Unit I Part C

Garbage Collection

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

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.

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.

How can an object be unreferenced?
There are many ways:

1. By nulling the reference

```
Employee e=new Employee();
e=null;
```

2. By assigning a reference to another

```
Employee e1=new Employee();
Employee e2=new Employee();
 e1=e2;
```

3. By anonymous object etc.

```
new Employee();
```

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.

Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up.

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 TestGarbage
{
        public void finalize()
        {
                System.out.println("object is garbage collected");
        }
        public static void main(String args[])
        {
                TestGarbage s1=new TestGarbage();
                TestGarbage s2=new TestGarbage();
                s1=null;
                s2=null;
                System.gc();
        }
}
```

Note: Neither finalization nor garbage collection is guaranteed.

Returning the Object

```
class Rectangle
{
        int length;
        int breadth;
        Rectangle(int l,int b)
        {
                length = l;
                breadth = b;
        }
        Rectangle getRectangleObject()
        {
                Rectangle rect = new Rectangle(10,20);
                return rect;
        }
}
class RetOb
{
        public static void main(String args[])
        {
                Rectangle ob1 = new Rectangle(40,50);
                Rectangle ob2;
                ob2 = ob1.getRectangleObject();
                System.out.println("ob1.length : " + ob1.length);
                System.out.println("ob1.breadth: " + ob1.breadth);
                System.out.println("ob2.length : " + ob2.length);
                System.out.println("ob2.breadth: " + ob2.breadth);
         }
}
```

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.
It makes the code compact but complex to understand.

```
class RecursionExample
{
        int n1=0,n2=1,n3=0;
        void printFibo(int count)
        {
                if(count>0)
                {
                        n3 = n1 + n2;
                        n1 = n2;
                        n2 = n3;
                        System.out.print(" "+n3);
                        printFibo(count-1);
                }
        }
        public static void main(String[] args)
        {
                int count=15;
                RecursionExample re= new RecursionExample();
                System.out.print(re.n1+" "+re.n2);//printing 0 and 1
                re.printFibo(count-2);//n-2 because 2 numbers are already printed
        }
}
```

Java Inner Classes

Java inner class or **nested class** is a class which is declared inside the class or interface.
We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
Additionally, it can access all the members of outer class including private data members and methods.

```
class TestOuter
{
	private int data=30;
	class Inner
	{
		void msg()
		{
			System.out.println("data is "+data);
		}
	}
	public static void main(String args[])
	{
		TestOuter obj=new TestOuter();
		TestOuter.Inner in=obj.new Inner();
		in.msg();
	}
}
```

Java Local inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

```
public class LocalInner
{
	private int data=30;//instance variable
	void display()
	{
		class Local
		{
			void msg()
			{
				System.out.println(data);
			}
		}
		Local l=new Local();
		l.msg();
	}
	public static void main(String args[])
	{
		LocalInner obj=new LocalInner();
		obj.display();
	}
}
```

Variable Argument (Varargs):

The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

Advantage of Varargs:

We don't have to provide overloaded methods so less code.

Rules for varargs:

While using the varargs, you must follow some rules otherwise program code won't compile. The rules are as follows:

- There can be only one variable argument in the method.
- Variable argument (varargs) must be the last argument.

```
class VarargsExample
{
	static void display(String... values)
	{
		System.out.println("display method invoked ");
		for(String s:values)
		{
			System.out.println(s);
		}
	}
	public static void main(String args[])
	{
		display();//zero argument
		display("hello");//one argument
		display("my","name","is","varargs");//four arguments
	}
}
```