Yo no he venido a hablar de mi libro

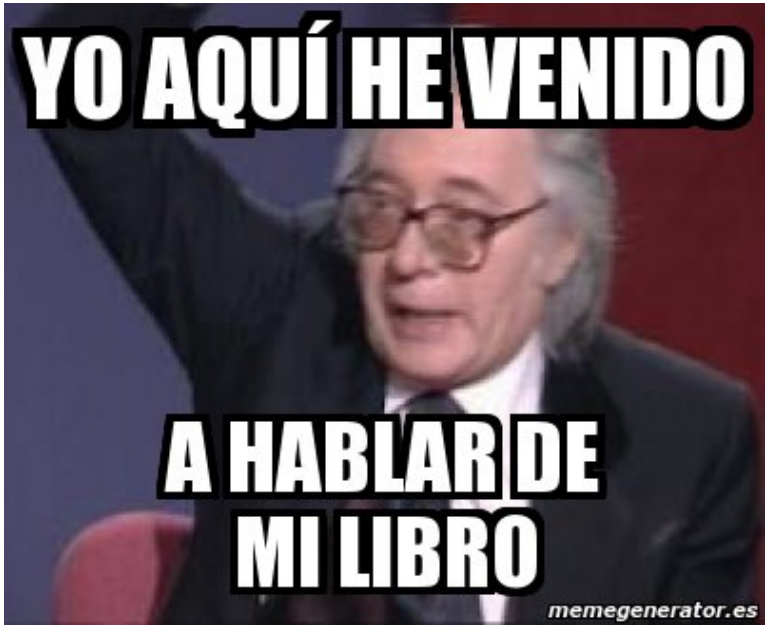
Víctor Rampérez Martín

ACM Lightning Talks 2021

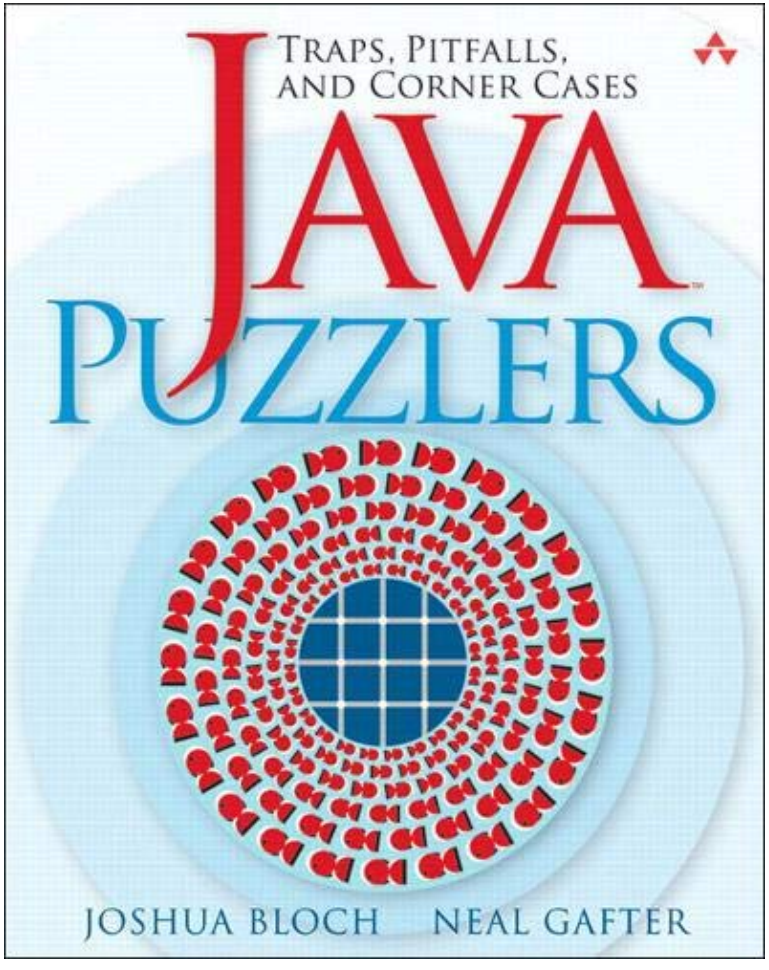
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

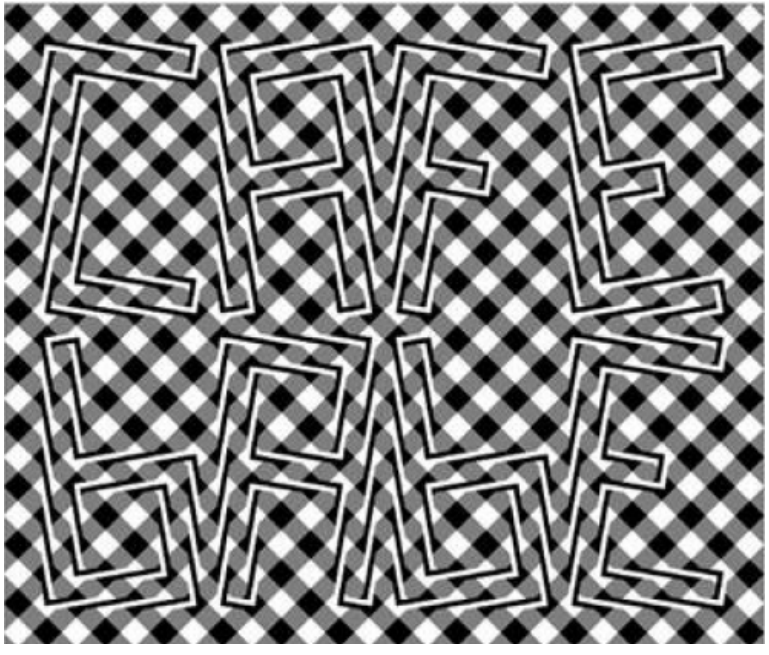
April 20, 2021

Yo no he venido a hablar de mi libro...



... yo he venido a hablar del libro(s) de otro(s)





DISCLAIMER!

- Todos los ejemplos aquí presentados se han extraído de conferencias de Joshua Bloch & Neal Gafter
- Todo el código mostrado en esta presentación compila correctamente

1. “Let’s start with a hello world”

```
public class HelloWorld {

  public static void main(String[] args) {

    "Hello World!".chars().forEach(System.out::print);
  }
}
```

1. Solution

- a) `Hello World!`
- b) Lanza una excepción
- c) **Ninguna de las anteriores:** **`72101108108111328711111410810033`**

1. “Let’s start with a hello world”

```
public IntStream chars()
```

Returns a stream of `int` zero-extending the `char` values from this sequence ...

2. “Life’s Persistent Questions”

```
public class SimpleQuestion {

  static boolean yesOrNo(String s) {
    s = s.toLowerCase();
    if (s.equals("yes") || s.equals("y") || s.equals("t"))
      s = "true";
    return Boolean.getBoolean(s);
  }

  public static void main(String[] args) {
    System.out.println(
      yesOrNo("true") + " " + yesOrNo("YeS"));
  }
}
```

2. Solution

- a) **`false false`**
- b) `true false`
- c) `true true`
- d) `ninguna de las anteriores`

The `Boolean.getBoolean` method does not do what you think it does

2. “Life’s Persistent Questions”

```
public static boolean getBoolean(String name)
```

The **java.lang.Boolean.getBoolean(String name)** returns true if and only if the system property named by the argument exists and is equal to the string “true”. A system property is accessible through getProperty, a method defined by the System class.
If there is no property with the specified name, or if the specified name is empty or null, then false is returned.

2. Possible Solution

```
public class SimpleQuestion {

  static boolean yesOrNo(String s) {
    s = s.toLowerCase();
    if (s.equals("yes") || s.equals("y") || s.equals("t"))
      s = "true";
    return Boolean.parseBoolean(s);
  }

  public static void main(String[] args) {
    System.out.println(yesOrNo("true") + " " + yesOrNo("YeS"));
  }
}
```

2. But this is even better...

```
static boolean yesOrNo(String s) {
  s = s.toLowerCase();
  return s.equals("yes") || s.equals("y") || s.equals("t") ||
       s.equals("true");
}

public static void main(String[] args) {
  System.out.println(yesOrNo("true") + " " + yesOrNo("YeS"));
}
```

1 & 2 Moral

- Strange and terrible methods lurk in libraries
  - Some have innocuous sounding names
- If your code misbehaves
  - Make sure you're calling the right methods
  - **Read the library documentation**
- Don't use similar names for wildly different behaviors (API designers)

3. “Searching for the one... and 130”

```
public class Searching {

  public static void main(String[] args) {
    String[] strings = {"0", "1", "2", "3", "129", "130"};

    // Translate String array into List of Integer
    List<Integer> integers = new ArrayList<Integer>();
    for (String s : strings) {
      integers.add(Integer.valueOf(s));
    }

    System.out.println(
      Collections.binarySearch(integers, 1, cmp) + ", "
      + Collections.binarySearch(integers, 130, cmp));
  }

  static Comparator<Integer> cmp = new Comparator<Integer>() {
    public int compare(Integer o1, Integer o2) {
      return o1 < o2 ? -1 : (o1 == o2 ? 0 : 1);
    }
  };
}
```

3. Solution

- a) `1, 5`
- b) **`1, -6 (en la práctica)`**
- c) `0, 0`
- d) **`ninguna de las anteriores: sin especificar (en teoría)`**

The Comparator is broken; autoboxing is tricky

3. "Searching for the one... and 130"

- The comparator is broken; autoboxing is tricky

```
public static Integer valueOf(String s) throws NumberFormatException {
        return Integer.valueOf(parseInt(s, 10));
}
```

```
public static Integer valueOf(int i)
```

Returns an Integer instance representing the specified int value. If a new Integer instance is not required, this method should generally be used in preference to the constructor Integer(int), as this method is likely to yield significantly better space and time performance by caching frequently requested values. This method will always cache values in the range -128 to 127, inclusive, and may cache other values outside of this range.

How do we fix it?

```
// Possible solution 1
static Comparator<Integer> cmp = new Comparator<Integer>() {

  public int compare(Integer o1, Integer o2) {
    return o1 < o2 ? -1 : (o1 > o2 ? 1 : 0);
  }
};
```

```
// Possible solution 2
static Comparator<Integer> cmp = new Comparator<Integer>() {

  public int compare(Integer o1, Integer o2) {
    return o1 < o2 ? -1 : (o1.equals(o2) ? 0 : 1);
  }
};
```

(Arguably) better

```
static Comparator<Integer> cmp = new Comparator<Integer>() {

  public int compare(Integer o1Boxed, Integer o2Boxed) {
    // Unbox arguments to force value comparison
    int o1 = o1Boxed;
    int o2 = o2Boxed;

    return o1 < o2 ? -1 : (o1 == o2 ? 0 : 1);
  }
};
```

Moral:

- Autoboxing blurs but does not erase distinction between primitives and boxed primitives
- Only four of the six comparison operators work on boxed primitives
  - $>, <, <=, >=$ work
  - $==, != $ do not work!
- It's very hard to test for broken comparators

4. “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());
  }
}
```

4. Solution

- a) `1`
- b) **`100`**
- c) `tira una excepción`
- d) `ninguna de las anteriores`

The set contains `Short`, but we are removing `Integer` values

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`

5. “More joy of Sets”

```
public class UrlSet {

  private static final String[] URL_NAMES = {
      "http://dssc.fi.upm.es/",
      "https://population.io/",
      "http://muss.fi.upm.es/",
      "http://www.catsthatlooklikehitler.com/",
      "https://apolloinrealtime.org/",
      "http://dssc.fi.upm.es/"};

  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());
  }
}
```

5. Solution

- a) 5
- b) **4 (normalmente, asumiendo que estás conectado a Internet)**
- c) 3
- d) **ninguna de las anteriores: varía de ejecución en ejecución**

`URL`'s `equals` y `hashCode` are broken

- El método `equals` y `hashCode` de la clase `URL` están rotos

```
public boolean equals(Object obj)
```

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.

5. Possible solution

```
public class UrlSet {

  private static final String[] URL_NAMES = {
      private static final String[] URL_NAMES = {
      "http://dssc.fi.upm.es/",
      "https://population.io/",
      "http://muss.fi.upm.es/",
      "http://www.catsthatlooklikehitler.com/",
      "https://apolloinrealtime.org/",
      "http://dssc.fi.upm.es/"};

  public static void main(String[] args) throws MalformedURLException,
       URISyntaxException {

    Set<URI> favorites = new HashSet<URI>();

    for (String urlName : URI_NAMES) {
      favorites.add(new URI(urlName));
    }

    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

6. “When words collide”

```
public class PrintWords {

  public static void main(String[] args) {

    System.out
        .println(Words.FIRST + " " + Words.SECOND + " " + Words.THIRD);
  }
}

public class Words { // Compile against this version
  public static final String FIRST = "the";
  public static final String SECOND = null;
  public static final String THIRD = "set";

}

public class Words { // Run against this version
  public static final String FIRST = "physics";
  public static final String SECOND = "chemistry";
  public static final String THIRD = "biology";
}
```

6. Solution

- a) `the null set`
- b) `physics chemistry biology`
- c) `tira una excepción`
- d) **`ninguna de las anteriores:  the chemistry set`**

Constant variables are inlined.

6. “When words collide”

- Constant variables are inlined
- Constant variables: final primitive or String variable, whose value is a compile-time constant (See JLS 4.12.4, 13.4.9, 15.28)
- A final String variable initialized to null is not a constant variable

```
public class Words {
  public static final String FIRST = "the";// Constant
  public static final String SECOND = null;// Not a constant
  public static final String THIRD = "set";// Constant

}
```

6. How to prevent constants from being inlined?

```
// Return its argument
private static String ident(String s) {
  return s;
}

// None of this fields are constant variables
public class Words {
  public static final String FIRST = ident("the");
  public static final String SECOND = ident(null);
  public static final String THIRD = ident("set");

}
```

Moral

- Constant variables are inlined (primitives and Strings, null is not a constant)
- Use constant variables only for entities whose value will never change

7. Is Schrodinger's cat alive?

```
public class SchrodingersCat {

  public static final SchrodingersCat CAT = new SchrodingersCat();
  private SchrodingersCat() { }

  private static final Boolean LIVING = true;

  private final Boolean alive = LIVING;

  private final Boolean lives() {
    return alive;
  }

  public static void main(String[] args) {
    System.out.println(CAT.lives() ? "of course" : "RIP");
  }
}
```

7. Solution

- a) `of course`
- b) `RIP`
- c) `Varía de ejecución en ejecución`
- d) **`ninguna de las anteriores: java.lang.NullPointerException`**

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

7. Another look

```
public class SchrodingersCat {

  // Recursive class initialization
  public static final SchrodingersCat CAT = new SchrodingersCat();
  private SchrodingersCat() { }

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

  private final Boolean alive = LIVING;

  private final Boolean lives() {
    return alive;
  }

  public static void main(String[] args) {
    System.out.println(CAT.lives() ? // Auto-unboxing
      "of course" : "RIP");
  }
}
```

7. Possible solution

```
public class SchrodingersCat {

  private static final Boolean LIVING = true;

  public static final SchrodingersCat CAT = new SchrodingersCat();
  private SchrodingersCat() { }

  private final Boolean alive = LIVING;

  private final Boolean lives() {
    return alive;
  }

  public static void main(String[] args) {
    System.out.println(CAT.lives() ? "of course" : "RIP");
  }
}
```

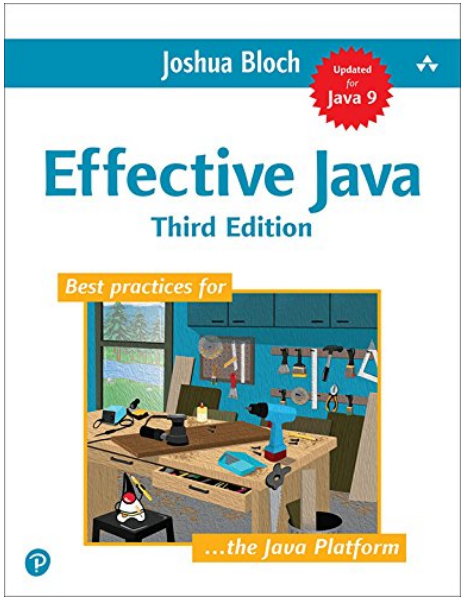
7. But this is even better

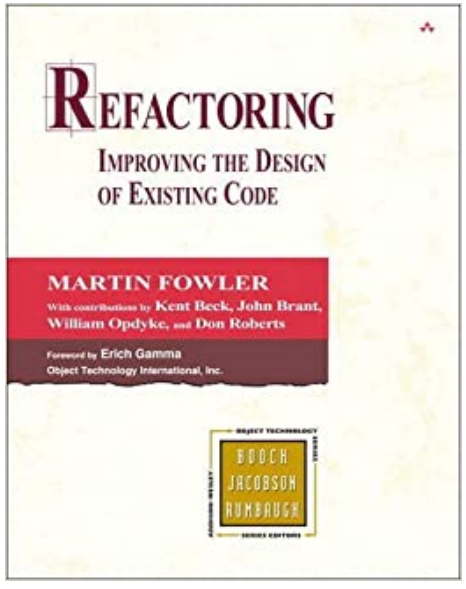
```
public class SchrodingersCat {

  private static final boolean LIVING = true;

  public static final SchrodingersCat CAT = new SchrodingersCat();
  private SchrodingersCat() { }

  private final boolean alive = LIVING;

  private final boolean lives() {
    return alive;
  }

  public static void main(String[] args) {
    System.out.println(CAT.lives() ? "of course" : "RIP");
  }
}
```

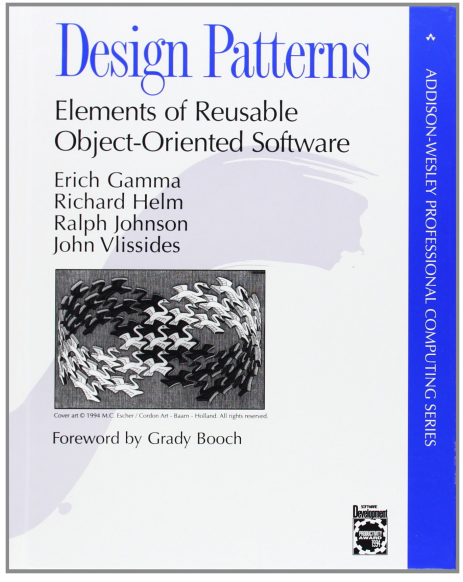
Moral

- Wrapped primitive aren't primitives
  - Prefer primitives to wrapped primitives
- 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

Bonus: más libros







Bonus: otros recursos

- Repo con código de los ejemplos y slides
  - `https://github.com/vramperez/java_puzzlers`
- Videos
  - `https://www.youtube.com/watch?v=V1vQf4qyMXg&ab_channel=UserGroupsatGoogle`
  - `https://www.youtube.com/watch?v=hSfylUXhpkA&t=1957s&ab_channel=DevoxxFR`
  - `https://www.youtube.com/watch?v=wEhu57pih5w&ab_channel=GoogleTechTalks`
  - `https://www.youtube.com/watch?v=wDN_EYUvUq0&ab_channel=GoogleTechTalks`
- Webs
  - `https://refactoring.guru/es`