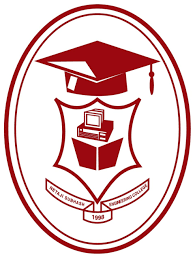
**Netaji Subhash Engineering College**

****

*CA2 Report Writing: INHERITANCE*

**Name: Sanjoy Saha**

**Department: CSE**

**Year: 3rd Semester: 5th**

**Section: A**

**Class Roll: 3**

**Univ. Roll: 10900120003**

**Subject : Object Oriented Programming**

**Subject Code: PCC-CS501**

**Inheritance**

**Q. What is Inheritance?**

**Inheritance** is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

**Types of Inheritance:**

There are different types of inheritance viz., Single Inheritance, Multiple Inheritance, Multilevel Inheritance, Hierarchical Inheritance.

1. **Single Inheritance:** When a derived class inherits only from one base class, it is known as single inheritance.
2. **Multiple Inheritance:** When a derived class inherits from multiple base classes it is known as multiple inheritance.
3. **Multilevel Inheritance:** The transitive nature of inheritance is itself reflected by this form of inheritance. When a class is derived from a class that is a derived class then it is referred to as multilevel inheritance.
4. **Hierarchical Inheritance:** When more than one derived class is created from a single base class. It is known as hierarchical inheritance.
5. Hence, it may be a combination of Multilevel and Multiple Inheritances or Hierarchical and Multilevel or Multileve and Multiple

**Terms frequently used in Inheritance**

* **Class:** It is a user-defined template or blueprint from which objects are created.
* **Derived/Sub/Child class:**It is a class that is derived from another class. Also known as extended class.
* **Base/Super/Parent class:**It is a class from which the derived class inherits its features.
* **Reusability:**It allows the reuse of the methods and fields of the existing class when creating a new class.

**Single Inheritance**

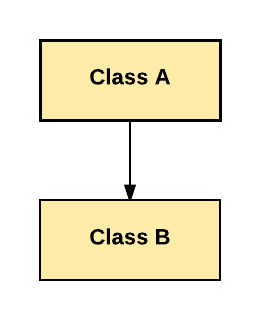
## Q. What Does Single Inheritance Mean?

## Single inheritance is the simplest type of inheritance in java. In this, a class inherits the properties from a single class. The class which inherits is called the derived class or child class or subclass, while the class from which the derived class inherits is called the base class or superclass or parent class. So, in single inheritance, we have only one derived class and one base class.

**Example:**

class Shape {

public void display() {

 System.out.println("Inside display");

}

}

class Rectangle extends Shape {

public void area() {

System.out.println("Inside area");

}

}

public class Tester {

public static void main(String[] arguments) {

Rectangle rect = new Rectangle();

rect.display();

rect.area();

}

**Multiple Inheritance**

## Q. What Does Multiple Inheritance Mean?

## Multiple inheritance a feature of some object-oriented programming languages in which a class or an object inherits characteristics and properties from more than one parent class or object. This is contrary to the single inheritance property, which allows an object or class to inherit from one specific object or class. Although there are certain benefits associated with multiple inheritance, it does increase ambiguity and complexity when not designed or implemented properly.

## Multiple Inheritance in Java

**Example:**

**No example because Java does not Multiple Inheritance**

**Multilevel Inheritance**

**Multilevel Inheritance in java** involves inheriting a class, which already inherited some other class. Correlating it with a real-life scenario, we’ve often seen some of our habits and thoughts match precisely with our parents. And similarly, their habits match with their parents, i.e., our grandparents.

We can create hierarchies in Java with as many layers of Inheritance as we want. This means we can utilize a subclass as a superclass. In this case, each subclass inherits all of the traits shared by all of its superclasses.

**Example:**

class Car{

public Car()

{

System.out.println("Class Car");

}

public void vehicleType()

{

System.out.println("Vehicle Type: Car");

}

}

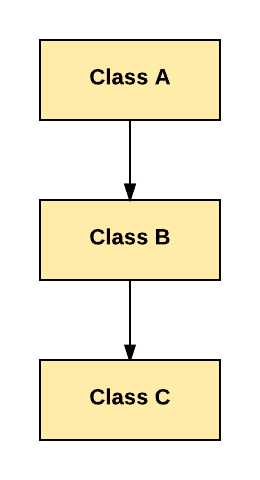
class Maruti extends Car{

public Maruti()

{

System.out.println("Class Maruti");

}



public void brand()

{

System.out.println("Brand: Maruti");

}

public void speed()

{

System.out.println("Max: 90Kmph");

}

}

public class Maruti800 extends Maruti{

public Maruti800()

{

System.out.println("Maruti Model: 800");

}

public void speed()

{

System.out.println("Max: 80Kmph");

}

public static void main(String args[])

{

Maruti800 obj=new Maruti800();

obj.vehicleType();

obj.brand();

obj.speed();

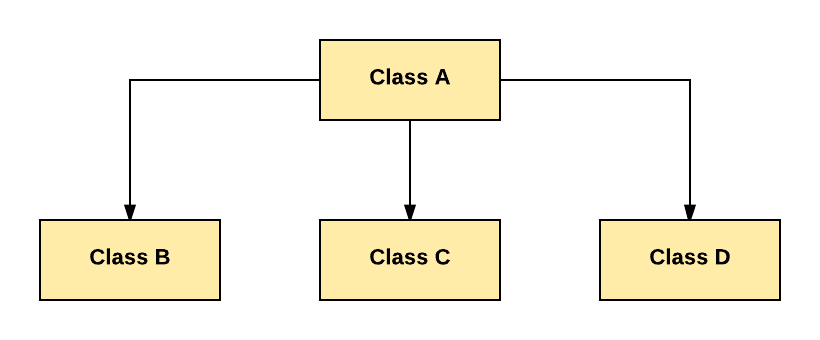
}

}

**Hierarchical Inheritance**

Q. What does Hierarchical Inheritance mean?

The type of inheritance in which more than one derived class inherits the properties of the same base class is called hierarchical inheritance. There are multiple child classes and a single parent class. All the child classes will inherit the methods and fields present in the parent class.



**Example:**

class A

{

public void methodA()

{

System.out.println("method of Class A");

}

}

class B extends A

{

public void methodB()

{

System.out.println("method of Class B");

}

}

class C extends A

{

public void methodC()

{

System.out.println("method of Class C");

}

}

class D extends A

{

public void methodD()

{

System.out.println("method of Class D");

}

}

class JavaExample

{

public static void main(String args[])

{

B obj1 = new B();

C obj2 = new C();

D obj3 = new D();

//All classes can access the method of class A

obj1.methodA();

obj2.methodA();

obj3.methodA();

}

**Method Overriding in Java**

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

**Example:**

Class Parent {

void show() {

System.out.println(“Parent’s show()”);

}

}

Class Child extends Parent {

void show() {

System.out.println(“Child’s show()”);

}

class Main {

public static void main(String[] args) {

Parent obj1 = new Parent();

obj1.show;

Parent obj2 = new Child();

obj2();

}

# Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

## Usage of Java super Keyword

1. ***super can be used to refer immediate parent class instance variable.***
2. ***super can be used to invoke immediate parent class method.***
3. ***super() can be used to invoke immediate parent class constructor.***

**Example:**

class Animal{

String color="white";

String name;

Animal(String name) {

this.name = name;

}

class Dog extends Animal{

String color="black";

Dog(String s) {

super(s);

}

void printColor(){

System.out.println(color);//prints color of Dog class

System.out.println(super.color);//prints color of Animal class

System.out.println(super.name);

}

}

class TestSuper1{

public static void main(String args[]){

Dog d=new Dog("Rocky");

d.printColor();

}}

**Can super class variable refer subclass object?**

Yes, the super class reference variable can hold the sub class object actually, it is widening in case of objects (Conversion of lower datatype to a higher datatype).

But, using this reference you can access the members of super class only, if you try to access the sub class members a compile time error will be generated.

## Example

In the following Java example, we have two classes namely Person and Student. The Person class has two instance variables name and age and one instance method displayPerson() which displays the name and age.

The Student extends the person class and in addition to the inherited name and age it has two more variables branch and student\_id. It has a method displayData() which displays all four values.

In the main method, we are assigning the subclass object with the super class reference variable

class Person{

   private String name;

   private int age;

   public Person(String name, int age){

      this.name = name;

      this.age = age;

   }

   public void displayPerson() {

      System.out.println("Data of the Person class: ");

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

      System.out.println("Age: "+this.age);

   }

}

public class Student extends Person {

   public String branch;

   public int Student\_id;

   public Student(String name, int age, String branch, int Student\_id){

      super(name, age);

      this.branch = branch;

      this.Student\_id = Student\_id;

   }

   public void displayStudent() {

      System.out.println("Data of the Student class: ");

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

      System.out.println("Age: "+this.age);

      System.out.println("Branch: "+this.branch);

      System.out.println("Student ID: "+this.Student\_id);

   }

   public static void main(String[] args) {

      Person person = new Student("Krishna", 20, "IT", 1256);

      person.displayPerson();

   }

}

## Output :

Data of the Person class:

Name: Krishna

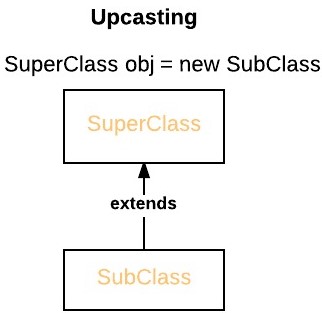
Age: 20

**Dynamic Method Dispatch or Runtime Polymorphism**

Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

* When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
* At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
* A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed. Here is an example that illustrates dynamic method dispatch:



**Example:**

class Apple

{

void display()

{

System.out.println("Inside Apple's display method");

}

}

class Banana extends Apple

{

void display() // overriding display()

{

System.out.println("Inside Banana's display method");

}

}

class Cherry extends Apple

{

void display() // overriding display()

{

System.out.println("Inside Cherry's display method");

}

}

class Fruits\_Dispatch

{

public static void main(String args[])

{

Apple a = new Apple(); // object of Apple

Banana b = new Banana(); // object of Banana

Cherry c = new Cherry(); // object of Cherry

Apple ref; // taking a reference of Apple

ref = a; // r refers to a object in Apple

ref.display(); // calling Apple's version of display()

ref = b; // r refers to a object in Banana

ref.display(); // calling Banana's version of display()

ref = c; // r refers to a object in Cherry

ref.display(); // calling Cherry's version of display()

}

}

***Thank You!  
With Due Respect***

-------------------------------