OOP Class
Object as parameter, Argument Passing, Recursion, call by value & call by reference

References:

- Java: The Complete Reference, Book & other online resources

Object as parameter

```
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));
        }
}
```

o/p

- ob1 == ob2: true
- ob1 == ob3: false

```
// Java program to demonstrate one object
class Box
{       double width, height, depth;
        Box(Box ob)
        {
                width = ob.width;
                height = ob.height;
                depth = ob.depth;
        }
        Box(double w, double h, double d)
        {
                width = w;
                height = h;
                depth = d;
        }
                double volume()
        {
                return width * height * depth;
        }
}
```

```
public class Test
{	public static void main(String args[])
	{	// creating a box with all dimensions specified
		Box mybox = new Box(10, 20, 15);
		// creating a copy of mybox
		Box myclone = new Box(mybox);
		double vol;
		// get volume of mybox
		vol = mybox.volume();
		System.out.println("Volume of mybox is " + vol);
		// get volume of myclone
		vol = myclone.volume();
		System.out.println("Volume of myclone is " + vol);
	}
}
```

o/p

Volume of mybox is 3000.0
Volume of myclone is 3000.0

Argument Passing

- In general, there are two ways that a computer language can pass an argument to a subroutine.
  - *call-by-value*
  - *call-by-reference*

- *call-by-value*: This approach copies the *value* of an argument into the formal parameter of the subroutine. Therefore, changes made to the parameter of the subroutine have no effect on the argument.
- *call-by-reference*: a reference to an argument (not the value of the argument) is passed to the parameter. Inside the subroutine, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will affect the argument used to call the subroutine

Overview

S. No.	Call by Value	Call by Reference
1.	A copy of actual parameters is passed into formal parameters.	Reference of actual parameters is passed into formal parameters.
2.	Changes in formal parameters will not result in changes in actual parameters.	Changes in formal parameters will result in changes in actual parameters.
3.	Separate memory location is allocated for actual and formal parameters.	Same memory location is allocated for actual and formal parameters.

*pass by **reference***

cup =

fillCup( )

*pass by **value***

cup =

fillCup( )

www.penjee.com

```
// Primitive types are passed by value.
class Test {
void meth(int i, int j) {
        i *= 2;
        j /= 2;
                }
        }
class CallByValue {
        public static void main(String args[]) {
        Test ob = new Test();
        int a = 15, b = 20;
        System.out.println("a and b before call: " +a + " " + b);
        ob.meth(a, b);
        System.out.println("a and b after call: " +a + " " + b);
                                        }
                }
```

The output from this program is shown here:

```
a and b before call: 15 20
a and b after call: 15 20
```

Objects are passed through their references.

```
class Test {
        int a, b;
        Test(int i, int j) {
        a = i;  b = j;
                }
        // pass an object
        void meth(Test o) {
        o.a *= 2; o.b /= 2;
        }
        }
class PassObjRef {
        public static void main(String args[]) {
        Test ob = new Test(15, 20);
        System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);
}
}
```

This program generates the following output:

ob.a and ob.b before call: 15 20

ob.a and ob.b after call: 30 10

As you can see, in this case, the actions inside **meth( )** have affected the object used as an argument.

Recursion

- Recursion is the process of defining something in terms of itself. As it relates to Java programming, recursion is the attribute that allows a method to call itself.
- A method that calls itself is said to be *recursive*.

```
public class RecursionExample2
{
        static void p()
        {       int count=0
                count++;
        if(count<=5)
        {
          System.out.println("hello "+count);
                p();
        }
 }
  public static void main(String[] args)
        {
                 p();
        }
  }
```

Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- **It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.**
- **It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.**

finalize() method

Just before destroying an object, Garbage Collector calls the finalize() method on the object to perform cleanup activities. Once the finalize() method completes, Garbage Collector destroys that object. The finalize() method is invoked each time before the object is garbage collected. The finalize() method is called by garbage collection thread before collecting objects. It's the last chance for any object to perform cleanup utility.

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public class GarbageCollectionDemo
{
   public void finalize()
   {
       System.out.println ("Garbage Collection performed by JVM");
   }
   public static void main (String args[])
   {
       GarbageCollectionDemo s1 = new GarbageCollectionDemo ();
       GarbageCollectionDemo s2 = new GarbageCollectionDemo ();
       s1 = null;
       s2 = null;
       System.gc();
   }
}
```