Still More Java™ Puzzlers

Joshua Bloch
Neal Gafter

Google™

Java™

Sun microsystems

Introduction

- **Eight *more* Java™ programming language puzzles**
  - Short program with curious behavior
  - What does it print? (multiple choice)
  - The mystery revealed
  - How to fix the problem
  - The moral
- Covers language and core libraries
  - No GUI, enterprise, or Tiger features

1. “Long Division”

```
public class LongDivision {
    private static final long MILLIS_PER_DAY
        = 24 * 60 * 60 * 1000;
    private static final long MICROS_PER_DAY
        = 24 * 60 * 60 * 1000 * 1000;

    public static void main(String[] args) {
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

What Does It Print?

(a) `5`

(b) `1000`

(c) `5000`

(d) Throws an exception

What Does It Print?

(a) `5`

(b) `1000`

(c) `5000`

(d) Throws an exception

Computation *does* overflow

Another Look

```
public class LongDivision {
    private static final long MILLIS_PER_DAY
        = 24 * 60 * 60 * 1000;
    private static final long MICROS_PER_DAY
        = 24 * 60 * 60 * 1000 * 1000; // >> Integer.MAX_VALUE

    public static void main(String[] args) {
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

How Do You Fix It?

```
public class LongDivision {
    private static final long MILLIS_PER_DAY
        = 24L * 60 * 60 * 1000;
    private static final long MICROS_PER_DAY
        = 24L * 60 * 60 * 1000 * 1000;

    public static void main(String[] args) {
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

The Moral

- When working with large numbers watch out for overflow—it's a silent killer
- Just because variable is big enough to hold result doesn't mean computation is of correct type
- When in doubt, use `long`

2. “No Pain, No Gain”

```
public class Rhymes {
   private static Random rnd = new Random();
   public static void main(String[] args) {
      StringBuffer word = null;
      switch(rnd.nextInt(2)) {
         case 1:  word = new StringBuffer('P');
         case 2:  word = new StringBuffer('G');
         default: word = new StringBuffer('M');
      }
      word.append('a');
      word.append('i');
      word.append('n');
      System.out.println(word);
   }
}
```

Thanks to madbot (also known as Mike McCloskey)

What Does It Print?

(a) **`Pain`**, **`Gain`**, or **`Main`** (varies at random)

(b) **`Pain`** or **`Main`** (varies at random)

(c) **`Main`** (always)

(d) None of the above

What Does It Print?

(a) **`Pain`**, **`Gain`**, or **`Main`** (varies at random)

(b) **`Pain`** or **`Main`** (varies at random)

(c) **`Main`** (always)

(d) None of the above: **`ain`** (always)

The program has three separate bugs.
One of them is quite subtle.

Another Look

```
public class Rhymes {
   private static Random rnd = new Random();
   public static void main(String[] args) {
      StringBuffer word = null;
      switch(rnd.nextInt(2)) {  // No breaks!
         case 1:  word = new StringBuffer('P');
         case 2:  word = new StringBuffer('G');
         default: word = new StringBuffer('M');
      }
      word.append('a');
      word.append('i');
      word.append('n');
      System.out.println(word);
   }
}
```

How Do You Fix It?

```
public class Rhymes {
   private static Random rnd = new Random();
   public static void main(String[] args) {
      StringBuffer word = null;
      switch(rnd.nextInt(3)) {
         case 1:  word = new StringBuffer("P"); break;
         case 2:  word = new StringBuffer("G"); break;
         default: word = new StringBuffer("M"); break;
      }
      word.append('a');
      word.append('i');
      word.append('n');
      System.out.println(word);
   }
}
```

The Moral

- Use common idioms
  - If you must stray, consult the documentation
- Chars are not strings; they're more like ints
- Always remember breaks in **`switch`** statement
- Watch out for fence-post errors
- Watch out for sneaky puzzlers

3. “The Name Game”

```
public class NameGame {
    public static void main(String args[]) {
        Map m = new IdentityHashMap();
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
        System.out.println(m.size());
    }
}
```

What Does It Print?

(a) `0`

(b) `1`

(c) `2`

(d) It varies

What Does It Print?

(a) `0`

(b) `1`

(c) `2`

(d) It varies

We're using an **`IdentityHashMap`**, but string literals are interned (they cancel each other out)

Another Look

```
public class NameGame {
    public static void main(String args[]) {
        Map m = new IdentityHashMap();
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
        System.out.println(m.size());
    }
}
```

How Do You Fix It?

```
public class NameGame {
    public static void main(String args[]) {
        Map m = new HashMap();
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
        System.out.println(m.size());
    }
}
```

The Moral

- **`IdentityHashMap`** not a general-purpose **`Map`**
  - Don't use it unless you *know* it's what you want
  - Uses identity in place of equality
  - Useful for *topology-preserving transformations*
- (String literals are interned)

4. “More of The Same”

```
public class Names {
    private Map m = new HashMap();
    public void Names() {
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
    }

    public int size() { return m.size(); }

    public static void main(String args[]) {
        Names names = new Names();
        System.out.println(names.size());
    }
}
```

What Does It Print?

(a) `0`

(b) `1`

(c) `2`

(d) It varies

What Does It Print?

(a) `0`

(b) `1`

(c) `2`

(d) It varies

No programmer-defined constructor

Another Look

```
public class Names {
    private Map m = new HashMap();
    public void Names() {    // Not a constructor!
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
    }

    public int size() { return m.size(); }

    public static void main(String args[]) {
        Names names = new Names(); // Invokes default!
        System.out.println(names.size());
    }
}
```

How Do You Fix It?

```
public class Names {
    private Map m = new HashMap();
    public Names() {    // No return type
        m.put("Mickey", "Mouse");
        m.put("Mickey", "Mantle");
    }

    public int size() { return m.size(); }

    public static void main(String args[]) {
        Names names = new Names();
        System.out.println(names.size());
    }
}
```

The Moral

- It is possible for a method to have the same name as a constructor
- **Don't ever do it**
- Obey naming conventions
  - `field, method(), Class, CONSTANT`

5. “Shades of Gray”

```
public class Gray {
    public static void main(String[] args){
        System.out.println(X.Y.Z);
    }
}

class X {
    static class Y {
        static String Z = "Black";
    }
    static C Y = new C();
}

class C {
    String Z = "White";
}
```

Thanks to Prof. Dominik Gruntz, Fachhochschule Aargau

What Does It Print?

(a) **`Black`**

(b) **`White`**

(c) Won't compile

(d) None of the above

What Does It Print?

(a) `Black`

(b) `White`

(c) Won't compile

(d) None of the above

Field `Y` *obscures* member class `Y` (JLS 6.3.2)
The rule: variable > type > package

Another Look

```
public class Gray {
    public static void main(String[] args){
        System.out.println(X.Y.Z);
    }
}

class X {
    static class Y {
        static String Z = "Black";
    }
    static C Y = new C();
}

class C {
    String Z = "White";
}
```

The rule: variable > type > package

How Do You Fix It?

```
public class Gray {
    public static void main(String[] args){
        System.out.println(Ex.Why.z);
    }
}

class Ex {
    static class Why {
        static String z = "Black";
    }
    static See y = new See();
}

class See {
    String z = "White";
}
```

The Moral

- Obey naming conventions
  - **`field, method(), Class, CONSTANT`**
  - Single-letter uppercase names reserved for *type variables* (new in J2SE 1.5)
- Avoid name reuse, except overriding
  - Overloading, shadowing, hiding, obscuring

6. “It’s Elementary”

```
public class Elementary {
    public static void main(String[] args) {
        System.out.println(54321 + 54321);
    }
}
```

What Does It Print?

(a) `-22430`

(b) `59753`

(c) `10864`

(d) `108642`

What Does It Print?

(a) `-22430`

(b) `59753`

(c) `10864`

(d) `108642`

Program doesn't say what you think it does!

Another Look

```
public class Elementary {
    public static void main(String[] args) {
        System.out.println(5432l + 5432l);
    }
}
```

l - the numeral one

l - the lowercase letter el

How Do You Fix It?

We won't insult your intelligence

The Moral

- **Always** use uppercase el (`L`) for long literals
  - Lowercase el makes the code unreadable
  - `5432L` is clearly a long, `5432l` is misleading
- Never use lowercase el as a variable name
  - Not this: `List l = new ArrayList();`
  - But this: `List list = new ArrayList();`

7. “Down For The Count”

```
public class Count {
    public static void main(String[] args) {
        final int START = 2000000000;
        int count = 0;
        for (float f = START; f < START + 50; f++)
            count++;
        System.out.println(count);
    }
}
```

What Does It Print?

(a) `0`

(b) `50`

(c) `51`

(d) None of the above

What Does It Print?

(a) `0`

(b) `50`

(c) `51`

(d) None of the above

The termination test misbehaves due to floating point “granularity”

Another Look

```
public class Count {
    public static void main(String[] args) {
        final int START = 2000000000;
        int count = 0;
        for (float f = START; f < START + 50; f++)
            count++;
        System.out.println(count);
    }
}

// (float) START == (float) (START + 50)
```

How Do You Fix It?

```
public class Count {
    public static void main(String[] args) {
        final int START = 2000000000;
        int count = 0;
        for (int f = START; f < START + 50; f++)
            count++;
        System.out.println(count);
    }
}
```

The Moral

- Don’t use floating point for loop indices
- Not every **`int`** can be expressed as a **`float`**
- Not every **`long`** can be expressed as a **`double`**
- If you must use floating point, use **`double`**
  - unless you’re certain that **`float`** provides enough precision *and* you have a compelling performance need (space or time)

8. “Line Printer”

```
public class LinePrinter {
   public static void main(String[] args) {
      // Note: \u000A is Unicode representation for newline
      char c = 0x000A;
      System.out.println(c);
   }
}
```

What Does It Print?

(a) Two blank lines

(b) `10`

(c) Won't compile

(d) It varies

What Does It Print?

(a) Two blank lines

(b) `10`

(c) Won't compile: Syntax error!

(d) It varies

Unicode escape breaks comment in two

Another Look

```
// Unicode escapes are processed before comments!
public class LinePrinter {
   public static void main(String[] args) {
      // Note: \u000A is unicode representation for newline
      char c = 0x000A;
      System.out.println(c);
   }
}

// This is what the parser sees
public class LinePrinter {
   public static void main(String[] args) {
      // Note:
is Unicode representation for newline
      char c = 0x000A;
      System.out.println(c);
   }
}
```

How Do You Fix It?

```
public class LinePrinter {
   public static void main(String[] args) {
      // Escape sequences (like \n) are fine in comments
      char c = '\n';
      System.out.println(c);
   }
}
```

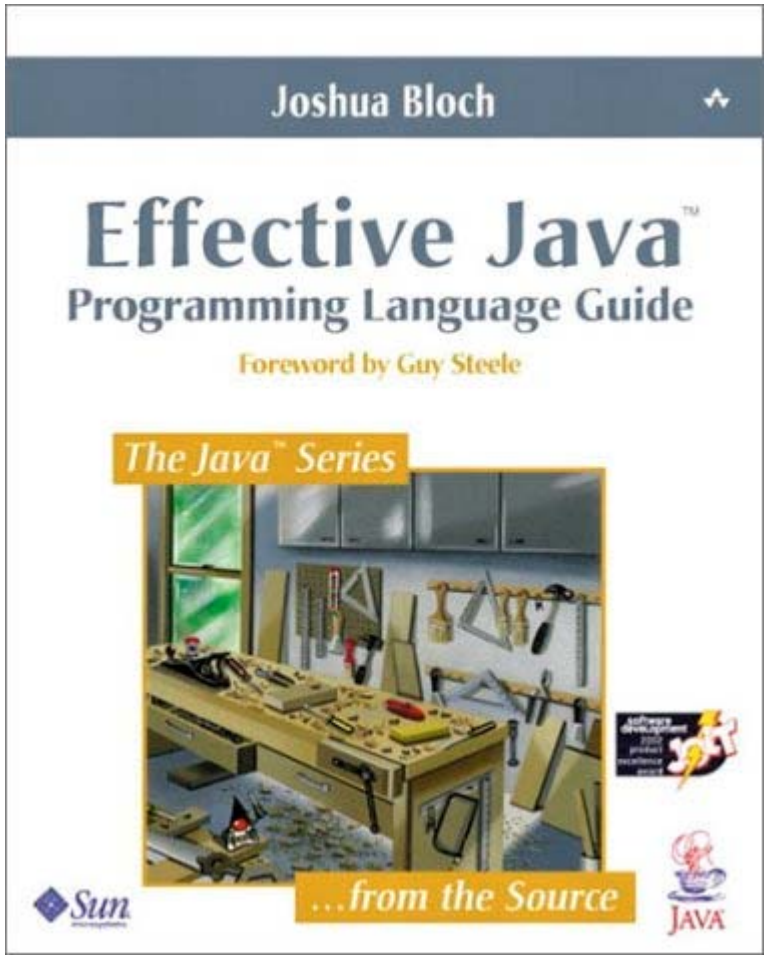
The Moral

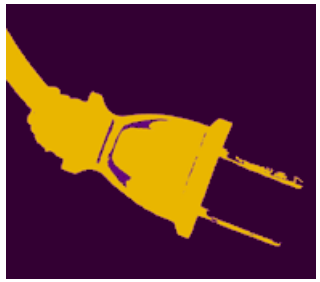
- **Unicode escapes are dangerous**
  - Equivalent to the character they represent!
- Use escape sequences instead, if possible
- If you must use Unicode escapes, use with care
  - **`\u000A`** (newline) can break string literals, char literals, and single-line comments
  - **`\u0022`** (`"`) can terminate string literals
  - **`\u0027`** (`'`) can terminate character literals

Conclusion

- Java platform is simple and elegant
  - But it has a few sharp corners—avoid them!
- Keep programs clear and simple
- If you aren't sure what a program does, it probably doesn't do what you want
- Don't code like my brother

Shameless Commerce Division





Send Us Your Puzzlers!

- If you have a puzzler for us, send it to

  javapuzzlers@gmail.com

Still More Java™ Puzzlers

Don’t code like my brother.

Joshua Bloch
Neal Gafter

Google™

Java™

Sun microsystems