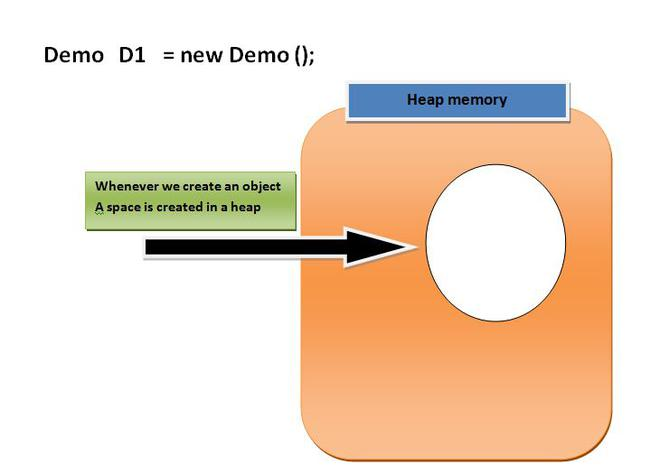
**JAVA PHASE 2**

**REFERENCE VARIABLE**

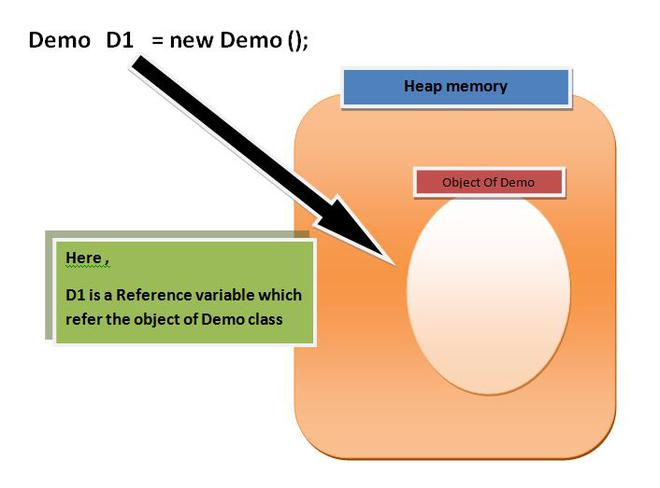
When we create an object (instance) of class then space is reserved in heap memory.

**Demo D1 = new Demo();**

****

Now, The space in the heap Memory is created but the question is how to access that space?.

Then, We create a Pointing element or simply called Reference variable which simply points out the Object(the created space in a Heap Memory).



**Imp points of Reference variable**

1. Reference variable is used to point object/values.

2. Classes, interfaces, arrays etc are reference types in Java. Reference variables hold the objects/values of reference types in Java.

3. Reference variable can also store **null** value. By default, **if no object is passed to a reference variable then it will store a null value.**

4. You can access object members using a reference variable using dot syntax.

**reference variable name .instance variable\_name / method\_name**

**Example**

**D1.display();**

**Or**

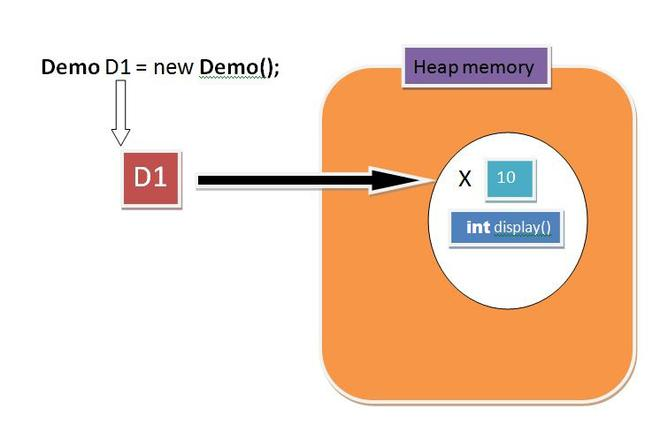
**D1.X;**

Here D1 is reference variable(or object in other term).

display is any function or method

And

X is a variable.



**Oops concepts**

OOPS (object oriented programming System) concepts are as follows:

Class

Objects

Pillars of OOPs:

Abstraction

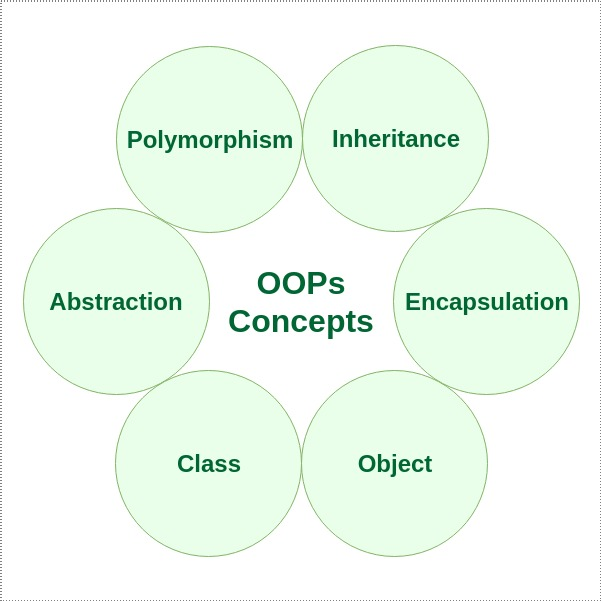
Encapsulation

Inheritance

Polymorphism

Compile-time polymorphism

Runtime polymorphism



**CLASS**

A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times.

This includes classes for objects occurring more than once in your code. In general, class declarations can include these components in order:

**Modifiers**: A class can be public or have default access(will discuss later)

**Class name**: The class name should begin with the initial letter capitalized by convention.

**Superclass (if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends (inheritance).

A class can only extend (subclass) one parent.

**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.

**Body**: The class body is surrounded by braces, { }.

**An object** is a basic unit of Object-Oriented Programming that represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. The objects are what perform your code, they are the part of your code visible to the viewer/user. An object mainly consists of:

**State**: It is represented by the attributes of an object. It also reflects the properties of an object.

**Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.

**Identity**: It is a unique name given to an object that enables it to interact with other objects.

**Method**: A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping it, which is why they are considered time savers. In Java, every method must be part of some class, which is different from languages like C, C++, and Python.

**public class Itsme**

{

String Employee\_name;

float Employee\_salary;

void set(String n, float p)

{

Employee\_name = n;

Employee\_salary = p;

}

**void get()**

{

System.out.println("Employee name is: " +Employee\_name );

System.out.println("Employee CTC is: " + Employee\_salary);

}

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

{

Itsme G=new Itsme();

G.set("Rathod Avinash", 10000.0f);

G.get();

}

}

**Pillar 1: Encapsulation**

It is defined as the wrapping up of data and functions under a single unit. It is the mechanism that binds together the code and the data it manipulates.

Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically, in encapsulation, the variables or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.

In encapsulation, the data in a class is hidden from other classes, which is similar to what **data-hiding** does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.

*Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.*

**Pillar 2: Inheritance**

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features (fields(variables) and methods) of another class.

Let us discuss some frequently used important terminologies:

**Superclass**: The class whose features are inherited is known as superclass (also known as base or parent class).

**Subclass**: The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

**Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**Why use inheritance in java**

1)For Method Overriding (so runtime polymorphism can be achieved).

2)For Code Reusability.

**SYNTAX**

class Subclass-name extends Superclass-name

{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Example of two class with only fields(variables) inherited

**class Employee**

{

float salary=40000;

}

class Programmer extends Employee

{

int bonus=10000;

public static void main(String args[])

{

Programmer p=new Programmer();

System.out.println("Programmer salary is:"+p.salary);

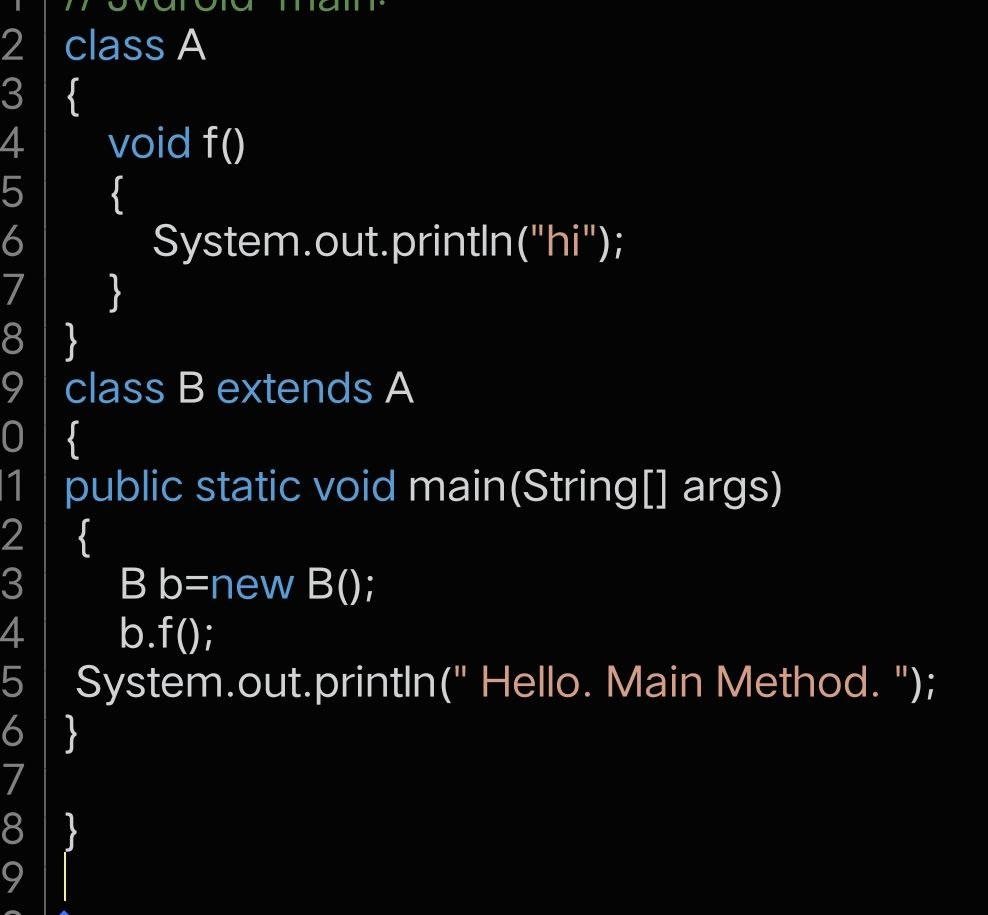
System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

Remember

we only used variables in parent class and child class to explain inheritance but this is not a good way to approach it because data hiding concepts stated that use of variables only done by using the functions.



**Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java:

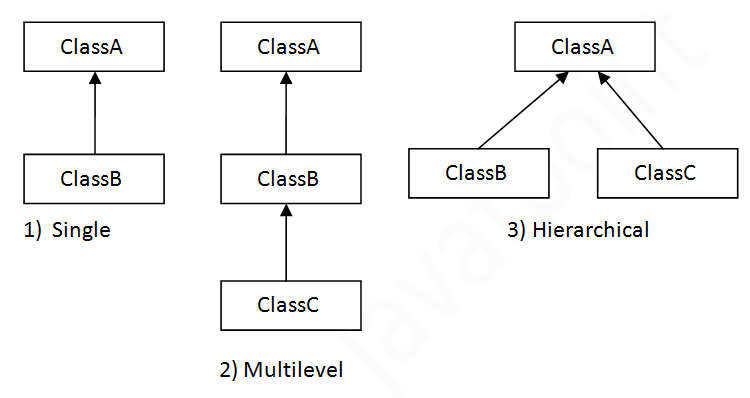
single

multilevel

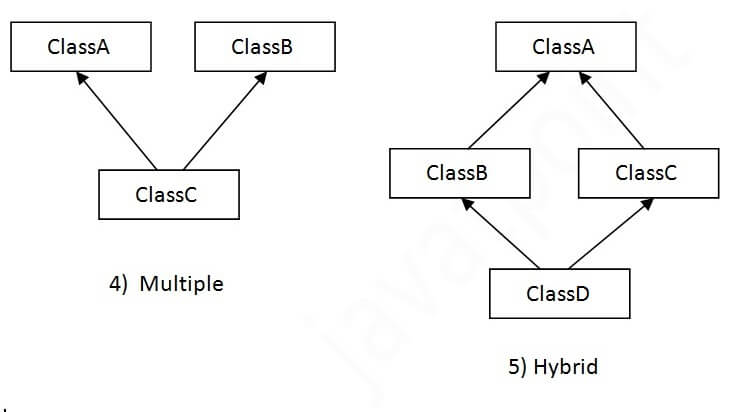
and

hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



Multiple inheritance is not supported in Java through class.it only supported through Interface only.



**Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

**class Animal**

{

void eat()

{

System.out.println("eating...");}

}

**class Dog extends Animal**

{

void bark()

{

System.out.println("barking...");}

}

class TestInheritance

{

public static void main(String args[])

{

Dog d=new Dog();

d.bark();

d.eat();

}

}

**Multilevel Inheritance**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

**class Animal**

{

void eat()

{

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

}

}

**class Dog extends Animal**

{

void bark()

{

System.out.println("barking...");}

}

**class BabyDog extends Dog**

{

void weep()

{

System.out.println("weeping...");}

}

**class TestInheritance2**

{

public static void main(String args[])

{

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}

}

**Hierarchical Inheritance**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

**class Animal**

{

void eat()

{

System.out.println("eating...");}

}

**class Dog extends Animal**

{

void bark()

{

System.out.println("barking...");}

}

**class Cat extends Animal**

{

void meow()

{

System.out.println("meowing...");}

}

class TestInheritance3

{

public static void main(String args[])

{

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}

}