Java 8
New and Noteworthy

Presented By: Andrey Loskutov

Agenda

ADVANTEST

- Motivation
- Java 8 Highlights
  - Lambdas
  - Lambdas
  - Lambdas
  - Default methods
  - Streams
- Known issues

Motivation

ADVANTEST

Java release 8 represents (from Java 7 point of view):

- 2 years work on the next major Java release
- Better support for scripting languages (faster "invokedynamic")
- Java bytecode specification changes
- Java language syntax changes
- lot of new API's
- various other smaller enhancements…

*Note for C/C++ programmers:

… code written for Java 1.1 (1997) is still supported on 1.8 (2014)

ADVANTEST

“Classic” way to write some small function pointer in Java

```
Runnable helloWorld = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello world");
    }
};

// later in other place:
helloWorld.run();
```

Lambda’s way to do the same:

```
Runnable helloWorld = () -> System.out.println("Hello lambda");
```

This looks much cleaner, but where is the catch?

Lambdas: Motivation

ADVANTEST

Motivation

- Function pointers for Java (functions as objects)
- Alternative for anonymous classes
- Cleaner code
- “Fluent” programming style with Streams

Caution

- Lambdas is NOT same as anonymous classes
- Complex expressions are hard to read
- Compiler/debugger/tool support is still evolving
- Some subtle identity issues

Syntax in four words

- Lambda parameters -> Lambda body

Lambdas: Syntax

ADVANTEST

https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.27

```
() -> {}                    // No parameters; result is void
() -> 42                    // No parameters, expression body
() -> null                  // No parameters, expression body
() -> { return 42; }        // No parameters, block body with return
() -> { System.gc(); }      // No parameters, void block body

() -> {                     // Complex block body with returns
  if (true) return 12;
  else {
    int result = 15;
    for (int i = 1; i < 10; i++)
      result *= i;
    return result;
  }
}

(int x) -> x+1                // Single declared-type parameter
(int x) -> { return x+1; }    // Single declared-type parameter
(x) -> x+1                    // Single inferred-type parameter
x -> x+1                      // Parentheses optional for
                              // single inferred-type parameter
(String s) -> s.length()        // Single declared-type parameter
(Thread t) -> { t.start(); }    // Single declared-type parameter
s -> s.length()                 // Single inferred-type parameter
t -> { t.start(); }             // Single inferred-type parameter

(int x, int y) -> x+y       // Multiple declared-type parameters
(x, y) -> x+y               // Multiple inferred-type parameters
(x, int y) -> x+y           // Illegal: can't mix inferred and declared types
(x, final y) -> x+y         // Illegal: no modifiers with inferred types
```

Lambdas: SAM

ADVANTEST

SAM aka @FunctionalInterface - Single Abstract Method

Many existing classes:

- JDK: Runnable, Callable, Function, Predicate, Consumer, Supplier …
- Eclipse: ICoreRunnable, ISafeRunnable, DisposeListener, IPropertyChangeListener …

```
@FunctionalInterface
public interface DisposeListener extends SWTEventListener {

/**
 * Sent when the widget is disposed.
 *
 * @param e an event containing information about the dispose
 */
public void widgetDisposed(DisposeEvent e);
```

```
...
Control control = getParentControl();
Job refreshJob = createRefreshJob();

control.addDisposeListener(e -> refreshJob.cancel());
```

Lambdas: Surprises

ADVANTEST

- Are lambdas Objects?
- But there is no new!
- Is `()->42 == ()->42` ?
- It depends!
- What about `() -> return this` ?
- Lamda expressions do not introduce new scope!

```
int x = 42;
Consumer<Integer> c = (i) -> {
    // Does not compile
    // int x = i;
};
```

- “Lambda expression's local variable cannot redeclare another local variable defined in an enclosing scope.”

Lambdas: Capture

ADVANTEST

Non capturing vs capturing lambdas

```
AtomicReference<Long> ref = new Atomicreference(42);

Runnable nonCapturing = () -> System.out.println("Hello");

Runnable capturing = () -> System.out.println(ref.get());
```

- Non capturing lambdas have no state
- Non capturing lambdas **can** be reused by JVM
- Capturing lambdas can have state thanks to captured objects

"Any local variable, formal parameter, or exception parameter used but not declared in a lambda expression must either be declared final or be effectively final."

Lambdas: Method References

ADVANTEST

We have now method and constructor references

```
List<Integer> list = Arrays.asList(1, 3, 1, 2);

HashSet<Integer> set = list.stream().collect(HashSet::new, HashSet::add,
HashSet::addAll);

Supplier<HashSet<Integer>> supplier = HashSet::new;

BiConsumer<HashSet<Integer>, ? super Integer> biConsumer = HashSet::add;

BiConsumer<HashSet<Integer>, HashSet<Integer>> combiner = HashSet::addAll;


set = list.stream().collect(supplier, biConsumer, combiner);

// Good old Java API to do the same
set = new LinkedHashSet<>(list);

Consumer<Object> println = System.out::println;
list.forEach(println);

Comparator<Integer> comparator = Integer::compare;
list.sort(comparator);
```

Looks differently to lambdas!

Implemented same way in the bytecode (invokedynamic)

Lambdas vs Inner: Use

ADVANTEST

Pro

- Lambdas are good for one-line callbacks
- Lambdas should not be bigger than few lines

Contra

- Complex lambda code can be hard to read
- Lambdas **always** produce hard to read stack traces
- Lambdas ~~can~~ make debugging hard

Lambdas vs Inner: Stack traces

ADVANTEST

Lambda anti-pattern: https://bugs.eclipse.org/498051

Huston, we have a problem…

```
Message: Unhandled event loop exception
java.lang.ArrayIndexOutOfBoundsException: null
 at java.lang.System.arraycopy(null:-2)
 at java.util.Arrays.copyOfRange(null:-1)
 at java.util.Arrays.copyOfRange(null:-1)
 at
org.eclipse.ui.internal.ide.ChooseWorkspaceDialog.lambda$7(ChooseWorkspaceDialog.java:3
80)
 at java.util.stream.ReferencePipeline$3$1.accept(null:-1)
 at java.util.ArrayList$ArrayListSpliterator.forEachRemaining(null:-1)
 at java.util.stream.AbstractPipeline.copyInto(null:-1)
 at java.util.stream.AbstractPipeline.wrapAndCopyInto(null:-1)
 at java.util.stream.ReduceOps$ReduceOp.evaluateSequential(null:-1)
 at java.util.stream.AbstractPipeline.evaluate(null:-1)
 at java.util.stream.ReferencePipeline.collect(null:-1)
 at
org.eclipse.ui.internal.ide.ChooseWorkspaceDialog.createUniqueWorkspaceNameMap(ChooseWo
rkspaceDialog.java:382)
```

OK, may be if we look at the code we could get an idea?

Lambdas vs Inner: Stack traces

ADVANTEST

Lambda anti-pattern: https://bugs.eclipse.org/498051

The “code”. **The** code. Try to find the error or count lambdas ☺

```
List<String[]> splittedWorkspaceNames = Arrays.asList(launchData.getRecentWorkspaces()).stream()
        .filter(s -> s != null && !s.isEmpty()).map(s -> s.split(Pattern.quote(fileSeparator)))
        .collect(Collectors.toList());
for (int i = 1; !splittedWorkspaceNames.isEmpty(); i++) {
    final int c = i;
    Function<String[], String> stringArraytoName = s -> String.join(fileSeparator,
            Arrays.copyOfRange(s, s.length - c, s.length));
    List<String> uniqueNames = splittedWorkspaceNames.stream().map(stringArraytoName)
            .collect(Collectors.groupingBy(s -> s, Collectors.counting())).entrySet().stream()
            .filter(e -> e.getValue() == 1).map(e -> e.getKey()).collect(Collectors.toList());
    splittedWorkspaceNames.removeIf(a -> {
        String joined = stringArraytoName.apply(a);
        if (uniqueNames.contains(joined)) {
            uniqueWorkspaceNameMap.put(joined, String.join(fileSeparator, a));
            return true;
        }
        return false;
    });
}
```

NEVER DO THIS!!!

Lambdas vs Inner: Performance

ADVANTEST

Lambdas	Inner classes
No .class files Runtime linkage Variables capture Invocation	Compile time generation Runtime class loading Instantiation Invocation

Some observations about Lambdas

- Reduce compile time (for huge projects)
- Reduce I/O and ClassLoader overhead
- Increase outer class size (more methods generated)
- Capture part: slower before JIT optimization starts
- Invocation: exactly same performance

Lambdas vs Inner: Memory

ADVANTEST

- Simple capturing lambda in a static context and anonymous class have same memory footprint (16 bytes)
- Same lambda in a non static context uses 24 bytes
- Lambdas itself don’t have state
- Lambdas don’t automatically keep reference to outer class
- Capturing and non-capturing lambdas
  - Captured variables are references to “outer” world
  - Captured objects can have state
  - Capturing lambdas are garbage collected, non capturing not
  - Non-capturing lambdas can be “reused” by JVM

Default methods

ADVANTEST

„Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.“

First attempt:

```
interface Figure {
    int getX();
    int getY();
}
```

Adding domain logic:

```
interface Figure {
    int getX();
    int getY();

    default Point getPoint() {
        return new Point(getX(), getY());
    }
}
```

etc …

API Changes: Streams

ADVANTEST

java.util.stream.Stream<T>

All collection classes support streams now:

Collection

- default Stream<T> stream()
- default Stream<T> parallelStream()

```
Stream<Integer> transform(List<Integer> numbers) {
    return numbers.stream().map(e -> e * 2);
}
```

Easy to switch from single threaded computation to parallel:

```
Stream<Integer> transform(List<Integer> numbers) {
    return numbers.parallelStream().map(e -> e * 2);
}
```

Lazy evaluation!

```
// The stream have not calculated anything so far!
Stream<Integer> stream = transform(numbers);

// Calculation starts here!
System.out.println(stream.collect(toList()));
```

Known problems

ADVANTEST

- Javac and ecj compiler have sometimes different opinion about valid Java code ☺
- In case you see strange compile errors, call me
- SpotBugs (formerly known as FindBugs) can’t analyze lambdas yet.
- JDT/SpotBugs: we accept high quality patches!
  - https://wiki.eclipse.org/JDT_Core_Committer_FAQ
  - https://github.com/spotbugs/spotbugs

Links

ADVANTEST

- Official Java 8 Documentation
  - http://www.oracle.com/technetwork/java/javase/8-relnotes-2226341.html
  - http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html
- Lambdas
  - http://www.lambdafaq.org
  - http://www.oracle.com/technetwork/java/jvmls2013kuksen-2014088.pdf
  - https://de.slideshare.net/PeterLawrey/streams-and-lambdas-the-good-the-bad-and-the-ugly
- All examples and this presentation

  git clone https://code.google.com/p/java7examples/