Quiz yourself: Initializing enums in Java code

JAVA SE

Quiz yourself: Initializing enums in Java code

There are rules about initializing enums. You need to know them.

by Mikalai Zaikin and Simon Roberts

April 12, 2021

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

This question investigates the initialization of enums, which are a data type that allows for a variable to be a set of predefined constants; the variable must be equal to one of the defined constants.

Given the following two Java types from a unit testing framework:

```
enum Outcome {
    SUCCESS, FAILURE;
    {System.out.print("IO ");}
    Outcome() {System.out.print("CO ");}
    static {System.out.print("SO ");}
}

class TestCase {
    Outcome result;
    public TestCase() {
        System.out.print("CTC ");
    }
    { result = Outcome.SUCCESS; System.out.pr
}
```

And this test case scenario:

```
TestCase tc1 = new TestCase();
TestCase tc2 = new TestCase();
```

What will be the system output? Choose one.

A. `IO CO SO CTC ITC CTC ITC` The answer is A.

B. `SO IO CO ITC CTC ITC CTC` The answer is B.

C. `ITC CTC ITC CTC SO IO CO` The answer is C.

D. `SO IO CO IO CO ITC CTC ITC CTC` The answer is D.

E. `IO CO IO CO SO ITC CTC ITC CTC` The answer is E.

Answer. When Java code calls a constructor, the following sequence happens:

1. If the target class has not yet been loaded, it will be loaded (this includes verification) and then initialized.
2. Memory for the instance will be allocated and zeroed.
3. Flow of control is passed to the `Object` class.
4. For each class from `Object` down to the class being instantiated
    a. Instance initialization is performed.
    b. A constructor is run, or perhaps constructors are run.

There are some elements of this that are not important to this question, such as why more than one constructor might be executed for a single class. This relates to the use of the `this(...)` construct in any given constructor.

What matters here is that, in general, a class must be initialized *before* instances are made. You will see shortly that enums are a special case (but note that programmer-created code can never call a constructor on an enum type).

The description so far actually leaves out some very important details, including what happens during *class* initialization and during *instance* initialization. Let's look at class initialization first.

Suppose you have the following class:

```
class X {
  static int x = 99;
  static { x ++; }
  static int y = x;
}
```

The static blocks and the initialized declarations of static members will execute in sequence from top to bottom. As a

result, the value of both `x` and `y` will be `100`.

A similar process occurs for the instance initialization of an object. Consider the following class:

```
public class X {
  int x = 10;
  {
    x++;
  }
  public X() {
    System.out.println("x is " + x);
  }
  {
    x++;
  }
}
```

In this situation, instantiating the class `X` will print `x is 12`. The code blocks that contain `x++` are both instance initializers, and they, along with instance initializations like the line `int x = 10;` all execute, in order from top to bottom of the class, before control passes to a constructor.

With that information, you can determine that even though the instance initializer block comes after the constructor in the `TestCase` class, the output caused by the construction of the two `TestCase` instances will be `ITC CTC ITC CTC` and this observation means option A is incorrect.

Next, you must determine what happens when the enum type `Outcome` is loaded and initialized. The first use of this class comes during the initialization of the first of the two `TestCase` objects, so of course the process precedes the output described above. The enum contains three messages: one from a static initializer, one from an instance initializer, and one from the constructor.

In view of the preceding discussion, it would be natural to expect that the static initializer message would be printed, followed by the instance initializer, then the constructor message, and then the last two would be repeated. However, this is not the case.

Java Language Specification section 8.9, "Enum types," explains that values of an enum are `public static final` fields in the enum class being declared. This creates something of a dilemma because to complete the static initialization, you need the instances that will represent the actual values of the enum.

Because of this dilemma, the two instances in this example (one for `SUCCESS` and one for `FAILURE`) are constructed and initialized and assigned to their respective `public static final` fields before the static initialization has been completed. More importantly, these two instances are fully created as the very first step in the static initialization process. Because of this, the two constants (`SUCCESS` and

`FAILURE`) must be placed at the top of the enum. So, the static initializer block must be placed *after* them, and it must execute after the initialization of the instances.

Consequently, the order of the messages in this part of the code will be

```
IO CO IO CO SO
```

This makes option E correct and options A, B, C, and D incorrect.

In fact, this behavior is not really specific to an enum, though the structure of an enum forces the situation. Consider the following class:

```
public class X {
  public static final X self = new X();
  static {
    System.out.println("Static Init");
  }
  {
    System.out.println("Instance Init");
  }
  public static void main(String[] args) {
    new X();
  }
}
```

The class `X` will exhibit the same behavior for the same reason. That is, when first instantiated, static initialization cannot proceed past the initialization of the variable `self` until an instance of `X` has been created. Consequently, the output of this code when it is run will be

```
Instance Init
Static Init
Instance Init
```

Although this concern can occur in any class that initializes static fields with instances of itself, an enum constructor is also prohibited from accessing `static` fields (unless they are `final` compile-time constants) in the *same* enum. This is because those static fields will not have been initialized at the point at which the constructor is called. Thus, in this enum declaration

```
public enum AnEnum {
  A, B;
  static int x = 100;
  private AnEnum() {
    System.out.println("x is " + x); // Fails
  }
}
```

The attempt to read the value of `x` in the constructor causes a compilation failure.

Conclusion: The correct answer is option E.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Share this Page