1. create multilevel inheritance for

//Vehicle

//Four\_wheeler

//Petrol\_Four\_Wheeler

//FiveSeater\_Petrol\_Four\_Wheeler

//Baleno\_FiveSeater\_Petrol\_Four\_Wheeler

a)

class Vehicle {

void display() {

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

}

}

class FourWheeler extends Vehicle {

void display() {

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

}

}

class PetrolFourWheeler extends FourWheeler {

void display() {

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

}

}

class FiveSeaterPetrolFourWheeler extends PetrolFourWheeler {

void display() {

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

}

}

class BalenoFiveSeaterPetrolFourWheeler extends FiveSeaterPetrolFourWheeler {

void display() {

System.out.println("This is a Baleno Five Seater Petrol Four Wheeler");

}

}

public class MultilevelInheritance {

public static void main(String[] args) {

BalenoFiveSeaterPetrolFourWheeler baleno = new BalenoFiveSeaterPetrolFourWheeler();

baleno.display();

}

}

Output:

This is a Baleno Five Seater Petrol Four Wheeler

1. Demonstrate the use of the super keyword

a)

class automobile {

String brand = "Generic";

void display() {

System.out.println("Vehicle Brand: " + brand);

}

}

class Car extends automobile {

String brand = "Toyota";

void display() {

super.display();

System.out.println("Car Brand: " + brand);

}

}

public class SuperKeywordDemo {

public static void main(String[] args) {

Car car = new Car();

car.display();

}

}

Output:

Vehicle Brand: Generic

Car Brand: Toyota

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

String hospitalName = "City Hospital";

String location = "Downtown";

void displayInfo() {

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

}

}

class Patient extends Hospital {

String patientName;

Patient(String patientName) {

this.patientName = patientName;

}

void showDetails() {

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

displayInfo();

}

}

public class HospitalPatient {

public static void main(String[] args) {

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

patient.showDetails();

}

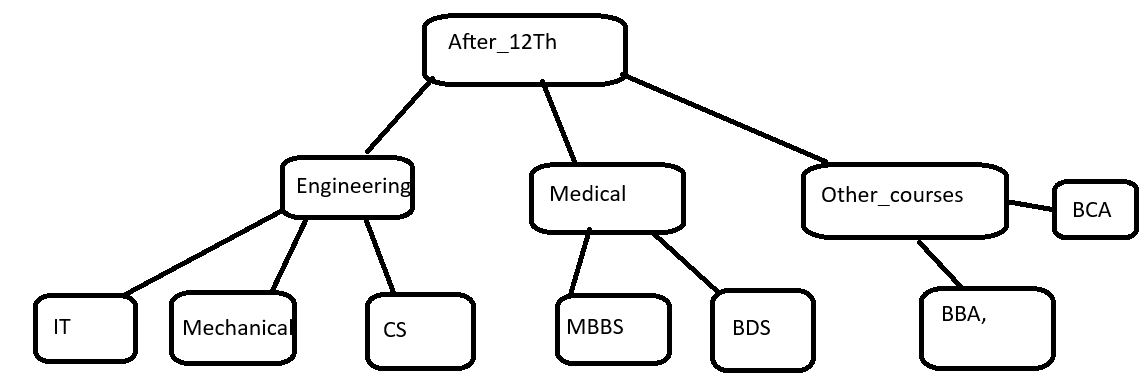
}

Output:

Patient: John Doe

Hospital: City Hospital, Location: Downtown

1. Create Hierarchical inheritance



1. class After12th {

void display () {

System.out.println("Options available after 12th:");

}

}

class Engineering extends After12th {

void showEngineering() {

System.out.println("Engineering Courses: IT, Mechanical, CS");

}

}

class Medical extends After12th {

void showMedical() {

System.out.println("Medical Courses: MBBS, BDS");

}

}

class OtherCourses extends After12th {

void showOtherCourses() {

System.out.println("Other Courses: BCA, BBA");

}

}

public class HierarchicalInheritance {

public static void main(String[] args) {

Engineering eng = new Engineering();

Medical med = new Medical();

OtherCourses other = new OtherCourses();

eng.display();

eng.showEngineering();

med.display();

med.showMedical();

other.display();

other.showOtherCourses();

}

}

Output :

Options available after 12th:

Engineering Courses: IT, Mechanical, CS

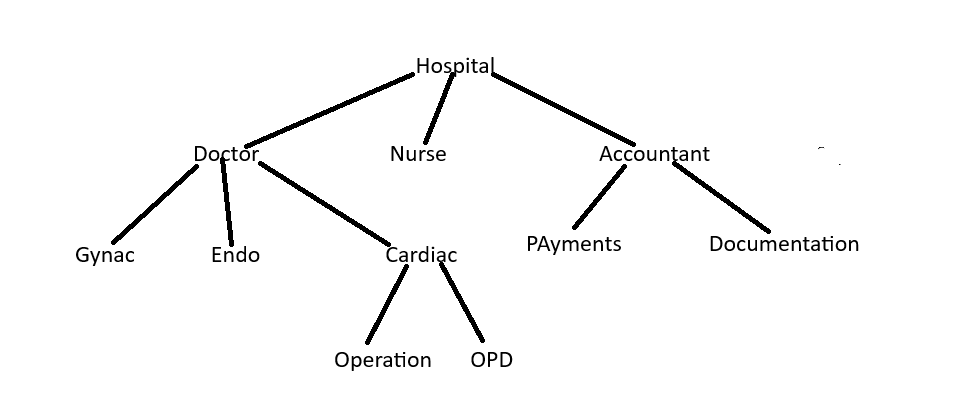
Options available after 12th:

Medical Courses: MBBS, BDS

Options available after 12th:

Other Courses: BCA, BBA

1. Create practice on this



a)  
  
 class Hospital {

void show () {

System.out.println("Hospital");

}

}

class Doctor extends Hospital {

void showDoctor() {

System.out.println(" Doctor");

}

}

class Nurse extends Hospital {

void showNurse() {

System.out.println(" Nurse");

}

}

class Accountant extends Hospital {

void showAccountant() {

System.out.println(" Accountant");

}

}

class Gynac extends Doctor {

void showGynac() {

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

}

}

class Endo extends Doctor {

void showEndo() {

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

}

}

class Cardiac extends Doctor {

void showCardiac() {

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

}

}

class Operation extends Cardiac {

void showOperation() {

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

}

}

class OPD extends Cardiac {

void showOPD() {

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

}

}

class Payments extends Accountant {

void showPayments() {

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

}

}

class Documentation extends Accountant {

void showDocumentation() {

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

}

}

public class HospitalHierarchy {

public static void main(String[] args) {

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

Hospital h = new Hospital();

h.show();

Doctor d = new Doctor();

d.showDoctor();

Gynac g = new Gynac();

g.showGynac();

Endo e = new Endo();

e.showEndo();

Cardiac c = new Cardiac();

c.showCardiac();

Operation op = new Operation();

op.showOperation();

OPD opd = new OPD();

opd.showOPD();

Nurse n = new Nurse();

n.showNurse();

Accountant a = new Accountant();

a.showAccountant();

Payments p = new Payments();

p.showPayments();

Documentation doc = new Documentation();

doc.showDocumentation();

}

}

Output:

Hospital Hierarchy

Hospital

Doctor

Gynac Department

Endo Department

Cardiac Department

Operation Department

OPD Department

Nurse

Accountant

Payments Section

Documentation Section

Polymorphism

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)

a) 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 PolymorphismCalculator {

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println("Sum of 2 integers: " + calc.add(5, 10));

System.out.println("Sum of 3 integers: " + calc.add(5, 10, 15));

System.out.println("Sum of 2 doubles: " + calc.add(5.5, 10.5));

}

}

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

a) class Shape {

void area() {

System.out.println("Calculating area of a shape");

}

}

class Circle extends Shape {

double radius;

Circle(double radius) {

this.radius = radius;

}

@Override

void area() {

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

}

}

class Rectangle extends Shape {

double length, width;

Rectangle (double length, double width) {

this.length = length;

this.width = width;

}@Override

void area() {

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

}

}

public class ShapePolymorphism {

public static void main(String[] args) {

Shape circle = new Circle(5);

Shape rectangle = new Rectangle(4, 6);

circle.area();

rectangle.area();

}

}

OutPut:

1. Create a Bank class with a method getInterestRate() create subclasses: SBIàreturn 6.7% ICICIàreturn 7.0% HDFCàreturn 7.5%

a) class Bank {

double getInterestRate() {

return 0.0;

}

}

class SBI extends Bank {

@Override

double getInterestRate() {

return 6.7;

}

}

class ICICI extends Bank {

@Override

double getInterestRate() {

return 7.0;

}

}

class HDFC extends Bank {

@Override

double getInterestRate() {

return 7.5;

}

}

public class BankInterestRate {

public static void main(String[] args) {

Bank sbi = new SBI();

Bank icici = new ICICI();

Bank hdfc = new HDFC();

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

System.out.println("ICICI Interest Rate: " + icici.getInterestRate() + "%");

System.out.println("HDFC Interest Rate: " + hdfc.getInterestRate() + "%");

}

}

Output:

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

a)

class Vehicle {

Vehicle() {

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

}

void start() {

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

}

}

class Bike extends Vehicle {

Bike() {

super();

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

}

@Override

void start() {

System.out.println("Bike starting with kick");

}

}

public class RuntimePolymorphism {

public static void main(String[] args) {

Vehicle bike = new Bike();

bike.start();

}

}

Combined question

4.1Create 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.
* Write code to store all objects in an array and use polymorphism to invoke their performFunction().

ans)

abstract class SmartDevice {

abstract void turnOn();

abstract void turnOff();

abstract void performFunction();

}

class SmartPhone extends SmartDevice {

@Override

void turnOn() {

System.out.println("SmartPhone turning on");

}

@Override

void turnOff() {

System.out.println("SmartPhone turning off");

}

@Override

void performFunction() {

System.out.println("SmartPhone: Calling and Browsing");

}

}

class SmartWatch extends SmartDevice {

@Override

void turnOn() {

System.out.println("SmartWatch turning on");

}

@Override

void turnOff() {

System.out.println("SmartWatch turning off");

}

@Override

void performFunction() {

System.out.println("SmartWatch: Tracking fitness and time");

}

}

class SmartSpeaker extends SmartDevice {

@Override

void turnOn() {

System.out.println("SmartSpeaker turning on");

}

@Override

void turnOff() {

System.out.println("SmartSpeaker turning off");

}

@Override

void performFunction() {

System.out.println("SmartSpeaker: Playing music and responding to voice commands");

}

}

public class SmartDevicePolymorphism {

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();

}

}

}

**4.2.**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).

ans)

interface Bank {

void deposit(double amount);

void withdraw(double amount);

double getBalance();

}

abstract class Account implements Bank {

protected String accountNumber;

protected double balance;

public Account(String accountNumber, double initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

}

@Override

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: " + amount);

} else {

System.out.println("Invalid deposit amount!");

}

}

@Override

public double getBalance() {

return balance;

}

}

class SavingsAccount extends Account {

private static final double MIN\_BALANCE = 500;

public SavingsAccount(String accountNumber, double initialBalance) {

super(accountNumber, initialBalance);

}

@Override

public void withdraw(double amount) {

if (amount > 0 && (balance - amount) >= MIN\_BALANCE) {

balance -= amount;

System.out.println("Withdrawn from SavingsAccount: " + amount);

} else {

System.out.println("Withdrawal denied! Minimum balance of " + MIN\_BALANCE + " must be maintained.");

}

}

}

class CurrentAccount extends Account {

private static final double OVERDRAFT\_LIMIT = 1000;

public CurrentAccount(String accountNumber, double initialBalance) {

super(accountNumber, initialBalance);

}

@Override

public void withdraw(double amount) {

if (amount > 0 && (balance - amount) >= -OVERDRAFT\_LIMIT) {

balance -= amount;

System.out.println("Withdrawn from CurrentAccount: " + amount);

} else {

System.out.println("Withdrawal denied! Overdraft limit exceeded.");

}

}

}

public class BankDemo {

public static void main(String[] args) {

SavingsAccount savings = new SavingsAccount("SA123", 2000);

savings.deposit(500);

savings.withdraw(1000);

savings.withdraw(1200); // This should fail

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

System.out.println();

CurrentAccount current = new CurrentAccount("CA456", 1000);

current.deposit(200);

current.withdraw(1500); // Allowed due to overdraft

current.withdraw(800); // This should fail

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

}

}

**4.3**

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.

Ans)

class Vehicle {

void start() {

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

}

static void startVehicle(Vehicle vehicle) {

vehicle.start();

}

}

class Car extends Vehicle {

@Override

void start() {

System.out.println("Car starting with key");

}

}

class Bike extends Vehicle {

@Override

void start() {

System.out.println("Bike starting with kick");

}

}

class Truck extends Vehicle {

@Override

void start() {

System.out.println("Truck starting with heavy engine");

}

}

public class VehiclePolymorphism {

public static void main(String[] args) {

Vehicle car = new Car();

Vehicle bike = new Bike();

Vehicle truck = new Truck();

Vehicle.startVehicle(car);

Vehicle.startVehicle(bike);

Vehicle.startVehicle(truck);

}

}

**4.4.**

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().

ans)

abstract class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

abstract String 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;

}

@Override

String getRoleInfo() {

return "Student: Name=" + name + ", Age=" + age + ", Course=" + course + ", Roll Number=" + 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;

}

@Override

String getRoleInfo() {

return "Professor: Name=" + name + ", Age=" + age + ", Subject=" + subject + ", Salary=" + salary;

}

}

class TeachingAssistant extends Student {

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

super(name, age, course, rollNumber);

}

@Override

String getRoleInfo() {

return "Teaching Assistant: Name=" + name + ", Age=" + age + ", Course=" + course + ", Roll Number=" + rollNumber;

}

}

public class PersonRoles {

public static void main(String[] args) {

Person[] people = {

new Student("Alice", 20, "Computer Science", 101),

new Professor("Bob", 45, "Mathematics", 75000),

new TeachingAssistant("Charlie", 25, "Physics", 102)

};

for (Person person : people) {

System.out.println(person.getRoleInfo());

}

}

}

4.5.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.

ans)

interface Drawable {

void draw();

}

abstract class Shape {

abstract double area();

}

class Circle extends Shape implements Drawable {

double radius;

Circle(double radius) {

this.radius = radius;

}

@Override

double area() {

return Math.PI \* radius \* radius;

}

@Override

public void draw() {

System.out.println("Drawing Circle with radius " + radius);

}

}

class Rectangle extends Shape implements Drawable {

double length, width;

Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

double area() {

return length \* width;

}

@Override

public void draw() {

System.out.println("Drawing Rectangle with length " + length + " and width " + width);

}

}

class Triangle extends Shape implements Drawable {

double base, height;

Triangle(double base, double height) {

this.base = base;

this.height = height;

}

@Override

double area() {

return 0.5 \* base \* height;

}

@Override

public void draw() {

System.out.println("Drawing Triangle with base " + base + " and height " + height);

}

}

public class DrawableShape {

public static void main(String[] args) {

Drawable[] shapes = { new Circle(5), new Rectangle(4, 6), new Triangle(3, 4) };

for (Drawable shape : shapes) {

shape.draw();

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

}

}

}