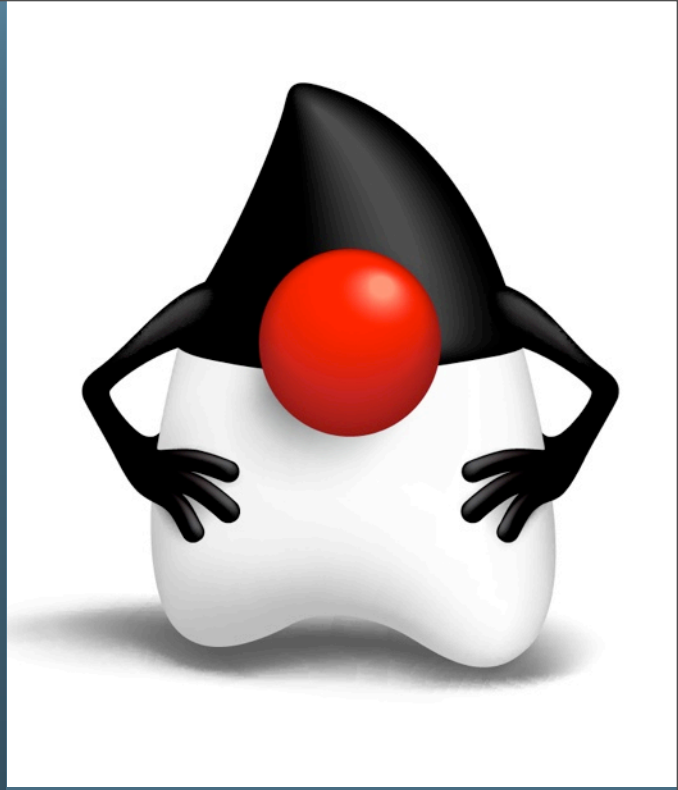
Project Lambda in Java SE 8

Daniel Smith
Java Language Designer



ORACLE

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

The Java Programming Language

- Around 9,000,000 developers worldwide
- 17 years old
- 4 major revisions (1996, 2000, 2005, 2013...)
- [Insert staggering number] of companies very heavily invested
- Formally standardized and evolved via community

Evolving a Major Language

- Adapting to change
- Righting what's wrong
- Maintaining compatibility
- Preserving the core

Project Lambda: Function Values in Java

Code as Data

```
(define f
  (lambda (x) (* x x)))

(map nums f)
```

```
Object subclass: Widget [
  draw: canvas [ ... ]
  click [ ... ]
]

gui add:(Widget new).
```


Status Quo in Java 2

```
interface Runnable {
  void run();
}

Thread hello = new Thread(new Runnable() {
  public void run() {
    System.out.println(“Hello, world!”);
  }
});
```


Status Quo in Java 5

```
interface Predicate<T> {
  boolean accept(T arg);
}

lines.removeAll(new Predicate<String>() {
  public boolean accept(String line) {
    return line.startsWith(“#”);
  }
});
```

What We Wish It Looked Like

```
interface Predicate<T> {
  boolean accept(T arg);
}

lines.removeAll(line -> line.startsWith(“#”));
```


Why Functions in Java? Better Libraries

- *Lots* of applications...
- Our priorities:
  - Collections
  - Concurrency

```
public class ForkBlur extends RecursiveAction {
  private int[] mSource;
  private int mStart;
  private int mLength;
  private int[] mDestination;

  public ForkBlur(int[] src, int start, int length, int[] dst) {
    mSource = src;
    mStart = start;
    mLength = length;
    mDestination = dst;
  }

  // Average pixels from source, write results into destination.
  protected void computeDirectly() {
    for (int index = mStart; index < mStart + mLength; index++) {
      mDestination[index] = blur(index, mSource);
    }
  }

  protected static int sThreshold = 10000;

  protected void compute() {
    if (mLength < sThreshold) {
      computeDirectly();
      return;
    }

    int split = mLength / 2;

    invokeAll(new ForkBlur(mSource, mStart, split, mDestination),
              new ForkBlur(mSource, mStart + split, mLength - split, mDestination));
  }

}
```


Java 8 Language Concepts & Features

- Lambda expressions
- Functional interfaces
- Target typing
- Method references
- Default methods

Lambda Expressions

Lambda Expressions

```
x -> x+1

(s,i) -> s.substring(0,i)

(Integer i) -> list.add(i)

() -> System.out.print("x")

cond -> cond ? 23 : 57
```

```
widget -> {
  if (flag) widget.poke();
  else widget.prod();
}

(int x, int y) -> {
  assert x < y;
  return x*y;
}
```


Variable Capture

- Lambdas can refer to variables declared outside the body
- These variables can be final or "effectively final"
  - Works for anonymous classes, too

```
void cut(List<String> l,
         int len) {

  l.updateAll(s ->
    s.substring(0, len));

}
```


Meaning of Names in Lambdas

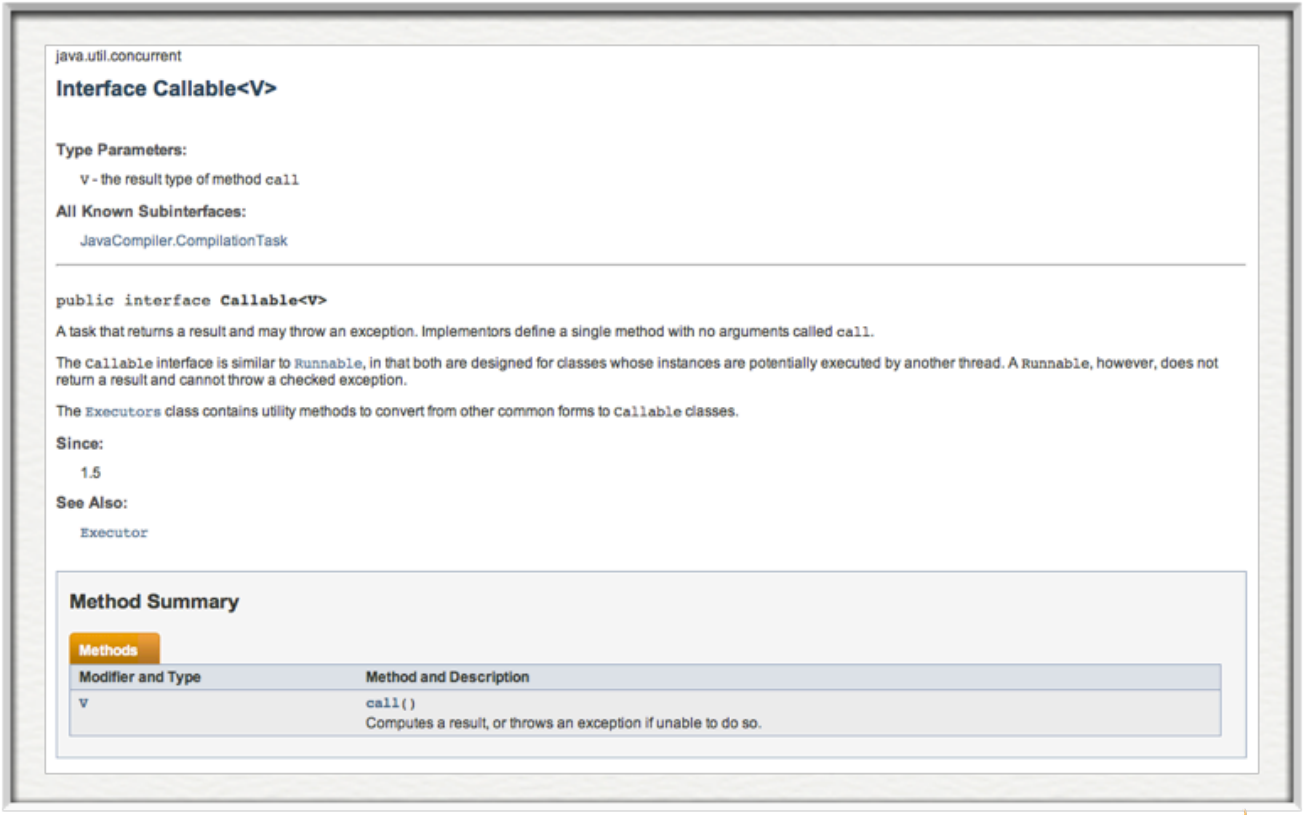
- Anonymous classes introduce a new “level” of scope
  - ‘`this`’ means the inner class instance
  - ‘`ClassName.this`’ is used to get to the enclosing class instance
  - Inherited names can shadow outer-scope names
- Lambdas reside in the same “level” as the enclosing context
  - this refers to the enclosing class
  - No new names are inherited
  - Like local variables, parameter names can’t shadow other locals

Functional Interfaces

Function Types in Java?

```
String -> int

(String, int, boolean) -> List<? extends Integer>

(String, Number) -> Class<?> throws IOException
```


Function Types in Java: Functional Interfaces



java.util.concurrent

Interface Callable<V>

Type Parameters:

V - the result type of method `call`

All Known Subinterfaces:

JavaCompiler.CompilationTask

`public interface Callable<V>`

A task that returns a result and may throw an exception. Implementors define a single method with no arguments called `call`.

The `Callable` interface is similar to `Runnable`, in that both are designed for classes whose instances are potentially executed by another thread. A `Runnable`, however, does not return a result and cannot throw a checked exception.

The `Executors` class contains utility methods to convert from other common forms to `Callable` classes.

Since:

1.5

See Also:

`Executor`

Method Summary

Methods

Modifier and Type	Method and Description
V	`call()` Computes a result, or throws an exception if unable to do so.

 | |

Common Existing Functional Interfaces

- java.lang.Runnable
- java.util.concurrent.Callable<V>
- java.security.PrivilegedAction<T>
- java.util.Comparator<T>
- java.io.FileFilter
- java.nio.file.PathMatcher
- java.lang.reflect.InvocationHandler
- java.beans.PropertyChangeListener
- java.awt.event.ActionListener
- javax.swing.event.ChangeListener

Attributes of Functional Interfaces

- Parameter types
- Return type
- Method type arguments
- Thrown exceptions
- An expressive, reifiable type name (possibly generic)
- An informal contract

Shiny New Functional Interfaces*

- java.util.functions.Predicate<T>
- java.util.functions.Factory<T>
- java.util.functions.Block<T>
- java.util.functions.Mapper<T, R>
- java.util.functions.BinaryOperator<T>

* Names and concepts in libraries are still tentative

Declare Your Own

```
/** Creates an empty set. */
public interface SetFactory {
  <T> Set<T> create();
}

/** Performs a blocking, interruptible action. */
public interface BlockingTask<T> {
  <T> T run() throws InterruptedException;
}
```


Target Typing

Assigning a Lambda to a Variable

```
// Runnable: void run()
Runnable r =
            () -> System.out.println(“hi”);

// Predicate<String>: boolean test(String arg)
Predicate<String> pred =
            s -> s.length() < 100;
```


Target Typing Errors

```
Object o =
            () -> System.out.println(“hi”);

// Predicate<String>: boolean test(String arg)
Predicate<String> pred =
            () -> System.out.println(“hi”);
```


Target Typing in Java 7

```
long[][] arr =
            { { 1, 2, 3 }, { 4, 5, 6 } };

List<? extends Number> nums =
            Collections.emptyList();

Set<Map<String, Object>> maps =
            new HashSet<>();
```


Target Typing for Invocations

```
class Thread {
  public Thread(Runnable r) { ... }
}

// Runnable: void run()
new Thread(() -> System.out.println("hi"));
```

Target Typing for Invocations

```
interface Stream<T> {
  Stream<T> filter(Predicate<T> pred);
}

Stream<String> strings = ...;

// Predicate<T>: boolean test(T arg)
strings.filter(s -> s.length() < 100);
```


A Recipe for Disaster
(Or: A Recipe for Awesome)

- Target typing
- Overload resolution
- Type argument inference

```
<T> int m(Predicate<T> p);
int m(FileFilter f);
<S,T> int m(Mapper<S,T> m);

m(x -> x == null);
```


Other Target Typing Contexts

```
Object o =
        (Runnable) () -> System.out.println("hi");

Runnable r =
        condition() ? null : () -> System.gc();

Mapper<String, Runnable> m =
        s -> () -> System.out.println(s);
```


Method References

Boilerplate Lambdas

```
(x, y, z) -> Arrays.asList(x, y, z)

(str, i) -> str.substring(i)

() -> Thread.currentThread().dumpStack()

(s) -> new File(s)
```

Method (and Constructor) References

```
(x, y, z) -> Arrays.asList(x, y, z)
Arrays::asList
(str, i) -> str.substring(i)
String::substring
() -> Thread.currentThread().dumpStack()
Thread.currentThread()::dumpStack
(s) -> new File(s)
File::new
```

Resolving a Method Reference

- Target type provides argument types
- Named method is searched for using those argument types
  - Searching for an instance method, the first parameter is the receiver
- Return type must be compatible with target return

Method References & Generics

```
Mapper<Byte, Set<Byte>> m1 = Collections::singleton;

// SetFactory: <T> Set<T> create()
SetFactory f2 = Collections::emptySet;

Mapper<Queue<Float>, Float> m2 = Queue::peek;

Factory<Set<String>> f3 = HashSet::new;
```


Default Methods

Evolving APIs

New concrete methods: Good

```
abstract class Widget {
  abstract double weight();
  abstract double volume();

  double density() {
    return weight()/volume();
  }
}
```

New abstract methods: Bad

```
interface Widget {
  double weight();
  double volume();

  double density();
}
```


Workaround: Garbage Classes

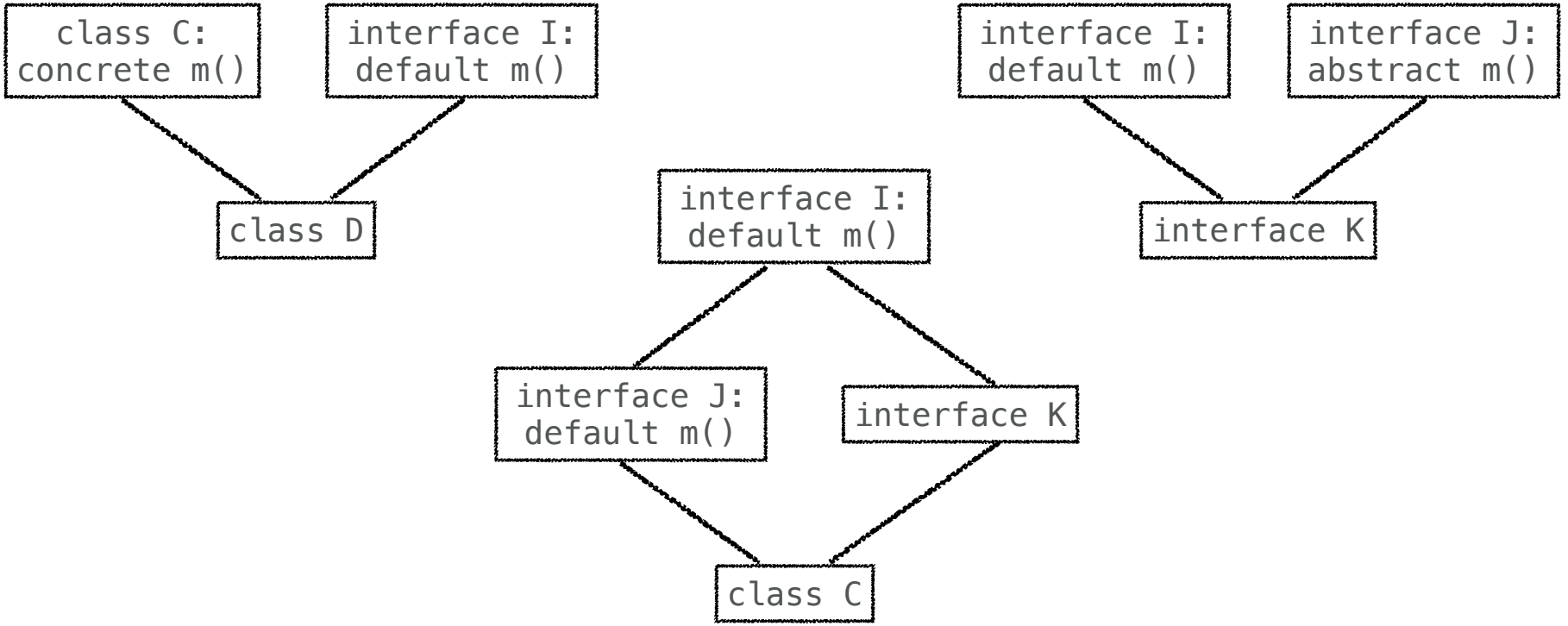
- Not really a class
- Non-idiomatic invocation syntax
- Non-virtual

```
class Widgets {

  static double density(Widget w) {
    return w.weight()/w.volume();
  }

}
```


Default Methods: Code in Interfaces

```
interface Widget {
  double weight();
  double volume();

  default double density() {
    return weight()/volume();
  }
}
```


Multiple Inheritance?



 | |

Evolving the Java Standard API

```
interface Enumeration<E> extends Iterator<E> {
  boolean hasMoreElements();
  E nextElement();

  default boolean hasNext() { return hasMoreElements(); }
  default E next() { return getNext(); }
  default void remove() { throw new UnsupportedOperationException(); }

  default void forEachParallel(Block<T> b) { ... }
}
```

Summary

Goals for Project Lambda

- Make dramatic & necessary enhancements to the programming model
- Smooth some rough edges in the language
- Preserve compatibility
- Maintain the essence of the Java language

Learning More

- **OpenJDK:** openjdk.java.net/projects/lambda
- **JSR 335:** www.jcp.org/en/jsr/detail?id=335
- **Me**: daniel.smith@oracle.com

- Download it and try it out!

