Theory: Final variables

9 minutes 0 / 4 problems solved

Sometimes, you need to use a variable that should not be modified during the program. Such variables are known as **constants**. Java provides a special keyword called `final` to declare them. The only difference between a regular variable and a final variable is that we cannot modify the value of a final variable once assigned. Hence final variables must be used only for the values that we want to remain **constant** throughout the execution of the program.

§1. Final variables

The following code demonstrates two final variables: `PI` which represents a well-known math constant and `HELLO_MSG` which represents a string text.

```
final double PI = 3.1415;
final String HELLO_MSG = "Hello";

System.out.println(PI); // 3.1415
System.out.println(HELLO_MSG); // Hello
```

Both variables cannot be modified since they are marked as final, but they can be accessed as many times as we need it.

A good practice is to represent a final variable in all caps using an underscore to separate words. It allows you to distinguish them from regular variables. But sometimes you will also see final variables, written in lowercase: this is also admissible for **local final variables**.

> Note, that the compiler will produce an error when trying to modify the value of a final variable.

Here is an example:

```
final double PI = 3.1415;
PI = 3.1416; // error line
```

The Java compiler outputs the message: `cannot assign a value to final variable PI`.

> Important, if a final variable has not been assigned before using it, the compiler also will produce an error.

Here is an example:

```
final boolean FALSE;
System.out.println(FALSE); // error line
```

If you've not assigned a value to a final variable before using it, the compiler will also produce the error `"variable might not have been initialized"`. To fix it, just assign a value before accessing the value of a final variable:

```
final boolean FALSE;       // not initialized
FALSE = false;             // initialized
System.out.println(FALSE); // no errors here
```

Notice that the value of a final variable can be reassigned to a regular variable without any restrictions:

```
final int count = 10;
int cnt = count;
cnt = 20; // no errors here, cnt is not final
```

Current topic:

Final variables

Topic depends on:

✓ String Stage 2

✓ Primitive and reference types Stage 2

Topic is required for:

Static members

Lambda expressions

The value of a regular variable can be changed any time as always.

§2. Final reference variables

The `final` keyword can be legally used with reference variables. In this case, the final keyword means that it is not possible to reassign a reference to the variable.

Here is an example with the `StringBuilder` class which is a mutable version of `String`.

```
final StringBuilder builder = new StringBuilder();
builder = new StringBuilder(); // error line
```

In this code, the second line won't compile since we are trying to reassign a reference to the final variable `builder`. But there is one important point.

> Note, that it is always possible to change the internal state of an object point by a final reference object, i.e. the constant is only the variable itself (the reference), not the object to which it refers.

So, the following code is absolutely correct:

```
final StringBuilder builder = new StringBuilder(); // ""
builder.append("Hello!"); // it works
System.out.println(builder.toString()); // Hello!
```

As you can see, this code changed the internal state of an object (`""` → `"Hello!"`) referenced by a final variable. When we invoked `append()` method we changed not the object itself but just the value of its fields. `append()` method is one of the main operations on a `StringBuilder` that are not available in String. It converts its argument to a `String` and then appends its characters to the character sequence. We will discuss the `StringBuilder` class and its methods later in further topics.

Since Java 11, it is also possible to use `final` with `var` to use the automatic type inference for the constant variable.

```
final var FINAL_VAR = 10; // int
final var MSG = "Hello!"; // String
```

§3. When to use final variables

We hope you understand how the `final` keyword works for local variables. Now it's time to figure out when to use it.

Some programmers mark all variables that they do not want to modify as `final`. In this case, the program will contain a lot of such variables.

```
final Scanner scanner = new Scanner(System.in);
final int a = scanner.nextInt();
final int b = scanner.nextInt();
System.out.println(a + b);
```

This approach allows you to write programs with the minimum number of mutable variables which usually leads to fewer errors. In addition, the Java compiler can optimize code with final variables effectively and your program can be a little faster. But this is not always predictable behavior and needs some advanced knowledge.

There is also a contra-argument: massive use of the `final` keyword makes your code less readable (boilerplate code).

Thus, in our learning examples, we will not always write the `final` keyword, but sometimes you will see such examples. In your solutions, you can write or avoid this keyword. During your real work as a programmer, we hope that the issue of using finals will be standardized for all programmers in the project.

Table of contents:

Interesting, that the `final` keyword can be used in some different contexts, not only for declaring constants. We will learn other ways to use this keyword in the next topics.

Report a typo

775 users liked this theory. **11** didn't like it. **What about you?**

Start practicing

Comments (15) Hints (0) Useful links (0) Show discussion