ETH Zürich
Chair of Software Engineering

Java and C# in Depth

Carlo A. Furia, Marco Piccioni, Bertrand Meyer

Exercise Session – Week 7

Project submission today

- Make sure assistants have access to SVN
  - Not necessary to send solution via email
- Make sure you are on Piazza
  - Several questions have been asked there already
- There will be no project presentations
  - Not even for final project phase

Quiz 1: Java final

```java
public class Quiz1 {
   private final int primitiveInt = 42;
   private final int lazyPrimitiveInt;
   private final Integer wrappedInt = 42;
   private final String stringValue = "42";
   public Quiz1() { lazyPrimitiveInt = 42; }

   public int getPrimitiveInt() { return primitiveInt; }
   public int getLazyPrimitiveInt() { return lazyPrimitiveInt; }
   public int getWrappedInt() { return wrappedInt; }
   public String getStringValue() { return stringValue; }

   public void changeField(String name, Object value) {
       Field field = Quiz1.class.getDeclaredField(name);
       field.setAccessible(true);
       field.set(this, value);
       println("reflection: " + name + " = " + field.get(this));
   }
}
```

Quiz 1: Java final (cont.)

```
public static void main(String[] args) {
   Quiz1 test = new Quiz1();
   test.changeField("primitiveInt", 84);
   println("direct: primitiveInt = " + test.getPrimitiveInt());
   test.changeField("lazyPrimitiveInt", 84);
   println("direct: lazyPrimitiveInt = " +
                                  test.getLazyPrimitiveInt());
   test.changeField("wrappedInt", 84);
   println("direct: wrappedInt = " + test.getWrappedInt());
   test.changeField("stringValue", "84");
   println("direct: stringValue = " + test.getStringValue());
}
```

```
reflection: primitiveInt = 84
direct: primitiveInt = 42
reflection: lazyPrimitiveInt = 84
direct: lazyPrimitiveInt = 84
reflection: wrappedInt = 84
direct: wrappedInt = 84
reflection: stringValue = 84
direct: stringValue = 42
```

Java Language Specification 17.5.3

17.5.3. Subsequent Modification of final Fields

In some cases, such as deserialization, the system will need to change the final fields of an object after construction. final fields can be changed via reflection and other implementation-dependent means …

Even then, there are a number of complications. If a final field is initialized to a compile-time constant expression (15.28) in the field declaration, changes to the final field may not be observed, since uses of that final field are replaced at compile time with the value of the constant expression.

Another problem is that the specification allows aggressive optimization of final fields. Within a thread, it is permissible to reorder reads of a final field with those modifications of a final field that do not take place in the constructor.

Quiz 2: Java reflection peformance

```
Object object = new Object();
Class<Object> c = Object.class;

for (int i = 0; i < 1000000; i++) {
   object.toString();
}
Method method = c.getMethod("toString");
for (int i = 0; i < 1000000; i++) {
   method.invoke(object);
}
for (int i = 0; i < 1000000; i++) {
   method = c.getMethod("toString");
   method.invoke(object);
}
```

142 ms

121 ms

308 ms

Running each variant on its own

- JIT compiler likely to influence benchmark results.
- Running each loop in its own program:
  - direct call: 149 ms
  - reflection cached lookup: 156 ms
  - reflection with lookup: 370 ms

How to do micro-benchmarks

- Know your JVM and JIT compiler
- Include a warmup-phase
  - run code, so it can be compiled and optimized
  - no measurement yet
- Run benchmark program multiple times
  - background tasks can interfere
- Make sure compiler and GC are not interfering
  - print output of compiler and GC operations (e.g. -XX:+PrintCompilation flag for JVM)

Quiz 3: Java magic

```
static void magic() {
   ...
}

public static void main(String args[]) {
   magic();
   System.out.format("The magic is %s", false);
}
```

Format as string

Output:

```
The magic is true
```

Quiz 3: Java magic (cont.)

```
static void magic() {
   Field field;
   field = Boolean.class.getField("FALSE");
   field.setAccessible(true);
   Field modifiersField =
               Field.class.getDeclaredField("modifiers");
   modifiersField.setAccessible(true);
   modifiersField.setInt(
          field, field.getModifiers() & ~Modifier.FINAL);
   field.set(null, true);
}
```

- Boolean values are autoboxed to Boolean.TRUE/Boolean.FALSE
- Boolean.FALSE changed to *true*
- When *false* is autoboxed (e.g. when printed), it referes to *true*

Quiz 4: Easy C# reflection

Is there an easier way in C# to do a call using reflection?

```
Object o = "abc";

result = o.GetType().InvokeMember(
  "Equals", BindingFlags.InvokeMethod, null,
  o, new object[]{ "def" });

result = ((dynamic)o).Equals("def");
```

Delegates in Java using reflection

- Delegates could be emulated in Java with reflection:
  - Passing Method objects
  - Need to pass target explicitly (for non-static functions)
  - No multicast delegates
  - No type safety

- Not the *Java way*

Example code

C# code

```
public delegate void Print (String s);
public void Display(Print pMethod)
{
  pMethod("Some text");
}
```

Corresponding Java code using reflection

```
public void display(Object o, Method m)
{
  try {
    m.invoke(o, new Object[]{"Some text"});
  } catch(Exception e) {}
}
```

Delegates in Java using innner classes

- Create an interface for each delegate type
- Create an anonymous inner class for each instantiation of the delegate
- Declared inside a method
  - Access to locals declared as `final`
  - Access to member variables
- Drawbacks
  - More verbose then delegates
  - Captures reference to outer class

Example code

```java
public interface Printable {
   public void print(String s);
}
public class Example {
   public Example() {
       display(new Printable() {
             public void print(String s) {
                   System.out.println(s);
             }
       });
   }

   public void display(Printable p) {
       p.print("Some text");
   }
}
```

Capturing variables in inner classes

```
public void Capture()
{
  final String prefix = "abc";
  display(new Printable() {
    public void print(String s) {
      System.out.println(prefix + s + postfix);
    }
  });
}

private String postfix;
```

Closures in Java 8

- Java 8 expected in September 2013
- Support for lambda-expressions
  - http://www.lambdafaq.org/
  - http://openjdk.java.net/projects/lambda/
- Basic syntax:

```
(int x, int y) -> x + y
(x, y) -> x - y
() -> 42
(String s) -> System.out.println(s)
x -> 2 * x
c -> { int s = c.size(); c.clear(); return s; }
```

Closures as Functional Interfaces

- Function Interface: one abstract method
- Type of lambda expression based on target
- Can directly instantiate a Function interface

```
FileFilter java = (File f) -> f.getName().endsWith(".java");
```

Type FileFilter

```
new Thread(() -> {
  connectToService();
  sendNotification();
}).start();
```

Type Runnable

Type ActionListener

```
ActionListener l = (ActionEvent e) -> do(e.getModifiers());
```

Variable capturing in Java closures

- Can capture local variables of surrounding context that are effectively final
- A variable is *effectively final*, if it could be declared as final without compiler error
- Captured variables cannot be modified in closure

```
Callable<String> helloCallable(String name) {
  String hello = "Hello";
  return () -> (hello + ", " + name);
}
```

Library support for Java closures

- Existing `Collection` class extended with `stream()` function
- Stream interface offers several chainable operations like `filter`, `forEach`, `sorted`, ...

```
List<Shape> blue = shapes.stream()
                         .filter(s -> s.getColor() == BLUE)
                         .into(new ArrayList<>());

int sum = shapes.stream()
                .filter(s -> s.getColor() == BLUE)
                .map(s -> s.getWeight())
                .sum();
```

Questions?