**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department Spring 2024, Lab Manual – 06**

|  |  |
| --- | --- |
| **Course Code: CL-1004** | **Course : Object Oriented Programming Lab** |
| **Instructor(s) :** | **Ali Fatmi** |

LAB - 6

Inheritance and Types

Types of Inheritance in Java

**Inheritance** is the most powerful feature of [object-oriented programming](https://www.javatpoint.com/what-is-object-oriented-programming). It allows us to inherit the properties of one class into another class. In this section, we will discuss **types of inheritance in Java** in-depth with real-life examples. Also, we will create Java programs to implement the concept of different types of inheritance.

Inheritance

[**Inheritance**](https://www.javatpoint.com/inheritance-in-java)is a mechanism of driving a new class from an existing class. The existing (old) class is known as **base class** or **super class** or **parent class**. The new class is known as a **derived class** or **sub class** or **child class**. It allows us to use the properties and behavior of one class (parent) in another class (child).

A class whose properties are inherited is known as **parent class** and a class that inherits the properties of the parent class is known as **child class**. Thus, it establishes a relationship between parent and child class that is known as parent-child or **Is-a** relationship.

Suppose, there are two classes named **Father** and **Child** and we want to inherit the properties of the Father class in the Child class. We can achieve this by using the **extends** keyword.

//inherits the properties of the Father class

**class** Child **extends** Father

{

//functionality

}

# 

## When we should use inheritance?

Inheritance provides the **reusability** of code especially when there is a large scale of code to reuse. It also establishes the relationship between different classes that is known as a **Is-a** relationship. We can also use it if we want to achieve **method overriding**.

# Points to Remember

# Constructor cannot be inherited in Java.

# Private members do not get inherited in Java.

# Cyclic inheritance is not permitted in Java.

# Assign parent reference to child objects.

# Constructors get executed because of super() present in the constructor.

# Types of Inheritance

# Java supports the following four types of inheritance:

# Single Inheritance

# Multi-level Inheritance

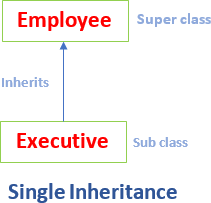
# Hierarchical Inheritance

# Hybrid Inheritance

# Note: Multiple inheritance is not supported in Java

## Single Inheritance

In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes it is also known as **simple inheritance**.



In the above figure, Employee is a parent class and Executive is a child class. The Executive class inherits all the properties of the Employee class.

Let's implement the single inheritance mechanism in a Java program.

**class** Employee

{

**float** salary=34534\*12;

}

**public class** Executive **extends** Employee

{

**float** bonus=3000\*6;

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

{

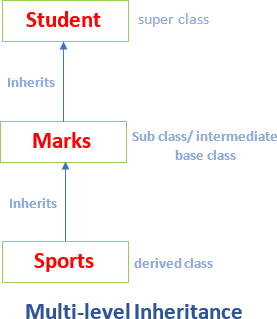
Executive obj=**new** Executive(); System.out.println("Total salary credited: "+obj.salary); System.out.println("Bonus of six months: "+obj.bonus);

}

}

Total salary credited: 414408.0 Bonus of six months: 18000.0

## Multi-level Inheritance

In **multi-level inheritance**, a class is derived from a class which is also derived from another class is called multi-level inheritance. In simple words, we can say that a class that has more than one parent class is called multi-level inheritance. Note that the classes must be at different levels. Hence, there exists a single base class and single derived class but multiple intermediate base classes.

In the above figure, the class Marks inherits the members or methods of the class Students. The class Sports inherits the members of the class Marks. Therefore, the Student class is the parent class of the class Marks and the class Marks is the parent of the class Sports. Hence, the class Sports implicitly inherits the properties of the Student along with the class Marks.

**MultilevelInheritanceExample.java**

**class** Student

{

**int** reg\_no;

**void** getNo(**int** no)

{

reg\_no=no;

}

**void** putNo()

{

System.out.println("registration number= "+reg\_no);

}

}

//intermediate sub class

**class** Marks **extends** Student

{

**float** marks;

**void** getMarks(**float** m)

{

marks=m;

}

**void** putMarks()

{

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

}

}

//derived class

**class** Sports **extends** Marks

{

**float** score;

**void** getScore(**float** scr)

{

score=scr;

}

**void** putScore()

{

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

}

}

**public class** MultilevelInheritanceExample

{

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

{

# Sports ob=new Sports();

# ob.getNo(0987);

ob.putNo(); ob.getMarks(78); ob.putMarks(); ob.getScore(68.7); ob.putScore();

}

}

**Output:**

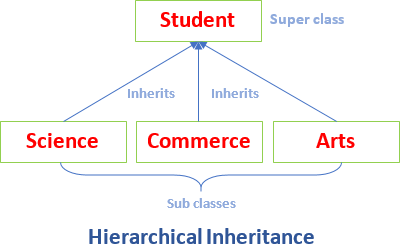
registration number= 0987

marks= 78.0

score= 68.7

## Hierarchical Inheritance

If a number of classes are derived from a single base class, it is called **hierarchical inheritance**.



In the above figure, the classes Science, Commerce, and Arts inherit a single parent class named Student.

Let's implement the hierarchical inheritance mechanism in a Java program.

**HierarchicalInheritanceExample.java**

//parent class

**class** Student

{

**public void** methodStudent()

{

System.out.println("The method of the class Student invoked.");

}

}

**Class** Science **extends** Student

{

**public void** methodScience()

{

System.out.println("The method of the class Science invoked.");

}

}

**class** Commerce **extends** Student

{

**public void** methodCommerce()

{

System.out.println("The method of the class Commerce invoked.");

}

}

**class** Arts **extends** Student

{

**public void** methodArts()

{

System.out.println("The method of the class Arts invoked.");

}

}

**public class** HierarchicalInheritanceExample

{

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

{

Science sci = **new** Science(); Commerce comm = **new** Commerce(); Arts art = **new** Arts();

//all the sub classes can access the method of super class sci.methodStudent();

comm.methodStudent(); art.methodStudent();

}

}

**Output:**

The method of the class Student invoked. The method of the class Student invoked. The method of the class Student invoked.

## Hybrid Inheritance

Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.

# 

In the above figure, GrandFather is a super class. The Father class inherits the properties of the GrandFather class. Since Father and GrandFather represents single inheritance. Further, the Father class is inherited by the Son and Daughter class. Thus, the Father becomes the parent class for Son and Daughter. These classes represent the hierarchical inheritance. Combinedly, it denotes the hybrid inheritance.

Let's implement the hybrid inheritance mechanism in a Java program.

**Daughter.java**

//parent class

**class** GrandFather

{

**public void** show()

# {

System.out.println("I am grandfather.");

}

}

//inherits GrandFather properties

**class** Father **extends** GrandFather

{

**public void** show()

{

System.out.println("I am father.");

}

}

//inherits Father properties

**class** Son **extends** Father

{

**public void** show()

{

System.out.println("I am son.");

}

}

//inherits Father properties

**public class** Daughter **extends** Father

{

**public void** show()

{

System.out.println("I am a daughter.");

}

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

{

Daughter obj = **new** Daughter(); obj.show();

}

}

**Output:**

I am daughter.

# Multiple Inheritance (not supported)

Java does not support multiple inheritances due to ambiguity. For example, consider the following Java program.

**Demo.java**

**class** Wishes

{

**void** message()

# {

System.out.println("Best of Luck!!");

}

}

**class** Birthday

{

**void** message()

{

System.out.println("Happy Birthday!!");

}

}

**public class** Demo **extends** Wishes, Birthday //considering a scenario

{

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

{

Demo obj=**new** Demo();

//can't decide which classes' message() method will be invoked obj.message();

}

}

The above code gives error because the compiler cannot decide which message() method is to be invoked. Due to this reason, Java does not support multiple inheritances at the class level but can be achieved through an **interface**.

# Lab Tasks

# Single Inheritance:

# Question: In a hospital management system, you're tasked with modeling the hierarchy of medical professionals. How would you utilize single inheritance to represent different types of medical staff such as Doctors, Nurses, and Pharmacists, while maintaining common attributes like name and specialization? Consider additional functionalities like scheduling appointments, administering medications, and updating patient records.

# Multi-level Inheritance:

# Question: You're developing a software application for a car dealership where various roles handle different aspects of the business. How would you implement multi-level inheritance to represent roles such as Salesperson, Sales Manager, and General Manager, each with increasing levels of authority and responsibilities? Assume that each role has specific tasks such as handling customer inquiries, managing sales teams, and overseeing overall operations.

# Hierarchical Inheritance:

# Question: In a school management system, you're tasked with modeling the hierarchy of academic departments. How would you utilize hierarchical inheritance to represent departments such as Science, Humanities, and Business, each with its own sub-departments and faculty members? Consider attributes like department name, faculty members, and courses offered, as well as behaviors like scheduling classes or managing resources.

# Hybrid Inheritance:

# You're designing an online marketplace where users can buy and sell various products. How would you utilize hybrid inheritance to represent different types of users such as Buyers, Sellers, and Admins, each inheriting from a base User class and implementing interfaces like PaymentProcessor and ProductManager? Assume that each user type has specific functionalities such as making purchases, listing products for sale, and managing user accounts.