**Experiment: 5**

PART A

(PART A: TO BE REFERRED BY STUDENTS)

**Aim: To study types of inheritance, method overriding, abstraction-abstract class, abstract method.**

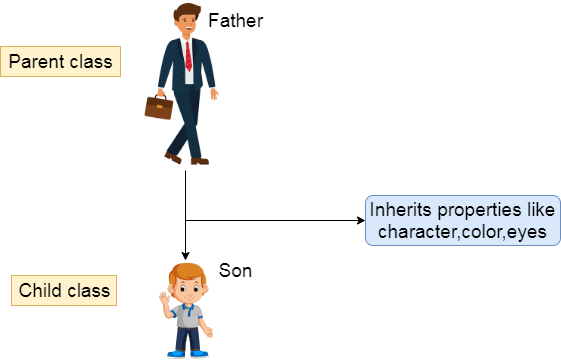
**Learning Outcomes: Learner would be able to**

1. Understand the syntax of Inheritance in JAVA
2. Interpret the types of inheritance concepts
3. Applying the method overloading and overriding(Dynamic Polymorphism) concepts in inheritance

**Theory:**

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

***extends Keyword***

****extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

**class Super {**

**..... ..... }**

**class Sub extends Super**

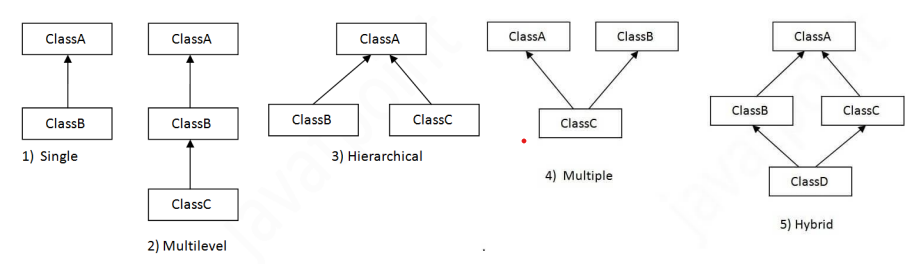
**{ ..... .....**

**}**

**Types of inheritance in Java**

Based on class, there can be three types of inheritance in java: **single, multilevel and hierarchical. However,** Java does **not support multiple inheritances with classes**. Thus to achieve **multiple inheritances only with the help of Interfaces**

In java programming, multiple and hybrid inheritance is supported through interface only



1. **Single inheritance:** When one class inherits another class, it is known as single level inheritance
2. **Multilevel inheritance:** is a process of deriving a class from another derived class
3. **Hierarchical inheritance:** is defined as the process of deriving more than one class from a base class.
4. **Multiple inheritance** is defined as the process of deriving a class from more than one base class.
5. **Hybrid inheritance:** Hybrid inheritance is a combination of simple, multiple inheritance and hierarchical inheritance.

**Polymorphism**

Polymorphism is the ability to present the same interface for differing underlying forms (data types). With polymorphism, each of these classes will have different underlying data. Precisely, Poly means ‘many’ and morphism means ‘forms’.

**Types of Polymorphism IMP**

1. Compile Time Polymorphism (Static)

2. Runtime Polymorphism (Dynamic)

**Compile Time Polymorphism:** The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Example - **Method Overloading**

**Method Overloading:** Method overloading is a technique, which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. The return type of the overloaded function.

2. The type of the parameters passed to the function.

3. The number of parameters passed to the function.

**Runtime Polymorphism:** Runtime polymorphism is also known as dynamic polymorphism. **Function Overriding is an example of runtime polymorphism**.

class Parent {

void myMethod() {

//original implementation

}

}

class Child extends Parent {

@override

void myMethod() {

//new implementation

}

}

Function Overriding means when the child class contains the method, which is already present in the parent class. Hence, the child class overrides the method of the parent class. In case of function overriding, parent and child classes both contain the same function with a different definition. The call to the function is determined at runtime is known as runtime polymorphism.

**Task 1:**

For the following Problem Statements write programs **using classes, objects and methods**

|  |  |
| --- | --- |
| 1. | A class “student” defines student with data members name, roll number and date of birth and methods as Inputdata(). Inputdata() accepts the input for instance variable. Derived the class “marks” from class “student” defines marks of various subjects, total percentage and grade of the student and methods as readdata() reads the marks out of 100, compute() calculates the percentage and assigns the grades accordingly, show() displays the name, roll number and date of birth, marks in various subjects, total, percentage and grade. |
| 2 | Create a class “Person” with fields “name” and “age” and a method “display()” that prints the name and age of the person.  Create a subclass “Employee” that extends “Person” and adds a field “salary” and a method “display\_Emp()” that prints the name, age, and salary of the employee.  Create a subclass “Manager” that extends “Employee” and adds a field “department” and a method “display\_Manager()” that prints the name, age, salary, and department of the manager.  Create an object of the “Employee” class and call the “display () and display\_Emp()” method.  Create an object of the “Manager” class and call the “display () and display\_Manager ()” method. |
| 3 | Create a class “Vehicle” with a method “drive ()” that displays a message “Driving a vehicle”. Create two subclasses “Car” and “Bike” that extend “Vehicle” and implement the “drive\_car ()” and “drive\_bike ()” method to display “Driving a car” and “Driving a bike” respectively. |
| 4 | Create a class “Person” with a method “speak ()” that displays a message “The person is speaking”. Create two subclasses “Student” and “Teacher” that extend “Person” and perform the “speak ()” method to display “The student is asking a question” and “The teacher is giving a lecture” respectively. |
| 5 | The program now uses a "Flight" base class and its subclasses "PassengerFlight" and "CargoFlight." Each subclass overrides the “fly()” method to provide specific behavior. |
| 6 | Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape. |
| 7 | Write a Java program to create an abstract class Instrument with abstract methods play() and tune(). Create subclasses for Glockenspiel and Violin that extend the Instrument class and implement the respective methods to play and tune each instrument. |

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the portal at the end of the practical. The filename should be **JAVA\_batch\_rollno\_experimentno Example: JAVA\_A1\_A001\_P1**

|  |  |
| --- | --- |
| **Roll No.: C126** | **Name: Rushabh Abhay Shah** |
| **Prog/Yr/Sem: BTI 4th Year 8th Sem** | **Batch: 2021-2027** |
| **Date of Experiment: 15-02-2025** | **Date of Submission: 15-02-2025** |

**Task 1: Program with output**

1.

import java.util.Scanner;

public class q1 {

    public static void main(String[] args) {

        Marks student1 = new Marks();

        student1.InputData();

        student1.ReadData();

        student1.Compute();

        student1.Show();

    }

}

class Student {

    String name;

    int rollNumber;

    String dateOfBirth;

    void InputData() {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter name: ");

        name = sc.nextLine();

        System.out.print("Enter roll number: ");

        rollNumber = sc.nextInt();

        sc.nextLine(); // consume the newline character

        System.out.print("Enter date of birth: ");

        dateOfBirth = sc.nextLine();

    }

}

class Marks extends Student {

    int[] marks = new int[5];

    int total;

    double percentage;

    String grade;

    void ReadData() {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter marks for 5 subjects out of 100:");

        for (int i = 0; i < 5; i++) {

            marks[i] = sc.nextInt();

        }

    }

    void Compute() {

        total = 0;

        for (int i = 0; i < 5; i++) {

            total += marks[i];

        }

        percentage = (total / 500.0) \* 100;

        if (percentage >= 90)

            grade = "A";

        else if (percentage >= 80)

            grade = "B";

        else if (percentage >= 70)

            grade = "C";

        else if (percentage >= 60)

            grade = "D";

        else

            grade = "F";

    }

    void Show() {

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

        System.out.println("Roll Number: " + rollNumber);

        System.out.println("Date of Birth: " + dateOfBirth);

        System.out.print("Marks: ");

        for (int i = 0; i < 5; i++) {

            System.out.print(marks[i] + " ");

        }

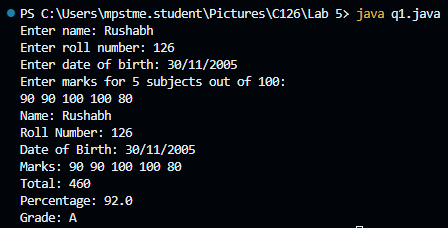
        System.out.println("\nTotal: " + total);

        System.out.println("Percentage: " + percentage);

        System.out.println("Grade: " + grade);

    }

}



2.

import java.util.Scanner;

public class q2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Employee emp = new Employee();

        System.out.print("Enter name for Employee: ");

        emp.name = sc.nextLine();

        System.out.print("Enter age for Employee: ");

        emp.age = sc.nextInt();

        System.out.print("Enter salary for Employee: ");

        emp.salary = sc.nextDouble();

        emp.display();

        emp.display\_Emp();

        sc.nextLine(); // Consume the leftover newline character

        Manager mgr = new Manager();

        System.out.print("Enter name for Manager: ");

        mgr.name = sc.nextLine();

        System.out.print("Enter age for Manager: ");

        mgr.age = sc.nextInt();

        System.out.print("Enter salary for Manager: ");

        mgr.salary = sc.nextDouble();

        sc.nextLine(); // Consume the leftover newline character

        System.out.print("Enter department for Manager: ");

        mgr.department = sc.nextLine();

        mgr.display();

        mgr.display\_Manager();

    }

}

class Person {

    String name;

    int age;

    void display() {

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

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

    }

}

class Employee extends Person {

    double salary;

    void display\_Emp() {

        super.display();

        System.out.println("Salary: " + salary);

    }

}

class Manager extends Employee {

    String department;

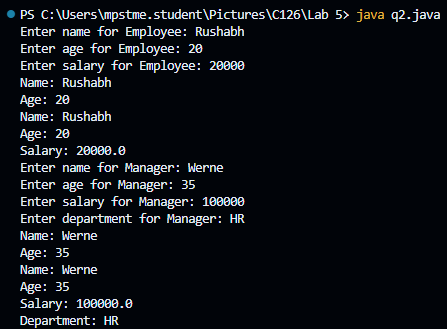
    void display\_Manager() {

        super.display\_Emp();

        System.out.println("Department: " + department);

    }

}



3.

public class q3 {

    public static void main(String[] args) {

        Car car = new Car();

        car.drive();

        car.drive\_car();

        Bike bike = new Bike();

        bike.drive();

        bike.drive\_bike();

    }

}

class Vehicle {

    void drive() {

        System.out.println("Driving a vehicle");

    }

}

class Car extends Vehicle {

    void drive\_car() {

        System.out.println("Driving a car");

    }

}

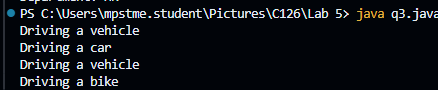
class Bike extends Vehicle {

    void drive\_bike() {

        System.out.println("Driving a bike");

    }

}



4.

public class q4 {

    public static void main(String[] args) {

        Student student = new Student();

        student.speak();

        Teacher teacher = new Teacher();

        teacher.speak();

    }

}

class Person {

    void speak() {

        System.out.println("The person is speaking");

    }

}

class Student extends Person {

    void speak() {

        System.out.println("The student is asking a question");

    }

}

class Teacher extends Person {

    void speak() {

        System.out.println("The teacher is giving a lecture");

    }

}



5.

public class q5 {

    public static void main(String[] args) {

        PassengerFlight passengerFlight = new PassengerFlight();

        passengerFlight.fly();

        CargoFlight cargoFlight = new CargoFlight();

        cargoFlight.fly();

    }

}

class Flight {

    void fly() {

        System.out.println("Flight is flying");

    }

}

class PassengerFlight extends Flight {

    @Override

    void fly() {

        System.out.println("Passenger flight is flying");

    }

}

class CargoFlight extends Flight {

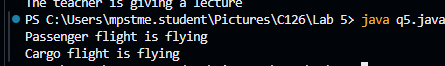
    @Override

    void fly() {

        System.out.println("Cargo flight is flying");

    }

}



6.

public class q6 {

    public static void main(String[] args) {

        Sphere sphere = new Sphere(5);

        sphere.calculateVolume();

        sphere.calculateSurfaceArea();

        Cube cube = new Cube(4);

        cube.calculateVolume();

        cube.calculateSurfaceArea();

    }

}

abstract class Shape3D {

    abstract void calculateVolume();

    abstract void calculateSurfaceArea();

}

class Sphere extends Shape3D {

    double radius;

    Sphere(double radius) {

        this.radius = radius;

    }

    void calculateVolume() {

        double volume = (4.0 / 3) \* Math.PI \* Math.pow(radius, 3);

        System.out.println("Volume of Sphere: " + volume);

    }

    void calculateSurfaceArea() {

        double surfaceArea = 4 \* Math.PI \* Math.pow(radius, 2);

        System.out.println("Surface Area of Sphere: " + surfaceArea);

    }

}

class Cube extends Shape3D {

    double side;

    Cube(double side) {

        this.side = side;

    }

    void calculateVolume() {

        double volume = Math.pow(side, 3);

        System.out.println("Volume of Cube: " + volume);

    }

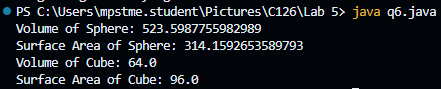
    void calculateSurfaceArea() {

        double surfaceArea = 6 \* Math.pow(side, 2);

        System.out.println("Surface Area of Cube: " + surfaceArea);

    }

}



7.

public class q7 {

    public static void main(String[] args) {

        Glockenspiel glockenspiel = new Glockenspiel();

        glockenspiel.play();

        glockenspiel.tune();

        Violin violin = new Violin();

        violin.play();

        violin.tune();

    }

}

abstract class Instrument {

    abstract void play();

    abstract void tune();

}

class Glockenspiel extends Instrument {

    void play() {

        System.out.println("Playing Glockenspiel");

    }

    void tune() {

        System.out.println("Tuning Glockenspiel");

    }

}

class Violin extends Instrument {

    void play() {

        System.out.println("Playing Violin");

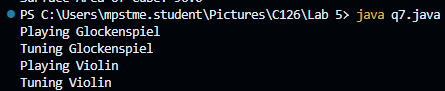
    }

    void tune() {

        System.out.println("Tuning Violin");

    }

}



**Conclusion (Learning Outcomes):** Reflect on the questions answered by you jot down your learnings about the Topic:

* **Learnt about Inheritance**
* **How to implement Inheritance in Java ?**

**Homework Questions:**

* + - 1. Examine the output of below given snippet code and write your observations or comments.

**class** D{

**protected** **int** x = 1;

**protected** **void** setX(**int** a){ x=a; }

**protected** **int** getX(){ **return** x;}

}

**class** B **extends** D{

**protected** **int** x = 3;

**public** **int** getX(){ **return** x; }

**public** **int** getB(){ **return** x; }

}

**public** **class** C {

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

// **TODO** Auto-generated method stub

D a = **new** D();

B b = **new** B();

System.***out***.println(a.getX());

System.***out***.println(b.getB());

System.***out***.println(b.getX());

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

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

}

}

**OUTPUT**

**1**

**3**

**3**

**1**

**3**

* + - 1. Examine the output of below given snippet code and write your observations or comments.

**class** X

{

**public** X() { System.***out***.println("Class A Constructor"); }

}

**class** y **extends** X

{

**public** y() { System.***out***.println("Class B Constructor"); }

}

**class** z **extends** X

{

**public** z() { System.***out***.println("Class C Constructor"); }

}

**public** **class** MainClass {

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

z a=**new** z();

y b=new y();

}

**OUTPUT**

**Class A Constructor**

**Class C Constructor**

**Class A Constructor**

**Class B Constructor**

* + - 1. Demonstrate the method override and overloading concepts in hierarchical inheritance.

Answer:

Method Overloading occurs when multiple methods have the same name but different parameters in the same class. Method Overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.

class Animal {

public void sound() {

System.out.println("Animal makes a sound");

}

// Overloaded method

public void sound(int times) {

for (int i = 0; i < times; i++) {

System.out.println("Animal makes a sound");

}

}

}

class Dog extends Animal {

// Method Overriding

@Override

public void sound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

// Method Overriding

@Override

public void sound() {

System.out.println("Cat meows");

}

}

public class Main {

public static void main(String[] args) {

Animal animal = new Animal();

Dog dog = new Dog();

Cat cat = new Cat();

animal.sound(); // Calls Animal's sound()

animal.sound(2); // Calls Animal's overloaded sound(int)

dog.sound(); // Calls Dog's overridden sound()

cat.sound(); // Calls Cat's overridden sound()

}

}

**Observations:**

Method Overloading: The Animal class has two sound() methods, one with no parameters and one with an int parameter. This is method overloading.

Method Overriding: Both Dog and Cat override the sound() method of Animal to provide their specific implementations.

**Output:**

Animal makes a sound

Animal makes a sound

Animal makes a sound

Dog barks

Cat meows