Java Puzzlers No. 2

Luka Marasović

Java Puzzlers

- **What?**
  - Short programs in Java that appears to do one thing but actually does something else
- **Workflow?**
  - Puzzler walk-through
  - Voting on answer
  - Solution
  - Analysis and explanation
  - Insights

Code examples

- Java SE
- All puzzlers compile*
    - (imports and Java version)

Goal

- Have fun!

```java
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

```
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

What does it print?

a) truetrue
b) truefalse
c) falsetrue
d) falsefalse
e) Any of the above

```java
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

What does it print?

a) truetrue
b) truefalse
c) falsetrue
d) falsefalse (99.99%)
e) Any of the above

```
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

```
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

t.run() is not starting new thread

```
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

```
public class TreadLightly {

    public static void main(String[] args) {
        Thread t = new Thread() {
            public void run() {
                System.out.print(Boolean.getBoolean("true"));
            }
        };
        t.run();
        System.out.print(Boolean.getBoolean("false"));
    }
}
```

getBoolean(String name) - Returns true if and only if the system property named by the argument exists and is equal to the string "true"

Puzzler 1 insights

- If Thread didn't have a public run method, it would be impossible for programmers to invoke it accidentally
- Methods should have names that describe their primary functions
- API Designers: When in doubt, leave it out

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

What does it print?

a) 1
b) -1
c) Order of iteration not guaranteed
d) None of the above

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

What does it print?
a) 1
b) -1
c) Order of iteration not guaranteed
d) None of the above
***Java 6: 65535 Java 7:1**

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                              = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

EnumMap - represented internally as arrays, maintain natural order of their keys and are extremely compact and efficient

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                              = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

EnumMap - represented internally as arrays, maintain natural order of their keys and are extremely compact and efficient

*In Java 6 entry IS iterator

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

```
public class ItsOver9000 {

    public static void main(String...strings) {
        Map<Value, Integer> map
                = new EnumMap<Value, Integer>(Value.class);
        map.put(Value.ONE, 1);
        map.put(Value.MINUS_ONE, -1);

        Iterator<Map.Entry<Value, Integer>> iterator
                            = map.entrySet().iterator();

        Entry<Value, Integer> entry = iterator.next();
        iterator.next();

        System.out.println((int)(char)(int)entry.getValue());
    }
}
enum Value { ONE, MINUS_ONE }
```

Java uses two's complement arithmetic, -1 has all bits set

Sign extension is performed if the type of the original value is signed; zero extension if it is a char

Puzzler 2 insights

- API designers: Never make API worse to perform better
- Sign extension: If you can’t tell what a program does by looking at it, it probably doesn’t do what you want

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

What does it print?
a) 100100
b) 100d
c) d100
d) dd
e) None of the above

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

What does it print?

a) 100100
b) 100d
c) d100
d) dd
e) None of the above

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

IEEE 754 Standard: If either operand is NaN,
then the result of == is false but the result of != is true

```java
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

```
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

public static int abs(int a)

Note that if the argument is equal to the value of Integer.MIN_VALUE, the most negative representable int value, the result is that same value, which is negative

```
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

```
public class Asymmetric {

    public static void main(String[] args) {
        char x = 'd';
        int value = 100;
        System.out.print(Double.NaN == Double.NaN ? 100 : x);
        System.out.print(Math.abs(Integer.MIN_VALUE) > 0 ? value : x);
    }
}
```

Ternary conditional operator: Binary numeric promotion is applied to char if other operand is not constant expression representable in char

Puzzler 3 insights

- Numerical equality operators follow IEEE 754 standards
- Watch for asymmetry of two's-complement arithmetic
- Java integer arithmetic overflows silently
- Use the same type for the second and third operands in conditional expressions

```
public class MaximumEffort {

    public static void main(String...strings) {
        LegacyTypedList<Integer> list
                = new LegacyTypedList<Integer>();
        list.add((Integer)1);
        list.add((Object)1);

        System.out.print(list.typedList.size());
        System.out.println(list.objectList.size());
    }
}

class TypedList<E> {
     List<E> typedList = new ArrayList<E>();

     void add(E e) { typedList.add(e); }
}

class LegacyTypedList<E> extends TypedList<E> {
    List<Object> objectList = new ArrayList<Object>();

    void add(Object object) { objectList.add(object); }
}
```

```
public class MaximumEffort {

    public static void main(String...strings) {
        LegacyTypedList<Integer> list
                = new LegacyTypedList<Integer>();
        list.add((Integer)1);
        list.add((Object)1);

        System.out.print(list.typedList.size());
        System.out.println(list.objectList.size());
    }
}

class TypedList<E> {
     List<E> typedList = new ArrayList<E>();

     void add(E e) { typedList.add(e); }
}

class LegacyTypedList<E> extends TypedList<E> {
    List<Object> objectList = new ArrayList<Object>();

    void add(Object object) { objectList.add(object); }
}
```

What does it print?

a) 20
b) 11
c) 02
d) None of the above

```
public class MaximumEffort {

    public static void main(String...strings) {
        LegacyTypedList<Integer> list
                = new LegacyTypedList<Integer>();
        list.add((Integer)1);
        list.add((Object)1);

        System.out.print(list.typedList.size());
        System.out.println(list.objectList.size());
    }
}

class TypedList<E> {
     List<E> typedList = new ArrayList<E>();

     void add(E e) { typedList.add(e); }
}

class LegacyTypedList<E> extends TypedList<E> {
    List<Object> objectList = new ArrayList<Object>();

    void add(Object object) { objectList.add(object); }
}
```

What does it print?

a) 20

b) 11

c) 02

d) None of the above

```
public class MaximumEffort {

    public static void main(String...strings) {
        LegacyTypedList<Integer> list
                = new LegacyTypedList<Integer>();
        list.add((Integer)1);
        list.add((Object)1);

        System.out.print(list.typedList.size());
        System.out.println(list.objectList.size());
    }
}

class TypedList<E> {
     List<E> typedList = new ArrayList<E>();

    void add(E e) { typedList.add(e); }
}

class LegacyTypedList<E> extends TypedList<E> {
    List<Object> objectList = new ArrayList<Object>();

   void add(Object object) { objectList.add(object); }
}
```

```
public class MaximumEffort {

    public static void main(String...strings) {
        LegacyTypedList<Integer> list
                = new LegacyTypedList<Integer>();
        list.add((Integer)1);
        list.add((Object)1);

        System.out.print(list.typedList.size());
        System.out.println(list.objectList.size());
    }
}

class TypedList<E> {
     List<E> typedList = new ArrayList<E>();

     void add(E e) { typedList.add(e); }
}

class LegacyTypedList<E> extends TypedList<E> {
    List<Object> objectList = new ArrayList<Object>();

    void add(Object object) { objectList.add(object); }
}
```

Type erasure - all type parameters in generic types replaced with bounds or Object if type parameters are unbounded

Puzzler 4 insights

- API decision: Evolution not revolution - possibility of gradual migration to generics
- Configure compiler errors/warnings to avoid accidental override

```java
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

```java
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

What does it print?

a) 5
b) 8
c) 22
d) 32
e) None of the above

```java
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

What does it print?

a) 5
b) 8
c) 22
d) 32
e) None of the above

```
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

Finally block is always* executed when control leaves the try block

```java
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

```
public class DoubleRainbow {

    public static void main(String[] args) {
        int length = "Abcd\u0022.length() + \u0022efgh".length();
        System.out.println(selectValue(length));
    }

    public static int selectValue(int value) {
        try {
            return value / 4;
        } finally {
            return value;
        }
    }
}
```

Compiler translates Unicode escapes into the characters they represent before it parses the program into tokens

Hello world?

```
\u0070\u0075\u0062\u006C\u0069\u0063\u0020\u0063\u006C\u0061\u0073\u0073\u0020
\u0048\u0065\u006C\u006C\u006F\u0057\u006F\u0072\u006C\u0064\u0020\u007B
\u0070\u0075\u0062\u006C\u0069\u0063\u0020\u0073\u0074\u0061\u0074\u0069\u0063
\u0076\u006F\u0069\u0064\u0020\u006D\u0061\u0069\u006E\u0028
\u0053\u0074\u0072\u0069\u006E\u0067\u005B\u005D\u0020\u0061\u0072\u0067\u0073\u0029\u0020
\u007B\u0020\u0053\u0079\u0073\u0074\u0065\u006D\u002E\u006F\u0075\u0074\u002E
\u0070\u0072\u0069\u006E\u0074\u006C\u006E\u0028\u0022\u0048\u0065\u006C\u006C\u006F\u0022
\u002B\u0020\u0022\u0057\u006F\u0072\u006C\u0064\u0022\u0029\u003B\u007D\u007D
```

Puzzler 5 insights

- Don’t use Unicode escapes for ASCII characters

Thank you!