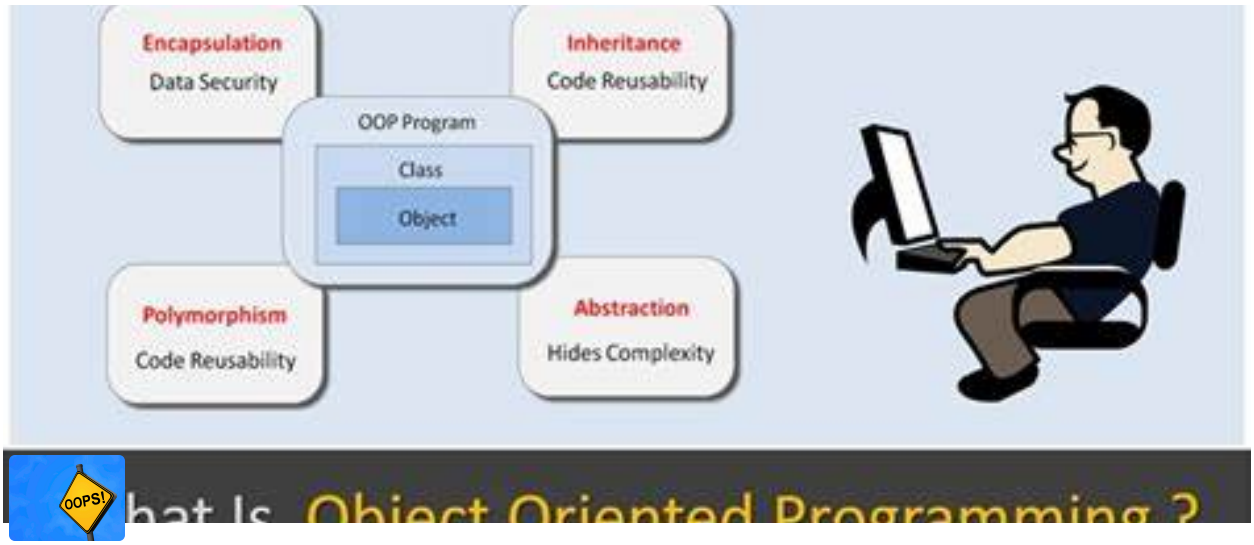


OOPS_Java

Why java is not purely objected oriented programming langugage?

There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

1. Encapsulation/Data Hiding
2. Inheritance
3. Polymorphism
4. Abstraction
5. All predefined types are objects.
6. All user defined types are objects
7. All operations performed on objects must be only through methods exposed/defined at the objects.

   Java supports property 1, 2, 3, 4 and 6 but fails to support property 5 and 7 given above. Java language is not a Pure Object Oriented Language as it contain these properties:

▼ Objects are the real world entity ex- car,water molecule, pen,humans etc

▼ Class is a group of these entities [real world entities]

▼ Class is a blueprint of objects

▼ Class does not occupy memory

---

1. A Class in Java can contain:
    - Data member
    - Methods
    - Constructor
    - Nested Class
    - Interface

```
access_modifier class <class_name>
{
    data member;
    method;
    constructor;
    nested class;
    interface;
}
```

How much memory does a class occupy?

Classes do not use memory. They merely serve as a template from which items are made. Now, objects actually initialize the class members and methods when they are created, using memory in the process.

Components of Java Classes:

Modifiers: A class can be public or has default access
Class keyword: class keyword is used to create a class.
Class name: The name should begin with an initial letter (capitalized by conve
ntion).
Superclass(if any): The name of the class's parent (superclass), if any, preced

ed by the keyword extends. A class can only extend (subclass) one parent.
Interfaces(if any): A comma-separated list of interfaces implemented by the cl ass, if any, preceded by the keyword implements. A class can implement more than one interface.
Body: The class body is surrounded by braces, { }.

What is OOPS ?

▼ It is a style of writing better and optimal code

▼ The main aim of OOPS is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

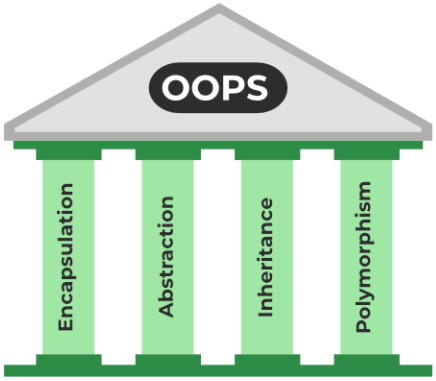

```
public class P1{
public static void main(String[] args) { //execution of program begins from mai
n function
    Pen p1=new Pen(); //created a pen object called p1
    p1.setColor("Blue");
    System.out.println(p1.color);
```

```
    p1.setTip(5);
    System.out.println(p1.tip);

    p1.setColor("Yellow");
    System.out.println(p1.color);

    //we can set color like this also
    p1.color="Orange";
    System.out.println(p1.color);

    p1.tip=33;
    System.out.println(p1.tip);


}
}
class Pen{
    //properties+function
    String color;
    int tip;

    void setColor(String newColor){
        color=newColor;
    }

    void setTip(int newTip){
        tip=newTip;
    }
}

class Student{
    String name;
    int age;
    float cgpa;

    void calcPercentage(int phy,int chem,int math){
```

```
        float percentage=(phy+chem+math)/3;
    }
}
```

4 Pillars of OOPS

- Encapsulation
- Inheritance
- Abstraction
- Polymorphism



Access Modifier :Defines the **access type** of the method i.e. from where it can be accessed in your application.

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package Non-Subclass	No	No	No	Yes

```
public class P2_Bank_Details {
    public static void main(String[] args) {
        BankAccount myAcc=new BankAccount();
        myAcc.username="Azaan";
        // myAcc.password="ajflasjf" this line will show error
        myAcc.setPassword("hkhsfhskh"); //you cannot access password but ca
n set the password
    }
}

class BankAccount{
    public String username;
    private String password;
    public void setPassword(String pwd){
        password=pwd;
    }

}
```

▼ Objects are created in **Heap memory**

Initializing a Java object

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 class constructor.

```
// Class Declaration

public class Dog {
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed, int age,
            String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName() { return name; }

    // method 2
    public String getBreed() { return breed; }

    // method 3
    public int getAge() { return age; }

    // method 4
    public String getColor() { return color; }

    @Override public String toString()
```

```
    {
        return ("Hi my name is " + this.getName()
                + ".\nMy breed,age and color are "
                + this.getBreed() + "," + this.getAge()
                + "," + this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy
            = new Dog("tuffy", "papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}
```

```
Hi my name is tuffy.
My breed,age and color are papillon,5,white
```

Initialize by using method/function:

```
public class GFG {
    // sw=software
    static String sw_name;
    static float sw_price;

    static void set(String n, float p)
    {
        sw_name = n;
        sw_price = p;
    }

    static void get()
    {
        System.out.println("Software name is: " + sw_name);
```

```
        System.out.println("Software price is: "
                           + sw_price);
    }

    public static void main(String args[])
    {
        GFG.set("Visual studio", 0.0f);
        GFG.get();
    }
}
```

Output :

```
Software name is: Visual studio
Software price is: 0.0
```

Anonymous Objects in Java

Anonymous objects are objects that are instantiated but are **not stored in a reference variable**.

- They will be destroyed after method calling.

Class	Object
Class is the blueprint of an object. It is used to create objects.	An object is an instance of the class.
No memory is allocated when a class is declared.	Memory is allocated as soon as an object is created.
A class is a group of similar objects.	An object is a real-world entity such as a book, car, etc.
Class is a logical entity.	An object is a physical entity.
A class can only be declared once.	Objects can be created many times as per requirement.
An example of class can be a car.	Objects of the class car can be BMW, Mercedes, Ferrari, e

Getters and Setters :

Get : It is used to get the value

```
public class P3_Getter_Setters{
public static void main(String[] args) { //execution of program begins from
main function
    Pen p1=new Pen(); //created a pen object called p1
    p1.setColor("Blue");
    System.out.println(p1.getColor());  //we replace p1.color=p1.getColor()

    p1.setTip(5);
    System.out.println(p1.getColor());

    p1.setColor("Yellow");
    System.out.println(p1.getColor());

    //we can set color like this also
    p1.color="Orange";
    System.out.println(p1.getColor());

    p1.tip=33;
    System.out.println(p1.tip);



}
}
class Pen{
    //properties+function
    String color;
    int tip;

    String getColor(){
        return this.color;
        //this means current object --> matlab joh bhi object hamm call krr ra
he hai uski properties ki hamm baat krr rahe hai
```

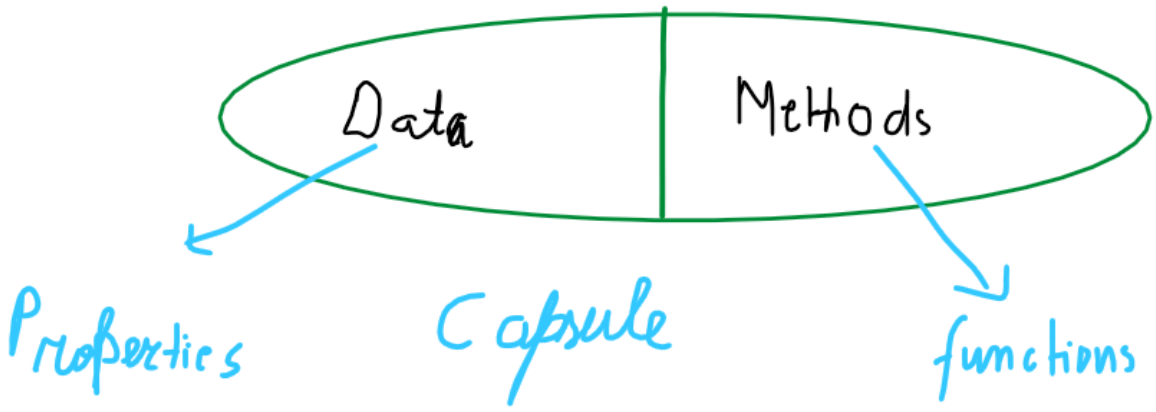
```
    }

    int getTip(){
        return this.tip;
    }
    void setColor(String newColor){
        color=newColor;
    }

    void setTip(int newTip){
        tip=newTip;
    }
}
```

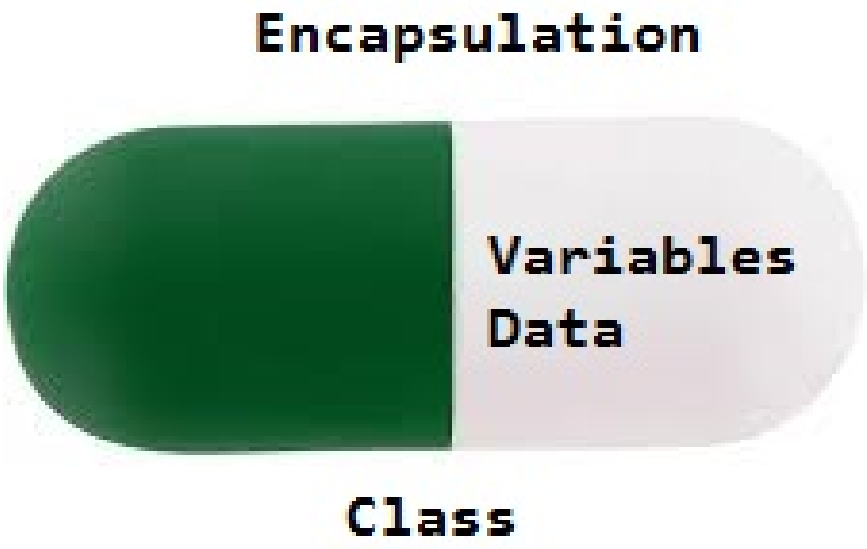
Set : It is used to modify the value

→ this keyword referred the current object

Encapsulation

Encapsulation is defined as the wrapping up of data and methods under a single unit.It also implements data hiding





→Data hiding is done using access specifier like protected and private.

Here's an example of encapsulation:

```
class Person {
    private String name;
    private int age;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }
}
```

```
public class Main {
    public static void main(String[] args)
    {
        Person person = new Person();
        person.setName("John");
        person.setAge(30);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

→ **Advantage of encapsulation is security**

Constructor

Constructor is a special method which is invoked automatically at the time of object creation

1. What is a constructor in Java?

> A constructor in Java is a special method used to initialize objects.

2. Can a Java constructor be private?

> Yes, a constructor can be declared private. A private constructor is used in restricting object creation.

How Java Constructors are Different From Java Methods?

- ▼ Constructors have the same name as class or structure
- ▼ Constructors don’t have a return type (not even void)
- ▼ Constructors are only called once ,at object creation
- ▼ Memory allocation happens when constructor is called

▼ A constructor in Java **can not be abstract, final, static, or Synchronized.**

```
package Day13_OOPS;

public class P4_Constructor {
    public static void main(String[] args) {
        Student s1=new Student();
        Student s2=new Student("Azaan");
        Student s3=new Student(123);
        //Student s4=new Student("Azaan",808) there is no constructor exist

        // this concept means when you write particular constructor respectiv
e constructor is called this phenomena is called constructor overloading li
ke above s1,s2,s3 and this is example of polymorphism
    }
}

class Student {
    String name;
    int roll;

    //constructor
    // NOTE-if you do not write constructor then java automatically make it i
nternally but there is no initialization
    Student(){  //non parametrized constructor
        System.out.println("Constructor is called....");
    }

    Student(String name){ //parametrized constructor
        this.name=name;
    }

    Student(int roll){
        this.roll=roll;
```

```
    }
}
```

Types of Constructors in Java

Now is the correct time to discuss the types of the constructor, so primarily there are three types of constructors in Java are mentioned below:

- Default Constructor
- Parameterized Constructor
- Copy Constructor

1. Default Constructor in Java

A constructor that has no parameters is known as default the constructor. A default constructor is invisible. And if we write a constructor with no arguments, the compiler does not create a default constructor. It is taken out. It is being overloaded and called a parameterized constructor. The default constructor changed into the parameterized constructor. But Parameterized constructor can't change the default constructor.

Example:

```
// Java Program to demonstrate
// Default Constructor
import java.io.*;

// Driver class
class GFG {

    // Default Constructor
    GFG() { System.out.println("Default constructor"); }

    // Driver function
    public static void main(String[] args)
    {
```

```
        GFG hello = new GFG();
    }
}
```

Output

```
Default constructor
```

> Note: Default constructor provides the default values to the object like 0, null, etc. depending on the type.

2.Parameterized Constructor in Java

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Example:

```
// Java Program for Parameterized Constructor
import java.io.*;
class Geek {
    // data members of the class.
    String name;
    int id;
    Geek(String name, int id)
    {
        this.name = name;
        this.id = id;
    }
}
class GFG {
    public static void main(String[] args)
    {
        // This would invoke the parameterized constructor.
        Geek geek1 = new Geek("avinash", 68);
        System.out.println("GeekName :" + geek1.name
                           + " and GeekId :" + geek1.id);
```

```
    }
}
```

Output :

```
GeekName :avinash and GeekId :68
```

Remember: Does constructor return any value?

There are no "return value" statements in the constructor, but the constructor returns the current class instance. We can write 'return' inside a constructor.

3. Copy Constructor in Java

Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

```
import java.util.*;
public class P5_Copy_Constructor {
    public static void main(String[] args) {
        Student s1=new Student();
        s1.name="Azaan";
        s1.roll=2432;
        s1.password="abcd";

        s1.marks[0]=100;
        s1.marks[1]=68;
        s1.marks[2]=34;


        Student s2=new Student(s1); //copy constructor
        s2.password="xyz";
        s1.marks[2]=99;  //copy krne ke baad s1 ke marks change but still ch
ange reflect karega in s2 because its referenced is passed

        for(int i=0;i<3;i++){
            System.out.println(s2.marks[i]+" ");  //here s2 marks are also chan
ged because referenced is passed →refer video
```

```
        }
    }
}

class Student {
    String name;
    int roll;
    String password;
    int marks[];

    //shallow copy constructor
    Student (Student s1){
        marks=new int[3];
        this.name=s1.name;
        this.roll=s1.roll;
        this.marks=s1.marks;

    }

    Student(){  //non parametrized constructor
        marks=new int[3];
        System.out.println("Constructor is called....");
    }

    Student(String name){
        marks=new int[3];
        this.name=name;
    }

    Student(int roll){
        marks=new int[3];
        this.roll=roll;
    }
}
```

Deep Copy

In this changes are not reflected

```
import java.util.*;

public class P6_Deep_Copy_Constructor {
    public static void main(String[] args) {
        Student s1=new Student();
        s1.name="Azaan";
        s1.roll=2432;
        s1.password="abcd";

        s1.marks[0]=100;
        s1.marks[1]=68;
        s1.marks[2]=34;


        Student s2=new Student(s1); //copy constructor
        s2.password="xyz";
        s1.marks[2]=100;

        for(int i=0;i<3;i++){
            System.out.println(s2.marks[i]+" ");
        }
    }
}

class Student {
    String name;
    int roll;
    String password;
    int marks[];

    //deep copy constructor
    Student(Student s1){
```

```
        marks=new int[3];
        this.name=s1.name;
        this.roll=s1.roll;

        for(int i=0;i<marks.length;++i){
            this.marks[i]=s1.marks[i];
        }

    }

    Student(){  //non parametrized constructor
        marks=new int[3];
        System.out.println("Constructor is called....");
    }

    Student(String name){
        marks=new int[3];
        this.name=name;
    }

    Student(int roll){
        marks=new int[3];
        this.roll=roll;
    }
}
```

Read this :Deep, Shallow and Lazy Copy with Java Examples - GeeksforGeeks

→There is no destructor in java because the work is done by garbage collector

Inheritance

Inheritance is when properties and methods of base class are passed on to a derived class

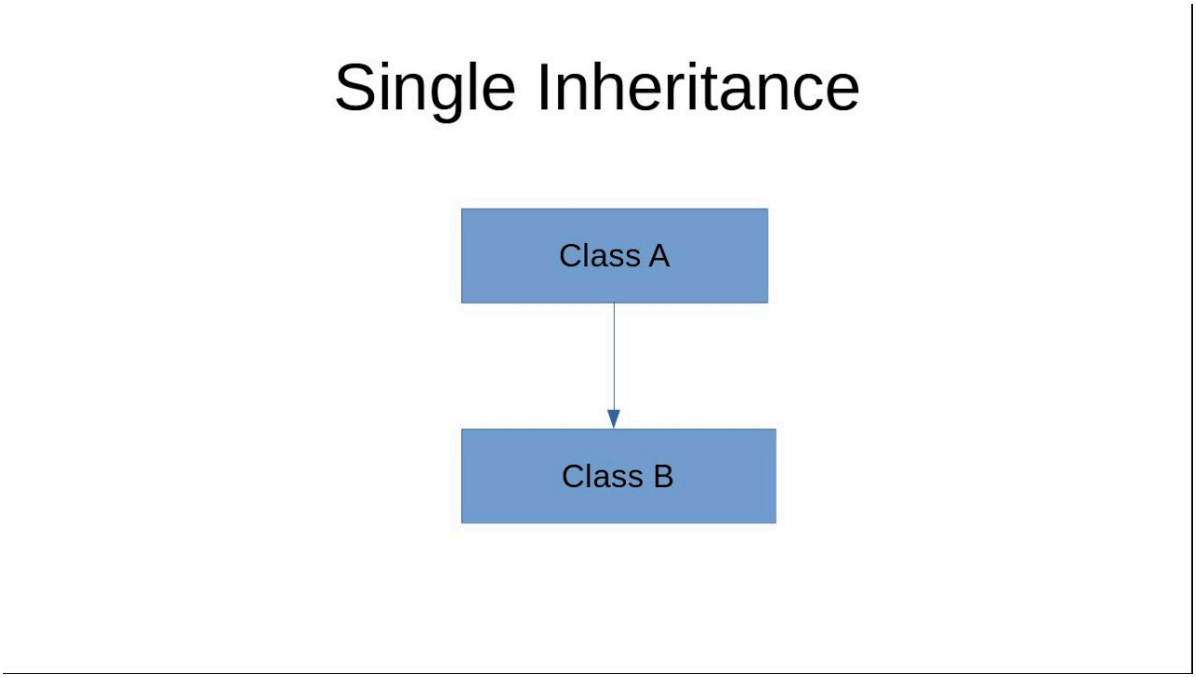
Why inheritance ?

▼ Code Reusability

▼ Method Overriding

• **Super Class/Parent Class:** The class whose features are inherited is known as a superclass(or a base class or a parent class).

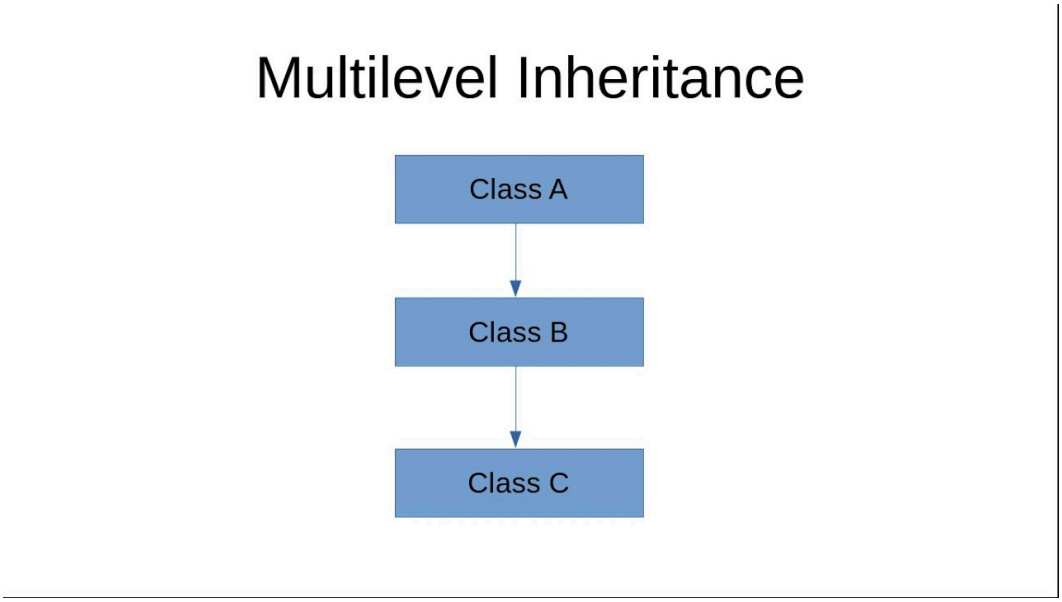
Single Inheritance




```
public class P7_Single_Level_Inheritance {
    public static void main(String[] args) {
        Fish shark=new Fish();
        shark.eat();
    }
}
//Base class
class Animal{
    String color;
    void eat(){
        System.out.println("Eats");
    }
```

```
    void breathe(){
        System.out.println("breathes");
    }
}

//Derived class
class Fish extends Animal{
    int fins;
    void swim(){
        System.out.println("Swims in water");
    }
}
```

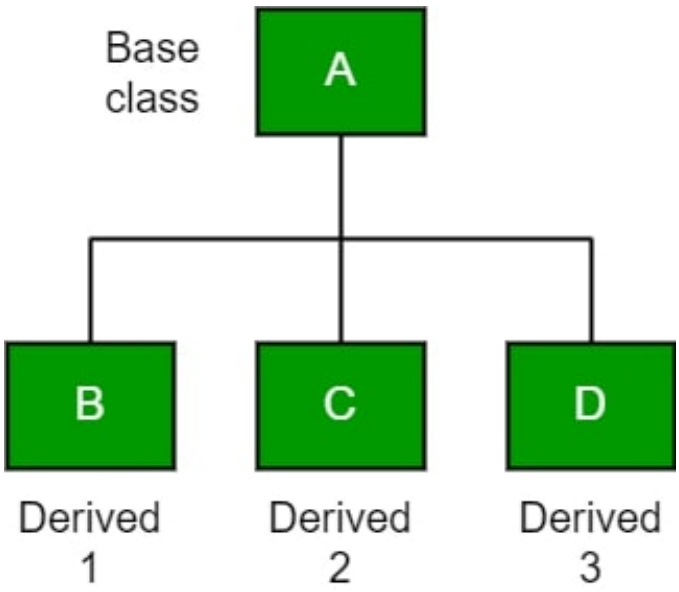
Multilevel Inheritance




```
public class P8_Multilevel_Inheritance {
    public static void main(String[] args) {
        Dog dobby=new Dog();
        dobby.eat();
        dobby.legs=4;
        System.out.println(dobby.legs);
```

```
    }
}

//Base class
class Animal{
    String color;
    void eat(){
        System.out.println("Eats");
    }
    void breathe(){
        System.out.println("breathes");
    }
}

class Mammal extends Animal{
    int legs;
}

class Dog extends Mammal{
    String breath;
}
```

Hierarchical Inheritance

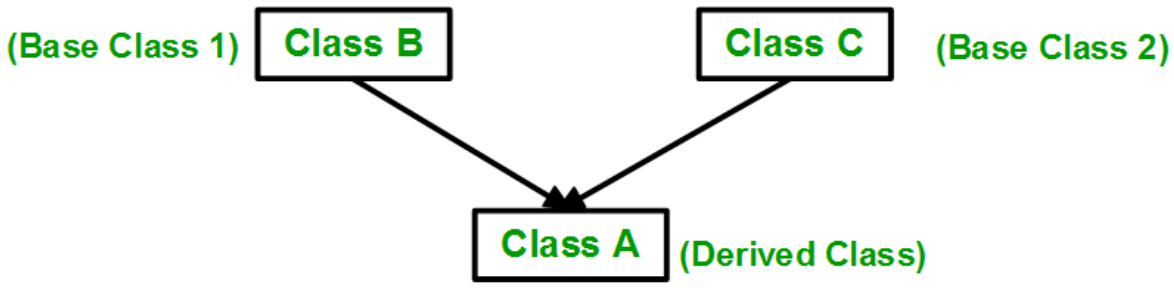



```
public class P9_Heirarchial_Inheritance {
    public static void main(String[] args) {

    }
}

//Base class
class Animal{
    String color;
    void eat(){
        System.out.println("Eats");
    }
    void breathe(){
        System.out.println("breathes");
    }
}

class Mammal extends Animal{
    void walk(){
        System.out.println("Walks");
    }
}
```

```
class Fish extends Animal{
   void swim(){
      System.out.println("Swim");
   }
}
class Bird extends Animal{
   void fly{
      System.out.println("Fly");
   }
}
```

Multiple Inheritance

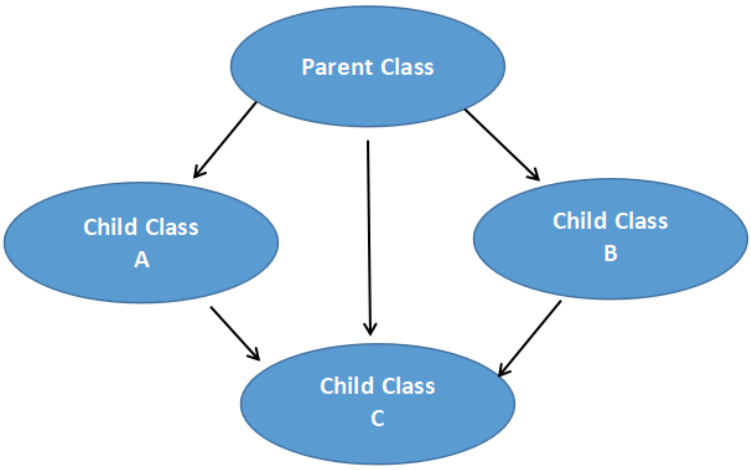
In **Multiple inheritances**, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support **multiple inheritances** with classes. In Java, we can achieve multiple inheritances only through **Interfaces**. In the image below, Class C is derived from interfaces A and B.




```
import java.io.*;
import java.lang.*;
import java.util.*;

interface one {
   public void print_geek();
}
```

```
interface two {
    public void print_for();
}

interface three extends one, two {
    public void print_geek();
}
class child implements three {
    @Override public void print_geek()
    {
        System.out.println("Geeks");
    }

    public void print_for() { System.out.println("for"); }
}

// Drived class
public class Main {
    public static void main(String[] args)
    {
        child c = new child();
        c.print_geek();
        c.print_for();
        c.print_geek();
    }
}
```

Hybrid Inheritance

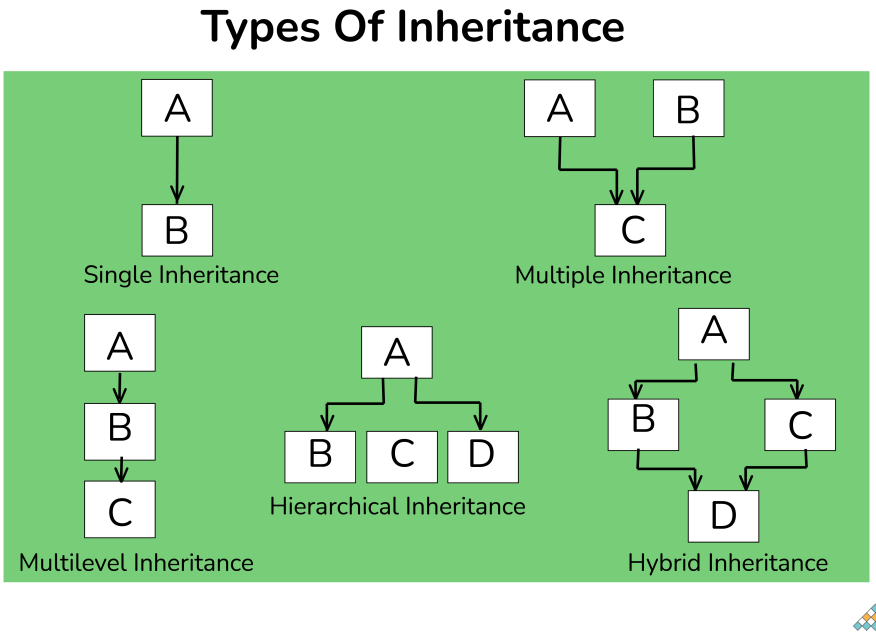



```
public class P10_Hybrid_Inheritance {
    public static void main(String[] args) {

    }
}

//Base class
class Animal{
    String color;
    void eat(){
        System.out.println("Eats");
    }
    void breathe(){
        System.out.println("breathes");
    }
}

class Mammal extends Animal{
    void walk(){
        System.out.println("Walks");
    }
}

class Bird extends Animal{
```

```
    void fly{
        System.out.println("Fly");
    }
}
class Dog extends Mammal{
    void breed(){
        System.out.println("German shepherd");
    }
}

class German extends Dog{
    void coatColor(){
        System.out.println("Double coat");
    }
}
```

Advantages of Inheritance

- Code reusability
- Reduce redundancy
- Ease of understanding



Polymorphism

When try to do the same task in two different ways is called polymorphism

There are two types of polymorphism :—

Compile Time/Static Polymorphism

Method Overloading

Run Time /Dynamic Polymorphism

Method Overriding

Method/Function Overloading

Multiple functions with the same name but different parameters is called Method Overloading

```
public class P11_Method_Overloading_Compile_Time_Polymorphism {
    public static void main(String[] args) {
        Calculator calc=new Calculator();
```

```
        System.out.println(calc.sum(1,2));
        System.out.println(calc.sum((float)1.3,(float)7.9));
        System.out.println(calc.sum(1,2,99));

    }
}

class Calculator{
    int sum(int a,int b){
        return a+b;
    }

    float sum(float a,float b){
        return a+b;
    }

    int sum(int a,int b,int c){
        return a+b+c;
    }
}
//Function overloading is an example of compile time polymorphism
```

Method Overriding

Parent and child classes both contains the same function but with different definition

```
package Day13_OOPS;

public class P12_Method_Overriding_Runtime_Polymorphism {
    public static void main(String[] args) {
        Deer d=new Deer();
        d.eat();
    }
}
```

```
class Animal{
   void eats(){
      System.out.println("Eats Anything:");
   }
}

class Deer extends Anmial{
  @Override
   void eat(){
      System.out.println("Eats grass:");
   }
}
```

Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essential units are not displayed to the user.

> Ex: A car is viewed as a car rather than its individual components.

In Java, abstraction is achieved by **interfaces** and **abstract classes**. We can achieve **100%** abstraction using interfaces.

Real-Life Example:

> Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

```
// Java Program to implement
// Java Abstraction
```

```
// Abstract Class declared
abstract class Animal {
    private String name;

    public Animal(String name) { this.name = name; }

    public abstract void makeSound();

    public String getName() { return name; }
}

// Abstracted class
class Dog extends Animal {
    public Dog(String name) { super(name); }

    public void makeSound()
    {
        System.out.println(getName() + " barks");
    }
}

// Abstracted class
class Cat extends Animal {
    public Cat(String name) { super(name); }

    public void makeSound()
    {
        System.out.println(getName() + " meows");
    }
}

// Driver Class
public class AbstractionExample {
    // Main Function
    public static void main(String[] args)
```

```
    {
        Animal myDog = new Dog("Buddy");
        Animal myCat = new Cat("Fluffy");

        myDog.makeSound();
        myCat.makeSound();
    }
}
```

Package

Package is a group of similar types of **classes ,interfaces and sub-class.**

Ex-Import java.util.*

- Package is a mechanism to encapsulate group of classes, subclasses, and interfaces.
- Package helps in naming conflicts.
- Ex- there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Package provide access control.
- Packages can be considered as data encapsulation/hiding.
- Package is created by package keyword.
- https://www.youtube.com/watch?v=av816KIz8nM&list=PLu0W_9lII9agS67Uits0UnJyrYiXhDS6q&index=65

Abstract Class

▼ Cannot create an instance/object of abstract class

▼ Can have abstract/non-abstract methods

▼ Can have constructors

```
package Day13_OOPS;
```

```
public class P13_Abstract_Class {
    public static void main(String[] args) {
        Horse h=new Horse();
        h.eat();
        h.walk();

        Chicken c=new Chicken();
        c.eat();
        c.walk();
    }
}

abstract class Animal{
    void eat(){
        System.out.println("animal eats");
    }
    abstract void walk(); //we cannot write implementation in abstract
}

class Horse extends Animal{
    void Walks(){
        System.out.println("Walks on 4 legs");
    }
}
class Chicken extends Animal{
    void walk(){
        System.out.println("Walks on 2 legs");
    }
}
```

Interface

Interface is a blueprint of class

Advantages of interface :

- Implement Multiple inheritance
- Achieve 100% abstraction

▼ All methods are public, abstract and without implementation

```
public class P15_Interfaces_Implementation{
    public static void main(String[] args) {
        Queeen q=new Queeen();
        q.moves();
    }
}

interface ChessPlayer{
    void moves();// yha hamne bss function likha but implementation nhi
}

class Queeen implements ChessPlayer{
    public void moves(){
        System.out.println("Up,Down,Left,Right,Diagonal -(in all direction)");
    }
}

class Rook implements ChessPlayer{
    public void moves(){
        System.out.println("Up,Down,Left,Right");
    }
}
```

Q. Why Multiple Inheritance is not supported through a class in Java, but it can be possible through the interface?

> Multiple Inheritance is not supported by class because of ambiguity. In the case of interface, there is no ambiguity because the implementation of the method(s) is provided by the implementing class up to Java 7. From Java 8, interfaces also have implementations of methods. So if a class implements

> two or more interfaces having the same method signature with implementation, it is mandated to implement the method in class also.

Watch this : https://www.youtube.com/watch?v=HnaVobvfSyc

Static Keyword

static keyword is used to share same variable or method of a given class

We can make below all these static :

1. Function
2. Properties
3. Block
4. Nested Classes

```
import java.util.*;

public class P14_Static_Keyword {
    public static void main(String[] args) {
        Student s1=new Student();
        s1.schoolName="st.thomas";

        Student s2=new Student();
        System.out.println(s2.schoolName);  //Output: st.thomas

        Student s3=new Student();
        System.out.println(s3.schoolName);   //Output: st.thomas
    }
}

class Student{
    String name;
    int roll;

    static String schoolName;
```

```
    void getName(String name){
        this.name=name;
    }

    String getName(){
        return this.name;
    }
}
```

Advantages of static keyword

- Memory efficiency : It means memory allocated for all static variables only once

Read this : static Keyword in Java - GeeksforGeeks

Static Method: Access the static data using class name. Declared inside class with **static** keyword.

```
//Static Method
static void method_name(){
body // static area
}
```

In Java, a static method is like a special function that belongs to a class rather than to a specific object created from that class. You can think of it as a method that's not tied to any particular instance (object) of the class.

Here's a simple way to understand it:

1. **No Need for Objects**: You can call a static method without creating an object of the class. You just use the class name to call it.
2. **Shared by All Objects**: Static methods are shared by all the objects of a class. Any object of the class can use the same static method.
3. **Common Tasks**: Use static methods when you have a task that doesn't depend on the specific properties or data of an object but is related to the

class as a whole.

For example, if you have a `Math` class in Java, you can have a static method like `Math.max()` to find the maximum of two numbers. You don't need to create a `Math` object; you can directly call `Math.max(5, 8)` to get the result.

Super keyword

Super keyword is used to refer immediate parent class object

Use of super keyword :

→ to access parent's properties

→ to access parent's functions

→ to access parent's constructor

```
import java.util.*;

public class P15_Super_Keyword {
    public static void main(String[] args) {
        Horse h=new Horse();

    }
}

class Animal{
    Animal(){
        System.out.println("Animal constructor is called");
    }
}

class Horse extends Animal{
    Horse(){
        super();
        System.out.println("horse constructor is called ");
```

```
    }
}
```

Wrapper class

wrapper class is use to convert primitive data types to objects (process called autoboxing) and objects to primitive data types(process called unboxing)