Passing and Returning Objects in Java

Although Java is [strictly pass by value](https://www.geeksforgeeks.org/g-fact-31-java-is-strictly-pass-by-value/), the precise effect differs between whether a [primitive type](https://www.geeksforgeeks.org/data-types-in-java/) or a reference type is passed.

When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that’s sort of a hybrid between pass-by-value and pass-by-reference. Basically, a parameter cannot be changed by the function, but the function can ask the parameter to change itself via calling some method within it.

* While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
* This effectively means that objects act as if they are passed to methods by use of call-by-reference.
* Changes to the object inside the method do reflect in the object used as an argument.

In Java we can pass objects to methods. For example, consider the following program :

|  |
| --- |
| // Java program to demonstrate objects  // passing to methods.  class ObjectPassDemo  {      int a, b;        ObjectPassDemo(int i, int j)      {          a = i;          b = j;      }        // return true if o is equal to the invoking      // object notice an object is passed as an      // argument to method      boolean equalTo(ObjectPassDemo o)      {          return (o.a == a && o.b == b);      }  }    // Driver class  public class Test  {      public static void main(String args[])      {          ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);          ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);          ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);            System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));          System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));      }  } |

Run on IDE

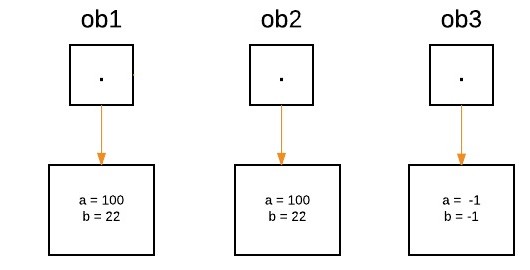
Output:

ob1 == ob2: true

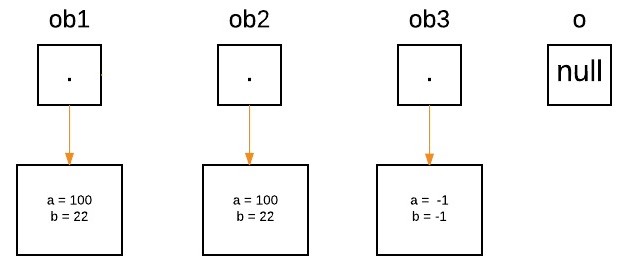
ob1 == ob3: false

**Illustrative images for the above program**

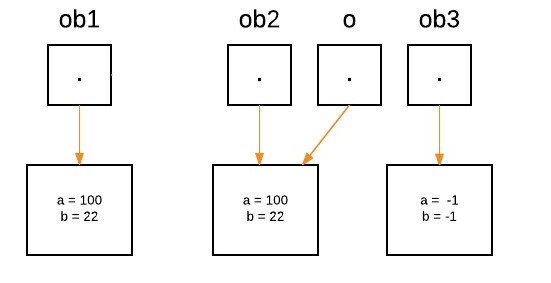
* Three objects ‘ob1’ , ‘ob2’ and ‘ob3’ are created:
* ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
* ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
* ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/one.jpeg)

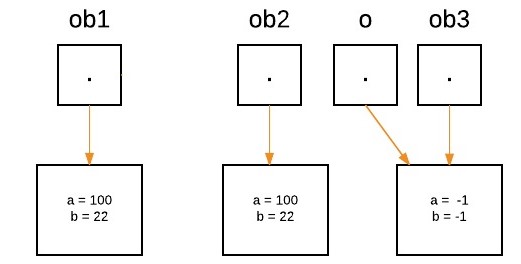
* From the method side, a reference of type Foo with a name a is declared and it’s initially assigned to null.
* boolean equalTo(ObjectPassDemo o);

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/two.jpeg)

* As we call the method equalTo, the reference ‘o’ will be assigned to the object which is passed as an argument, i.e. ‘o’ will refer to ‘ob2’ as following statement execute.
* System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/three.jpeg)

* Now as we can see, equalTo method is called on ‘ob1’ , and ‘o’ is referring to ‘ob2’. Since values of ‘a’ and ‘b’ are same for both the references, so if(condition) is true, so boolean true will be return.
* if(o.a == a && o.b == b)
* Again ‘o’ will reassign to ‘ob3’ as the following statement execute.
* System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/four.jpeg)

* Now as we can see, equalTo method is called on ‘ob1’ , and ‘o’ is referring to ‘ob3’. Since values of ‘a’ and ‘b’ are not same for both the references, so if(condition) is false, so else block will execute and false will be return.