

About **a** f Ω in **≡**

Declarations

MAY 21, 2015

Class

- There can be **only one public class** per source code file.
- If there is a public class in a file, the name of the file must match the name of the public class.
- Files with no public classes can have a name that does not match any of the classes in the file.
- A file can have more than one nonpublic class.

Interface

- All **interface methods** are **implicitly public** and abstract. In other words, you do not need to actually type the public or abstract modifiers in the method declaration, but the method is still always public and abstract.
- All variables defined in an interface are implicitly public, static, and final, in other words, interfaces can declare only constants, not instance variables.

• Interface methods must not be static, final, strictfp, or native.

Some valid interface declarations:

```
public abstract interface Rollable { } // abstract is redundant as
                                       // interfaces are implicitly abstract
public interface Rollable { } // public modifier is required if you
                             // want the interface to have public
                              // rather than default access
interface Rollable { } // interface with default access
public interface Bounceable {
    // following initializations/declarations are all
    // legal and identical
    int b = 8;
    public static final int b = 8;
   public final int b = 8;
    static final int b = 8;
    final int b = 8;
    void bounce();
    public void bounce();
    abstract void bounce();
    public abstract void bounce();
    abstract public void bounce();
```

NOTE: The above points state that interfaces have very little flexibility in how the methods and variables defined in the interface are declared. This is also one of the **rare differences between an interface and an abstract class**.

Var-args

Use var-args when you want to pass **variable number of arguments** to a method.

Rules for Var-args:

Let's look at some legal and illegal var-arg declarations:

Enums

Java lets you restrict a variable to having one of only a few pre-defined values, in other words, one value from an **enumerated list**. (The items in the enumerated list are called, surprisingly, **enums**.)

It's not required that enum constants be in all caps, but it's a good idea. Enums can be declared as their own separate class, or as a class member, however they must not be declared within a method.

An enum outside a class:

NOTE: The preceding code can be part of a single file. (Remember, the file must be named CoffeeTest1.java because that's the name of the public class in the file.) The key point to remember is that an enum that isn't enclosed in a class can be declared with only the **public or default modifier**, just like a class (non-inner).

An enum inside a class:

NOTE: The syntax for accessing an enum's members depends on where the enum was declared.

Conceptually what are enums:

The most important thing to remember is that **enums are not Strings or ints**. Each of the enumerated CoffeeSize types are actually **instances of CoffeeSize**. In other words, BIG is of type CoffeeSize. Think of an enum as a kind of class, that looks something (but not exactly) like this:

NOTE: Each of the enumerated values, BIG, HUGE, and OVERWHELMING, are instances of type CoffeeSize. They're represented as static and final, which in the Java world are constants.

Declaring Constructors, Methods, and Variables in an enum

You can add constructors, instance variables, methods, and something really strange known as a "constant specific class body".

Why we need an enum constructor?

Imagine you want to know the actual size, in ounces, that map to each of the three CoffeeSize constants. For example, you want to know that BIG is 8 ounces, HUGE is 10 ounces, and OVERWHELMING is 16 ounces. You could make some kind of a lookup table, using some other data structure, but that would be a poor design and hard to maintain. The simplest way is to treat your enum values (BIG, HUGE, and OVERWHELMING), as objects that can each have their own instance variables. Then you can assign those values at the time the enums are initialized, by passing a value to the enum constructor.

What is "constant specific class body"?

Imagine this scenario: you want enums to have two methods—one for ounces and one for lid code (a String). Now imagine that most coffee sizes use the same lid code, "B", but the OVERWHELMING size uses type "A". You can define a getLidCode() method in the CoffeeSize enum that returns "B", but then you need a way to override it for OVERWHELMING. You don't want to do some hard-to-maintain if/then code in the getLidCode() method, so the best approach might be to somehow have the OVERWHELMING constant override the getLidCode() method.

The below code snippet describes two of the above preceding points:

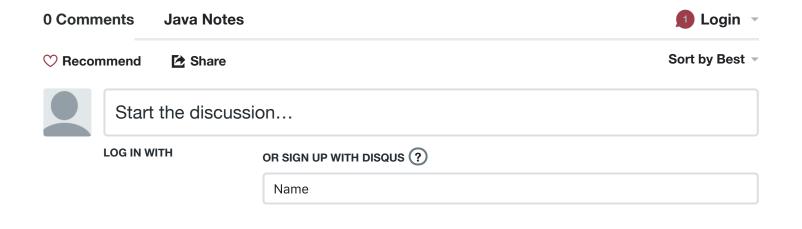
```
enum CoffeeSize {
    BIG(8),
    HUGE (10),
    OVERWHELMING(16) { // start a code block that defines
                       // the "body" for this constant
        public String getLidCode() {
                                       // override the method
                                        // defined in CoffeeSize
            return "A";
    }; // the semicolon is REQUIRED when more code follows
    CoffeeSize(int ounces) {
                                // constructor
       this.ounces = ounces;
    private int ounces;
    public int getOunces() {
        return ounces;
    public String getLidCode() {
                                   // this method is overridden
                                   // by the OVERWHELMING constant
       return "B"; // the default value we want to return for
                    // CoffeeSize constants
    }
```

Some points to note:

- You can NEVER invoke an enum constructor directly. The enum constructor is invoked automatically, with the arguments you define after the constant value.
- Every enum has a static method, values(), that returns an array of the enum's values in the order they're declared.

Variables and Literals « Previous Access Control Next »

Carefully curated by Ram swaroop. Powered by Jekyll with Type Theme.



Be the first to comment.

ALSO ON JAVA NOTES

Overloading | Java Concepts

1 comment • 3 years ago

Anagh Hegde — public class MyTest { public
static void main(String[] args) { MyTest test =
new MyTest(); int i = 9; test.TestOverLoad(i); } ...

Nested Classes | Java Notes

2 comments • 3 years ago

Amit Satpathy — Ram, it's really great to find your short and precise writeups on tech. Keep up the passion.

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy