L42

Friday, February 4, 2022 1:08 AM

Final fields, Final methods

Final fields

Q1. In Java, we can classify variables or references into 5 different types:

1. **Parameters** - these are the variables/references that appear in a method signature inside the parenthesis().
2. **Local variables/references** - the variables/references that are declared inside a method are called local variables. Their scope/visibility is restricted only to their enclosing method block.
3. **Instance fields** - these are the fields declared in the class. They hold the state information of the class. Every instance of the class holds acopy of these fields.
4. **Static fields** - these are the fields declared in the class with a static keyword. Only one copy of such fields exists in memory and are shared by (or available to) all the instances of that class.
5. **Constants** - these are the variables/references declared in the class with both static and final keywords. Only one copy of such fields exist in memory like the **static fields**. However, these variables/references cannot be reassigned another value once they have been assigned a value.

Note: If a variable is only declared as final and not static, then it is not called a constant. It is simply called a **final variable.**

Select all the correct statements from the below code:

```
public class A {
	private int value1; // statement 1
	private final int value2; // statement 2
	private static int value3; // statement 3
	private static final int value4; // statement 4
}
```

- [ ] `value1` declared in **statement 1** is called a `parameter`.
- [ ] Only one copy of `value2` declared in **statement 2** is shared by all instances of class `A`.
- [ ] `value3` declared in **statement 3** is called a `instance field`.
- [x] `value4` declared in **statement 4** is called a `constant`.

Q2. Imagine a variable or a reference to be a **cup** (something like a coffee cup).

We have **2 types** of **cups**. We will call the **cups** which are of type primitive as variables and the **cups** which are of type classes or arrays as references.

Meaning

int age; // ***age*** *primitive* String firstName; // ***firstName*** *reference* int[] marksArr; // ***marksArr*** *reference* String[] cityNamesArr; // ***cityNamesArr*** *reference*

Once we mark one of the cups as final, they can be **assigned a value only once.** For example:

```
final int age;
age = 20; // this is a valid statementage = 30; // compiler will give an error on this line stating the final variable age is already assigned a value
```

The same applies to cups which are of type references.

In case of references, the cup once assigned a reference cannot be assigned a new reference. However, the object which the reference points to (or refers to) can undergo changes by operations (method calls) on the object.

Same is the case with arrays, since arrays are also objects. For example:

```
final int[] marksArr1 = {20, 30, 40};
final int[] marksArr2 = {10, 20, 30};
marksArr1[0] = 70;	// this is a valid statement, // since we are not changing the value in the cup marksArr1// we are changing the value inside the object pointed by the cup marksArr1marksArr1 = marksArr2;
	// compiler will throw an error on this line// stating final variable marksArr1 cannot be reassigned
```

Select all the correct statements from the below code:

```
class A {
	private final int id;
	private final String name;
	private String comments;
	public A(int id, String name, String comments) {
		this.id = id;	// statement 1this.name = name;	// statement 2this.comments = comments;	// statement 3}
	public void setComments(String comments) {
		this.comments = comments;	// statement 4}
}
public class TestA {
	public static void main(String[] args) {
		A a1 = new A(3, "Car", "Red Car");
		final A a2 = new A(4, "Jeep", "Green Jeep");
		a2.setComments("Black Jeep");	// statement 5a2 = a1;	// statement 6}
}
```

- [ ] **Statements 1** and **2** result in compilation errors. Since both these fields are declared as `final`, they must initialized during declaration only.
- [ ] **Statement 3** will result in a compilation error. Since fields `id` and `name` are declared as `final` the third field `comments` must also be declared `final`.
- [x] **Statement 4** will **not** result in a compilation error, since field `comments` is not declared as `final`.
- [ ] **Statement 5** will result in a compilation error. Since reference `a2` is declared as `final` we cannot change the value of `comments` in `a2`.
- [x] **Statement 6** will result in a compilation error.

Final methods

Q1. In Java, when we do not want a method in a class to be overridden in its subclasses, we declare that method as final.

We have many final methods in Object class. For example below code displays one of them:

```
public class Object {
        ...
        ...
        public final void wait() throws InterruptedException {
                wait(0);
        }
        ...
}
```

Select all the correct statements from the below code:

```
public class A {
        public void method1() { /*do something */}        // statement 1public final void method2() { /*do something */}  // statement 2public void method3() { /*do something */}        // statement 3}
public class B extends A {
        public void method1() { /*do something else */}   // statement 4public void method2() { /*do something else */}  // statement 5public final void method3() { /*do something else */}   // statement 6}
```

- [x] `method1` of class `A` in **statement 1** is overridden in class `B` in **statement 4**.
- [ ] **Statement 5** will not give any compilation error.
- [ ] Since class `A` has one method declared as `final`, we should also declare the class `A` as `final`.
- [ ] Statement 6 will result in a compilation error. Since the `method3` is not marked as `final` in class `A`, the subclass `B` cannot mark the method as `final` when it wants to override.