Closures Cookbook

Neal Gafter
Google

TS-5579

{=>JAVA}

Java

Sun microsystems

Google is open to multiple parallel investigations of Closures but is not currently prepared to commit to any particular proposal

This talk shows how one approach can reduce boilerplate in programs

Goal:

Use and recognize common idioms with “BGGA Closures” to eliminate boilerplate in uses of an API

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

BGGA Review: Closure Expression

```
Runnable r = { => doWhatever(); };
r.run();
```

BGGA Review: Closure Expression

```
Runnable r = { => doWhatever(); };
r.run();

Callable<String> cs = { => "result" };
```

BGGA Review: Closure Expression

```
Runnable r = { => doWhatever(); };
r.run();

Callable<String> cs = { => "result" };

Comparator<String> reverse =
   { String s1, String s2 => s2.compareTo(s1) };
```

BGGA Review: Closure Expression

```
Runnable r = { => doWhatever(); };
r.run();

Callable<String> cs = { => "result" };

Comparator<String> reverse =
   { String s1, String s2 => s2.compareTo(s1) };

ItemSelectable is = ...;
is.addItemListener(
   { ItemEvent e => doSomething(e, is); });
```

BGGA Review: Closure Expression

- Creates an object that represents
  - Code of the body
  - Lexical context
- An instance of some interface
- Few restrictions
  - May access locals, **`this`**
  - May **`return`** from enclosing method

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Aggregate Operations

- Input is an aggregate: array, list, map, etc.
- Bulk computation over data
- Part of computation caller-defined
- Can often be “automatically” parallelized

- sort, filter, map, reduce, fold, etc.

Aggregate Operations

```
// sequential code
double highestGpa = -Double.MAX_VALUE;
for (Student s : students) {
  if (s.graduationYear == THIS_YEAR) {
    int gpa = s.getGpa();
    if (highestGpa < gpa) highestGpa = gpa;
  }
}
```

```
// aggregate code
final double highestGpa =
  students
  .filter({ Student s => s.graduationYear==THIS_YEAR })
  .map({ Student s => s.getGpa() })
  .max();
```

Aggregate Operations

- See also TS-5515
  Let's Resync: What's New for Concurrency on the Java™ Platform, Standard Edition
  Brian Goetz
  Tuesday (yesterday) 6:00 PM
- See also Java Specification Request 166y

- Aggregate operations may be added with closures

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

An API

```
/** Record performance statistics of an operation. */
public void recordTiming(
  String opName,
  long nanoseconds,
  boolean succeeded) { ... }
```

An API... and its clients

```
long startTime = System.nanoTime();
boolean success = false;
try {
  // a series of statements to be timed
  success = true;
} finally {
  recordTiming(
    "opName", System.nanoTime() - startTime, success);
}
```

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- **Reducing boilerplate**
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Move the boilerplate into an API?

```
class Tracer {
  private final long startTime = System.nanoTime();
  private final String opName;
  private boolean success = false;
  public Tracer(String opName) {
    this.opName = opName;
  }
  public success() {
    this.success = true;
  }
  public void done() {
    recordTiming(
      opName, System.nanoTime() - startTime, success);
  }
}
```

New client...

```
Tracer tracer = new Tracer("opName");
try {
  // a series of statements to be timed
  tracer.success();
} finally {
  tracer.done();
}
```

Another example...

```
return ...compute result...;
```

...becomes

```
Tracer tracer = new Tracer("opName");
try {
  ResultType result = ...compute result...;
  tracer.success();
  return result;
} finally {
  tracer.done();
}
```

Another example...

```
if (quick path) {
  return ...compute result...;
}
// more statements
return ...compute result...;
```

...becomes

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    ResultType result = ...compute result...;
    tracer.success(); // easy to forget
    return result;
  }
  // more statements
  ResultType result = ...compute result...;
  tracer.success();
  return result;
} finally {
  tracer.done();
}
```

Change default to success

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    return ...compute result...;
  }
  // more statements
  return computation();
} catch (MyException1 ex) {
  tracer.failure();
  throw ex;
} catch (MyException2 ex) {
  tracer.failure();
  throw ex;
} finally {
  tracer.done();
}
```

Multicatch (separate proposal)

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    return ...compute result...;
  }
  // more statements
  return computation();
} catch (MyException1 | MyException2 ex) {
  tracer.failure();
  throw ex;
} finally {
  tracer.done();
}
```

Multicatch (separate proposal)

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    return ...compute result...;
  }
  // more statements
  return computation();
} catch (MyException1 | MyException2 ex) {
  tracer.failure();
  throw ex;
} finally {
  tracer.done();
}
```

But don't forget

`Error`, `RuntimeException`

Change default to success

```
void enclosingMethod() { // throws Nothing
  Tracer tracer = new Tracer("opName");
  try {
    if (quick path) {
      return ...compute result...;
    }
    // more statements
    return computation();
  } catch (Throwable ex) {
    tracer.failure();
    throw ex; // Error: Throwable not declared
  } finally {
    tracer.done();
  }
```

“Rethrow” feature (separate proposal)

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    return ...compute result...;
  }
  // more statements
  return computation();
} catch (final Throwable ex) {
  tracer.failure();
  throw ex; // "rethrow" same exceptions
} finally {
  tracer.done();
}
```

ARM blocks? (separate proposal)

```
Tracer tracer = new Tracer("opName");
try {
  if (quick path) {
    return ...compute result...;
  }
  // more statements
  return computation();
} catch (final Throwable ex) {
  tracer.failure();
  throw ex; // "rethrow" same exceptions
} finally {
  tracer.done();
}
```

“ARM blocks” don't appear capable of distinguishing normal from exceptional cases.

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Closure-based API

```
interface Block {
  void execute();
}
public void time(String opName, Block block) {
  long startTime = System.nanoTime();
  boolean success = false;
  try {
    block.execute();
    success = true;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

... and its client

```
public void time(String opName, Block block) ...
```

```
time("opName", {=>
  // a series of statements to be timed
});
```

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Propagating Exceptions

```
time("opName", {=>
  // statements that might throw MyException
});
```

Propagating Exceptions

```
time("opName", {=>
  // statements that might throw MyException
});
```

Error: a block can't throw MyException.

Propagating Exceptions

```
time("opName", {=>
  // statements that might throw MyException
});
```

```
Error: a block can't throw MyException.
```

```
interface Block {
  void execute();
}
```

Closure-based API

```
interface Block {
  void execute(); // throws Nothing
}
public void time(
    String opName, Block block) {
  long startTime = System.nanoTime();
  boolean success = false;
  try {
    block.execute();
    success = true;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

“Exception Transparency”

```
interface Block<X extends Exception> {
  void execute() throws X;
}
public <X extends Exception> void time(
    String opName, Block<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = false;
  try {
    block.execute();
    success = true;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

“Exception Transparency”

```
time("opName", {=>
  // statements that might throw MyException
});
```

```
// But what about zero or two exceptions?
```

“Exception Transparency” (BGGA)

```
interface Block<throws X> {
  void execute() throws X;
}
public <throws X> void time(
    String opName, Block<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = false;
  try {
    block.execute();
    success = true;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

“Exception Transparency” (BGGA)

```
time("opName", {=>
  // statements that might throw some exceptions
});
```

Exceptions thrown in the block are known by the compiler to be thrown by the invocation of “time”

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Completion

To time the body of a method

```
int f() {
  return ...compute result...; // return result from f
}
```

we write

```
int f() {
  time("opName", {=>
    return ...compute result...; // return result from f
  });
}
```

Completion

To time the body of a method

```
int f() {
  return ...compute result...; // return result from f
}
```

we write

```
int f() {
  time("opName", {=> //but the "success" status is wrong
    return ...compute result...; // return result from f
  });
}
```

Success Status

```
interface Block<throws X> {
  void execute() throws X;
}
public <throws X> void time(
    String opName, Block<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = false;
  try {
    block.execute();
    success = true; // might not be executed
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Success Status

```
interface Block<throws X> {
  void execute() throws X;
}
public <throws X> void time(
    String opName, Block<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex; // rethrow
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Completion

```
int f() {
  time("opName", {=>
    return ...compute result...; // return result from f
  });
}
```

Error: missing return statement.

A block that cannot complete normally

```
interface NothingBlock<throws X> {
  Nothing execute() throws X;
}
public <throws X> Nothing time2(
    String opName, NothingBlock<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    return block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Completion

```
int f() {
  time("opName", {=>
    // some statements
  });
  time2("opName", {=>
    return ...compute result...;
  });
}
```

A block that can complete normally

```
interface VoidBlock<throws X> {
  Void execute() throws X;
}
public <throws X> Void time3(
    String opName, VoidBlock<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    return block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Completion

```
int f() {
  time3("opName", {=>
    // some statements
  });
  time2("opName", {=>
    return ...compute result...;
  });
}
```

A block that cannot complete normally

```
interface NothingBlock<throws X> {
  Nothing execute() throws X;
}
public <throws X> Nothing time2(
    String opName, NothingBlock<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    return block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

A block that can complete normally

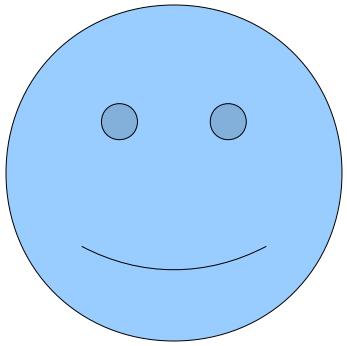
```
interface VoidBlock<throws X> {
  Void execute() throws X;
}
public <throws X> Void time3(
    String opName, VoidBlock<X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    return block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Combine them using generics (BGGA)

```
interface Block<R, throws X> {
  R execute() throws X;
}
public <R, throws X> R time(
    String opName, Block<R, X> block) throws X {
  long startTime = System.nanoTime();
  boolean success = true;
  try {
    return block.execute();
  } catch (final Throwable ex) {
    success = false;
    throw ex;
  } finally {
    recordTiming(
      "opName", System.nanoTime() - startTime, success);
  }
}
```

Completion

```
int f() throws MyException {
  time("opName", {=>
    // some statements that can throw MyException
  });
  time("opName", {=>
    return ...compute result...;
  });
}
```



Completion

```
int f() {
  time("opName", {=>
    if (quick path) return ...compute result...;
    // more statements
    return ...compute result...;
  });
}
```

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Simplified Syntax

```
time("opName", {=>
   ...;
});
```

Simplified Syntax

```
time("opName", {=>
   ...;
});
```

```
time("opName") {
   ...;
}
```

Agenda

- BGGA Review
- Aggregate Operations
- An API
- Boilerplate in its clients
- Reducing boilerplate
- Reducing boilerplate with closures
- Exceptions
- Completion
- Simplified syntax
- Additional Examples

Additional Examples: withStream

```
withStream (InputStream s : makeInputStream()) {
  // do something with s
}
```

Additional Examples: withStream

```
interface ClosableBlock<R, C extends Closeable, throws X>{
  R invoke(C c) throws X;
}

<R, C extends Closeable, throws X>
R withStream(C c, CloseableBlock<R, ? super C, X> block)
throws X, IOException {
  try {
    return block.invoke(c);
  } finally {
    c.close();
  }
}
```

Additional Examples: withStream

```
<R, C extends Closeable, throws X>
R withStream(C c, { C ==> R throws X } block)
throws X, IOException {
  try {
    return block.invoke(c);
  } finally {
    c.close();
  }
}
```

Additional Examples: eachEntry

```
Map<Key,Value> map = ...;

for eachEntry(Key k, Value v : map) {
    // do something with k, v
}
```

Additional Examples: eachEntry

```
<K,V,throws X>
void for eachEntry(
        Map<K,V> m,
        { K, V ==> void throws X } block) throws X {
    for (Map.Entry<K,V> e : m.entrySet()) {
        block.invoke(e.getKey(), e.getValue());
    }
}
```

Additional Examples: withLock

```
import java.util.concurrent.locks.Lock;

Lock lock = ...;

withLock (lock) {
    // do something while holding the lock
}
```

Additional Examples: withLock

```
<R,throws X>
R withLock(
        Lock lock,
        { ==> R throws X } block) throws X {
    lock.lock();
    try {
        return block.invoke();
    } finally {
        lock.unlock();
    }
}
```

THANK YOU

Closures Cookbook

Neal Gafter
Google

TS-5579

Java

Sun microsystems