COURSEWARE

Pass by Value/Reference

Contents

Overview

When passing values and objects as parameters in Java, it is important to understand how that data is being handled. There are generally two methods used in programming languages. These are:

- **pass by value** a copy of the contents of the actual parameter is stored in a new local variable within the method. If this value is changed, the original parameter remains the same.
- **pass by reference** a copy of the address where the actual parameter is stored. Changing this value within the method will change the original parameter as well.

Java is, ostensibly, an exclusively pass by value language, but the way it treats objects allows us more flexibility when we code.

Object References

Java stores data in two different ways, *Primitive* and *Non-Primitive* types.

Primitive types are the most basic data types available within the Java language. There are 8: (*boolean, byte, char, short, int, long, float and double*) Such types serve only one purpose — containing pure, simple values of a kind.

Non-primitive types, or *reference types*, are the more sophisticated members of the data type family. They don't store the value, but store a reference to that value. Java keeps the reference - the address - to that value, not the value itself.

All data types in Java that haven't been listed above are non-primitive types.

As a general rule of thumb, data types which start with a `lowercase` character are primitive types - all others are non-primitive.

This means that although we are still only passing by value, when we pass non-primitive types the method receives the reference stored as a value, and thus can update the original object.

Tutorial

Primitive types

```java
public class FooBar {

    public static void main(String[] args) {
            int personAge = 35;
            changePersonAge(personAge);
            System.out.println(personAge);
    }

    static void changePersonAge(int personAge) {
            personAge = 40;
    }
}
```

What will be the output of the above code?

▶ Click here for solution

Non-Primitive types

```java
public class FooBar {

    public static void main(String[] args) {
            Person personObj = new Person();
            personObj.name = "Matthew";
            changeToMatt(personObj);
            System.out.println(personObj.name);
    }

    static void changeToMatt(Person passedPerson) {
            passedPerson.name = "Matt";
    }
}
class Person {
    String name;
}
```

What will be the output of the above code?

▶ Click here for solution

Immutable Objects

What if we did the same test with an immutable String object? A string object is a non-primitive type, so a method passed one will receive a reference to the original data.

```java
public class FooBar {

    public static void main(String[] args) {
            String personName = "";
            changePersonName(personName);
            System.out.println(personName);
    }

    static void changePersonName(String personName) {
            personName = "Stephan";
    }
}
```

What will be the output of the above code?

▶ Click here for solution

Exercises

There are no exercises for this module.