1. create multilevel inheritance for

//Vehicle

//Four\_wheeler

//Petrol\_Four\_Wheeler

//FiveSeater\_Petrol\_Four\_Wheeler

//Baleno\_FiveSeater\_Petrol\_Four\_Wheeler

Program:

// Base class

class Vehicle {

void showVehicle() {

System.out.println("This is a Vehicle");

}

}

class FourWheeler extends Vehicle {

void showFourWheeler() {

System.out.println("This is a Four Wheeler");

}

}

class PetrolFourWheeler extends FourWheeler {

void showPetrol() {

System.out.println("Runs on Petrol");

}

}

class FiveSeaterPetrolFourWheeler extends PetrolFourWheeler {

void showFiveSeater() {

System.out.println("Has 5 seats");

}

}

class BalenoFiveSeaterPetrolFourWheeler extends FiveSeaterPetrolFourWheeler {

void showBaleno() {

System.out.println("This is Baleno model");

}

}

public class MultilevelExample {

public static void main(String[] args) {

BalenoFiveSeaterPetrolFourWheeler car = new BalenoFiveSeaterPetrolFourWheeler();

car.showVehicle();

car.showFourWheeler();

car.showPetrol();

car.showFiveSeater();

car.showBaleno();

}

}

1. Demonstrate the use of the super keyword

Program:

class Animal {

String name = "Animal";

Animal() {

System.out.println("Animal Constructor");

}

void sound() {

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

}

}

class Dog extends Animal {

String name = "Dog";

Dog() {

super(); // Calls Animal constructor

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

}

void display() {

System.out.println("Parent name: " + super.name);

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

}

void sound() {

super.sound(); // Calls parent method

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

}

}

public class SuperExample {

public static void main(String[] args) {

Dog d = new Dog();

d.display();

d.sound();

}

}

1. Create Hospital super class and access this class inside the patient child class and access properties from Hospital class.

Program:

class Hospital {

String hospitalName = "City Hospital";

void showHospitalInfo() {

System.out.println("Hospital: " + hospitalName);

}

}

class Patient extends Hospital {

String patientName;

Patient(String name) {

this.patientName = name;

}

void showPatientInfo() {

showHospitalInfo(); // from Hospital

System.out.println("Patient: " + patientName);

}

}

public class HospitalExample {

public static void main(String[] args) {

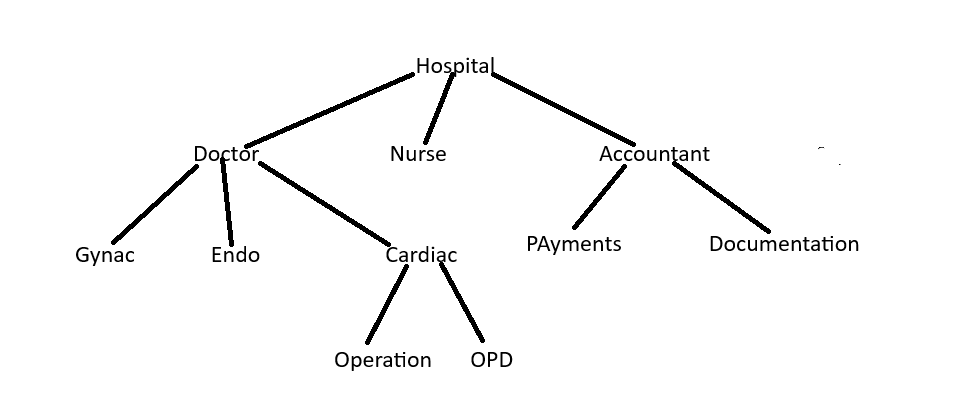
Patient p = new Patient("John Doe");

p.showPatientInfo();

}

}

1. Create Hierarchical inheritance
2. Create practice on this



Polymorphism

// Base Class

class Hospital {

void showHospital() {

System.out.println("Hospital Management System");

}

}

// Level 1

class Doctor extends Hospital {

void showDoctor() {

System.out.println("Doctor Department");

}

}

class Nurse extends Hospital {

void showNurse() {

System.out.println("Nurse Department");

}

}

class Accountant extends Hospital {

void showAccountant() {

System.out.println("Accountant Department");

}

}

// (Doctor's Specializations)

class Gynac extends Doctor {

void showGynac() {

System.out.println("Gynaecology Department");

}

}

class Endo extends Doctor {

void showEndo() {

System.out.println("Endocrinology Department");

}

}

class Cardiac extends Doctor {

void showCardiac() {

System.out.println("Cardiac Department");

}

}

(Cardiac Sub-Departments)

class Operation extends Cardiac {

void showOperation() {

System.out.println("Operation Theater - Cardiac");

}

}

class OPD extends Cardiac {

void showOPD() {

System.out.println("OPD - Cardiac");

}

}

(Accountant Sub-Departments)

class Payments extends Accountant {

void showPayments() {

System.out.println("Payments Section");

}

}

class Documentation extends Accountant {

void showDocumentation() {

System.out.println("Documentation Section");

}

}

// Main Class

public class HospitalHierarchy {

public static void main(String[] args) {

// Gynaecology

Gynac gynac = new Gynac();

gynac.showHospital();

gynac.showDoctor();

gynac.showGynac();

System.out.println();

// Cardiac -> Operation

Operation operation = new Operation();

operation.showHospital();

operation.showDoctor();

operation.showCardiac();

operation.showOperation();

System.out.println();

// Accountant -> Documentation

Documentation doc = new Documentation();

doc.showHospital();

doc.showAccountant();

doc.showDocumentation();

}

}

1. Create a class Calculator with the following overloaded add()

1.add(int a, int b)

2.add(int a, int b, int c)

3.add(double a, double b)

Program:

class Calculator {

int add(int a, int b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

double add(double a, double b) {

return a + b;

}

}

public class CalculatorExample {

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println(calc.add(2, 3));

System.out.println(calc.add(2, 3, 4));

System.out.println(calc.add(2.5, 3.7));

}

}

1. Create a base class Shape with a method area() that prints a message. Then create two subclasses Circle🡪override area() to calculator and print area of circle Rectangle🡪 override area() to calculate and print area of a rectangle

Program:

class Shape {

void area() {

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

}

}

class Circle extends Shape {

double radius;

Circle(double r) { radius = r; }

void area() {

System.out.println("Area of Circle: " + (Math.PI \* radius \* radius));

}

}

class Rectangle extends Shape {

double length, width;

Rectangle(double l, double w) { length = l; width = w; }

void area() {

System.out.println("Area of Rectangle: " + (length \* width));

}

}

public class ShapeExample {

public static void main(String[] args) {

Shape s1 = new Circle(5);

Shape s2 = new Rectangle(4, 6);

s1.area();

s2.area();

}

}

1. Create a Bank class with a method getInterestRate() create subclasses: SBI🡪return 6.7% ICICI🡪return 7.0% HDFC🡪return 7.5%

Program:

package assigment3;

class Bank {

double getInterestRate() {

return 0.0;

}

}

class SBI extends Bank {

double getInterestRate() {

return 6.7;

}

}

class ICICI extends Bank {

double getInterestRate() {

return 7.0;

}

}

// Subclass: HDFC

class HDFC extends Bank {

@Override

double getInterestRate() {

return 7.5;

}

}

public class BankMain {

public static void main(String[] args) {

Bank bank= new SBI();

System.out.println("SBI Interest Rate: " + bank.getInterestRate() + "%");

Bank bank2= new ICICI();

System.out.println("SBI Interest Rate: " + bank2.getInterestRate() + "%");

bank2.getInterestRate();

Bank bank3= new HDFC();

System.out.println("SBI Interest Rate: " + bank3.getInterestRate() + "%");

}

}

1. Runtime Polymorphism with constructor Chaining create a class vehicle with a constructor that prints “Vehicle Created”

Create a subclass Bike that override a method and uses super() in constructor

class Vehicle {

Vehicle() {

System.out.println("Vehicle Created");

}

void run() {

System.out.println("Vehicle is running");

}

}

class Bike extends Vehicle {

Bike() {

super(); // Calls Vehicle constructor

System.out.println("Bike Created");

}

@Override

void run() {

System.out.println("Bike is running fast");

}

}

public class ConstructorChainingExample {

public static void main(String[] args) {

Bike b = new Bike();

b.run();

}

}

Combined question

Create an abstract class SmartDevice with methods like turnOn(), turnOff(), and performFunction().  
Create child classes:

* SmartPhone: performs calling and browsing.
* SmartWatch: tracks fitness and time.
* SmartSpeaker: plays music and responds to voice commands.

abstract class SmartDevice {

abstract void turnOn();

abstract void turnOff();

abstract void performFunction();

}

class SmartPhone extends SmartDevice {

void turnOn() { System.out.println("SmartPhone turned ON"); }

void turnOff() { System.out.println("SmartPhone turned OFF"); }

void performFunction() { System.out.println("Calling and browsing internet"); }

}

class SmartWatch extends SmartDevice {

void turnOn() { System.out.println("SmartWatch turned ON"); }

void turnOff() { System.out.println("SmartWatch turned OFF"); }

void performFunction() { System.out.println("Tracking fitness and showing time"); }

}

class SmartSpeaker extends SmartDevice {

void turnOn() { System.out.println("SmartSpeaker turned ON"); }

void turnOff() { System.out.println("SmartSpeaker turned OFF"); }

void performFunction() { System.out.println("Playing music and responding to voice"); }

}

public class SmartDeviceTest {

public static void main(String[] args) {

SmartDevice[] devices = { new SmartPhone(), new SmartWatch(), new SmartSpeaker() };

for (SmartDevice device : devices) {

device.turnOn();

device.performFunction();

device.turnOff();

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

}

}

}

* Write code to store all objects in an array and use polymorphism to invoke their performFunction().

**2.**Design an interface Bank with methods deposit(), withdraw(), and getBalance().  
Implement this in SavingsAccount and CurrentAccount classes.

* Use inheritance to create a base Account class.
* Demonstrate method overriding with customized logic for withdrawal (e.g., minimum balance in SavingsAccount).

interface Bank {

void deposit(double amount);

void withdraw(double amount);

double getBalance();

}

class Account {

protected double balance;

Account(double initialBalance) {

balance = initialBalance;

}

}

class SavingsAccount extends Account implements Bank {

private static final double MIN\_BALANCE = 500;

SavingsAccount(double initialBalance) {

super(initialBalance);

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (balance - amount >= MIN\_BALANCE) {

balance -= amount;

} else {

System.out.println("Withdrawal denied! Minimum balance must be maintained.");

}

}

public double getBalance() {

return balance;

}

}

class CurrentAccount extends Account implements Bank {

CurrentAccount(double initialBalance) {

super(initialBalance);

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

balance -= amount;

}

public double getBalance() {

return balance;

}

}

public class BankTest {

public static void main(String[] args) {

Bank savings = new SavingsAccount(1000);

Bank current = new CurrentAccount(2000);

savings.withdraw(400);

current.withdraw(1500);

System.out.println("Savings Balance: " + savings.getBalance());

System.out.println("Current Balance: " + current.getBalance());

}

}

**3**

Create a base class Vehicle with method start().  
Derive Car, Bike, and Truck from it and override the start() method.

* Create a static method that accepts Vehicle type and calls start().
* Pass different vehicle objects to test polymorphism.

class Vehicle2 {

void start() { System.out.println("Vehicle starting"); }

}

class Car extends Vehicle2 {

void start() { System.out.println("Car starting with key"); }

}

class Bike2 extends Vehicle2 {

void start() { System.out.println("Bike starting with kick"); }

}

class Truck extends Vehicle2 {

void start() { System.out.println("Truck starting with button"); }

}

public class VehicleTest {

static void startVehicle(Vehicle2 v) {

v.start();

}

public static void main(String[] args) {

startVehicle(new Car());

startVehicle(new Bike2());

startVehicle(new Truck());

}

}

**4.**

Design an abstract class Person with fields like name, age, and abstract method getRoleInfo().  
Create subclasses:

* Student: has course and roll number.
* Professor: has subject and salary.
* TeachingAssistant: extends Student and implements getRoleInfo() in a hybrid way.
* Create and print info for all roles using overridden getRoleInfo().

abstract class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

abstract void getRoleInfo();

}

class Student extends Person {

String course;

int rollNumber;

Student(String name, int age, String course, int rollNumber) {

super(name, age);

this.course = course;

this.rollNumber = rollNumber;

}

void getRoleInfo() {

System.out.println("Student: " + name + ", Course: " + course + ", Roll No: " + rollNumber);

}

}

class Professor extends Person {

String subject;

double salary;

Professor(String name, int age, String subject, double salary) {

super(name, age);

this.subject = subject;

this.salary = salary;

}

void getRoleInfo() {

System.out.println("Professor: " + name + ", Subject: " + subject + ", Salary: " + salary);

}

}

class TeachingAssistant extends Student {

TeachingAssistant(String name, int age, String course, int rollNumber) {

super(name, age, course, rollNumber);

}

void getRoleInfo() {

System.out.println("Teaching Assistant: " + name + ", Course: " + course + ", Roll No: " + rollNumber + " (Assists Professor)");

}

}

public class PersonTest {

public static void main(String[] args) {

Person[] people = {

new Student("Alice", 20, "BSc CS", 101),

new Professor("Dr. John", 45, "Mathematics", 80000),

new TeachingAssistant("Bob", 24, "MSc Physics", 202)

};

for (Person p : people) {

p.getRoleInfo();

}

}

}

5.Create:

* Interface Drawable with method draw()
* Abstract class Shape with abstract method area()  
  Subclasses: Circle, Rectangle, and Triangle.
* Calculate area using appropriate formulas.
* Demonstrate how interface and abstract class work together.

interface Drawable {

void draw();

}

abstract class Shape implements Drawable {

abstract double area();

}

class Circle extends Shape {

double radius;

Circle(double radius) { this.radius = radius; }

double area() { return Math.PI \* radius \* radius; }

public void draw() { System.out.println("Drawing Circle"); }

}

class Rectangle extends Shape {

double length, width;

Rectangle(double length, double width) { this.length = length; this.width = width; }

double area() { return length \* width; }

public void draw() { System.out.println("Drawing Rectangle"); }

}

class Triangle extends Shape {

double base, height;

Triangle(double base, double height) { this.base = base; this.height = height; }

double area() { return 0.5 \* base \* height; }

public void draw() { System.out.println("Drawing Triangle"); }

}

public class ShapeTest {

public static void main(String[] args) {

Shape[] shapes = {

new Circle(5),

new Rectangle(4, 6),

new Triangle(3, 7)

};

for (Shape s : shapes) {

s.draw();

System.out.println("Area: " + s.area());

}

}

}