Swapping two objects with void function in Java

Martin Kozeny
CSCI 4501: Programming Language Structure
Spring 2011 University of New Orleans

February 27, 2011

1 Theory

Before I write a method in Java, which swap two objects, it is necessary to introduce some theory around that problem. According to [1], in Java is everything passed by value and objects are never passed at all. Moreover the values of variables are always primitives or references, never objects. [1] claims, that this results in the fact, that we cannot pass the object! So what the Java do when we call a function with object as a parameter? The answer is that Java pass **references to objects**. We can support this assertion with the second truth, that the values of variables are always primitives or references, never objects. Passing a reference by value actually means we can think of references how we think of primitives, to a large extent. All in all into method is passed a copy of reference to object. That means that, when this object is modified inside the method, this change will appeared also on references to this object outside method scope. But when it is to variable inside the function, which references to object, gived reference to another object, reference to original object of course expires. We can that fact demonstrate on the source code shown below.

This issue is also connected with the fact, that Java Virtual Machine has stack based architecture. That means when is some function called, firstly its parameters are pushed onto stack. Objects are passed as references to objects to function's parameters. If it is inside the scope of the function reassigned value of these parameters, this change will not appear in outer scope, because when leaving function the stack expires also with parameters.

2 Example

When we study methods $swap_i()$, where $i \in 1 \dots 5$, we can successfully predict, which function will swap two objects and this action appears also in main scope. If we look at $swap1()$, we say that reference to the instance of object b is passed to tmp and then to object b is passed reference to object a. After that is to variable a given reference to object on which refers variable $temp$. After leaving scope of this function, ends scope of this local variables (also paramaters!) and in outer scope is everything unchanged. Similarly it is in method $swap3()$ and $swap4()$. In $swap3()$ object referred with variable b is cloned but this object expired with end of the scope of method and the same thing happened to objects created by copy contructors in method $swap4()$. The only way to swap these objects is to change value of its field, than will be object, which is referred from outer scope also changed, because it is referred the same object from outer scope and also from inside context of method. Function $swap2()$ and $swap5()$ have the same effect, they only exchange value of object's fields and that's the only way, how to swap objects in void method.

```java
public class Main {
  public static void main(String[] args) {
    TestClass a = new TestClass(1);
    TestClass b = new TestClass(2);
    System.out.println("Value of object a before swap1: "+a.getValue());
    System.out.println("Value of object b before swap1: "+b.getValue());
    swap1(a,b);
    System.out.println("Value of object a after swap1: "+a.getValue());
    System.out.println("Value of object b after swap1: "+b.getValue());
    System.out.println();
    System.out.println("Value of object a before swap2: "+a.getValue());
    System.out.println("Value of object b before swap2: "+b.getValue());
    swap2(a,b);
    System.out.println("Value of object a after swap2: "+a.getValue());
    System.out.println("Value of object b after swap2: "+b.getValue());
    System.out.println();
    System.out.println("Value of object a before swap3: "+a.getValue());
    System.out.println("Value of object b before swap3: "+b.getValue());
    swap3(a,b);
    System.out.println("Value of object a after swap3: "+a.getValue());
    System.out.println("Value of object b after swap3: "+b.getValue());
    System.out.println();
    System.out.println("Value of object a before swap4: "+a.getValue());
    System.out.println("Value of object b before swap4: "+b.getValue());
    swap4(a,b);
    System.out.println("Value of object a after swap4: "+a.getValue());
    System.out.println("Value of object b after swap4: "+b.getValue());
    System.out.println();
    System.out.println("Value of object a before swap5: "+a.getValue());
    System.out.println("Value of object b before swap5: "+b.getValue());
    swap5(a,b);
    System.out.println("Value of object a after swap5: "+a.getValue());
    System.out.println("Value of object b after swap5: "+b.getValue());
  }
  public static void swap1(TestClass a, TestClass b)
  {
    TestClass tmp = b;
    b = a;
    a = tmp;
  }
  public static void swap2(TestClass a, TestClass b)
  {
    int tmpVal = b.getValue();
    b.setValue(a.getValue());
    a.setValue(tmpVal);
  }
  public static void swap3(TestClass a, TestClass b)
  {
    TestClass tmp = (TestClass) b.clone();
    b = (TestClass) a.clone();
    a = tmp;
  }
  public static void swap4(TestClass a, TestClass b)
  {
    TestClass tmp = new TestClass(b);
    b = new TestClass(a);
    a = new TestClass(tmp);
  }
  public static void swap5(TestClass a, TestClass b)
  {
    TestClass tmp = new TestClass();
    tmp.copyTestClassObject(b);
    b.copyTestClassObject(a);
    a.copyTestClassObject(tmp);
  }
}
```

```java
public class TestClass implements Cloneable{

  private int value;

  public TestClass(){}

  public TestClass(TestClass testClass)
  {
    this.value = testClass.getValue();
  }

  public TestClass(int value) {
    super();
    this.value = value;
  }

  public void copyTestClassObject(TestClass testClass)
  {
    this.value = testClass.getValue();
  }

  public int getValue() {
    return value;
  }

  public void setValue(int value) {
    this.value = value;
  }
  public Object clone()
    {
        try
        {
            return super.clone();
        }
        catch( CloneNotSupportedException e )
        {
            return null;
        }
    }
}
```

Output of calling

```
Value of object a before swap1: 1
Value of object b before swap1: 2
Value of object a after swap1: 1
Value of object b after swap1: 2

Value of object a before swap2: 1
Value of object b before swap2: 2
Value of object a after swap2: 2
Value of object b after swap2: 1

Value of object a before swap3: 2
Value of object b before swap3: 1
Value of object a after swap3: 2
Value of object b after swap3: 1

Value of object a before swap4: 2
Value of object b before swap4: 1
Value of object a after swap4: 2
Value of object b after swap4: 1

Value of object a before swap5: 2
Value of object b before swap5: 1
Value of object a after swap5: 1
Value of object b after swap5: 2
```

References

[1] Jon's homepage, Jon Skeet, software engineer at Google, `http://www.yoda.arachsys.com/java/passing.html#formal`.