the length or offset or offset plus length fall outside the buffer, or if the buffer parameter is invalid

See Also: `getData()`

reset()

Declaration:
```
public void reset()
```

Description:
Zero the `read/write pointer` as well as the `offset` and `length` state variables.

javax.microedition.io
DatagramConnection

Declaration
```
public interface DatagramConnection extends Connection
```

All Superinterfaces: `Connection`

Description
This interface defines the capabilities that a datagram connection must have.

Reminder: Since the CLDC Specification does not define any actual network protocol implementations, the syntax for datagram addressing is not defined in the CLDC Specification. Rather, syntax definition takes place at the level of J2ME profiles such as MIDP.

In the sample implementation that is provided as part of the CLDC reference implementation, the following addressing scheme is used:

The parameter string describing the target of a connection in the CLDC reference implementation takes the following form:

```
{protocol}://[{host}]:[{port}]
```

A datagram connection can be opened in a “client” mode or “server” mode. If the “//{host}” part is missing then the connection is opened as a “server” (by “server”, we mean that a client application initiates communication). When the “//{host}” part is specified, the connection is opened as a “client”.

Examples:

A datagram connection for accepting datagrams
datagram://:1234

A datagram connection for sending to a server:
datagram://123.456.789.12:1234

Note that the port number in “server mode” (unspecified host name) is that of the receiving port. The port number in “client mode” (host name specified) is that of the target port. The reply-to port in both cases is never unspecified. In “server mode”, the same port number is used for both receiving and sending. In “client mode”, the reply-to port is always dynamically allocated.

Also note that the allocation of datagram objects is done in a more abstract way than in Java 2 Standard Edition (J2SE). Instead of providing a concrete `DatagramPacket` class, an abstract `Datagram` interface is provided. This is to allow a single platform to support several different datagram interfaces simultaneously. Datagram objects must be allocated by calling the `newDatagram` methods of the `DatagramConnection` object. The resulting object is defined using another interface type called `javax.microedition.io.Datagram`.

Since: CLDC 1.0

Member Summary
Methods

Member Summary	
int	getMaximumLength()
int	getNominalLength()
Datagram	newDatagram(byte[] buf, int size)
Datagram	newDatagram(byte[] buf, int size, java.lang.String addr)
Datagram	newDatagram(int size)
Datagram	newDatagram(int size, java.lang.String addr)
void	receive(Datagram dgram)
void	send(Datagram dgram)

Inherited Member Summary
Methods inherited from interface Connection
close()

Methods

getMaximumLength()

Declaration:

```
public int getMaximumLength()
            throws IOException
```

Description:

Get the maximum length a datagram can be. Maximum length determines the maximum size of the datagram that can be created using the `newDatagram` method, and the maximum size of the datagram that can be sent or received.

Returns: The maximum length of a datagram.

Throws:

`java.io.IOException` - If an I/O error occurs.

getNominalLength()

Declaration:

```
public int getNominalLength()
            throws IOException
```

Description:

Get the nominal length of a datagram. Nominal length refers to the size of the datagram that is stored into the data buffer. Nominal length may be equal or less than the maximum length of the datagram.

Returns: The nominal length of a datagram.

Throws:

`java.io.IOException` - If an I/O error occurs.

send(Datagram)

Declaration:
```
public void send(javax.microedition.io.Datagram dgram)
            throws IOException
```

Description:
Send a datagram. The `Datagram` object includes the information indicating the data to be sent, its length, and the address of the receiver. The method sends `length` bytes starting at the current `offset` of the `Datagram` object, where `length` and `offset` are internal state variables of the `Datagram` object.

Parameters:
`dgram` - A datagram.

Throws:
`java.io.IOException` - If an I/O error occurs.

`java.io.InterruptedIOException` - Timeout or interrupt occurred.

receive(Datagram)

Declaration:
```
public void receive(javax.microedition.io.Datagram dgram)
            throws IOException
```

Description:
Receive a datagram. When this method returns, the internal buffer in the `Datagram` object is filled with the data received, starting at the location determined by the `offset` state variable, and the data is ready to be read using the methods of the `DataInput` interface.

This method blocks until a datagram is received. The internal `length` state variable in the `Datagram` object contains the length of the received datagram. If the received data is longer than the length of the internal buffer minus offset, data is truncated.

This method does not change the internal *read/write state variable of the Datagram object. Use method Datagram.reset to change the pointer before reading if necessary.*

Parameters:
`dgram` - A datagram.

Throws:
`java.io.IOException` - If an I/O error occurs.

`java.io.InterruptedIOException` - Timeout or interrupt occurred.

newDatagram(int)

Declaration:
```
public javax.microedition.io.Datagram newDatagram(int size)
            throws IOException
```

Description:
Create a new datagram object.

Parameters:
`size` - The size of the buffer needed for the datagram

Returns: A new datagram

Throws:
`java.io.IOException` - If an I/O error occurs.

`java.lang.IllegalArgumentException` - if the size is negative or larger than the maximum size

newDatagram(int, String)

Declaration:

```
public javax.microedition.io.Datagram newDatagram(int size, java.lang.String addr)
            throws IOException
```

Description:
Create a new datagram object.

Parameters:

`size` - The size of the buffer needed for the datagram

`addr` - The I/O address to which the datagram will be sent

Returns: A new datagram

Throws:

`java.io.IOException` - If an I/O error occurs.

`java.lang.IllegalArgumentException` - if the size is negative or larger than the maximum size, or if the address parameter is invalid

newDatagram(byte[], int)

Declaration:

```
public javax.microedition.io.Datagram newDatagram(byte[] buf, int size)
            throws IOException
```

Description:
Create a new datagram object.

Parameters:

`buf` - The buffer to be used for the datagram

`size` - The size of the buffer needed for the datagram

Returns: A new datagram

Throws:

`java.io.IOException` - If an I/O error occurs.

`java.lang.IllegalArgumentException` - if the size is negative or larger than the maximum size or the given buffer's length, or if the buffer parameter is invalid

newDatagram(byte[], int, String)

Declaration:

```
public javax.microedition.io.Datagram newDatagram(byte[] buf, int size,
            java.lang.String addr)
            throws IOException
```

Description:
Make a new datagram object.

Parameters:

`buf` - The buffer to be used for the datagram

`size` - The size of the buffer needed for the datagram

`addr` - The I/O address to which the datagram will be sent

Returns: A new datagram

Throws:

`java.io.IOException` - If an I/O error occurs.

`java.lang.IllegalArgumentException` - if the size is negative or larger than the maximum size or the given buffer's length, or if the address or buffer parameter is invalid

javax.microedition.io
InputConnection

Declaration
```
public interface InputConnection extends Connection
```

All Superinterfaces: Connection

All Known Subinterfaces: ContentConnection, StreamConnection

Description
This interface defines the capabilities that an input stream connection must have.

Since: CLDC 1.0

Member Summary	
Methods	
java.io.DataInputStream	openDataInputStream()
java.io.InputStream	openInputStream()

Inherited Member Summary
Methods inherited from interface Connection
close()

Methods

openInputStream()

Declaration:
```
public java.io.InputStream openInputStream()
            throws IOException
```

Description:
Open and return an input stream for a connection.

Returns: An input stream

Throws:
java.io.IOException - If an I/O error occurs

openDataInputStream()

Declaration:

```
public java.io.DataInputStream openDataInputStream()
            throws IOException
```

Description:

Open and return a data input stream for a connection.

Returns: An input stream

Throws:

`java.io.IOException` - If an I/O error occurs

javax.microedition.io
OutputConnection

Declaration
```
public interface OutputConnection extends Connection
```

All Superinterfaces: Connection

All Known Subinterfaces: ContentConnection, StreamConnection

Description
This interface defines the capabilities that an output stream connection must have.

Since: CLDC 1.0

Member Summary	
Methods	
java.io.DataOutputStream	openDataOutputStream()
java.io.OutputStream	openOutputStream()

Inherited Member Summary
Methods inherited from interface Connection
close()

Methods

openOutputStream()

Declaration:
```
public java.io.OutputStream openOutputStream()
            throws IOException
```

Description:
Open and return an output stream for a connection.

Returns: An output stream

Throws:
java.io.IOException - If an I/O error occurs

openDataOutputStream()

Declaration:

```
public java.io.DataOutputStream openDataOutputStream()
            throws IOException
```

Description:

Open and return a data output stream for a connection.

Returns: An output stream

Throws:

`java.io.IOException` - If an I/O error occurs

javax.microedition.io
StreamConnection

Declaration
`public interface` **`StreamConnection extends`** **`InputConnection`**`, OutputConnection`

All Superinterfaces: `Connection`, `InputConnection`, `OutputConnection`

All Known Subinterfaces: `ContentConnection`

Description
This interface defines the capabilities that a stream connection must have.

In a typical implementation of this interface (for instance in MIDP 2.0), all `StreamConnections` have one underlying `InputStream` and one `OutputStream`. Opening a `DataInputStream` counts as opening an `InputStream` and opening a `DataOutputStream` counts as opening an `OutputStream`. Trying to open another `InputStream` or `OutputStream` causes an `IOException`. Trying to open the `InputStream` or `OutputStream` after they have been closed causes an `IOException`.

The methods of `StreamConnection` are not synchronized. The only stream method that can be called safely in another thread is `close`.

Since: CLDC 1.0

Inherited Member Summary
Methods inherited from interface `Connection`
`close()`
Methods inherited from interface `InputConnection`
`openDataInputStream()`, `openInputStream()`
Methods inherited from interface `OutputConnection`
`openDataOutputStream()`, `openOutputStream()`

javax.microedition.io
StreamConnectionNotifier

Declaration
```
public interface StreamConnectionNotifier extends Connection
```

All Superinterfaces: Connection

Description
This interface defines the capabilities that a connection notifier must have.

Since: CLDC 1.0

Member Summary
Methods
StreamConnection acceptAndOpen()

Inherited Member Summary
Methods inherited from interface Connection
close()

Methods

acceptAndOpen()

Declaration:
```
public javax.microedition.io.StreamConnection acceptAndOpen()
            throws IOException
```

Description:
Returns a `StreamConnection` object that represents a server side socket connection. The method blocks until a connection is made.

Returns: A `StreamConnection` to communicate with a client.

Throws:
`java.io.IOException` - If an I/O error occurs.

acceptAndOpen()

CHAPTER 5

Constant Field Values

Contents

- java.lang.*
- java.util.*
- javax.microedition.*

java.lang.*

java.lang.Byte	
static byte MAX_VALUE	127
static byte MIN_VALUE	-128

java.lang.Character	
static int MAX_RADIX	36
static char MAX_VALUE	65535
static int MIN_RADIX	2
static char MIN_VALUE	0

java.lang.Double	
static double MAX_VALUE	1.7976931348623157E308d
static double NaN	0d/0d
static double NEGATIVE_INFINITY	-1d/0d
static double POSITIVE_INFINITY	1d/0d

java.lang.Float	
static float MAX_VALUE	3.4028234663852886E38f
static float MIN_VALUE	1.401298464324817E-45f
static float NaN	0f/0f
static float NEGATIVE_INFINITY	-1f/0f
static float POSITIVE_INFINITY	1f/0f

java.lang.Integer	
static int MAX_VALUE	2147483647
static int MIN_VALUE	-2147483648

java.lang.Long	
static long MAX_VALUE	9223372036854775807l
static long MIN_VALUE	-9223372036854775808l

java.lang.Math	
static double E	2.718281828459045d
static double PI	3.141592653589793d

java.lang.Short	
static short MAX_VALUE	32767
static short MIN_VALUE	-32768

java.lang.Thread	
static int MAX_PRIORITY	10
static int MIN_PRIORITY	1
static int NORM_PRIORITY	5

java.util.*

java.util.Calendar	
static int AM	0
static int AM_PM	9
static int APRIL	3
static int AUGUST	7
static int DATE	5
static int DAY_OF_MONTH	5
static int DAY_OF_WEEK	7
static int DECEMBER	11
static int FEBRUARY	1
static int FRIDAY	6
static int HOUR	10
static int HOUR_OF_DAY	11
static int JANUARY	0
static int JULY	6
static int JUNE	5
static int MARCH	2
static int MAY	4
static int MILLISECOND	14
static int MINUTE	12
static int MONDAY	2
static int MONTH	2
static int NOVEMBER	10
static int OCTOBER	9
static int PM	1
static int SATURDAY	7
static int SECOND	13
static int SEPTEMBER	8
static int SUNDAY	1

java.util.Calendar	
static int THURSDAY	5
static int TUESDAY	3
static int WEDNESDAY	4
static int YEAR	1

javax.microedition.*

javax.microedition.io.Connector	
static int READ	1
static int READ_WRITE	3
static int WRITE	2