



Classes & Objects in Java

Class in Java

- A class is a user-defined data type.
- A Class is collection of members like Attributes(Variables) & Methods(Functions)
- It is a template or blueprint from which objects are created.
- A class in java can contain:
 - data members(variables) & methods
 - constructors
 - block
 - class and interface



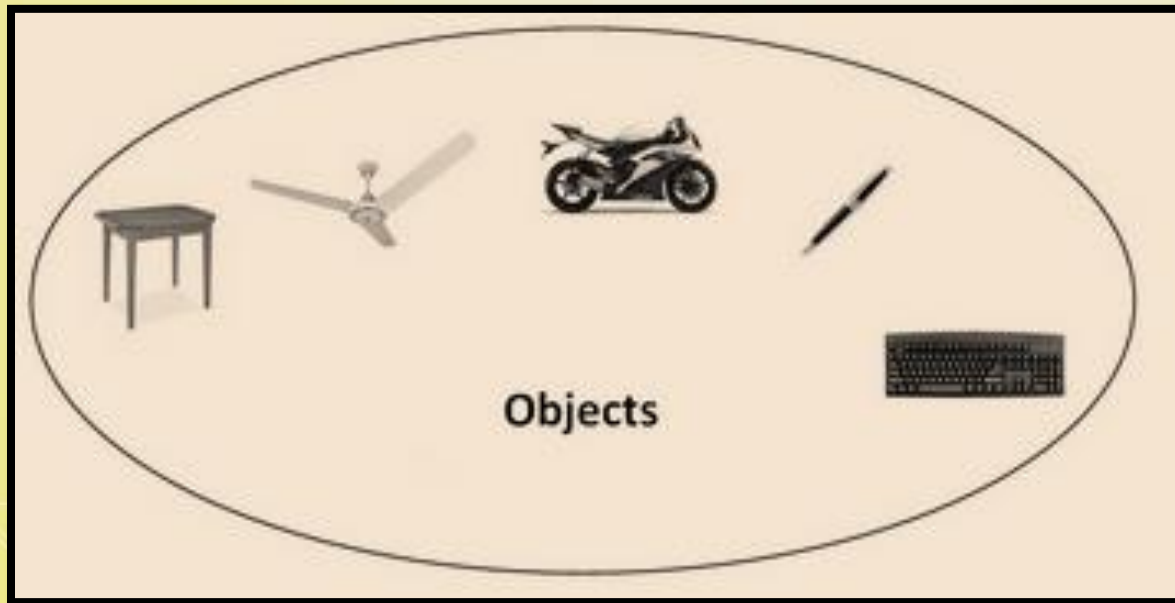
Object in Java

- An entity that has state and behavior is known as an object. e.g. chair, bike, marker, pen,
- It can be physical or logical (touchable or non-touchable)
- The example of logical object is banking system.
- An object has three characteristics:
 - state: represents data (value) of an object.
 - behavior: represents the behavior (functionality) of an object such as deposit, withdraw etc.
 - identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user.
 - But, it is used internally by the JVM to identify each object uniquely.



Object in Java

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



General form to declare a class

General form of a class:

```
class classname {  
    type  instance-variable-1;  
    ...  
    type  instance-variable-n;  
  
    type  method-name-1(parameter-list)      { ... }  
    type  method-name-2(parameter-list)      { ... }  
    ...  
    type  method-name-m(parameter-list)      { ... }  
}
```

Example of Class and Object

```
class Student_Info
{
    int id;//data member
    String name;//data member
    public static void main(String args[])
    {
        Student_Info s = new Student_Info();
        System.out.println(s.id);
        System.out.println(s.name);
    }
}
```

Output:

0

null



Object Creation

- There are three steps when creating an object from a class –
 - **Declaration** – A variable declaration with a variable name with an object type.
 - **Instantiation** – The 'new' keyword is used to create the object.
 - **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.



Example...

```
class Student_Info
```

```
{
```

```
    int id;//data member
```

```
    String name;//data member
```

```
    public static void main(String args[])
```

```
{
```

```
        Student_Info s = new Student_Info();
```

```
        s.id = 1;
```

```
        s.name = "ABC"
```

```
        System.out.println("ID:"+s.id);
```

```
        System.out.println("Name:"+s.name);
```

```
    }
```

```
}
```

Output:

ID:1

Name: ABC



new Keyword

Indicates that it does not
point to any object ←

null

S

```
Student s;  
s = new Student( );
```

S

s.id

s.name

combined

Student s = new Student();

Methods in Java

- In java, a method is like function i.e. used to expose behavior of an object.
- **Advantage of Method**
 - Code Reusability
 - Code Optimization



Simple method...

```
class Bank
{
    double p, r;
    int n;
    double simpleInterest()
    {
        return(p*r*n/100);
    }
}
```

```
class Bank_Demo
{
    public static void main(String[] args)
    {
        Bank b1 = new Bank();
        b1.p = 5000; b1.r = 12; b1.n = 5;
        double si = b1.simpleInterest();
        System.out.println("Simple interest is:" + si);
    }
}
```



Parameterized method

```
class Bank
{
    double p, r;
    int n;
    void setData(double pa, double ra, int ny)
    { p = pa; r = ra; n = ny; }
    double simpleInterest()
    {
        return(p*r*n/100);
    }
}
```

```
class Bank_Demo
```

```
{
    public static void main(String[] args)
    {
        Bank b1 = new Bank();
        Bank b2 = new Bank();
        b1.setData(5000,12 ,5);
        b2.setData(10000,12 ,5);
        double si = b1.simpleInterest();
        System.out.println("Simple interest is:" + si);
        System.out.println("Simple interest is:" +
            b2. simpleInterest());
    }
}
```



Method Overloading

- If a class has **multiple methods having same name but different Signature (parameters)** known as Method Overloading.

Different ways to overload the method

- By changing number of arguments
- By changing the data type



By changing the no. of arguments

```
class Calculation
{
    void sum(int a,int b)
    {
        System.out.println(a+b);
    }
    void sum(int a,int b,int c)
    {
        System.out.println(a+b+c);}
}
class CalDemo
{
    public static void main(String args[])
    {
        Calculation c=new Calculation();
        c.sum(10,10,10);
        c.sum(20,20);
    }
}
```



By changing data type....

```
class Calculation
{
    void sum(int a,int b)
    {
        System.out.println(a+b);
    }
    void sum(double a, double b)
    {
        System.out.println(a+b);
    }
    void sum(float a, float b)
    {
        System.out.println(a+b);
    }
}
```

```
class CalDemo
{
    public static void main(String args[])
    {
        Calculation c=new Calculation();
        c.sum(10,20);
        c.sum(20.5,30.5);
        c.sum(10.5f,20.6);
    }
}
```



Constructor in Java

- **Constructor in java** is a *special type of method* that is used to initialize the object.
- Constructor is *invoked at the time of object creation*.
- It constructs the values(provides data) for the object that is why it is known as constructor.
- **Rules for creating java constructor**
 1. Constructor name must be same as its class name
 2. Constructor must have no explicit return type
- **Types of java constructors**
 1. Default constructor (no-argument constructor)
 2. Parameterized constructor



Java Default Constructor

- A constructor that have no parameter is known as default constructor.
- Default constructor provides the default values to the object like 0, null etc. depending on the type.
- Syntax of default constructor:

```
<class_name>()  
{  
    -----  
}
```


- **Rule:** If there is no constructor in a class, **compiler** automatically creates a default constructor.



Example...

```
class Bank
{
    double p, r;
    int n;
    Bank()
    {
        p = 5000; r = 12; n = 5;
    }
    double simpleInterest()
    {
        return(p*r*n/100);
    }
}
```

```
class Bank_Demo
{
    public static void main(String[] args)
    {
        Bank b1 = new Bank();
        Bank b2 = new Bank();
        double si = b1.simpleInterest();
        System.out.println("Simple interest is:" + si);
        si = b2.simpleInterest();
        System.out.println("Simple interest is:" + si);
    }
}
```

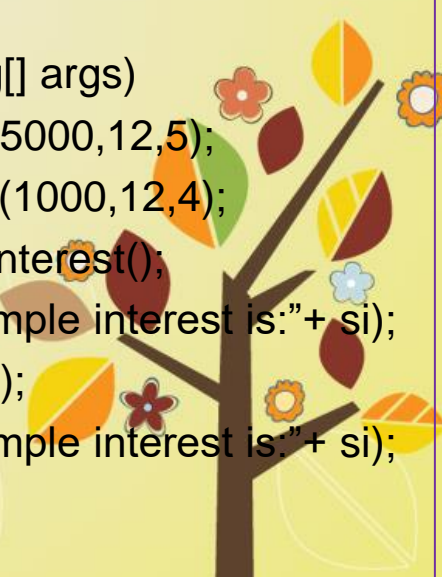


Parameterized Constructor

```
class Bank
{
    double p,r;
    int n;
    Bank(double pa, double ra, int ny)
    {
        p = pa; r = ra; n = ny;
    }
    double simpleInterest()
    {
        return(p*r*n/100);
    }
}
```

```
class Bank_Demo
```

```
{
    public static void main(String[] args)
    {
        Bank b1 = new Bank(5000,12,5);
        Bank b2 = new Bank(1000,12,4);
        double si = b1.simpleInterest();
        System.out.println("Simple interest is:" + si);
        si= b2. simpleInterest();
        System.out.println("Simple interest is:" + si);
    }
}
```



Can't Do...Compilation Error

```
class Student
{
    int id; String name;
    Student(int i,String n) { id = i; name = n; }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[])
    {
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");

        Student s3 = new Student(); X
        s1.display();
        s2.display();
        s3.display();
    }
}
```



Constructor Overloading in Java

```
class Student
{
    int id, age;
    String name;
    Student( )
    {
        id = 0; name = " "; age=0
    }
    Student(int i, String n)
    {
        id = i; name = n; age=18;
    }
    Student(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[])
{
    Student s1 = new Student(1,"XYZ");
    Student s2 = new Student(2,"ABC",25);
    Student s3 = new Student();
    s1.display();
    s2.display();
    s3.display();
}
}
```



Copy Constructor

```
class Bank
{
    double p,r;
    int n;
    Bank() //default constructor
    {
        p = 1000; r = 10; n = 5;
    }
    Bank(double pa, double ra, int ny) //Parameterized
    constructor
    {
        p = pa; r = ra; n = ny;
    }
    Bank(Bank b) // copy constructor
    {
        p = b.p; r = b.r; n = b.n;
    }
    double simpleInterest()
    {
        return(p*r*n/100);
    }
}
```

```
class Bank_Demo
{
    public static void main(String[] args)
    {
        Bank b1 = new Bank();
        Bank b2 = new Bank(1000,12,4);
        Bank b3 = new Bank(b2);
        Bank b4 = b1;
        double si = b1.simpleInterest();
        System.out.println("Simple interest is:" + si);
        si = b2. simpleInterest();
        System.out.println("Simple interest is:" + si);
        si = b3. simpleInterest();
        System.out.println("Simple interest is:" + si);
        si = b4. simpleInterest();
        System.out.println("Simple interest is:" + si);
    }
}
```

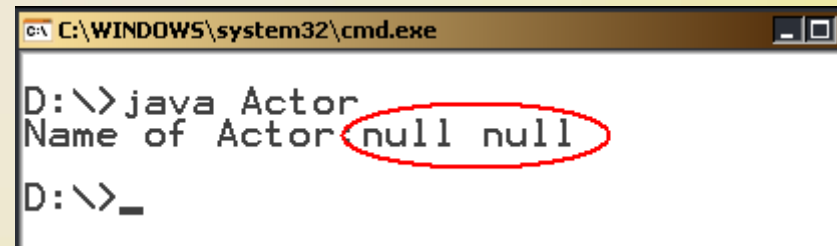


Use of **this** keyword

- **this** is a keyword in Java. Which can be used inside *method* or *constructor* of class.
- It(**this**) works as a reference to current object whose method or constructor is being invoked.

Example...(without this)

```
class Actor
{
    String lastname;
    String firstname;
    Actor(String lastname, String firstname)
    {
        lastname = lastname;
        firstname = firstname;
    }
    public static void main(String args[])
    {
        Actor a = new Actor("Patel","Saumya");
        System.out.println("Name of Actor:"+a.firstname
                           +" "+a.lastname);
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
D:\>java Actor
Name of Actor null null
D:\>_
```

Example...(with **this** keyword)

```
class Actor
```

```
{
```

```
    String lastname;
```

```
    String firstname;
```

```
    Actor(String lastname, String firstname)
```

```
{
```

```
        this.lastname = lastname;
```

```
        this.firstname = firstname;
```

```
}
```

```
    public static void main(String args[])
```

```
{
```

```
        Actor a = new Actor ("Patel","Saumya");
```

```
        System.out.println("Name of Actor:"+a.firstname  
                             +" "+a.lastname);
```

```
}
```

```
}
```

```
D:\>javac Actor.java
```

```
D:\>java Actor
```

```
Name of Actor:Saumya Patel
```



Difference between constructor and method

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

Instance Variable

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



Static Keyword

- The **static keyword** in java is used for memory management mainly.
- A variable or method which is declared with static keyword also known as Class variable or Class method because there is no need of object (instance) to use these.
- The static can be:
 - variable (also known as class variable)
 - method (also known as class method)
 - block
 - nested class
- If any variable declared as static, it is known static variable.
- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.



Ex. of Counter without static

```
class Counter_Demo
{
    int count=0;// will get memory when instance is created
    Counter_Demo()
    {
        count++;
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Counter_Demo c1=new Counter_Demo();
        Counter_Demo c2=new Counter_Demo();
        Counter_Demo c3=new Counter_Demo();
    }
}
```

Output:

1
1
1



Ex. of Counter with static

```
class Counter_Demo
{
    static int count=0;//memory only once and retain its value
    Counter_Demo()
    {
        count++;
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Counter_Demo c1=new Counter_Demo();
        Counter_Demo c2=new Counter_Demo();
        Counter_Demo c3=new Counter_Demo();
    }
}
```

Output:

1
2
3



Example of static variable

```
class Student
{
    int rollno;
    String name;
    static String college = "VSC";
    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(10,"XYZ");
        Student s2 = new Student(20,"ABC");
        s1.display();
        s2.display();
    }
}
```



Java static method

- If you apply **static** keyword with any method, it is known as static method.
- A static method can be invoked without the creating an instance (object) of a class.
- static method can access static data member and can change the value of it.
- **Note:** Static method can not access instance members of class.



Example...

```
class Calculate
{
    static int cube(int x)
    {
        return x*x*x;
    }
}

class Demo
{
    public static void main(String args[])
    {
        int result=Calculate.cube(5);
        System.out.println(result);
        result=Calculate.cube(10);
        System.out.println(result);
    }
}
```

Output:

125
1000



Static and non Static member

class A

```
{  
    int a=40;//non static  
    public static void main(String args[])  
    {  
        System.out.println(a);  
    }  
}
```

=====

class A

```
{  
    static int a=40;//static  
    public static void main(String args[])  
    {  
        System.out.println(a);  
    }  
}
```

Output:

Compile time error

Output:

40



Variable Argument (Varargs):

- The varargs allows the method to accept zero or multiple arguments.
- varargs is the better approach if we don't know how many arguments we will have to pass in the method.
- Syntax:

```
accessModifier methodName(datatype... arg)
{
    // method body
}
```

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

Examples of varargs that fail to compile:

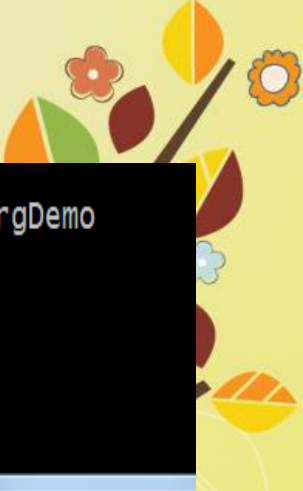
```
void method(String... a, int... b){} //Compile time error
void method(int... a, String b){} //Compile time error
```



Example:

```
class VarArgs
{
    void sum(int ... n)
    {
        int sum = 0;
        for(int a : n)
        {
            sum=sum+a;
        }
        System.out.println("Sum of "+n.length+ " nos = "+sum);
    }
}
```

```
class VarArgDemo
{
    public static void main(String args[])
    {
        VarArgs obj = new VarArgs();
        obj.sum();
        obj.sum(10,20);
        obj.sum(10,20,30,40);
        obj.sum(10,20,30,40,5,5);
    }
}
```



```
D:\B2_Java\X_Batch>java VarArgDemo
Sum of 0 nos = 0
Sum of 2 nos = 30
Sum of 4 nos = 100
Sum of 6 nos = 110
D:\B2_Java\X_Batch>_
```


Access Specifiers

- Java Access Specifiers (Visibility Specifiers) regulate access to classes, attributes and methods in Java.
- It determine whether attributes or methods in a class, can be used or invoked by another method in another class or sub-class.
- Access Specifiers can be used to restrict access.
- Access Specifiers are part of object-oriented programming.
- **Types Of Access Specifiers :**
 1. public
 2. private
 3. protected
 4. default(no specifier)



Access Specifiers...

Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes