Tricky stuff

in java grammar and javac

About me

Marek Parfianowicz

- Gdańsk University of Technology, ETI, distributed software systems
- Senior Software Engineer at Lufthansa Systems (2004-2012)
- Support Engineer and Software Developer at Spartez (since 2012)
  - main developer of the Atlassian Clover

- C/C++, Java, Groovy, Scala

Language tricks

“&” operator for casting

```
public class LambdaTest {
  public static void main(String[] args) throws Exception {
        Object o1 = (Runnable & Serializable) () ->
        System.out.println("I’m a serializable lambda");

        Object o2 = (Runnable) () ->
        System.out.println("I’m NOT serializable!!!");

        ObjectOutputStream out = new ObjectOutputStream(new ByteArrayOutputStream());
        out.writeObject(o1); // OK
        out.writeObject(o2); // java.io.NotSerializableException
  }
}
```

JDK5

“&” operator for generics

```
class MyClass<S extends Comparable & Runnable> {
    S s;
    MyClass(S s) { this.s = s; }
    S getComparableRunner() { return s; }
    static MyClass<ComparableRunner> cre
        return new MyClass<ComparableRun
    }
}
class ComparableRunner implements Comparable, Runnable {
    public void run() { }
    public int compareTo(Object o) { return 0; }
}
```

```
// instead of:

abstract class ComparableRunnable
    implements Comparable, Runnable { }

class MyClass<S extends ComparableRunnable> {
    // ...
}
```

JDK5

“|” operator for multiple catch (1)

```
public void myCatch(String name) throws IOException {
    try {
        if (name.equals("First"))
            throw new FirstException();
        else
            throw new SecondException();
    } catch (FirstException | SecondException e) {
        throw e;
    }
}
```

JDK7

“|” operator for multiple catch (2)

```
public void myRethrow(String name)
                    throws FirstException, SecondException {
        try {
           if (name.equals("First"))
               throw new FirstException();
           else
               throw new SecondException();
        } catch (IOException e) {
            throw e;
        }
}
```

JDK7

Hexadecimal floats

- “P” for a binary exponent
- “D” and “F” for double and float
- write exact value avoiding decimal rounding problems

```
public class HexFloat {
     double f3 = 0XAB1P0;
     double f4 = 0Xab1aP+20;
     double f5 = 0Xab1aP+20d;
     float f7 = 0Xab1aP+3f;
}
```

JDK5

String in switch

```
public String switchWithString(String name) {
        switch (name) {
        case "Moon":
                return "moon";
        case "Sun":
                return "star";
        default:
            return "unknown";
        }
}
```

~~if (“Moon”.equals(a)) { … }~~
~~else if …~~

JDK7

Try with resources

- java.lang.AutoCloseable

```
try (
    zipFile = new java.util.zip.ZipFile(zipFileName);
    bufferedWriter = java.nio.file.Files.newBufferedWriter(filePath, charset)
) {
    // … make some stuff …
}
```

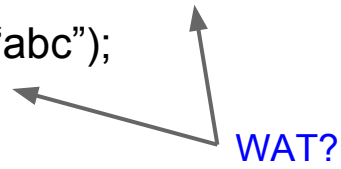
JDK7

Lambda functions (1)

Declaration of an argument. Calling lambda.

```
import java.util.function.Consumer;

void call(Consumer<String> consumer) {
    consumer.accept(“abc”);
}
```




```
// not possible :-(
void call((String -> void) consumer) {
   consumer(“abc”);
}
```

JDK8

Lambda functions (2)

Making lambda serializable:

```
Runnable r = (Runnable & Serializable) () -> System.out.println("Hello");
```

Lambda in ternary expressions:

```
Callable<Integer> c1 = flag ? () -> 23 : () -> 42;
```

Lambda vs operator priorities:

```
Produce<Integer> o = z != 0 ? y -> y < 0 ? y + 1 : y - 1 : y -> 3 * y;

                     z != 0     ? (y) -> { return (y < 0 ? y+1 : y-1); }
                                : (y) -> { return 3 * y; };
```

JDK8

Lambda functions (3)

Recursion in lambda functions:

```
class WithLambdaRecursion {
   static IntUnaryOperator factorial = i ->
         i == 0 ? 1 : i * WithLambdaRecursion.factorial.map(i - 1);
}
```

JDK8

Method references - constructors

- object constructors

```
import java.util.function.Supplier;
Supplier<String> sup = String::new;
String emptyString = sup.get();
```

- array constructors

```
interface ProduceStringArray {
    String[] produce(int n);
 }
ProduceStringArray creator = String[]::new;
String[] stringArray = creator.produce(10);
```

JDK8

Method references - constructors

- generic constructors

```
ArrayList::new                    // raw list
ArrayList::<String>new            // forbidden
ArrayList<String>::new            // generic list
ArrayList<String>::<String>new    // superfluous, but OK
```

JDK8

Compiler quirks

Generics with autoboxing (1)

- A compiler bug? A feature?
- A branch condition with autoboxed Boolean declared as a generic type

```
interface Data {
        public <T> T getValue();
}

public boolean testGetValue(Data source) {
    // shall we expect “Object” for source.getValue()?
}
```

Generics with autoboxing (2)

```
// fails under JDK6, compiles under JDK7+
public boolean testGetValue(Data source) {
        if (source.getValue())  // Implicit conversion to Boolean via autoboxing
        ...
}

// … but compilation of this code fails under JDK7 with a message:
//      Error: Operator && cannot be applied to java.lang.Object, boolean
public boolean testGetValue(Data source) {
        if (source.getValue() && true)
        …
}
```

Generics with autoboxing (3)

```
> javap -c GenericsWithAutoboxing.class

...

  public boolean testGetValue(Data);
        Code:
        0: aload_1
        1: invokeinterface #2,  1        // InterfaceMethod Data.getValue:()Ljava/lang/Object;
        6: checkcast       #3            // class java/lang/Boolean
        9: invokevirtual #4              // Method java/lang/Boolean.booleanValue:()Z
        12: ifeq       17
        15: iconst_1
        16: ireturn
        17: iconst_0
        18: ireturn
```

Cast + generics

- Compiler bug; occurs in JDK6+JDK7; javac is unable to parse () with <>

```
public class Util {
    @SuppressWarnings("unchecked")
    public static <T> T cast(Object x) {
            return (T) x;
    }
    static {
        Util.<Object>cast(null);   // OK
        (Util.<Object>cast(null)).getClass();  // Error: illegal start of expression
    }
}
```

http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6481655

?

Thank you