Experiment 6: Write a java program to demonstrate:
a) Call by value
b) Call by reference

6(a). Call by Value

Aim

To write a Java program to demonstrate Call by Value.

Theory

In Java, when a primitive data type is passed to a method, a copy of the value is passed. Any changes made inside the method do not affect the original variable.

Algorithm

1. Declare an integer variable.
2. Pass it to a method.
3. Modify the value inside the method.
4. Display values before and after method call.

Program Code

```
class CallByValue {

    void change(int x) {
        x = x + 10;
        System.out.println("Inside method: " + x);
    }

    public static void main(String[] args) {
        CallByValue obj = new CallByValue();

        int num = 50;
        System.out.println("Before method call: " + num);
        obj.change(num);
        System.out.println("After method call: " + num);
    }
}
```

Result

The value of the variable remains unchanged after method call, proving Call by Value.

6(b). Call by Reference

Aim

To write a Java program to demonstrate Call by Reference.

Theory

When an object is passed to a method, the reference of the object is passed.
Changes made inside the method affect the original object.

Algorithm

1. Create a class with a data member.
2. Pass the object to a method.
3. Modify the object data inside the method.
4. Display values before and after method call.

Program Code

```
class Student {
    int marks;
}
class CallByReference {

    void change(Student s) {
        s.marks = s.marks + 10;
        System.out.println("Inside method: " + s.marks);
    }

    public static void main(String[] args) {
        CallByReference obj = new CallByReference();

        Student s1 = new Student();
        s1.marks = 60;

        System.out.println("Before method call: " + s1.marks);
        obj.change(s1);
        System.out.println("After method call: " + s1.marks);
    }
}
```

Result

The value of the object changes after method call, proving Call by Reference.