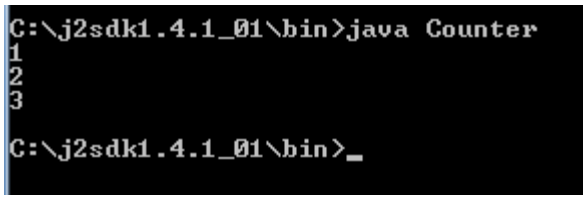
```
class Counter
{

//  int count=0;//will get memory when instance is created

static int count=0;//will get memory only once and retain its value

Counter()
{
count++;
System.out.println(count);
}

public static void main(String args[])
{

Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();

}
}
```



```
class student
{
  int rollno;
  String name;
  static String college ="GZSCCET";

  student(int r,String n)
{
  rollno = r;
  name = n;
 }
 void display ()
{
System.out.println(rollno+" "+name+" "+college);
}

 public static void main(String args[])
{
 student s1 = new student(111,"Kalyani");
 student s2 = new student(222,"Aryan");

 s1.display();
 s2.display();
 }
}
```

```
class Bike
{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike obj=new  Bike();
 obj.run();
 }
}//end of class


/*
   Output:Compile Time Error
*/


/*

class Bike
{
  final void run()
{
System.out.println("running");
}

}

class Honda extends Bike
{
```

```
  void run()
{
System.out.println("running safely with 100kmph");
}

  public static void main(String args[])
 {
  Honda honda= new Honda();
  honda.run();
 }
}

*/

/*

Output:Compile Time Error
*/




/*

final class Bike
{

}

class Honda1 extends Bike
{
 void run()
{
```

```
System.out.println("running safely with 100kmph");
}

  public static void main(String args[])
{
  Honda1 honda= new Honda();
  honda.run();
  }
}

*/

/*

Output:Compile Time Error

*/




/*

class Bike
{
  final void run()
{
System.out.println("running...");
}
}
class Honda2 extends Bike
{
   public static void main(String args[])
```

```
{
    new Honda2().run();
   }
}

*/

/*

Output:running...

*/
```

/*

What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. I know it sounds confusing. Don't worry we will discuss this in detail.

It is a feature that allows one interface to be used for a general class of actions.
An operation may exhibit different behavior in different instances.
The behavior depends on the types of data used in the operation.
It plays an important role in allowing objects having different internal structures to share the same external interface.
Polymorphism is extensively used in implementing inheritance.
Following concepts demonstrate different types of polymorphism in java.
1) Method Overloading
2) Method Overriding

Method Definition:
A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name.

1)Method Overloading:
In Java, it is possible to define two or more methods of same name in a class, provided that there argument list or parameters are different. This concept is known as Method Overloading.

I have covered method overloading and Overriding below. To know more about polymorphism types refer my post Types of polymorphism in java: Static, Dynamic, Runtime and Compile time Polymorphism.

1) Method Overloading

To call an overloaded method in Java, it is must to use the type and/or number of arguments to determine which version of the overloaded method to actually call.

Overloaded methods may have different return types; the return type alone is insufficient to distinguish two versions of a method. .

When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

It allows the user to achieve compile time polymorphism.

An overloaded method can throw different exceptions.

It can have different access modifiers.

Example:

```
*/

class Overload
{
   void demo (int a)
   {
     System.out.println ("a: " + a);
   }
   void demo (int a, int b)
   {
     System.out.println ("a and b: " + a + "," + b);
   }
   double demo(double a) {
     System.out.println("double a: " + a);
     return a*a;
   }
}
class MethodOverloading
{
   public static void main (String args [])
   {
```

```
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);
    }
}

/*
Here the method demo() is overloaded 3 times: first having 1 int parameter, second one has 2 int parameters and third one is having double arg. The methods are invoked or called with the same type and number of parameters used.

Output:

a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
Rules for Method Overloading

Overloading can take place in the same class or in its sub-class.
Constructor in Java can be overloaded
Overloaded methods must have a different argument list.
Overloaded method should always be the part of the same class (can also take place in sub class), with same name but different parameters.
The parameters may differ in their type or number, or in both.
They may have the same or different return types.
It is also known as compile time polymorphism.

*/
```

/*

2) Method Overriding

Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.
Example:

*/

```
public class BaseClass
{
   public void methodToOverride() //Base class method
   {
       System.out.println ("I'm the method of BaseClass");
   }
}

public class DerivedClass extends BaseClass
{
   public void methodToOverride() //Derived Class method
   {
       System.out.println ("I'm the method of DerivedClass");
   }
}

public class TestMethod
{
    public static void main (String args []) {
      // BaseClass reference and object
      BaseClass obj1 = new BaseClass();
```

```
        // BaseClass reference but DerivedClass object
        BaseClass obj2 = new DerivedClass();
        // Calls the method from BaseClass class
        obj1.methodToOverride();
        //Calls the method from DerivedClass class
        obj2.methodToOverride();
    }
}

/*

Output:

I'm the method of BaseClass
I'm the method of DerivedClass
Rules for Method Overriding:

applies only to inherited methods
object type (NOT reference variable type) determines which overridden method will be used at runtime
Overriding method can have different return type (refer this)
Overriding method must not have more restrictive access modifier
Abstract methods must be overridden
Static and final methods cannot be overridden
Constructors cannot be overridden
It is also known as Runtime polymorphism.

*/
```

```
/*

super keyword in Overriding:

When invoking a superclass version of an overridden method the super keyword is used.

Example:

*/

class Vehicle {
   public void move () {
       System.out.println ("Vehicles are used for moving from one place to another ");
   }
}

class Car extends Vehicle {
   public void move () {
    super. move (); // invokes the super class method
    System.out.println ("Car is a good medium of transport ");
   }
}

public class TestCar {
   public static void main (String args []){
      Vehicle b = new Car (); // Vehicle reference but Car object
      b.move (); //Calls the method in Car class
   }
}
```

/*

Output:

Vehicles are used for moving from one place to another

Car is a good medium of transport

*/