Object Oriented Programming with JAVA AMCA0201N

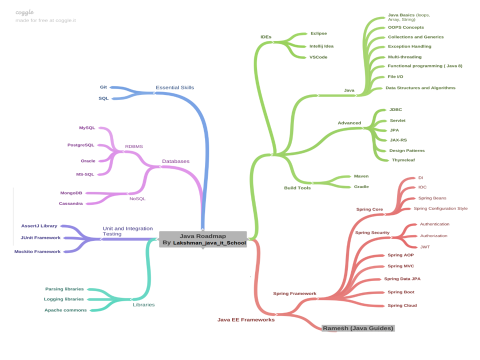
Unit 1 Introduction

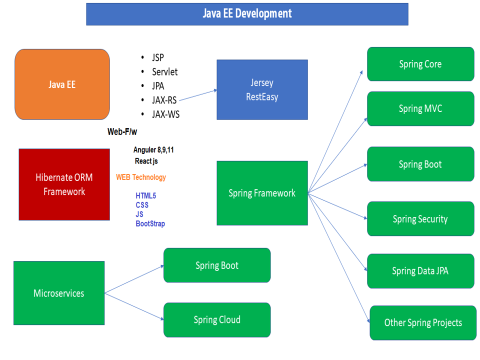
**Object Oriented Programming**: Introduction and Features: Abstraction, Encapsulation, Polymorphism, and Inheritance concepts, Need of OOP’s paradigm.

**Modeling Concepts:** Introduction, Class Diagram and Object Diagram.

**Control Statements:** Decision Making, Looping and Branching, Argument Passing Mechanism: Command Line Argument



. 



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  | |
|  | | Java introduction  June 1991. small team of sun engineers called **Green Team**. | |
| Auther |  | : | ***James Gosling*  Mike Sheridan**, and **Patrick Naughton** | |
| Vendor | | : | ***Sun Micro System(which has since*** [***merged into Oracle Corporation***](http://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle) in 2010 ***)*** | |
| Project name | | : | ***Green Project*** | |
| Type | | : | ***open source & free software*** | |
| Initial Name | | : | *OAK language* | |
| Present Name | | : | *Java* In 1995, Oak was renamed as **"Java"** | |
| Extensions | | : | *.java & .class & .jar* | |
| Initial version | | : | *jdk 1.0 (java development kit)* | |
| Present version | | : | Java 17 LTS long-term support **Sept 15, 2021.** | |
| Operating System | | : | *multi Operating System* | |
| Implementation Lang | | : | *c, cpp……* | |
| Symbol | | : | *coffee cup with saucer* | |
| Objective | | : | *To develop web applications* | |
| SUN | | : | ***Stanford Universally Network*** | |
| Slogan/Motto | | : | *WORA(write once run anywhere)* | |

Published in Time magazine called **Java one of the Ten Best Products of 1995**

## **OOPs (Object-Oriented Programming System)**

**Object-Oriented Programming** or **OOPs** refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc

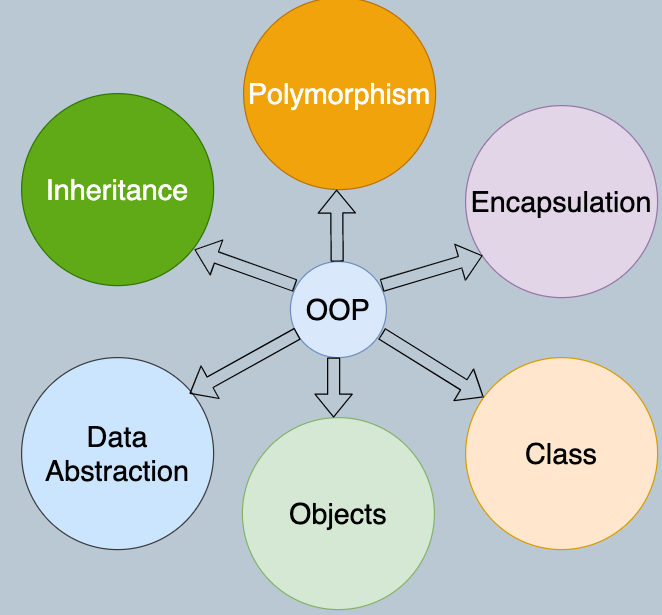
**OOPs concepts in Java**) is a programming paradigm/model based on the concept of “objects”. Object-Oriented Programming is a methodology to design a program using classes and objects.

OOPs. Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

Java OOPs concepts provide several advantages such as security, reusability, effective communication, developing complex software, maintenance, and efficiency.

.



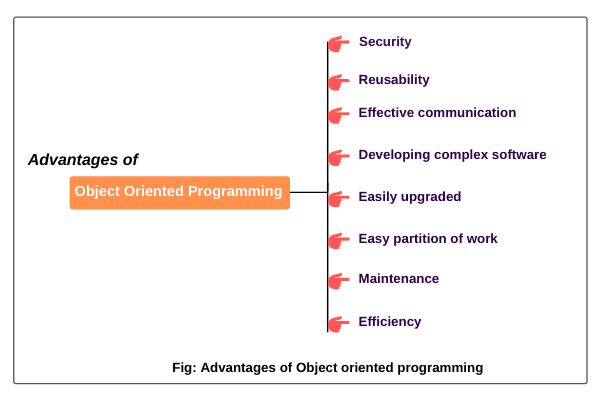
* [**Object**](https://www.javatpoint.com/object-and-class-in-java)
* **Class**
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition

### What are the benefits of OOP?

Benefits of OOP include:

* **Modularity.** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
* **Reusability.** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
* **Productivity.** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
* **Easily upgradable and scalable.**Programmers can implement system functionalities independently.
* **Interface descriptions.** Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
* **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and [internet protocols](https://www.techtarget.com/searchunifiedcommunications/definition/Internet-Protocol) are protected.
* **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface
* Code can be reused.
* Provides security through encapsulation and data hiding features.
* Debugging is easy[](https://www.scientecheasy.com/2020/07/oops-concepts-in-java.html/)

## **Advantages of OOPs Concepts in Java**

* Reduces code redundancy
* Improves code readability
* Low development cost
* Improved software quality
* Faster product development
* Reusability of code
* Polymorphism offers a lot of flexibility in OOPs
* OOPs is very helpful in solving very complex level of problems.

### **Features of OOP's Language**

* Better memory management
* Suitable for large projects
* Fairly efficient languages
* It implements real life scenario
* It is easy to maintain and modify existing code
* Implementation details are hidden from other modules

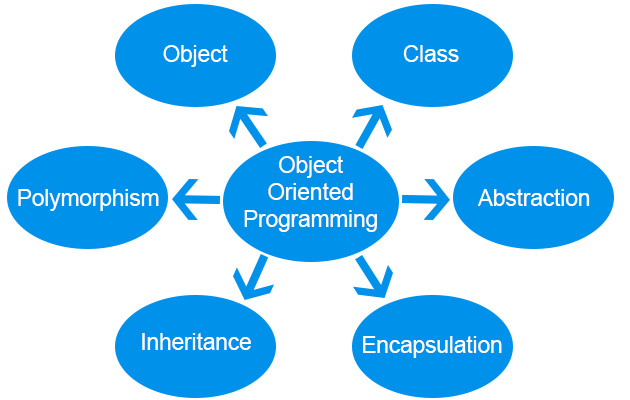
## **Pillars of OOPs**

The major concepts that we have discussed above are known as **pillars of OOPs**. There are **four** pillars on which OOP rests.

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

## **Pillars of OOPs**

* **Objects**
* **Classes**
* **Abstract class**
* Encapsulation
* **Inheritance**
* **Polymorphismc**



### **Object**

The Object is the real-time entity having some state and behaviour. In Java, Object is an instance of the class having the instance variables as the state of the object and the methods as the behaviour of the object. The object of a class can be created by using the new keyword in Java Programming language.

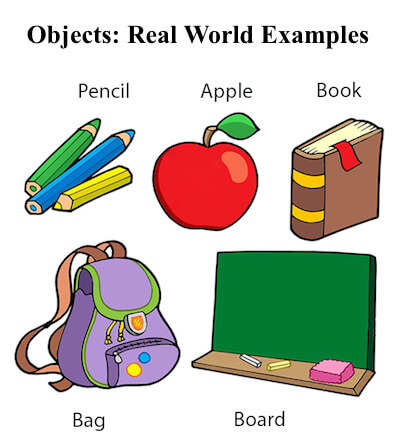
A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**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 and* **Identity**.
* The object is *an instance of a class*.
* An object is physical *entity*
* The new operator is used to dynamically allocates memory for an object

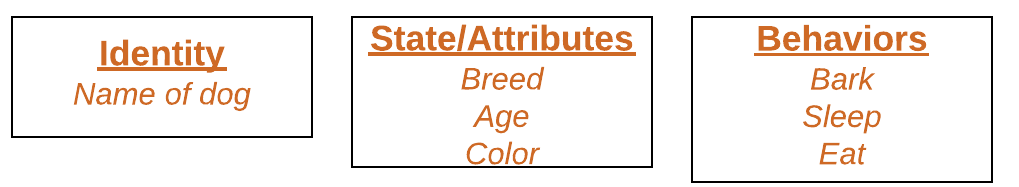
### Real-world examples

## **Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Chair, Bike, Marker, Pen, Table, Car, Book, Apple, Bag etc. It can be physical or logical (tangible and intangible).**

* chair, pen, table, keyboard, bike, etc
* 

## **Java Object**

* **State:** represents the data (value) of an object.
* **Behaviour:** represents the behaviour (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



They are characterized bys three features:

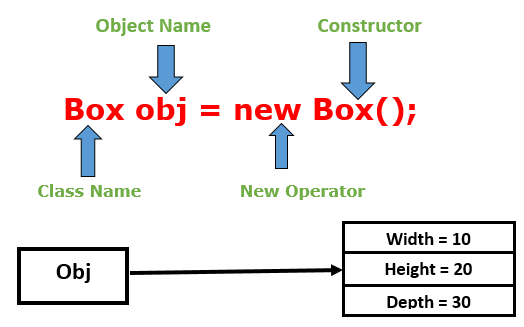
* Identity
* State
* Behavior

There are three steps to creating a Java object:

* Declaration of the object.
* Instantiation of the object.
* Initialization of the object.

Class c =new Class ();

Box obj = new Box();  
  
Here we can create an object of **Box class** and obj is a reference to an object

[](https://3.bp.blogspot.com/-Tc5TLF5W30Q/XM6V6YJleJI/AAAAAAAAC9s/m34II1aZ5dQhECsPxMJRYCPvrQnKIbmJgCLcBGAs/s1600/Screenshot+(380).png)

**Object:** Student  
**State:** name, age, gender  
**Behavior:** study, play, run

**Example**:

**Object**: Mobile

**State**: Name, Model No, Colour

**Behavior**: Calling, messaging, switch on, switch off, take photos, etc.

Syntax:

**ClassName ObjectName = new classConstructor( );**

**Mybook Myobj= new Mybook ();**

 This is the statement used for creating objects.

**System.out.println(Myobj.x);**

public class ObjectCreation

{

String FirstString = "Hello World";

public static void main(String[] args)

{

ObjectCreation obj = new ObjectCreation();

System.out.println(obj.FirstString);

}

}

class Box{

//declare member variables

double width;

double height;

double depth;

}

class BoxVolume{

public static void main(String args[]){

//creating an object of Box class

Box obj = new Box();

double vol;

//assign values to obj instance variable

obj.width = 10;

obj.height = 20;

obj.depth = 30;

//compute volume of Box

vol = obj.width \* obj.height \* obj.depth;

//printing the volume of Box

System.out.println("Volume of Box is: " +vol);

}

}

## **How to Declare, Create and Initialize an Object in Java**

A class is a blueprint for Object, you can create an object from a class. Let's take Student class and try to create Java object for it.  
  
Let's create a simple *Student* class which has *name* and *college* fields. Let's write a program to create declare, create and initialize a *Student* object in Java.

public class Student {

    private String name;

    private String college;

    public Student(String name, String college) {

        super();

        this.name = name;

        this.college = college;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getCollege() {

        return college;

    }

    public void setCollege(String college) {

        this.college = college;

    }

    public static void main(String[] args) {

        Student student = new Student("Ramesh", "BVB");

        Student student2 = new Student("Prakash", "GEC");

        Student student3 = new Student("Pramod", "IIT");

    }

}

*Student*objects are:

Student student = new Student("Ramesh", "BVB");

Student student2 = new Student("Prakash", "GEC");

Student student3 = new Student("Pramod", "IIT");

**Declaration:**The code *Student student;* declarations that associate a variable name with an object type.

**Instantiation:**The *new* keyword is a Java operator that creates the object.

**Initialization:** The *new* operator is followed by a call to a constructor, which initializes the new object.

### Declaring a Variable to Refer to an Object

General syntax:

type name;

string name;

int rollNum;

### Initializing an Object

The *new* keyword is followed by a call to a constructor, which initializes the new object. For example:

Student student = new Student("Ramesh", "BVB");

Student student2 = new Student("Prakash", "GEC");

Student student3 = new Student("Pramod", "IIT");

### Instantiating a Class

## **The *new* operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The *new* operator also invokes the object constructor.**

## **For example:**

Student student = new Student("Ramesh", "BVB");

Student student2 = new Student("Prakash", "GEC");

Student student3 = new Student("Pramod", "IIT");

## 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.

## What is Class?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. In short, **a class is the specification or template of an object**.

A **class** is a blueprint/template that defines the state and behaviour of objects. It is the logical entity that wraps all our code into a single unit and structures the code. In Java, we cannot compile our code without a **class**.

1-Class is a set of object which shares common characteristics/ behaviour and common properties/ attributes.  
2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.  
3. Class does not occupy memory.  
4. Class is a group of variables of different data types and group of methods.

A class is a template or blueprint for creating objects at run time. It is a logical entity that is a collection of objects. The class consists of fields and methods

* Keyword "**class**" is used in declaration of a class.

* class\_name is any valid java identifiers.

**Syntax of Class in Java**

class <class\_name>{

field;

method;

}

## Declaring Classes

You've seen classes defined in the following way:

class MyClass {

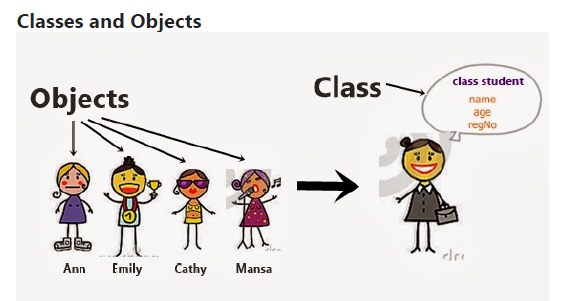
// field, constructor, and

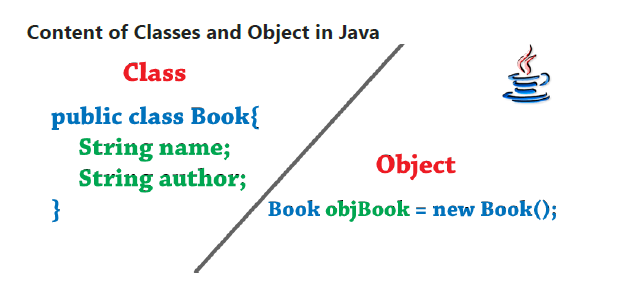
// method declarations

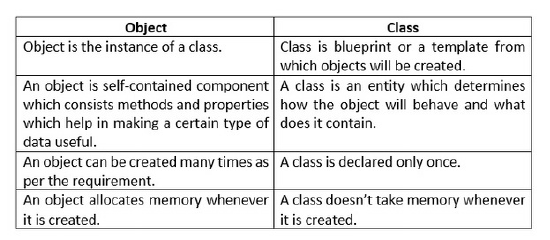
}

**class** **Athlete** {  
**public** String athleteName;  
**public** **double** athleteSpeed;  
**public** **int** athleteAge;  
}

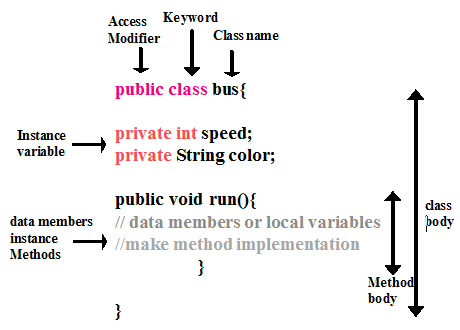
* Member variables in a class—these are called *fields*.
* Variables in a method or block of code—these are called *local variables*.
* Variables in method declarations—these are called *parameters*.

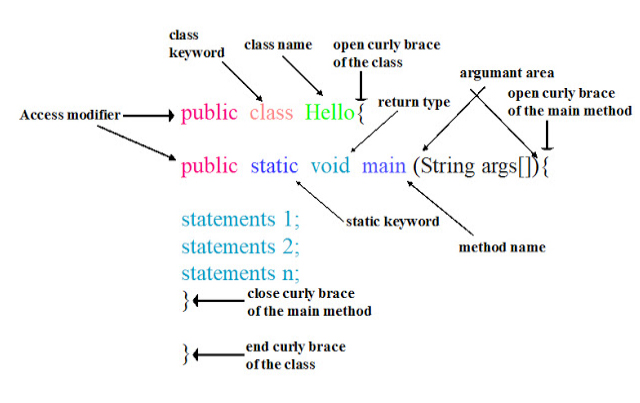


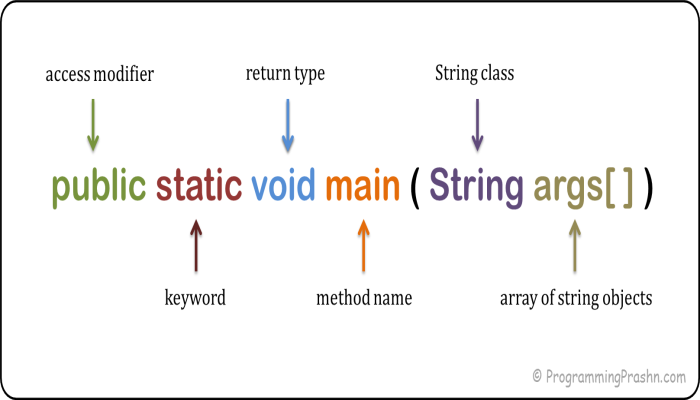


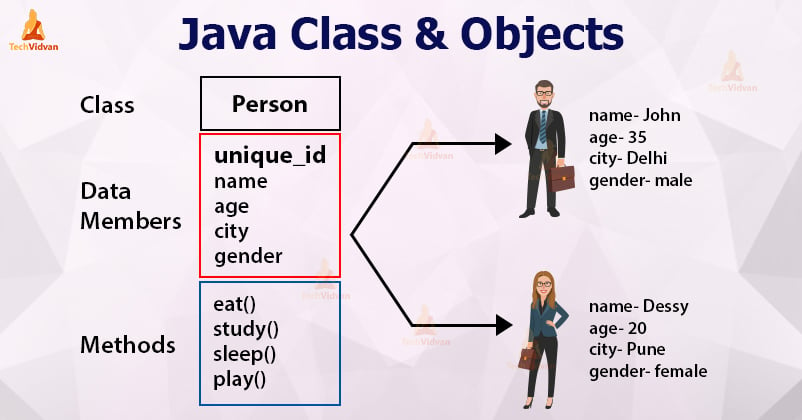












# Difference between object and class

There are many differences between object and class. A list of differences between object and class are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Object** | **Class** |
| 1) | Object is an **instance** of a class. | Class is a **blueprint or template** from which objects are created. |
| 2) | Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. | Class is a **group of similar objects**. |
| 3) | Object is a **physical** entity. | Class is a **logical** entity. |
| 4) | Object is created through **new keyword** mainly e.g. Student s1=new Student(); | Class is declared using **class keyword** e.g. class Student{} |
| 5) | Object is created **many times** as per requirement. | Class is declared **once**. |
| 6) | Object **allocates memory when it is created**. | Class **doesn't allocated memory when it is created**. |
| 7) | There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization. | There is only **one way to define class** in java using class keyword. |

**Example:**

Class: Person

Object: Male or Female

State: Name, Age

Behavior: Eating, Walking, Cooking, etc

1. **class** <class\_name>{
2. field;
3. method;
4. }

A class in java can contain:  
• data member  
• method  
• constructor  
• nested class and   
• interface

Syntax to declare a class:

access\_modifier class<class\_name>

{

data member;

method;

constructor;

nested class;

interface;

}

class <class\_name>{

    field;

    method;

}

1. **Modifiers**: A class can be public or has default access (Refer [this](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/) for details).
2. **Class keyword:**class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body is surrounded by braces, { }.

class Student {

    int id; // data member (also instance variable)

    String name; // data member (also instance variable)

    public static void main(String args[])

    {

        Student s1 = new Student(); // creating an object of

                                    // Student

        System.out.println(s1.id);

        System.out.println(s1.name);

    }

}

//Java Program to illustrate how to define a class and fields

//Defining a Student class.

class Student{

    //defining fields

    int id;//field or data member or instance variable

    String name;

    //creating main method inside the Student class

    public static void main(String args[]){

     //Creating an object or instance

     Student s1=new Student();//creating an object of Student

     //Printing values of the object

     System.out.println(s1.id);//accessing member through reference variable

     System.out.println(s1.name);

    }

   }

//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);

    }

   }

class Student{

    int id;

    String name;

   }

   class TestStudent3{

    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);

    }

   }

public class Person **{**

//Attributes

String name;

int age;

Person**(**String name, int age**){**

this.name = name;

this.age = age;

**}**

public void eating**()** **{**

System.out.println**(**"Eating"**)**;

**}**

public void walking**()** **{**

System.out.println**(**"Walking"**)**;

**}**

public static void main**(**String**[]** args**)** **{**

Person male = new Person**(**"Dev",35**)**;

Person female = new Person**(**"Devika",20**)**;

System.out.println**(**male.name + " " + male.age**)**;

male.eating**()**;

System.out.println**(**female.name + " " + female.age**)**;

female.walking**()**;

**}**

**}**

* In Java, we can not declare a top-level class as private. Java allows only public and default access specifiers for top-level classes. We can declare inner classes as private.
* We can include any type of the three ***variables in Java*** –*local, instance and static variables.*
* There can be only one public class in a single program and its name should be the same as the name of the Java file. There can be more than one non-public classes in a single Java file.
* A public class is visible to all classes from all the packages.
* A class with default access is visible only to the classes within the same package.
* We can also use the non-access modifiers for the class such as final, abstract and strictfp.
* We cannot create an object or instance of an abstract class.
* No subclasses or child class can be created from a class that is declared as final.
* A class cannot be declared both as final and abstract at the same time.

### **new keyword in Java**

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

### **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

1. **class** Student{
2. **int** rollno;
3. String name;
4. **void** insertRecord(**int** r, String n){
5. rollno=r;
6. name=n;
7. }
8. **void** displayInformation(){System.out.println(rollno+" "+name);}
9. }
10. **class** TestStudent4{
11. **public** **static** **void** main(String args[]){
12. Student s1=**new** Student();
13. Student s2=**new** Student();
14. s1.insertRecord(111,"Karan");
15. s2.insertRecord(222,"Aryan");
16. s1.displayInformation();
17. s2.displayInformation();
18. }
19. }



### **Object and Class Example: Employee**

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

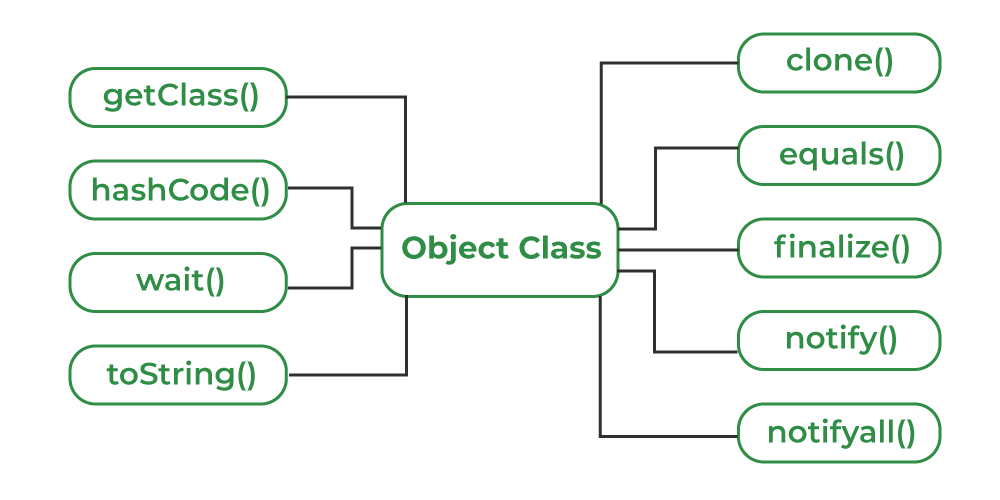
*File: TestEmployee.java*

1. **class** Employee{
2. **int** id;
3. String name;
4. **float** salary;
5. **void** insert(**int** i, String n, **float** s) {
6. id=i;
7. name=n;
8. salary=s;
9. }
10. **void** display(){System.out.println(id+" "+name+" "+salary);}
11. }
12. **public** **class** TestEmployee {
13. **public** **static** **void** main(String[] args) {
14. Employee e1=**new** Employee();
15. Employee e2=**new** Employee();
16. Employee e3=**new** Employee();
17. e1.insert(101,"ajeet",45000);
18. e2.insert(102,"irfan",25000);
19. e3.insert(103,"nakul",55000);
20. e1.display();
21. e2.display();
22. e3.display();
23. }
24. }

## **java.lang.Object Class in Java**

**Object** class is present in **java.lang** package. Every class in Java is directly or indirectly derived from the **Object** class. If a class does not extend any other class then it is a direct child class of **Object**

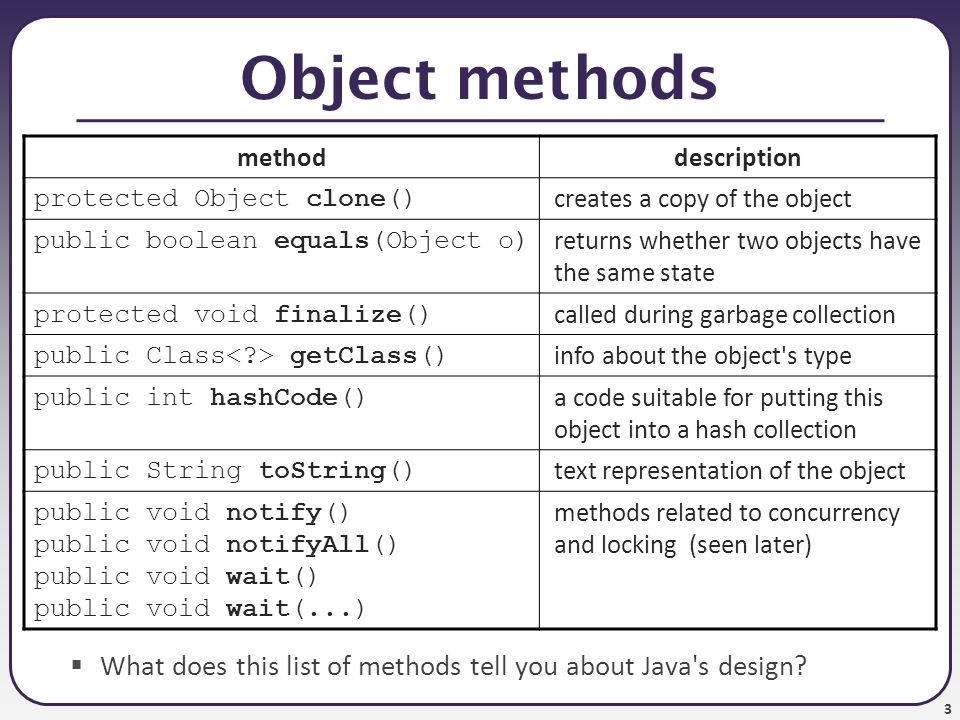
11 methods



## Using Object Class Methods

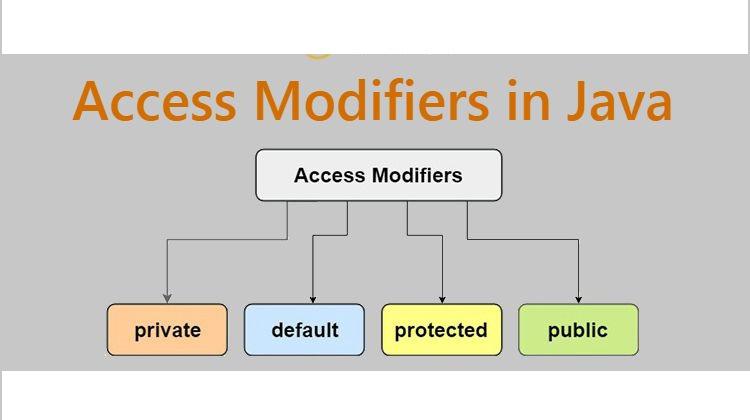
The Object class provides multiple methods which are as follows:

* tostring() method
* hashCode() method
* equals(Object obj) method
* finalize() method
* getClass() method
* clone() method
* wait(), notify() notifyAll() methods



# Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.



1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| Default | declarations are visible only within the package (package private) |
| Private | declarations are visible within the class only |
| Protected | declarations are visible within the package or all subclasses |
| Public | declarations are visible everywhere |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

1. class Data {
2. // private variable
3. private String name;
4. }
5. public class Main {
6. public static void main(String[] main){
7. // create an object of Data
8. Data d = new Data();
9. // access private variable and field from another class
10. d.name = "Programiz";
11. }
12. }

#### **Abstraction in Java**

* Abstraction is a process to represent only “relevant” or essential data and “hide” the unnecessary or background details of an object from the user.
* Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It’s used to create a boundary between the application and the client programs.
* Java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

The abstract method contains only method declaration but not implementation.

### **We can achieve abstraction in two ways:** a) Abstract Class b) Interface

##### **Abstract class**

An Abstract class in Java uses the ‘**abstract’** keyword. If we declare a class as abstract, we cannot instantiate it, which means we cannot create an abstract class object. Also, In an abstract class, there can be abstract as well as concrete methods.

We can achieve 0-100% abstraction using abstract class.

abstract class Person //abstract class

{

abstract void talk(); //abstract method

void walk() //non-abstract method

{

//code of method

}

}

##### **Interface**

An interface is a fully abstract class. It includes a group of abstract methods (methods without a body).

Interface is a blueprint of a class. An interface is a collection of abstract methods and static constants. Each method in an interface is public and abstract, but there is no constructor. Interfaces also help to achieve multiple inheritance in Java.

In Java, an **interface** is a blueprint or template of a class. It is much similar to the Java class but the only difference is that it has abstract methods and static constants

We can achieve 100% abstraction using interfaces.

We use the **interface** keyword to create an interface in Java

* All the methods in an interface should be declared as abstract.
* An interface is implicitly abstract. While declaring an interface, you do not need to use the keyword abstract.
* Each method of an interface is also implicitly abstract, so we need not use the abstract keyword while declaring methods inside an interface.
* Each method in an interface is implicitly public.
* An interface cannot contain instance fields. It can only contain the fields that are declared as both static and final.
* An interface can not be extended or inherited by a class; it is implemented by a class.
* An interface cannot implement any class or another interface.

**It cannot be instantiated just like the abstract class.**

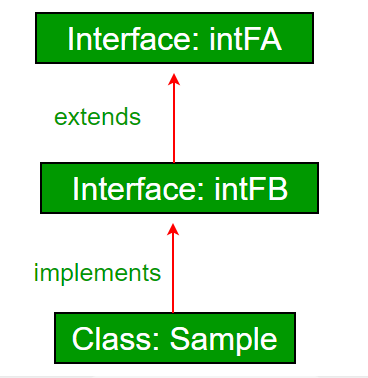
Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.



### **Syntax:**

1. **interface** <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

## **Internal addition by the compiler**

#### **The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.**

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



interface Language {

public void getType();

public void getVersion();

}

* Language is an interface.
* It includes abstract methods: getType() and getVersion().

interface interface1

{

    void method1();

}

interface interface2 extends interface1

{

    void method2();

}

public class ConcreteClass implements interface2{

    @Override

    public void method1() {

        System.out.println("In method 1");

    }

    @Override

    public void method2() {

        System.out.println("In method 2");

    }

}

* public class SimpleRoom implements Room {
* @Override
* public void showRoom() {
* System.out.println("Showing simple room");
* }
* }
* public class SpecialRoom implements Room {
* @Override
* public void showRoom() {
* System.out.println("Showing special Room");
* }
* }
* public class RoomInterfaceMain {
* public static void main(String[] args) {
* // You can assign interface reference variable to concrete implementation
* Room room=new SimpleRoom();
* room.showRoom();
* System.out.println("=================");
* // You can easily change implementation
* room=new SpecialRoom();
* room.showRoom();
* }
* }

## Abstract class vs Interface

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Abstract class** | **Interface** |
| Default method Implementation | It can have default method implementation | Interfaces are pure abstraction.It can not have implementation at all but in java 8, you can have default methods in interface. |
| Implementation | Subclasses use **extends**keyword to extend an abstract class and they need to provide implementation of all the declared methods in the abstract class unless the subclass is also an abstract class | subclasses use **implements**keyword to implement interfaces and should provide implementation for all the methods declared in the interface |
| Constructor | Abstract class can have constructor | Interface  can not have constructor |
| Different from normal java class | Abstract classes are almost same as java classes except you can not instantiate it. | Interfaces are altogether different type |
| Access Modifier | Abstract class methods can have public ,protected,private and default modifier | Interface methods are by default public. you can not use any other access modifier with it |
| Main() method | Abstract classes can have main method so we can run it | Interface do not have main method so we can not run it. |
| Multiple inheritance | Abstract class can extends one other class and can implement one or more interface. | Interface can extends to one or more interfaces only |
| Speed | It is faster than interface | Interface is somewhat slower as it takes some time to find implemented method in class |
| Adding new method | If you add new method to abstract class, you can provide default implementation of it. So you don’t need to change your current code | If you add new method to interface, you have to change the classes which are implementing that interface |

.

## **Data Hiding**

Data hiding is a way of restricting the access of our data members by out of the class by using private keyword achieve data hiding the implementation details. Encapsulation also provides a way for data hiding.

Data hiding using the private specifier

### **Encapsulation**

**Encapsulation in Java** is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

The **Java Bean** class is the example of a fully encapsulated class.

Encapsulation in java is the process of **binding related data(variables) and functionality(methods)** into a single unit called class. Encapsulation can be achieved by using [access modifier](https://java2blog.com/access-modifiers-java/) such as public, private, protected or default, so your class will be safe from unauthorized access by others

We can create fully encapsulated class by

* Making variables private
* Providing getters and setters methods for the accessing the variables.

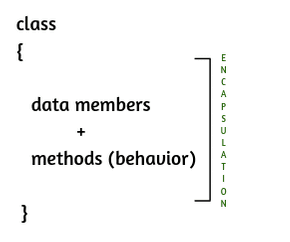
Encapsulation is also termed as data hiding because you are making variables private and variables can be only accessed through public getters and setters.

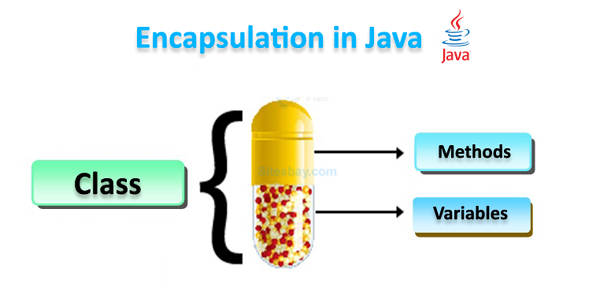
**We can implement Encapsulation in two ways:**

**1.** Declare the instance variables as private. We make them private, so no one from outside the class can access them directly. We can only set and get the values of these variables using the methods of the class.

**2.** Provide the getter and setter methods in the class. These methods set and get the values of the instance variables.

Now, let us see an example if Encapsulation in Java:





//A Java class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

package com.javatpoint;

public class Student{

//private data member

private String name;

//getter method for name

public String getName(){

return name;

}

//setter method for name

public void setName(String name){

this.name=name

}

}

//A Java class to test the encapsulated class.

package com.javatpoint;

class Test{

public static void main(String[] args){

//creating instance of the encapsulated class

Student s=new Student();

//setting value in the name member

s.setName("vijay");

//getting value of the name member

System.out.println(s.getName());

}

}

class Area {

// fields to calculate area

int length;

int breadth;

// constructor to initialize values

Area(int length, int breadth) {

this.length = length;

this.breadth = breadth;

}

// method to calculate area

public void getArea() {

int area = length \* breadth;

System.out.println("Area: " + area);

}

}

class Main {

public static void main(String[] args) {

// create object of Area

// pass value of length and breadth

Area rectangle = new Area(5, 6);

rectangle.getArea();

}

}

public class Employee{

    private int employeeId;

    private String employeeName;

    private String department;

    public int getEmployeeId() {

        return employeeId;

    }

    public void setEmployeeId(int employeeId) {

        this.employeeId = employeeId;

    }

    public String getEmployeeName() {

        return employeeName;

    }

    public void setEmployeeName(String employeeName) {

        this.employeeName = employeeName;

    }

    public String getDepartment() {

        return department;

    }

    public void setDepartment(String department) {

        this.department = department;

    }

    public static void main(String args[])

    {

        Employee e1=new Employee();

        e1.setEmployeeId(1);

        e1.setEmployeeName("John");

        e1.setDepartment("Sales");

        System.out.println("=============");

        System.out.println("Employee Id: "+e1.getEmployeeId());

        System.out.println("Employee Name: "+e1.getEmployeeName());

        System.out.println("Employee Department: "+e1.getDepartment());

    }

}

### Advantage of Encapsulation

The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

### Real life example of Encapsulation in Java

The common example of encapsulation is capsule. In capsule all medicine are encapsulated in side capsule.

## Benefits of encapsulation

* Provides abstraction between an object and its clients.
* Protects an object from unwanted access by clients.
* Example: A bank application forbids (restrict) a client to change an Account's balance.

## Difference between abstraction and encapsulation in java:

* Encapsulation means data hiding using getter and setters. [Abstraction](https://www.java2blog.com/2017/04/abstraction-java-example.html) means hiding implementation details using [abstract class](https://java2blog.com/abstract-class-java/ “abstract class”) and [interface](https://java2blog.com/interface-in-java-with-example/ “interface”).
* Abstraction is more of design level concept and Encapsulation is more of implementation level concept.

### **Inheritance**

The concept allows us to inherit or acquire the properties of an existing class (parent class) into a newly created class (child class). It is known as [**inheritance**](https://www.javatpoint.com/inheritance-in-java). It provides code reusability.

**Inheritance** is a process where child class acquired all the properties and behaviors of the parent class. Inheritance is used when one object is based on another object. Here parent class also called a superclass and child class called as a subclass.

**For Example,** Person is Parent class and Employee is a subclass of Person. which acquired all the properties and behavior of Person class.

***Advantage of inheritance***

* Code Reusability
* Runtime Polymorphism

***Points about Inheritance***

* **extends** the keyword used to implement inheritance.
* Java doesn’t support multiple inheritances. It’s possible by implementing multiple interfaces.

Inheritance has an **“IS-A”** relationship.

#### **Types of Inheritance in Java**

**1. Single Inheritance:** Single Inheritance is a child and parent class relationship where one class extends another class.

**2. Multilevel Inheritance:** Multilevel Inheritance is a child-parent relationship when a class extends the child class, and that child class becomes a parent class for another class, and so on. For example, class A extends class B, and class C extends class B.

**3. Hierarchical Inheritance:** Hierarchical Inheritance refers to a child-parent class relationship where more than one class can extend the same parent class. For example, class B extends class A, and class C extends class A.

**4. Multiple Inheritance:** Multiple Inheritance refers to a parent-child class relationship when one child class extends more than one parent class. This means, a child class can have more than one parent class. Java does not support multiple inheritance using classes, but with interfaces.

* Using inheritance, we can extend the functionalities of a class in another class.
* Using inheritance, we can define class in hierarchical order.

Inheritance is mainly used for code re-usability, you can reuse the field and methods of parent class and then add code for your specific requirement of child class.  
  
Keyword **extends** is used by child class to inherit the properties of parent class.

**Benefits of Inheritance**

* *Code re-usability.*
* *Hierarchical classification of objects.*
* *Runtime polymorphism by method overriding.*

Between a parent and child class, there exists a "**is a**" relationship, like "Apple is a Fruit". Here Apple class inherits some properties from Fruit class because an Apple is a Fruit.

class Animal {

   public String color;

   public float weight;

   public float length;

   public void walk() {

      // Code for walking

   }

}

class Tiger extends Animal {

   public boolean hasTail;

   public boolean isCarnivorous;

   public float runningSpeed;

   public void run() {

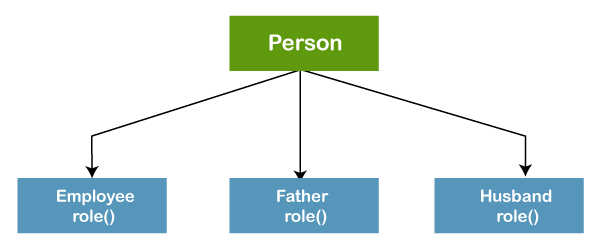
      // Do something

   }

}

### **Polymorphism**

The word [**polymorphism**](https://www.javatpoint.com/runtime-polymorphism-in-java) is derived from the two words i.e. **ploy** and **morphs**. Poly means many and morphs means forms. It allows us to create methods with the same name but different method signatures. It allows the developer to create clean, sensible, readable, and resilient code.



### Types of Polymorphism

There are 2 types of polymorphism:

* Static Polymorphism – This is also called as **compile-time** polymorphism.
* Dynamic Polymorphism –  This is called **runtime** polymorphism.

#### Static Polymorphism

When polymorphism is resolved during compile time, we call it as static polymorphism. One best example is **method overloading.**This means we can have multiple methods of the same name in a class but having different signatures(parameters).

##### **a. Static Polymorphism**

Polymorphism that the compiler resolves during the compile-time is called the static polymorphism. We can consider Method overloading as a static polymorphism example in Java.

Method Overloading allows us to have more than one method with the same name in a class with a different signature. The above example of polymorphism is the example of method overloading or static polymorphism in Java.

##### **b. Dynamic Polymorphism**

The other name for Dynamic Polymorphism is Dynamic Method Dispatch. Dynamic or runtime polymorphism is a technique in which the overridden method is resolved at runtime rather than the compile-time. That’s why it is called runtime polymorphism.

There are two types of polymorphism in java.

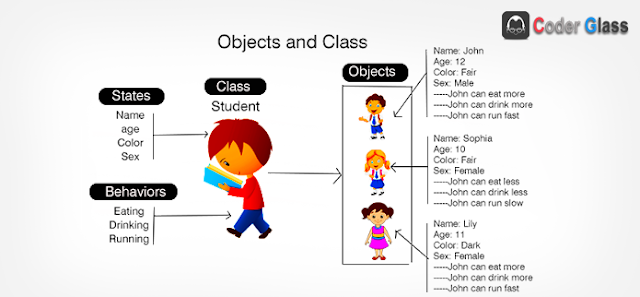
**Method overloading (compile time polymorphism)**

Here more than one methods have same names but different input parameters. They can either have different numbers of parameters or parameters of different types.

**Method overriding (run time polymorphism)**

Method overriding means a derived class is implementing a method of parent class. The method signature in derived class remains the same.

[](https://1.bp.blogspot.com/-fF4IcjEQbiE/XU-utqBUkSI/AAAAAAAAEnk/fZ38-T3S1BoBdPVpl8s0ur6aBa637fKsgCLcBGAs/s1600/Screenshot+(8).png)

[](https://1.bp.blogspot.com/-pFHJTng93q4/XU-utsYfSjI/AAAAAAAAEno/43JhWi3SrVAVmXc3UyKmhyXKH74esCKHACLcBGAs/s1600/object-class-diagrame-1.png)

[](https://1.bp.blogspot.com/-fF4IcjEQbiE/XU-utqBUkSI/AAAAAAAAEnk/fZ38-T3S1BoBdPVpl8s0ur6aBa637fKsgCLcBGAs/s1600/Screenshot+(8).png)

## Introduction

 Java is an Object-Oriented Language. As a language that has the ObjectOriented feature, Java supports the following fundamental concepts −

●    Polymorphism

●    Inheritance

●    Encapsulation

●    Abstraction

●    Classes

●    Objects

●    Instance

●    Method

●    Message Parsing

### Flow of Control

● Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

- Selection (Decision-making) statements (if-then, if-thenelse, switch)

– Looping statements (for, while, do-while)

– Branching statements (break, continue, return)

abc

### Selection Statements

● Selection statements allow you to control the flow of program execution on the basis of the outcome of an expression or state of a variable known during runtime.

● Selection statements can be divided into the following **categories**:

●    The if and if-else statements

●    The if-else statements

●    The if-else-if statements

●    The switch statements

### Iteration Statements

● Repeating the same code fragment several times until a specified condition is satisfied is called iteration.

● Iteration statements execute the same set of instructions until a termination condition is met.

● Java provides the following loop for **iteration statements:**

●    The while loop

●    The for loop

●    The do-while loop

●    The for each loop

### Jump Statements

● Jump statements are used to unconditionally transfer the program control to another part of the program.

● Java provides the following **jump statements:**

●    break statement

●    continue statement

●    return statement

