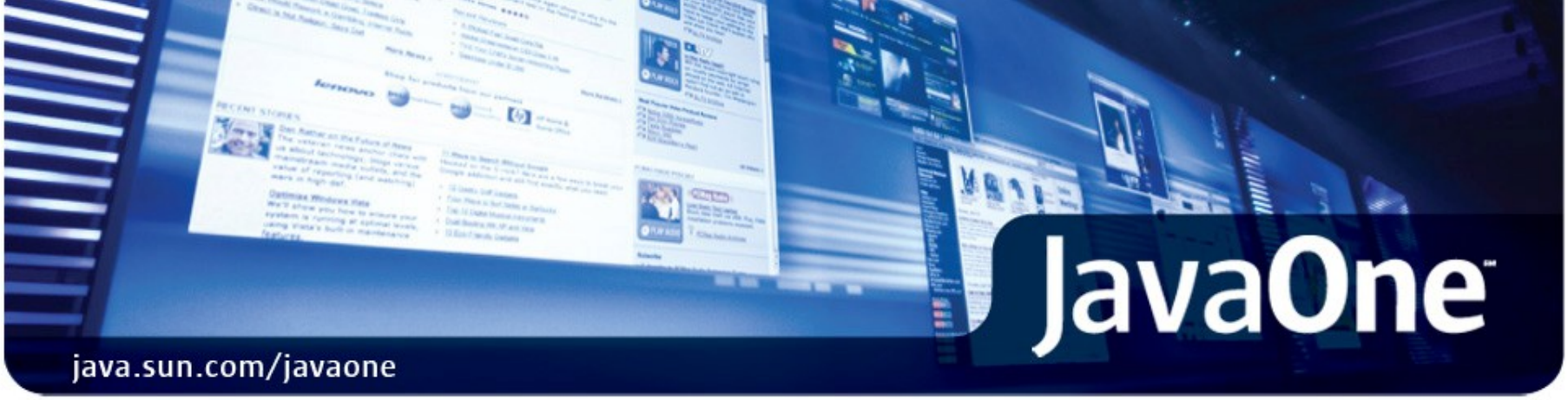



# Use of Java™ Technology-Based Class Loaders to Design and Implement a Java Platform, Micro Edition (JavaME™ Platform) Container

Christian Kurzke, Developer Platforms Architect, Motorola Inc.
Gustavo de Paula, Senior Consultant for Wireless Technologies, C.E.S.A.R

TS-7575, JavaSE™ and Cool Stuff

MOTOROLA C.E.S.A.R

Java

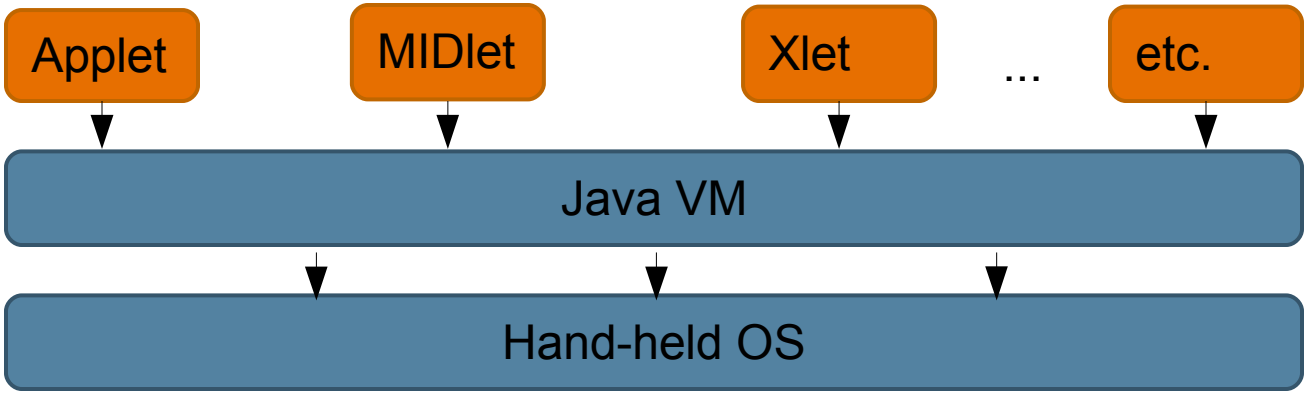
Sun microsystems

Learn the characteristics of an application container and understand how to design a Java™ Platform, Micro Edition (Java ME) container using Java Platform, Standard Edition (Java SE) Class Loaders

GOAL

# Motivation

- Today most of the hand-held devices in the world include a Java virtual machine…
- But the Java platform defines different application models, such as MIDlet, Applet, etc.

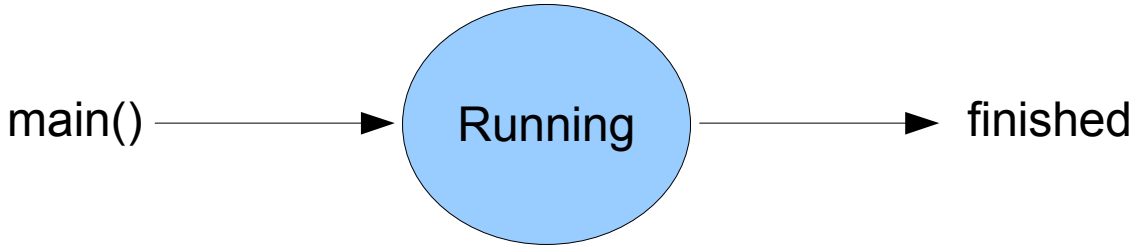



- Challenge: How to **design** a solution that enables **different Application Models** on top of the same Java virtual machine base?

# Agenda

- Application Model
- Application Container
- Java Class Loaders
- Motorola Java ME Platform Emulator Scenario
- Main Issues in Java ME Container Design
- Conclusions

# Application Model




## Usual Java Application Execution model is very simple:

- Run a Java Class
- There is a single "Running" State
  - Entered when `public static void main(String[]args)` is called
  - Exits when the method finishes
- All JRE is available to the Java class

## The Application Model defines

- The main **execution entity**
- The **life cycle** of this execution entity (states, events and transitions)
- The **runtime environment**

# Application Model
## Java ME Platform Architecture

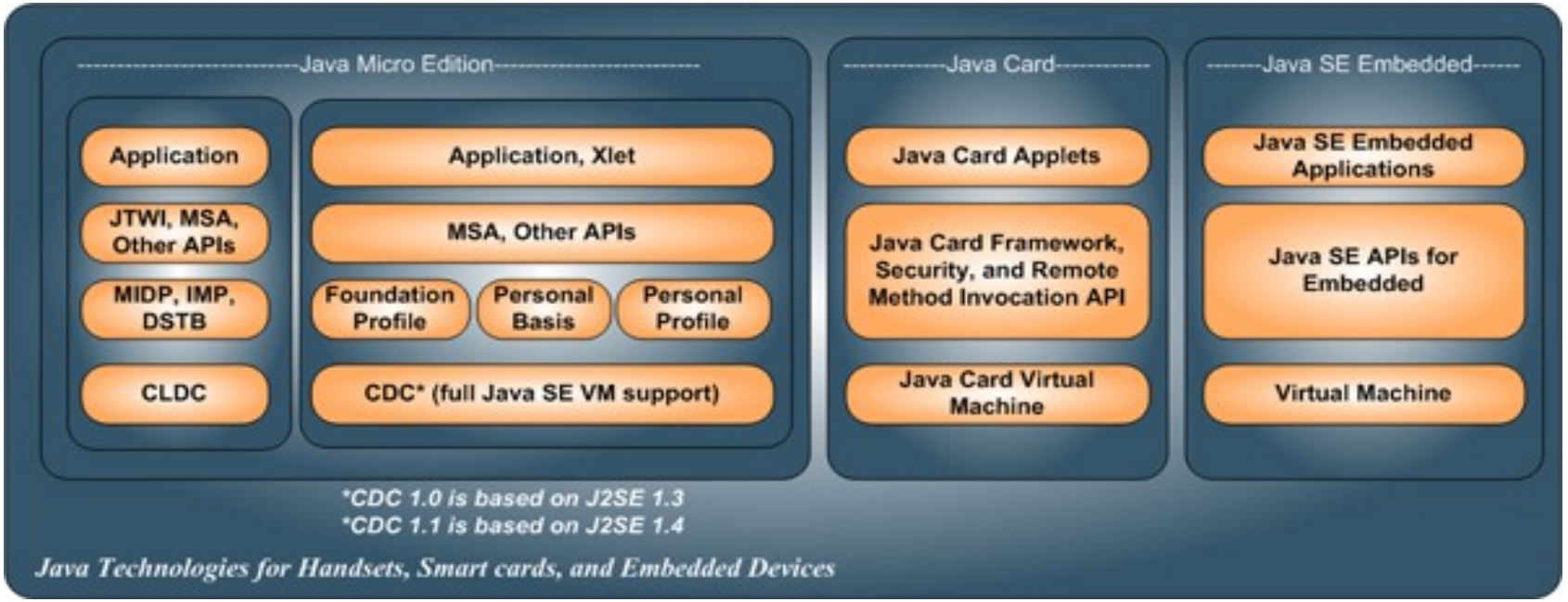
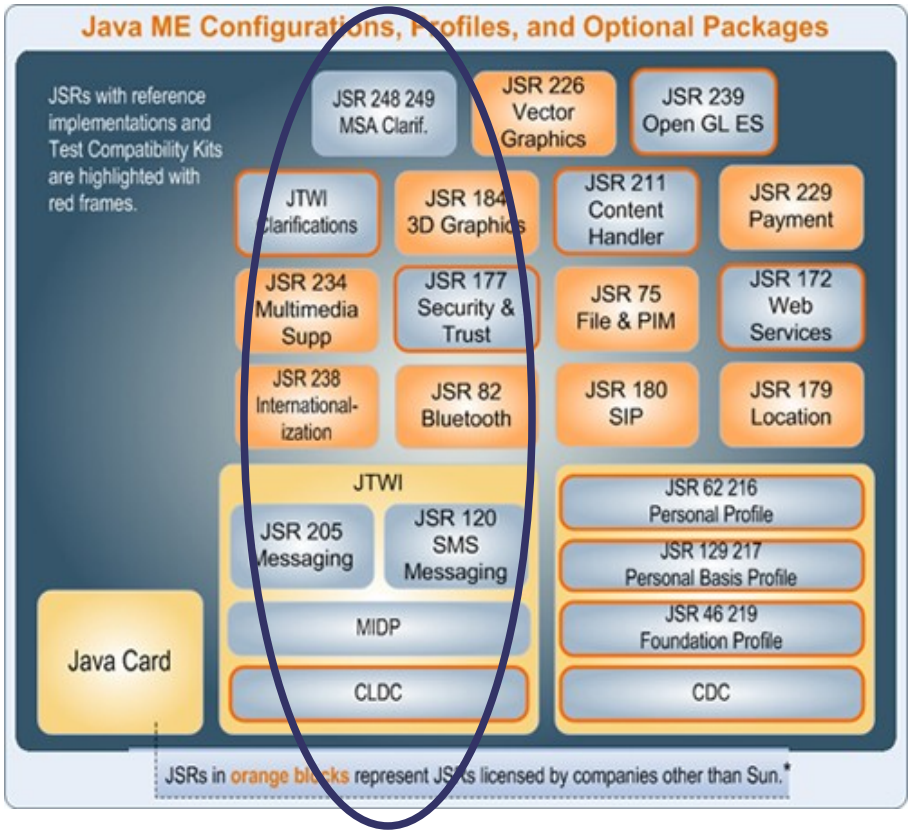



Figure 2 in REF [1]

- Java ME platform defines two **configurations** and a set of different **profiles**
- **Four** different application models are supported

# Application Model
## Mobile Information Device Profile Application Model

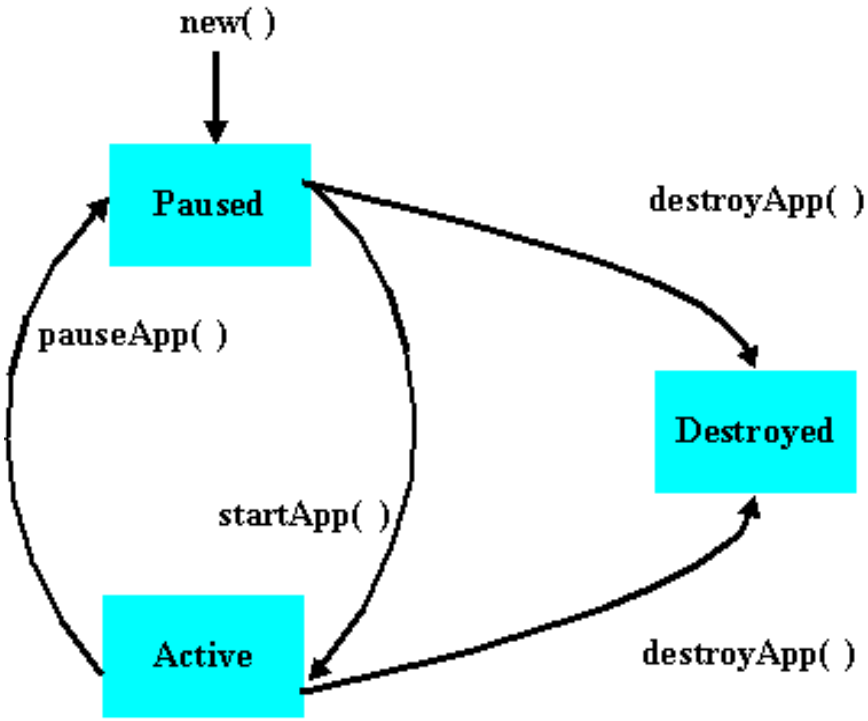
- MIDP is one of the Java ME platform Profiles
  - Defines a **“new” Application model** called MIDlet
- Runs on top of CLDC/KVM
- Defines the concept of **optional package**
  - Different devices might have a **different set of JSRs**
  - But: **All** devices support the same **MIDlet** Application model




See REF [2]

# Application Model
## MIDlet Application Model

- **Execution entity** is a MIDlet
  - extends the class `javax.microedition.midlet.MIDlet`
- **Life cycle** defines 3 states
  - Paused
  - Active
  - Destroyed
- The **runtime** platform is
  - MIDP/CLDC/KVM
  - Optional packages




See REF [3]

# Application Model
## MIDP/CLDC/KVM vs. Java SE Platform

### MIDP/CLDC/KVM

- Limited VM (limited error handling, threadgroups, etc.)
- No JNI support
- No user defined class loaders
- No reflection
- No `Object.finalize()`
- Managed Application model (MIDlet)

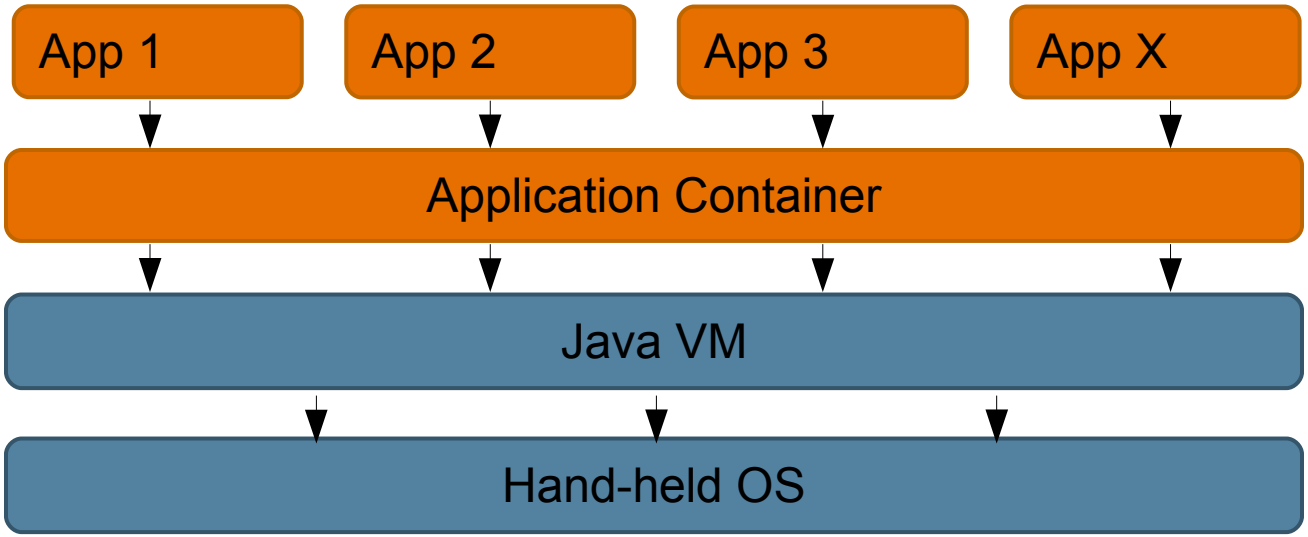
### Java SE platform

- Full Java SE technology-based API support
- Simplified Application model (not managed)
- All features that are not supported on Java ME platform

# Agenda

- Application Model
- Application Container
- Java Class Loaders
- Motorola Java ME Platform Emulator Scenario
- Main Issues in Java ME Container Design
- Conclusions

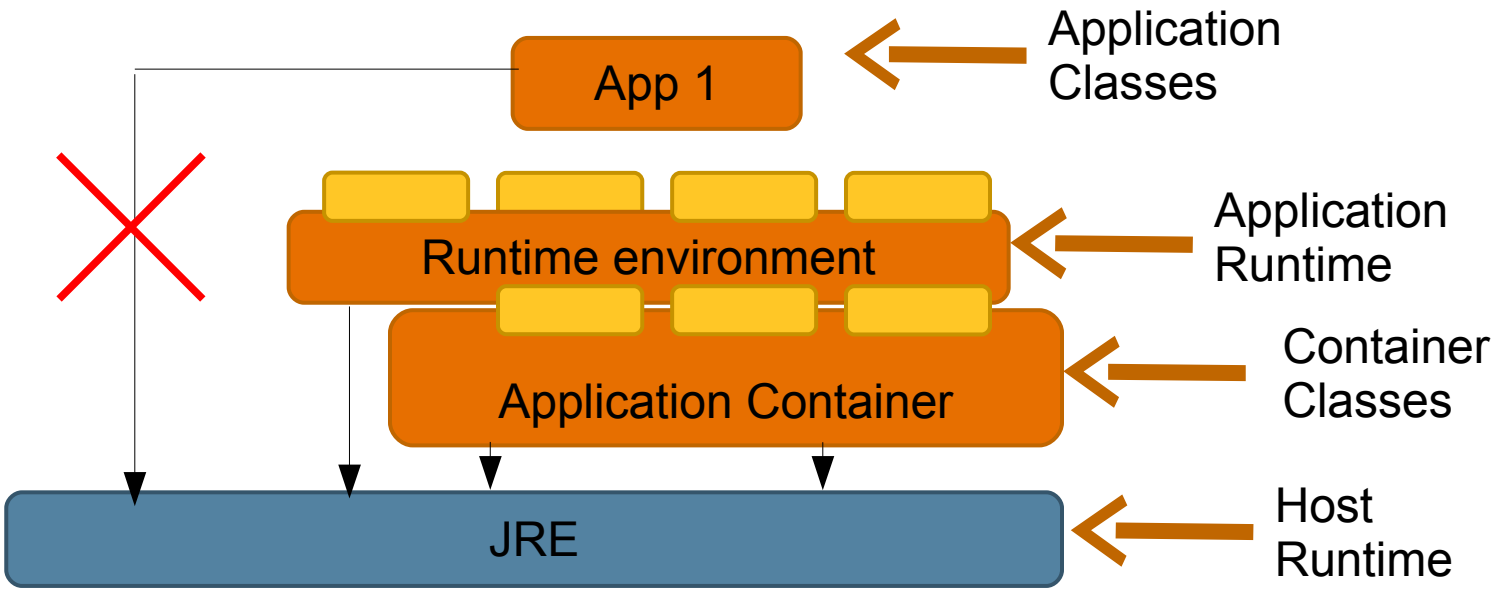
# Application Container




- The **container** is directly associated to a specific **life cycle**
- It is responsible to
  - **Implements** the Application model **specification**
  - Manage the application **life cycle**
  - Interact with "**external components**"
  - Provide the **runtime environment**
- Each application model, Xlet, MIDlet, etc., has a different container

# Application Container

- In order to design and implement a container, it is necessary to have
  - A mechanism to **isolate** the **application runtime** environment from the **host runtime** environment
  - A mechanism to load the **application classes** different from the **container execution classes**



# Application Container

- Java SE platform provides the a mechanism to implement those requirements
  - Java Class Loaders
- Java Class Loaders are part of the core Java SE platform Specification
  - Available since Java 1.0 platform

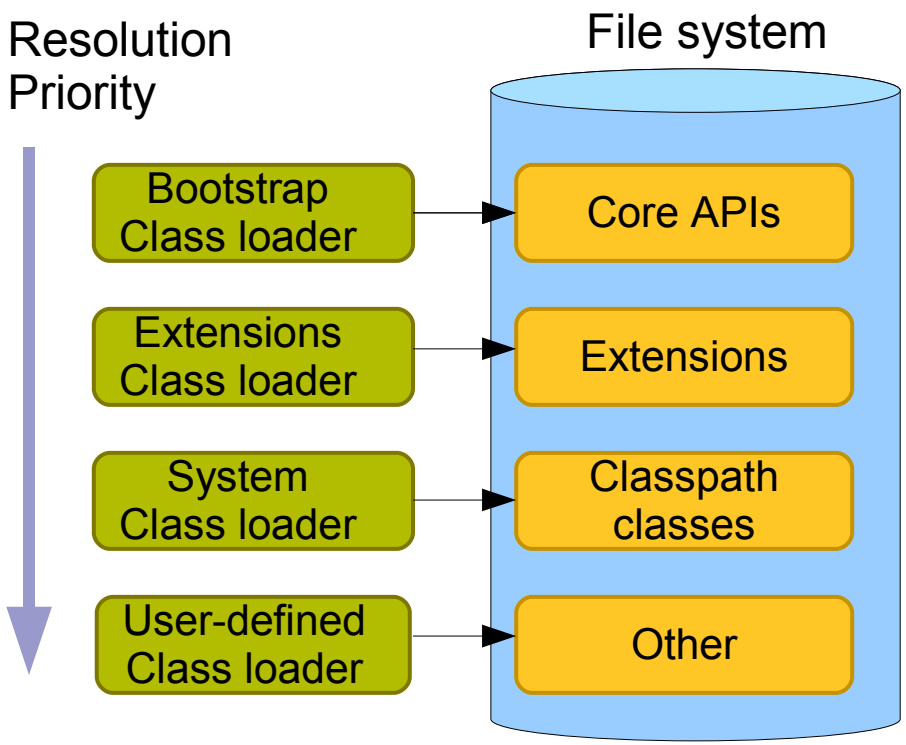
# Agenda

- Application Model
- Application Container
- Java Class Loaders
- Motorola Java ME Platform Emulator Scenario
- Main Issues in Java ME Container Design
- Conclusions

# Java Class Loaders

- A class loader is the VM component that has 3 main services
  - Know where to **find the class files**, optionally packaged as JAR files
  - **Find one specific class** inside the JARs
  - **Load** this class or resource inside the VM execution environment
- Class loader is deeply related to Java virtual machine
  - **Inheritance hierarchies** resolved and class definitions are located by VM
  - Classes are **interpreted** byte code
- Class **loading mechanism** is similar to the operating system **dynamic library loading** mechanism

# Java Class Loaders




## 4 different types of class loaders

- **Boot** class loader is the **initial** class loader
  - Implemented in native code
  - **Loads** all **"Core" APIs**
- User defined class loader extend the system class loader
  - Application classes **customized** loading mechanism
- **Core APIs** and extensions are "protected" inside the VM
  - Those classes cannot be loaded by other class loaders

# Java Class Loaders
## Code Sample - loadClass

```
// verify if it was already loaded
clazz = findLoadedClass(name);
if (clazz != null) {
        return clazz;
}
try {
        clazz = findClass(name);
} catch (ClassNotFoundException e) {
        clazz = findSystemClass(name);
}
if (resolve) {
        resolveClass(clazz);
}
return clazz;
```

Check if the class is already loaded

Find the Class

If can't find, check on system classpath

# Java Class Loaders
## Code Sample - findClass

```
String path = classname.replace('.', '/') + ".class";
byte[] raw = null;
for (int i = 0; i < CLASSPATH.length; i++) {
  JarEntry classFile=
      CLASSPATH[i].getJarEntry(path);
  if (classFile != null){
    try {
      raw = readFromJAR(CLASSPATH[i],
          classFile);
      break;
    } catch (IOException e) {
      throw new ClassNotFoundException("", e);
    }
  }
}
if (raw == null) {
       throw new ClassNotFoundException("" + name);
}
return defineClass(name, raw, 0,
      raw.length);
```

Get one entry on the class path and search file

Read the Class bytes

Define the class

# Java Class Loaders
## Class Loader on Java ME platform (CLDC)

- CLDC **does not** allow **user-defined** class loaders
- When a **MIDlet is running**, the only class loader is the **main KVM loader**
  - Similar to the bootstrap class loader
- KVM Class loader only load
  - Core KVM classes (CLDC, MIDP)
  - Optional packages
  - MIDlet JAR

# Agenda

- Application Model
- Application Container
- Java Class Loaders
- **Motorola Java ME Platform Emulator Scenario**
- Main Issues in Java ME Container Design
- Conclusions

# Motorola Java ME Platform Emulator Scenario

- Motorola supports **different device platforms** according to the target market
  - Low tier: Motorola OS
  - High tier: MOTOMAGX (Linux)
  - Multi-media device: UIQ (Symbian)
  - Productivity Device: Windows Mobile
- **MIDlet** Application Model is supported on **ALL platforms**
- But each platform has a different set of supported JSRs and hardware capabilities

# Motorola Java ME Platform Emulator Scenario

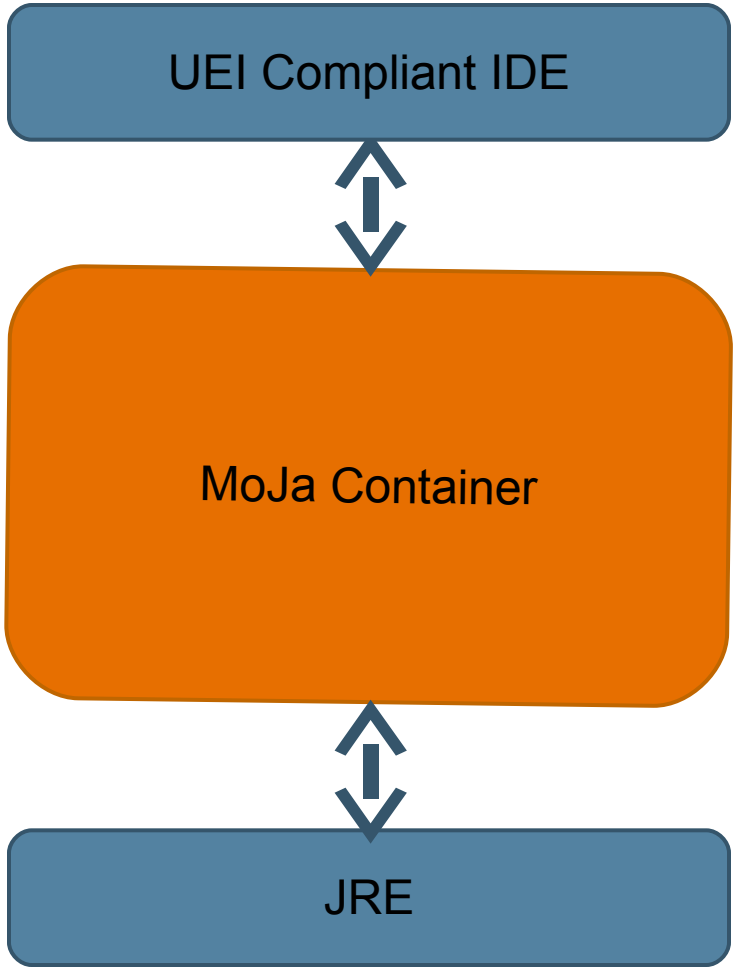
Linux OS
Media Streaming
WMA 2.0

Windows Mobile OS
Bluetooth API
Location API

Motorola OS
Crypto API
Bluetooth API

- A Motorola Java ME platform Emulator needs to **address** those **different kinds of devices**
  - **Same** Application model, but with **different** runtime platform
- An **application model container** is the best solution for that

# Motorola Java ME Platform Emulator
## MoJa Container Requirements

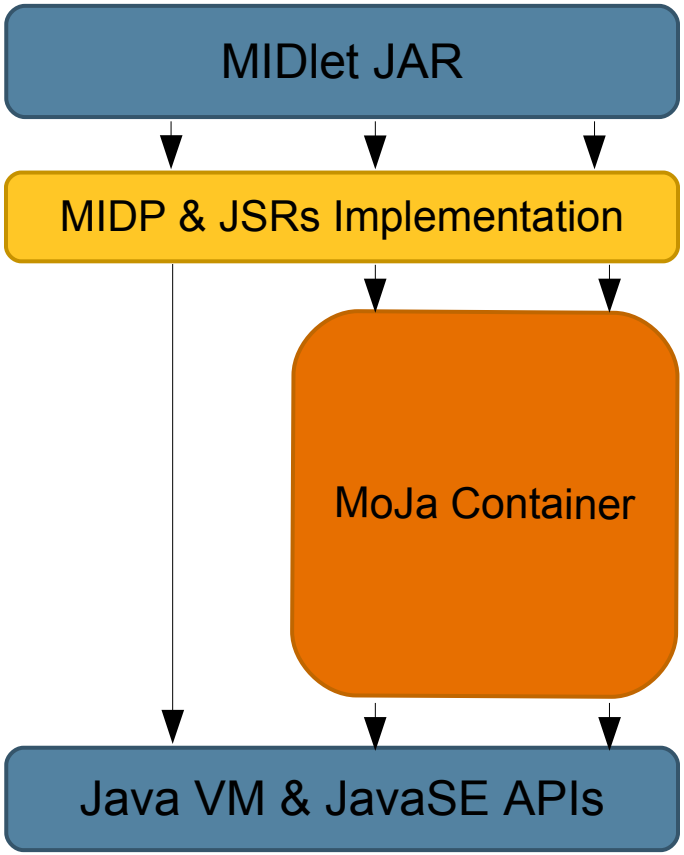



- **Motorola Java ME Container**
- **Single Application Model**, MIDlet, but different platforms and devices
- **Host runtime platform** is the **Java SE** platform
- Allow that **each device** can define its own **set of supported APIs**
- Integrated on any **UEI Compliant IDE**
- **Loads** the MIDlet and the JSRs classes

# Agenda

- Application Model
- Application Container
- Java Class Loaders
- Motorola Java ME Platform Emulator Scenario
- Main Issues in Java ME Container Design
- Conclusions

# Main Issues in Java ME Platform Container Design
## High Level View




- MoJa Container can access **ALL Java SE APIs**
- MIDP and JSRs implementation can
  - Access **each other**
  - Access the **container internal classes**
  - Access all **Java SE APIs**
- MIDlet JAR can
  - Access its **own classes**
  - Access the **MIDP & JSRs Interfaces**

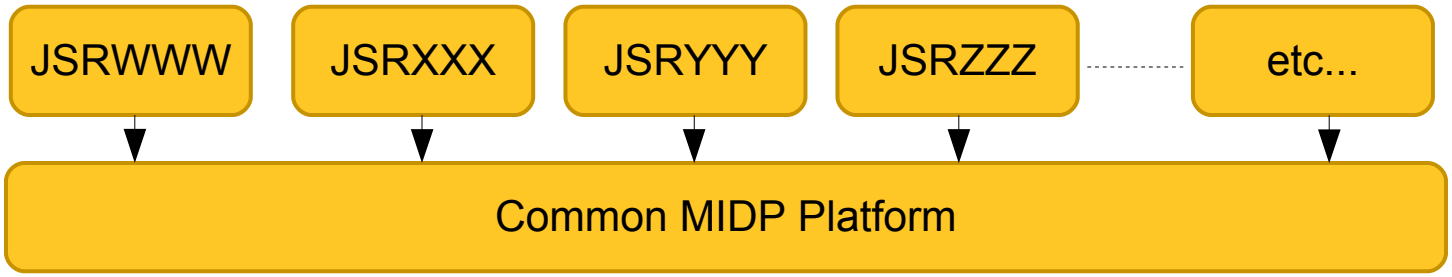
# Main Issues in Java ME Platform Container Design
## Four Issues

- How to represent each specific JSR?
- How to represent each Device?
- How to handle the MIDlet suite under execution?
- How to separate the Java ME platform from the Java SE platform?

# Issue 1: How to represent each specific JSR?

- CLDC/MIDP Architecture Define the Concept of **Optional Packages (JSR)**
  - Depends on CLDC and/or MIDP and there is **no dependency** between each other
- Solution
  - Define a **common platform** (supported by ALL devices)
  - Design each JSR as a **separated JAR** that depends only on the common platform

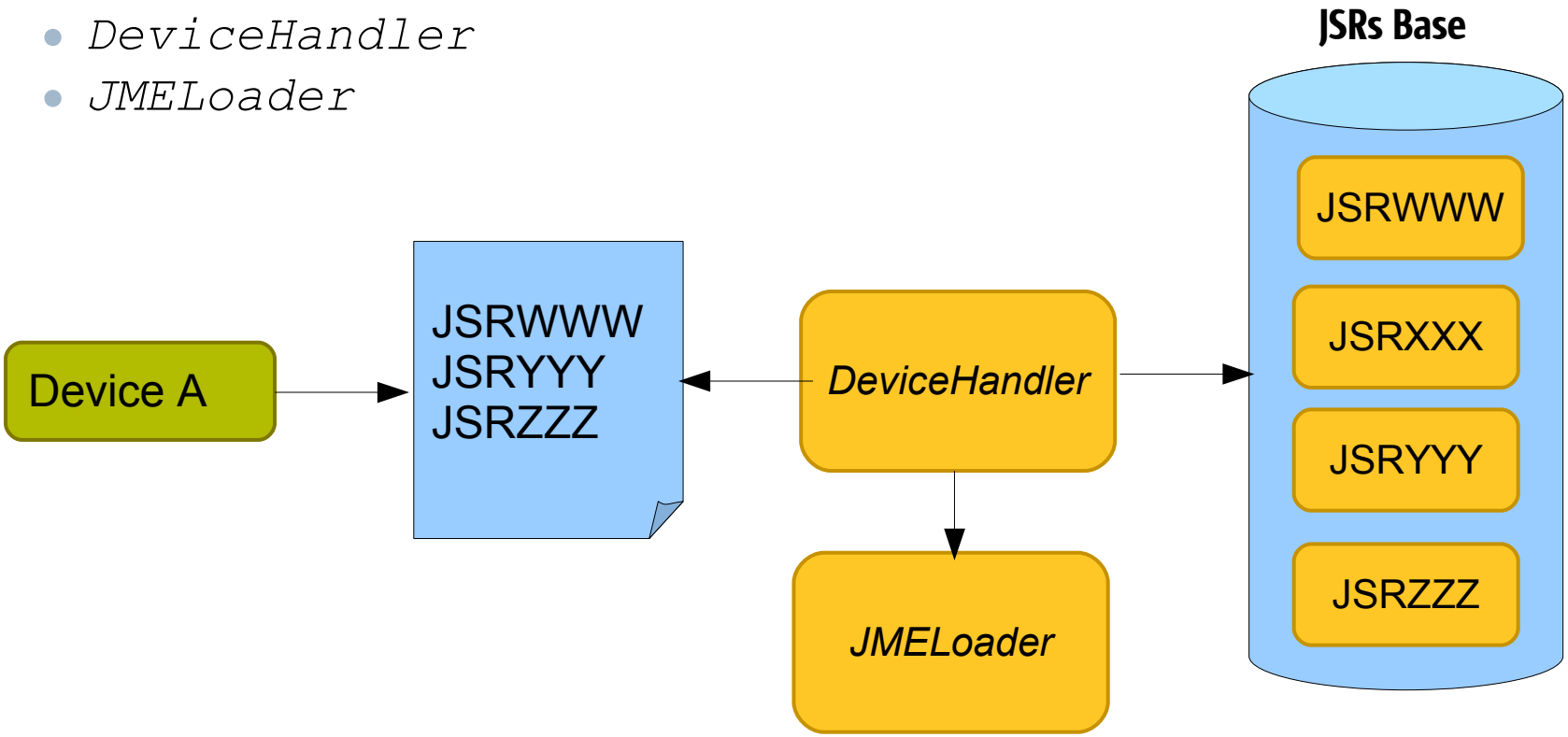
# How to represent each specific JSR?
## JSRs Logical View




- MoJa Common platform is CLDC, MIDP, JSR 75 (File System) and JSR 135
  - Common platform can be organized in a **single JAR file**
- Each other JSR can have its **own JAR**
- **MoJa Container** will have a **base of JSRs** that it supports

# Issue 2: How to represent each Device?

- Each device must **indicate** the **set of JSRs** that it supports from the **JSRs Base**
- MoJa Container has two main components that handles the device representation
  - *DeviceHandler*
  - *JMELoader*

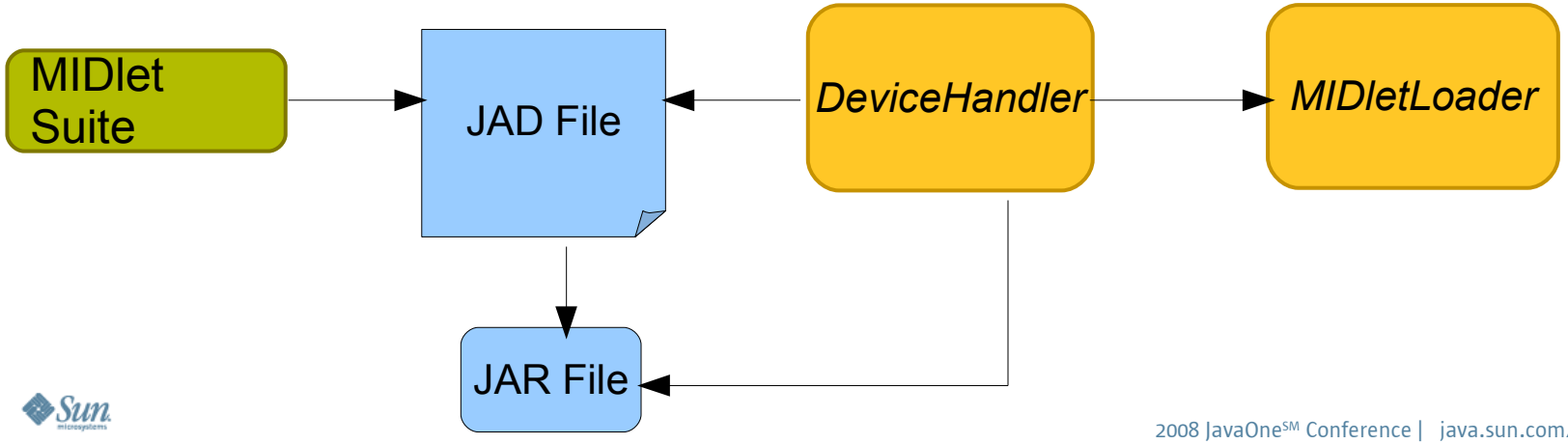


# How to represent each Device?

- Each device is represented by a **property/XML** file
  - List all the JSRs supported by that device
- *`DeviceHandler`* **is responsible to**
  - **Parser** the device property file that list all supported JSRs
  - **Go** to the JSRs base
  - **Locate** JSR JARs
  - **Start** the JMELoader pointing to ALL Device JSRs
- *`JMELoader`* **is responsible to**
  - **Find** classes inside the JSRs JARs
  - **Load** classes inside the JSRs JARs

# Issue 3: How to handle the MIDlet suite under execution?

- A MIDlet suite is represented by its
  - Java Application Descriptor (JAD File)
  - JAR File
- The **Suite JAR** has all **classes** and **resources** that are necessary to **execute** the MIDlet
- MoJa Container has two main components that handles the device representation
  - *DeviceHandler*
  - *MIDletLoader*



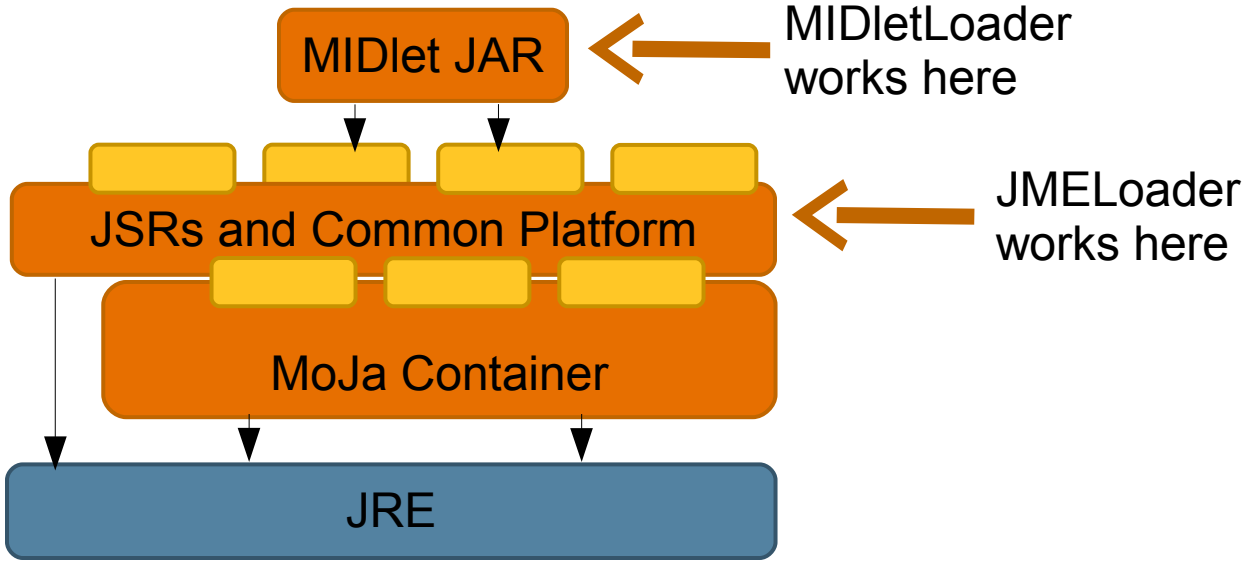
# How to handle the MIDlet suite under execution?

- When the MIDlet is executed, the *DeviceHandler* is responsible to
  - **Read / Parse** the Suite JAD file
  - **Locate** the Suite JAR File
  - **Start** the MIDletLoader pointing to the MIDlet JAR File
  - **Create** the MIDlet it self and **manage** its state transitions
- *MIDletLoader* is responsible to
  - **Find** classes inside the MIDlet JAR
  - **Load** classes inside the MIDlet JAR
- *MIDletLoader* the only responsible to load MIDlet classes / resources

# Issue 4: How to separate the Java ME platform from the Java SE platform?

## MIDlet Tries to Load a Class

- *MIDletLoader* and *JMELoader* Class Loaders both avoid the MIDlet from accessing any non Java ME class
- *JMELoader* class loader allow access to all Java SE classes
  - But *MIDletLoader* doesn't



# How to separate the Java ME platform from the Java SE platform?

## MIDlet Tries to Load a Class

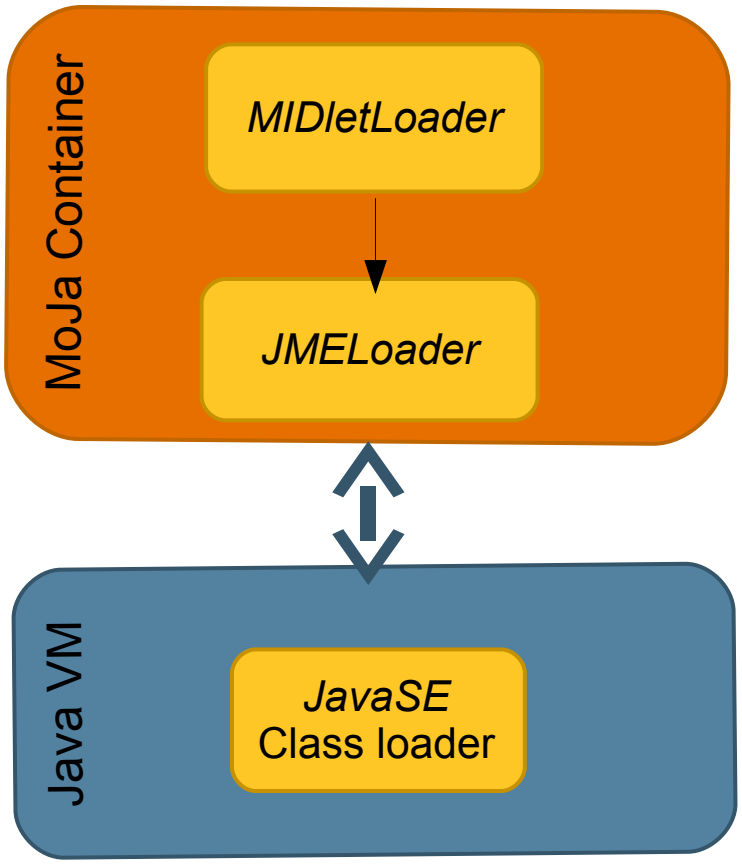
- `MIDletLoader` execute the following steps to load a class
  1. Check if the class is inside the MIDlet JAR
  2. If it is not, ask `JMELoader` to load the class (JSRs JARs)
  3. If the class is not found until step 2, a `ClassNotFoundException` is returned

- `JMELoader` execute the following steps
  1. Check if the class is inside the JSRs JARs;
  2. If it is not, check if the class is inside the Common Platform;
  3. If it is not, ask the system class loader to load the class;
  4. If the class is not found until step 3, a `ClassNotFoundException` is returned.

# Main Issues in Java ME Platform Container Design
## Summarized Answers

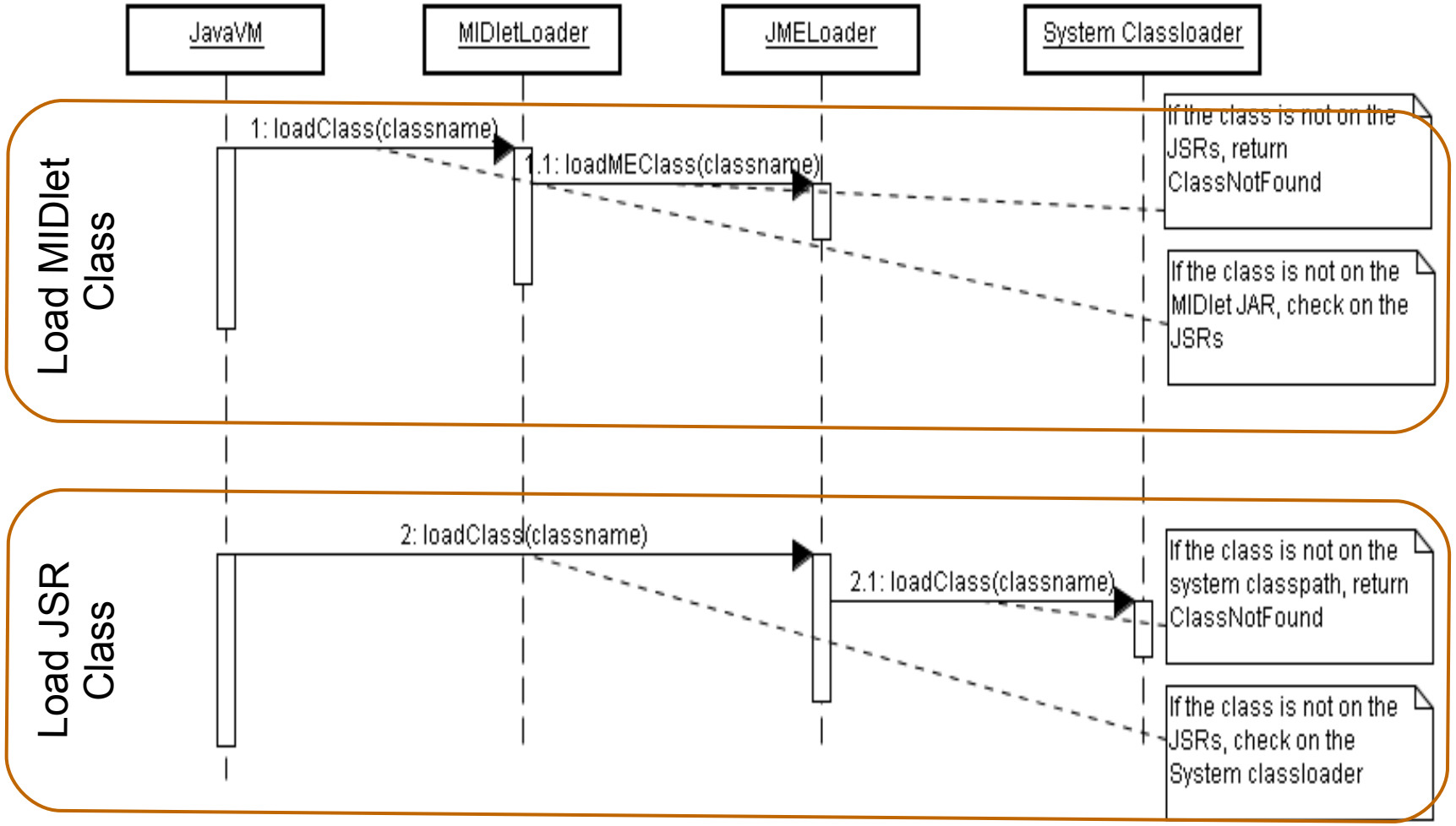
- How to represent each specific JSR?
  - MoJa has a common platform
  - Each JSR has it own JAR that depends on the common platform
- How to represent each Device?
  - Device property represents a device
  - `DeviceHandler` understands the device property file
  - `JMELoader` loads all JSRs JARs
- How to handle the MIDlet suite under execution?
  - `DeviceHandler` manage the MIDlet execution
  - `MIDletLoader` loads the MIDlet JAR
- How to separate the Java ME platform from the Java SE platform?
  - `MIDletLoader` only allow access to MIDlet and JSR classes
  - `JMELoader` allow access to JSRs and JRE classes

# MoJa Container Architecture
## High Level Logical View




- 3 levels of class loaders
- `MIDletLoader`
  - Knows how to load all MIDlet suite specific classes and resources
- `JMELoader`
  - Knows how to load all JSR specific classes and resources
- Java SE System Class Loader
  - Knows how to load all Java SE classes and resources

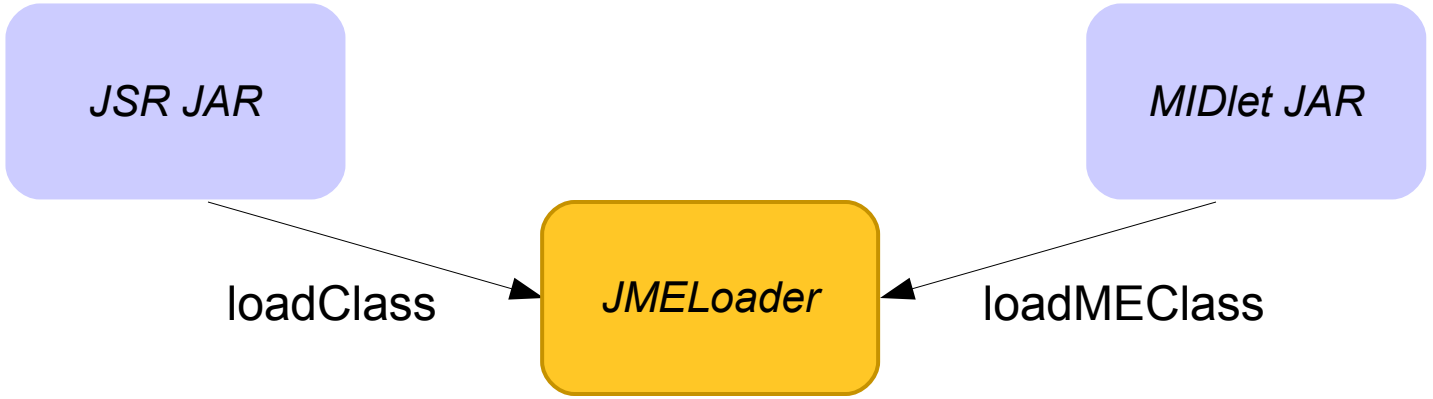
# MoJa Container Architecture
## Load Class Sequence Diagram



# MoJa Container Architecture
## JMELoader Entry Points

- While the MIDlet is under execution, there are two main entry points on the *JMELoader*
  - *loadClass()*: usual load class method. Used by all classes that were loaded by JMELoader
  - *loadMEClass()*: specific load class method that only check for classes inside the OPs & Platform.

- *loadMEClass()* method guarantees that only Java ME classes are accessed by the MIDlet



# MoJa Container Architecture
## MIDletLoader.loadClass() Code Sample

```
// verify if it was already loaded
clazz = findLoadedClass(name);
if (clazz != null) return clazz;

try {
  // try to find the class on MIDlet JAR classpath
  clazz = findClass(name);
} catch (ClassNotFoundException e) {
  try {
    // try to find the class on Java ME library
    clazz = getJMELoader().loadMEClass(name, resolve);
  } catch (Exception exc) {
    throw new ClassNotFoundException();
  }
  if (clazz == null) throw new ClassNotFoundException("");
}
return clazz;
```

Check MIDlet JAR

Check JSRs & Platform

# MoJa Container Architecture
## JMELoader.loadClass() Code Sample

```
// verify if it was already loaded
clazz = findLoadedClass(name);
if (clazz != null) return clazz;

// try to find the class on java me library
try {
  clazz = findClass(name);
} catch (ClassNotFoundException e) {
  //try to find the class from the system
  clazz = findSystemClass(name);
}
return clazz;
```


# Agenda

- Application Model
- Application Container
- Java Class Loaders
- Motorola Java ME Platform Emulator Scenario
- Main Issues in Java ME Container Design
- Conclusions

# Conclusions

- Java platform allows many different application models
- Each application model has its own particularities
- An application model container provides the necessary concepts to isolate each model particularities from the underlaying platform
- Java ME platform defines the MIDlet application models that can be implemented on top of a Java SE platform
- Java SE classes loader is a powerful mechanism that can be used to implement a Java ME container

# For More Information

(1)A Survey of Java ME Today (Update)
http://developers.sun.com/mobility/getstart/articles/survey

(2)Sun Mobile Device Technology - Introduction to Mobility Java Technology http://developers.sun.com/mobility/getstart/

(3)MIDP Javadocs
http://java.sun.com/javame/reference/apis/jsr118/

# THANK YOU

Christian Kurzke, Developer Platforms Architect, Motorola Inc.

Gustavo de Paula, Senior Consultant for Wireless Technologies, C.E.S.A.R

TS-7575, JavaSE and Cool Stuff