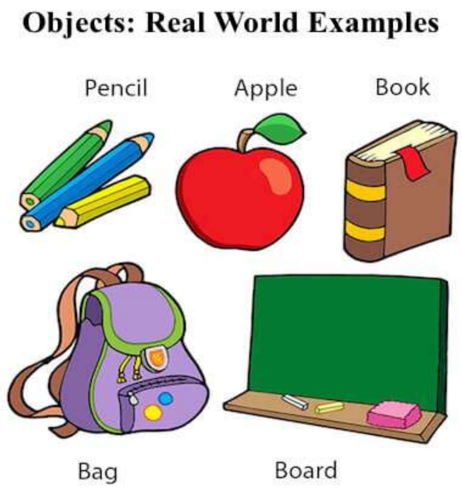
22. Object and Class in Java

In this page, we will learn about Java objects and classes. In object-oriented programming technique, we design a program using objects and classes. An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

22.1 What is an Object in Java

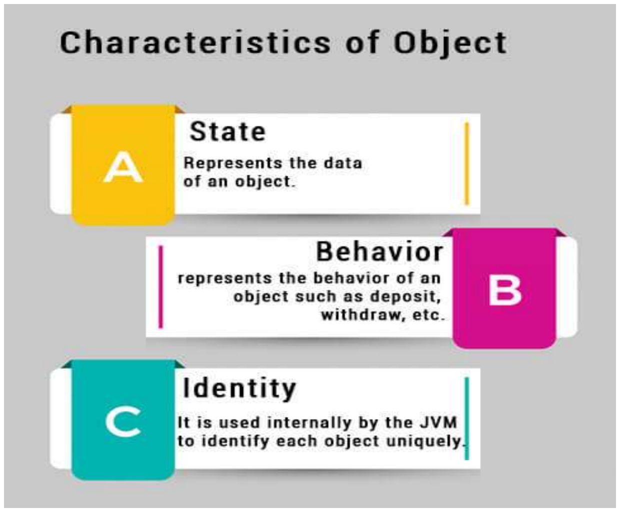


An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.


Characteristics of Object
A
State
Represents the data of an object.
Behavior
represents the behavior of an object such as deposit, withdraw, etc.
B
C
Identity
It is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

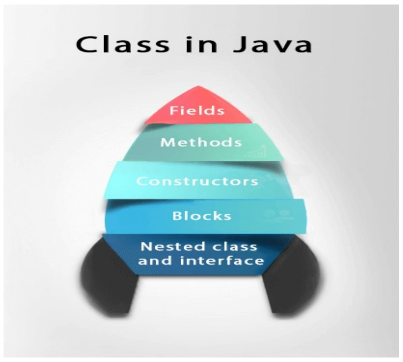
- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behavior.
- The object is an instance of a class.

22.2 What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface



Syntax to declare a class:

```
class <class_name>
{
        field;
        method;
}
```

22.3 Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

22.4 Method in Java

In Java, a method is like a function which is used to expose the behavior of an object. **Advantage of Method**

- Code Reusability
- Code Optimization

22.5 new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

22.6 Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value. Here, we are creating a main() method inside the class. In the below example we are not initialize any value to it so that the default values are stored in the instance variables.

Example:

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
public  class Student
{
      //defining fields
      //field or data member or instance variable
      int id;
```

```
     String name;
     //creating main method inside the Student class
     public static void main(String args[])
     {
          //Creating an object or instance
          //creating an object of Student
          Student s1=new Student();
          //Printing values of the object
          //accessing member through reference variable
          System.out.println(s1.id);
          System.out.println(s1.name);
     }
}
```

Output:

```
0
null
```

22.7 Object and Class Example: main outside the class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

Example:

```
//Java Program to demonstrate having the
//main method in another class
//Creating Student class.
class Student
{
     int id;
     String name;
}
//Creating another class TestStudent1
//which contains the main method
class TestStudent1
{
     public static void main(String args[])
     {
          Student s1=new Student();
          System.out.println(s1.id);
          System.out.println(s1.name);
     }
}
```

Output:

```
0
null
```

22.8 3-Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

22.8.1 Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

Example1:

```
class Student
{
	int id;
	String name;
}
public class Test
{
	public static void main(String args[])
	{
	Student s1=new Student();
	s1.id=101;
	s1.name="Sonoo";
	//printing members with a white space
	System.out.println(s1.id+" "+s1.name);
	}
}
```

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

Example2:

```
class Student
{
	int id;
	String name;
}
class Test
```

```
{
	public static void main(String args[])
	{
		//Creating objects
		Student s1=new Student();
		Student s2=new Student();
		//Initializing objects
		s1.id=101;
		s1.name="Sonoo";
		s2.id=102;
		s2.name="Amit";
		//Printing data
		System.out.println(s1.id+" "+s1.name);
		System.out.println(s2.id+" "+s2.name);
	}
}
```

Output:

```
101 Sonoo
102 Amit
```

22.8.2 Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

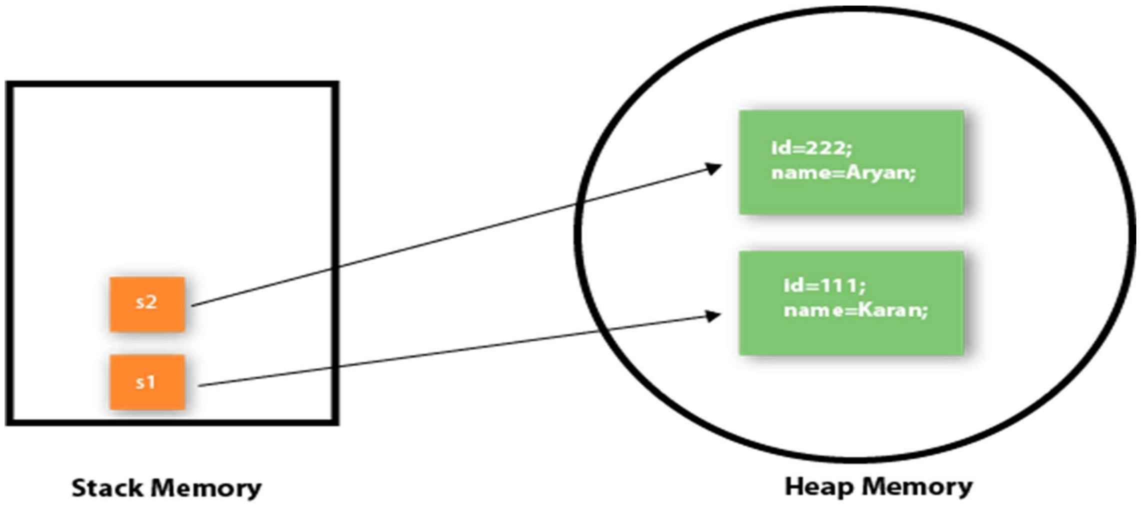
Example1:

```
class Student
{
	int rollno;
	String name;
	void insertRecord(int r, String n)
	{
		rollno=r;
		name=n;
	}
	void displayInformation()
	{
		System.out.println(rollno+" "+name);}
	}
class Test
{
	public static void main(String args[])
	{
		Student s1=new Student();
		Student s2=new Student();
```

```
            s1.insertRecord(111,"Karan");
            s2.insertRecord(222,"Aryan");
            s1.displayInformation();
            s2.displayInformation();
      }
}
```

Output:

```
111 Karan
222 Aryan
```



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

Example2:

Let's see an example where we are maintaining records of employees.

```
class Employee
{
      int id;
    String name;
    float salary;
    void insert(int i, String n, float s)
    {
        id=i;
        name=n;
        salary=s;
    }
    void display()
    {
      System.out.println(id+" "+name+" "+salary);
    }
}
public class TestEmployee
```

```
{
      public static void main(String[] args)
      {
          Employee e1=new Employee();
          Employee e2=new Employee();
          Employee e3=new Employee();
          e1.insert(101,"ajeet",45000);
          e2.insert(102,"irfan",25000);
          e3.insert(103,"nakul",55000);
          e1.display();
          e2.display();
          e3.display();
      }
}
```

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

Example3:

There is given another example that maintains the records of Rectangle class.

```
class Rectangle
{
      int length;
      int width;
      void insert(int l, int w)
      {
            length=l;
            width=w;
      }
      void calculateArea()
      {
            System.out.println(length*width);
      }
}
public class TestRectangle
{
      public static void main(String args[])
      {
            Rectangle r1=new Rectangle();
            Rectangle r2=new Rectangle();
            r1.insert(11,5);
            r2.insert(3,15);
            r1.calculateArea();
            r2.calculateArea();
      }
}
```

Output:

```
55
45
```

Example4:

```
//Java Program to demonstrate the working
//of a banking-system where we deposit and
//withdraw amount from our account.

//Creating an Account class which has
//deposit() and withdraw() methods
class Account
{
	int acc_no;
	String name;
	float amount;
	//Method to initialize object
	void insert(int a,String n,float amt)
	{
		acc_no=a;
		name=n;
		amount=amt;
	}
	//deposit method
	void deposit(float amt)
	{
		amount=amount+amt;
		System.out.println(amt+" deposited");
	}
	//withdraw method
	void withdraw(float amt)
	{
		if(amount<amt)
		{
			System.out.println("Insufficient Balance");
		}
		else
		{
			amount=amount-amt;
			System.out.println(amt+" withdrawn");
		}
	}
	//method to check the balance of the account
	void checkBalance()
	{
		System.out.println("Balance is: "+amount);
	}
	//method to display the values of an object
	void display()
	{
		System.out.println(acc_no+" "+name+" "+amount);
	}
}
//Creating a test class to deposit and
//withdraw amount
public class TestAccount
{
	public static void main(String[] args)
	{
		Account a1=new Account();
		a1.insert(832345,"Ankit",1000);
		a1.display();
		a1.checkBalance();
		a1.deposit(40000);
		a1.checkBalance();
		a1.withdraw(15000);
		a1.checkBalance();
	}
}
```

Output:

```
832345 Ankit 1000.0
Balance is: 1000.0
40000.0 deposited
Balance is: 41000.0
15000.0 withdrawn
Balance is: 26000.0
```

22.8.3 Object and Class Example: Initialization through a constructor

In Java , a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

22.8.3.1 Rules for creating Java constructor

There are two rules defined for the constructor.

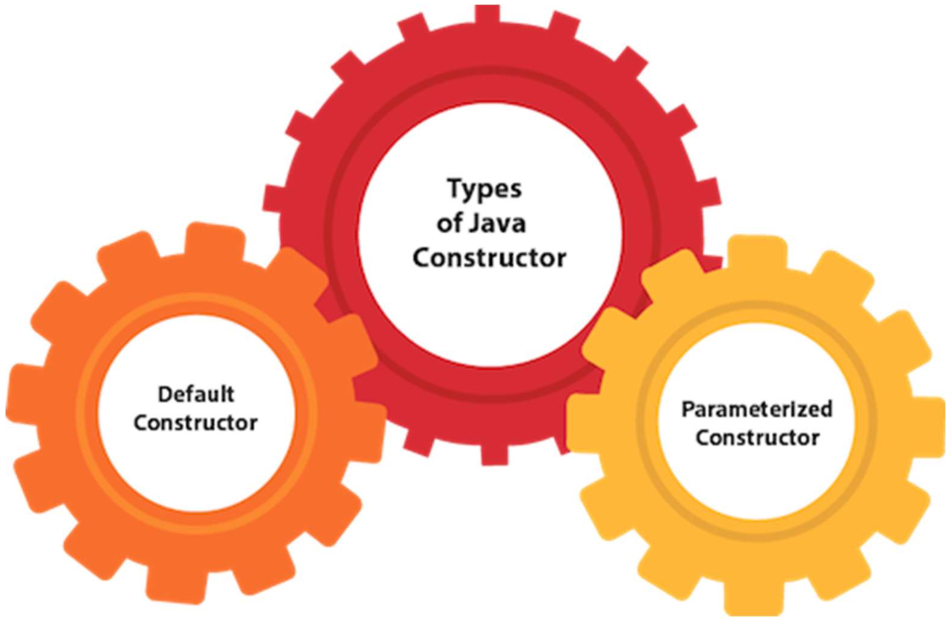
1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

22.8.3.2 Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



22.8.3.2.1 Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>()
{

}
```

Example of Default Constructor

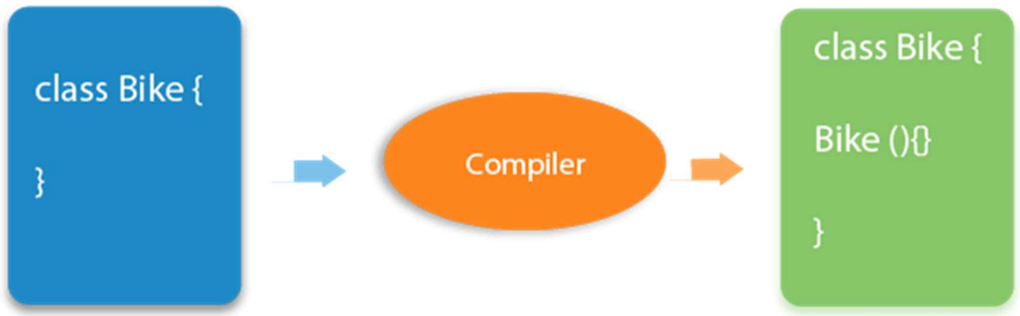
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
public class Bike1
{
	//creating a default constructor
	Bike1()
	{
		System.out.println("Bike is created");
	}
	//main method
	public static void main(String args[])
	{
		//calling a default constructor
		Bike1 b=new Bike1();
	}
}
```

Output:

```
Bike is created
```

Note: If there is no constructor in a class, compiler automatically creates a default constructor.



Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
public class Student3
{
	int id;
	String name;
	//method to display the value of id and name
	void display()
	{
		System.out.println(id+" "+name);
	}

	public static void main(String args[])
	{
		//creating objects
		Student3 s1=new Student3();
		Student3 s2=new Student3();
		//displaying values of the object
		s1.display();
		s2.display();
	}
}
```

Output:

```
0 null
0 null
```

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Example2:

```
class A
{
	int id;
	String name;
}
public class Lokesh
{
	public static void main(String args[])
	{
		A s1=new A();
		A s2=new A();
		System.out.println(s1.id+" "+s1.name);
		System.out.println(s2.id+" "+s2.name);
	}
}
```

Output:

```
0 null
0 null
```

22.8.3.2.1 Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the
//parameterized constructor
public class Student4
{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i, String n)
    {
          id = i;
          name = n;
    }
    //method to display the values
    void display()
    {
          System.out.println(id+" "+name);
    }

    public static void main(String args[])
    {
          //creating objects and passing values
          Student4 s1 = new Student4(111,"Karan");
          Student4 s2 = new Student4(222,"Aryan");
          //calling method to display
            //the values of   object
          s1.display();
          s2.display();
    }
}
```

Output:

```
111 Karan
222 Aryan
```

Example2:

```
class B
{
      int id;
    String name;
    public B(int i, String n)
    {
          id = i;
          name = n;
    }
}
public class A
{
    public static void main(String args[])
    {
         B s1 = new B(229,"Lokesh");
         B s2 = new B(235,"Bhavana");
         System.out.println(s1.id+" "+s1.name);
         System.out.println(s2.id+" "+s2.name);
    }
}
```

Output:

```
229 Lokesh
235 Bhavana
```

22.8.3.3 Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors
public class Student5
{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i, String n)
    {
         id = i;
         name = n;
    }
    //creating three arg constructor
    Student5(int i, String n, int a)
    {
         id = i;
         name = n;
         age=a;
    }
    void display()
    {
         System.out.println(id+" "+name+" "+age);
    }
    public static void main(String args[])
    {
           Student5 s1 = new Student5(229,"Lokesh");
```

```
            Student5 s2 = new Student5(235,"Bhavana",21);
            s1.display();
            s2.display();
            System.out.println(s1.id+" "+s1.name);
            System.out.println(s2.id+" "+s2.name+" "+s2.age);
    }
}
```

Output:

```
229 Lokesh 0
235 Bhavana 21
229 Lokesh
235 Bhavana 21
```

22.8.3.4 Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

22.8.3.5 Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

1. By constructor
2. By assigning the values of one object into another
3. By clone() method of Object class

In this example, we are going to copy the values of one object into another using Java constructor.

Example:

```
//Java program to initialize the values from
//one object to another object.
public class Student6
```

```
{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n)
    {
         id = i;
         name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s)
    {
         id = s.id;
         name =s.name;
    }
    void display()
    {
         System.out.println(id+" "+name);
    }

    public static void main(String args[])
    {
           Student6 s1 = new Student6(111,"Karan");
           Student6 s2 = new Student6(s1);
           s1.display();
           s2.display();
    }
}
```

Output:

```
111 Karan
111 Karan
```

22.8.3.6 Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

Example:

```
public class Student7
{
    int id;
    String name;
    Student7(int i, String n)
    {
         id = i;
         name = n;
    }
    Student7()
    {

    }
    void display()
    {
      System.out.println(id+" "+name);
    }

    public static void main(String args[])
    {
        Student7 s1 = new Student7(111,"Karan");
        Student7 s2 = new Student7();
        s2.id=s1.id;
        s2.name=s1.name;
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Karan
111 Karan
```

Q) Does constructor return any value?
Yes, it is the current class instance (You cannot use return type yet it returns a value).

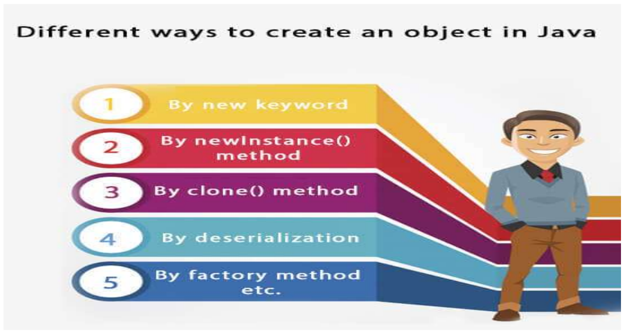
Q) Can constructor perform other tasks instead of initialization?
Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Q) Is there Constructor class in Java?
Yes.
Q) What is the purpose of Constructor class?
Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

22.9 What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method



- By deserialization
- By factory method etc.

We will learn these ways to create object later.

22.10 Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach. For example:

```
new Calculation();//anonymous object
```

Calling method through a reference:

```
Calculation c=new Calculation();
c.fact(5);
```

Calling method through an anonymous object:

```
new Calculation().fact(5);
```

Let's see the full example of an anonymous object in Java.

```
public class Calculation
{
	void fact(int  n)
	{
		int fact=1;
		for(int i=1;i<=n;i++)
		{
		fact=fact*i;
		}
		System.out.println("factorial is "+fact);
	}
	public static void main(String args[])
	{
		//calling method with anonymous object
		new Calculation().fact(5);
	}
}
```

Output:

```
factorial is 120
```

22.11 Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

```
int a=10, b=20;
```

Initialization of reference variables:

```
//creating two objects
Rectangle r1=new Rectangle(), r2=new Rectangle();
```

Let's see the example:

```
//Java Program to illustrate the use of
//Rectangle class which has length and
//width data members
class Rectangle
{
      int length;
      int width;
      void insert(int l, int w)
      {
            length=l;
            width=w;
      }
      void calculateArea()
      {
            System.out.println(length*width);
      }
}
public class TestRectangle2
{
      public static void main(String args[])
      {
            //creating two objects
            Rectangle r1=new Rectangle(),r2=new Rectangle();
            r1.insert(11,5);
            r2.insert(3,15);
            r1.calculateArea();
            r2.calculateArea();
      }
}
```

Output:

```
55
45
```