Java™ Puzzlers, Episode VI: The PhantomReference Menace. Attack of the Clone. Revenge of the Shift.

Joshua Bloch
Chief Java Architect
Google Inc.

William Pugh
Professor
University of Maryland

Session TS-2707

A long time ago
in a convention center
not very far away,
two space cadets took
laser pointers in hand...

Introduction

- Eight more Java™ platform puzzles
  - Short program with curious behavior
  - What does it print? (multiple choice)
  - The mystery revealed
  - How to fix the problem
  - The moral
- Covers language, core libraries, and more

1. “The Joy of Sets”

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1);
        }
        System.out.println(s.size());
    }
}
```

What Does It Print?

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1);
        }
        System.out.println(s.size());
    }
}
```

(a) `1`
(b) `100`
(c) Throws exception
(d) None of the above

What Does It Print?

(a) `1`

(b) `100`

(c) Throws exception

(d) None of the above

The set contains `Short` values, but we're removing `Integer` values

Another Look

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1); // int-valued expression
        }
        System.out.println(s.size());
    }
}
```

Another ‘nother Look

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for (short i = 0; i < 100; i++) {
            s.add(i);
            s.remove(i - 1); // int-valued expression
        }
        System.out.println(s.size());
    }
}

public interface Set<E>extends Collection<E> {
    public abstract boolean add(E e);
    public abstract boolean remove(Object o);
    ...
}
```

How Do You Fix It?

```
public class ShortSet {
    public static void main(String args[]) {
        Set<Short> s = new HashSet<Short>();
        for(short i = 0; i < 100; i++) {
            s.add(i);
            s.remove((short) (i - 1));
        }
        System.out.println(s.size());
    }
}
```

Moral

- **`Collection<E>.remove`** takes **`Object`**, not **`E`**
    - Also **`Collection.contains`**, **`Map.get`**
- Integral arithmetic always results in **`int`** or **`long`**
- Avoid mixing types
- Avoid **`short`**; prefer **`int`** and **`long`**
    - Arrays of **`short`** are the only compelling use case

2. “More Joy of Sets”

```
import java.net.*;
public class UrlSet {
    private static final String[] URL_NAMES = {
        "http://javapuzzlers.com",
        "http://apache2-snort.skybar.dreamhost.com",
        "http://www.google.com",
        "http://javapuzzlers.com",
        "http://findbugs.sourceforge.net",
        "http://www.cs.umd.edu"
    };
    public static void main(String[] args)
            throws MalformedURLException {
        Set<URL> favorites = new HashSet<URL>();
        for (String urlName : URL_NAMES)
            favorites.add(new URL(urlName));
        System.out.println(favorites.size());
    }
}
```

JavaOne

What Does It Print?

(a) 4

(b) 5

(c) 6

(d) None of the above

```
import java.net.*;
public class UrlSet {
    private static final String[] URL_NAMES = {
        "http://javapuzzlers.com",
        "http://apache2-snort.skybar.dreamhost.com",
        "http://www.google.com",
        "http://javapuzzlers.com",
        "http://findbugs.sourceforge.net",
        "http://www.cs.umd.edu"
    };
    public static void main(String[] args)
            throws MalformedURLException {
        Set<URL> favorites = new HashSet<URL>();
        for (String urlName : URL_NAMES)
            favorites.add(new URL(urlName));
        System.out.println(favorites.size());
    }
}
```

What Does It Print?

(a) 4 (typically, assuming you’re on the net)

(b) 5

(c) 6

(d) None of the above – it varies from run to run

`URL`’s `equals` and `hashCode` are broken

Another Look (1)

```
import java.net.*;
public class UrlSet {
    private static final String[] URL_NAMES = {
        "http://javapuzzlers.com",
        "http://apache2-snort.skybar.dreamhost.com",
        "http://www.google.com",
        "http://javapuzzlers.com",
        "http://findbugs.sourceforge.net",
        "http://www.cs.umd.edu"
    };
    public static void main(String[] args)
            throws MalformedURLException {
        Set<URL> favorites = new HashSet<URL>();
        for (String urlName : URL_NAMES)
            favorites.add(new URL(urlName));
        System.out.println(favorites.size());
    }
}
```

Same IP Address

Identical

Another Look (2) – `URL` Documentation

Two URL objects are equal if they have the same protocol, reference equivalent hosts, have the same port number on the host, and the same file and fragment of the file.

Two hosts are considered equivalent if both host names can be resolved into the same IP addresses; else if either host name can't be resolved, the host names must be equal without regard to case; or both host names equal to null.

Since hosts comparison requires name resolution, this operation is a blocking operation.

How Do You Fix It?

```
import java.net.*;
public class UriSet {
    private static final String[] URI_NAMES = {
        "http://javapuzzlers.com",
        "http://apache2-snort.skybar.dreamhost.com",
        "http://www.google.com",
        "http://javapuzzlers.com",
        "http://findbugs.sourceforge.net",
        "http://www.cs.umd.edu"
    };
    public static void main(String[] args)
            throws URISyntaxException {
        Set<URI> favorites = new HashSet<URI>();
        for (String uriName : URI_NAMES)
            favorites.add(new URI(uriName));
        System.out.println(favorites.size());
    }
}
```

Moral

- Do not use `URL` as a `Set` element or `Map` key
  - `equals` and `hashCode` aren’t well defined
  - They do not obey their general contracts!
- Use `URI` instead
  - Make `URL` from `URI` as necessary
- `equals` should not depend on environment

3. “Racy Little Number”

```
public class Test extends junit.framework.TestCase {
    int number;

    public void test() throws InterruptedException {
        number = 0;
        Thread t = new Thread(new Runnable() {
            public void run() {
                assertEquals(2, number);
            }
        });
        number = 1;
        t.start();
        number++;
        t.join();
    }
}
```

How Often Does This Test Pass?

```
public class Test extends junit.framework.TestCase {
    int number;

    public void test() throws InterruptedException {
        number = 0;
        Thread t = new Thread(new Runnable() {
            public void run() {
                assertEquals(2, number);
            }
        });
        number = 1;
        t.start();
        number++;
        t.join();
    }
}
```

(a) It always fails

(b) It sometimes passes

(c) It always passes

(d) It always hangs

How Often Does This Test Pass?

(a) It always fails

(b) It sometimes passes

(c) It always passes – but it tells us nothing

(d) It always hangs

JUnit doesn’t get a chance to see whether assertion succeeds or fails

Another Look

```
public class Test extends junit.framework.TestCase {
    int number;

    public void test() throws InterruptedException {
        number = 0;
        Thread t = new Thread(new Runnable() {
            public void run() {
                // JUnit never sees any thrown exception
               assertEquals(2, number);
            }
        });
        number = 1;
        t.start();
        number++;
        t.join();
    }
}
```

How Do You Fix It? (1)

```
volatile Exception exception;

volatile Error error;

// Triggers test case failure if any thread asserts failed
public void tearDown() throws Exception {
    if (error != null)
        throw error;
    if (exception != null)
        throw exception;
}
```

How Do You Fix It? (2)

```
Thread t = new Thread(new Runnable() {
    public void run() {
        try {
            assertEquals(2, number);
        } catch(Error e) {
            error = e;
        } catch(Exception e) {
            exception = e;
        }
    }
});
```

Moral

- JUnit does not support concurrency
- You must provide your own
  - If you don’t, you’ll get a false sense of security
- Also see TS-2220 Testing Concurrent Software
  - Describes new framework to better handle this situation
  - Thursday, 1:30 PM–2:30 PM

4. “Elvis Lives Again”

```
public class Elvis {
    // Singleton pattern: there's only one Elvis
    public static final Elvis ELVIS = new Elvis();
    private Elvis() { }

    private static final Boolean LIVING = true;

    private final Boolean alive = LIVING;

    public final Boolean lives() { return alive; }

    public static void main(String[] args) {
        System.out.println(ELVIS.lives() ?
            "Hound Dog" : "Heartbreak Hotel");
    }
}
```

What Does It Print?

(a) `Hound Dog`

(b) `Heartbreak Hotel`

(c) It varies

(d) None of the above

```
public class Elvis {
    // Singleton pattern: there's only one Elvis
    public static final Elvis ELVIS = new Elvis();
    private Elvis() { }

    private static final Boolean LIVING = true;

    private final Boolean alive = LIVING;

    public final Boolean lives() { return alive; }

    public static void main(String[] args) {
        System.out.println(ELVIS.lives() ?
            "Hound Dog" : "Heartbreak Hotel");
    }
}
```

What Does It Print?

(a) Hound Dog
(b) Heartbreak Hotel
(c) It varies
(d) None of the above – throws `NullPointerException`

Class initialization is tricky, and auto-unboxing happens where you least expect it

Another Look

```
public class Elvis {
    // Recursive class initialization
    public static final Elvis ELVIS = new Elvis();
    private Elvis() { }

    private static final Boolean LIVING = true; // Too late

    private final Boolean alive = LIVING;

    public final Boolean lives() { return alive; }

    public static void main(String[] args) {
        System.out.println(ELVIS.lives() ? // Auto-
  unboxing!
            "Hound Dog" : "Heartbreak Hotel");
    }
}
```

How Do You Fix It?

```java
public class Elvis {
    private Elvis() { }

    private static final Boolean LIVING = true;

    // Create singleton *after* initializing other fields
    public static final Elvis ELVIS = new Elvis();

    private final Boolean alive = LIVING;

    public final Boolean lives() { return alive; }

    public static void main(String[] args) {
        System.out.println(ELVIS.lives() ?
            "Hound Dog" : "Heartbreak Hotel");
    }
}
```

Moral

- Wrapped primitives aren’t primitives
    - They aren’t compile-time constants, either
- Auto-unboxing can occur when you least expect it
    - It can cause **`NullPointerException`**
- Never use **`Boolean`** as a three-valued return type
    - Almost guarantees **`NullPointerException`**
- Watch out for circularities in class initialization
    - Construct instances at end of class initialization

5. “Mind the Gap”

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE);
        int last = in.read();
        System.out.println(first + last);
    }
}
```

What Does It Print?

(a) 1

(b) 3

(c) Throws exception

(d) It varies

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE);
        int last = in.read();
        System.out.println(first + last);
    }
}
```

What Does It Print?

(a) 1 (in practice)

(b) 3

(c) Throws exception

(d) It varies from run to run (according to spec)

`skip` returns a value; ignore it at your peril. Also it is difficult to use correctly.

Another Look

```
import java.io.*;
public class Gap {
    private static final int GAP_SIZE = 10 * 1024;
    public static void main(String args[]) throws IOException {
        File tmp = File.createTempFile("gap", ".txt");
        FileOutputStream out = new FileOutputStream(tmp);
        out.write(1);
        out.write(new byte[GAP_SIZE]);
        out.write(2);
        out.close();
        InputStream in =
            new BufferedInputStream(new FileInputStream(tmp));
        int first = in.read();
        in.skip(GAP_SIZE); // Not guaranteed to skip entire gap
        int last = in.read();
        System.out.println(first + last);
    }
}
```

How Do You Fix It?

```
static void skipFully(InputStream in, long nBytes)
        throws IOException {
    long remaining = nBytes;
    while (remaining != 0) {
        long skipped = in.skip(remaining);
        if (skipped == 0)
            throw new EOFException();
        remaining -= skipped;
    }
}
```

Moral

- The **`skip`** method is hard to use and error prone
- Use your **`skipFully`** instead of skip
  - There is an RFE to add it to **`InputStream`**
- More generally, if an API is broken, wrap it
- For API designers
  - Don't violate the principle of least astonishment
  - Make it easy to do simple things

6. “Histogram Mystery”

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```

What Does It Print?

(a) `83`
(b) `C16`
(c) `S`
(d) None of the above

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```

What Does It Print?

(a) `83`

(b) `C16`

(c) `S`

(d) None of the above – throws
`ArrayOutOfBoundsException`

`Math.abs(int)` can return a negative number, and so can the `%` operator

Another Look

```
public class Histogram {
    private static final String[] words = // Carefully chosen!
      { "I", "recommend", "polygene", "lubricants" };
    // "polygenelubricants".hashCode() == Integer.MIN_VALUE
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words) {
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode())
                   % histogram.length;
                histogram[bucket]++;
            }
        }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```

How Do You Fix It?

```
public class Histogram {
    private static final String[] words =
      { "I", "recommend", "polygene", "lubricants" };
    public static void main(String[] args) {
        int[] histogram = new int[5];
        for (String word1 : words)
            for (String word2 : words) {
                String pair = word1 + word2;
                int bucket = Math.abs(pair.hashCode()
                   % histogram.length); // Math.abs follows %
                histogram[bucket]++;
            }
        int pairCount = 0;
        for (int freq : histogram)
            pairCount += freq;
        System.out.println('C' + pairCount);
    }
}
```

Moral

- `Math.abs` doesn't guarantee nonnegative result
  - `Integer.MIN_VALUE == -Integer.MIN_VALUE`
- The `%` operator is remainder, not mod; can be negative
- To translate a signed hash value to a bucket
  - `Math.abs(hashVal % buckets.length)`
  - Or `(hashVal >>> 1) % buckets.length`
  - Or `(hashVal & 0x7fffffff) % buckets.length`
  - Or use power-of-two length array `(hashVal & (buckets.length - 1))`

7. “A Sea of Troubles”

```
public class Hamlet {
    public static void main(String[] args) {
        Random rnd = new Random();
        boolean toBe = rnd.nextBoolean();
        Number result = (toBe || !toBe) ?
            new Integer(3) : new Float(1);
        System.out.println(result);
    }
}
```

What Does It Print?

```
public class Hamlet {
    public static void main(String[] args) {
        Random rnd = new Random();
        boolean toBe = rnd.nextBoolean();
        Number result = (toBe || !toBe) ?
            new Integer(3) : new Float(1);
        System.out.println(result);
    }
}
```

(a) `3`

(b) `1.0`

(c) Throws exception

(d) None of the above

What Does It Print?

(a) `3`

(b) `1.0`

(c) Throws an exception

(d) None of the above: `3.0`

The `? :` operator has strange behavior when applied to mismatched integral wrapper types

Another Look

```
public class Hamlet {
    public static void main(String[] args) {
        Random rnd = new Random();
        boolean toBe = rnd.nextBoolean();
        Number result = (toBe || !toBe) ?
            new Integer(3) : new Float(1);
        System.out.println(result);
    }
}
```

Another Look (2) – `?:` Spec (JLS 15.25)

The type of a conditional expression is determined as follows:

- If the second and third operands have the same type (which may be the null type), then that is the type of the conditional expression.
- If one of the second and third operands is of type boolean and the type of the other is of type Boolean, then the type of the conditional expression is boolean.
- If one of the second and third operands is of the null type and the type of the other is a reference type, then the type of the conditional expression is that reference type.
- **Otherwise, if the second and third operands have types that are convertible (§5.1.8) to numeric types, then there are several cases:**
  * If one of the operands is of type byte or Byte and the other is of type short or Short, then the type of the conditional expression is short.
  * If one of the operands is of type T where T is byte, short, or char, and the other operand is a constant expression of type int whose value is representable in type T, then the type of the conditional expression is T.
  * If one of the operands is of type Byte and the other operand is a constant expression of type int whose value is representable in type byte, then the type of the conditional expression is byte.
  * If one of the operands is of type Short and the other operand is a constant expression of type int whose value is representable in type short, then the type of the conditional expression is short.
  * If one of the operands is of type Character and the other operand is a constant expression of type int whose value is representable in type char, then the type of the conditional expression is char.
- **Otherwise, binary numeric promotion (§5.6.2) is applied to the operand types, and the type of the conditional expression is the promoted type of the second and third operands. Note that binary numeric promotion performs unboxing conversion (§5.1.8) and value set conversion (§5.1.13).**
- Otherwise, the second and third operands are of types S1 and S2 respectively. Let T1 be the type that results from applying boxing conversion to S1, and let T2 be the type that results from applying boxing conversion to S2. The type of the conditional expression is the result of applying capture conversion (§5.1.10) to lub(T1, T2) (§15.12.2.7).

How Do You Fix It?

```
public class Hamlet {
    public static void main(String[] args) {
        Random rnd = new Random();
        boolean toBe = rnd.nextBoolean();
        Number result;
        if (toBe || !toBe) {
            result = new Integer(3);
        } else {
            result = new Float(1);
        }
        System.out.println(result);
    }
}
```

Moral

- Avoid mixing types
- The `? :` operator has counterintuitive semantics when used with two different wrapper types
- If you must select between two wrapped integral types, use `if-else` instead of `? :`

8. “Ground Round”

```
public class Round {
    public static void main(String[] args) {
        Random rnd = new Random();
        int i = rnd.nextInt();
        if (Math.round(i) != i)
            System.out.println("Ground Round");
    }
}
```

How Often Does It Print `Ground Round`?

```
public class Round {
    public static void main(String[] args) {
        Random rnd = new Random();
        int i = rnd.nextInt();
        if (Math.round(i) != i)
            System.out.println("Ground Round");
    }
}
```

(a) Never
(b) Seldom
(c) Almost every time it’s run
(d) Every time it’s run

How Often Does It Print `Ground Round`?

(a) Never

(b) Seldom

(c) Almost every time it’s run – 96.7% of the time!

(d) Every time it’s run

Silent, lossy conversion from **`int`** to **`float`** in combination with **`Math.round(float)`** is killing us.

Another Look

```
public class Round {
    public static void main(String[] args) {
        Random rnd = new Random();
        int i = rnd.nextInt();
        if (Math.round(i) != i) // i "promoted" to float
            System.out.println("Ground Round");
    }
}
```

```
public static int round(float); // This one gets invoked
public static long round(double);
```

`float` has 8 exponent bits. You lose about one bit of precision for each one!

How Do You Fix It?

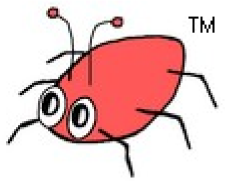
```
public class Round {
    public static void main(String[] args) {
        Random rnd = new Random();
        int i = rnd.nextInt();
        if (Math.round((double) i) != i)
            System.out.println("Ground Round");
    }
}
```

Moral

- **Silent "widening" conversion from `int` to `float` is lossy and dangerous**
  - Ditto for `long` to `double`
- The `float` type is seldom called for: use `double`
- Method overloading is dangerous, particularly when combined with unexpected "widening"

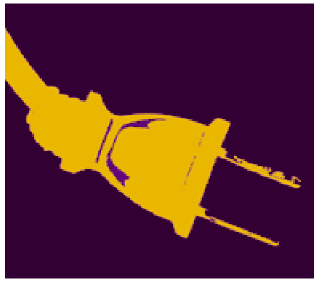
Conclusion

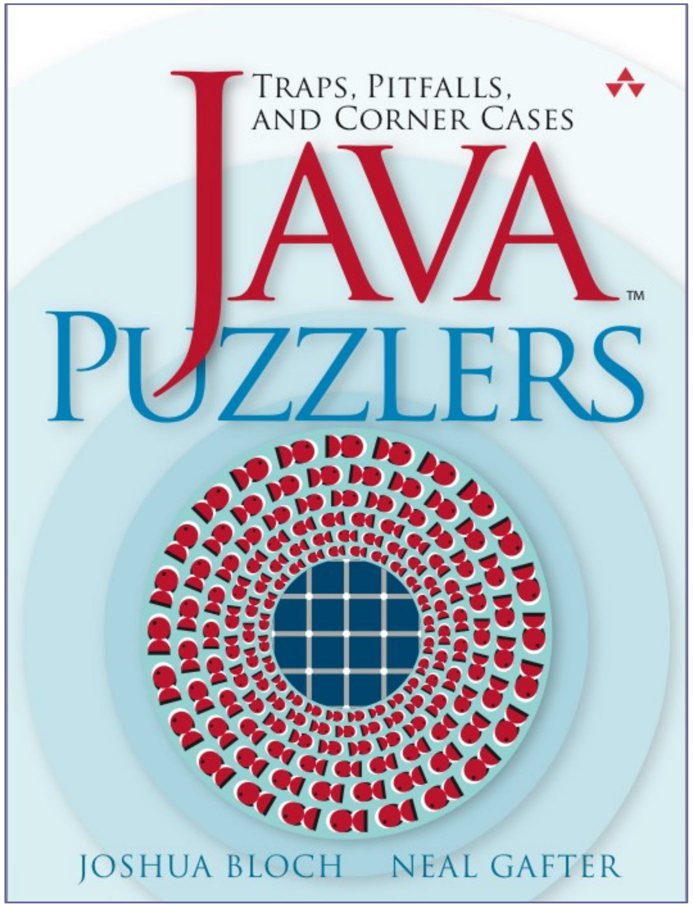
- Java platform is reasonably simple and elegant
  - But it has a few sharp corners—avoid them!
- Keep programs clear and simple
- If you aren't sure what a program does, it probably doesn't do what you want
- Use FindBugs™; it finds all 8 bugs in this talk!
- Don't code like my brother



Shameless Commerce Division

- 95 puzzles
- 52 illusions
- Tons of fun





Java™ Puzzlers, Episode VI: The PhantomReference Menace. Attack of the Clone. Revenge of the Shift.

Joshua Bloch
Chief Java Architect
Google Inc.

William Pugh
Professor
University of Maryland

Session TS-2707