1.create multilevel inheritance for

//Vehicle

//Four_wheeler

//Petrol_Four_Wheeler

//FiveSeater_Petrol_Four_Wheeler

//Baleno_FiveSeater_Petrol_Four_Wheeler

```
// Multilevel Inheritance Structure
class Vehicle {
    void type() {
        System.out.println("This is a Vehicle.");
    }
}

class FourWheeler extends Vehicle {
    void wheels() {
        System.out.println("It has four wheels.");
    }
}

class PetrolFourWheeler extends FourWheeler {
    void fuelType() {
        System.out.println("Runs on petrol.");
    }
}

class FiveSeaterPetrolFourWheeler extends PetrolFourWheeler {
    void seating() {
        System.out.println("Seating capacity: 5");
    }
}

class BalenoFiveSeaterPetrolFourWheeler extends FiveSeaterPetrolFourWheeler {
    void modelName() {
```

```
        System.out.println("Model: Baleno");
    }
}
```

// Main Class to Test the Inheritance Chain

```
public class Main {
    public static void main(String[] args) {
        BalenoFiveSeaterPetrolFourWheeler car = new BalenoFiveSeaterPetrolFourWheeler();

        car.type();
        car.wheels();
        car.fuelType();
        car.seating();
        car.modelName();
    }
}
```

2.Demonstrate the use of the super keyword

```
class Vehicle {
    String brand = "Generic Vehicle";

    Vehicle() {
        System.out.println("Vehicle constructor called.");
    }

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

class Car extends Vehicle {
    String brand = "Car";
```

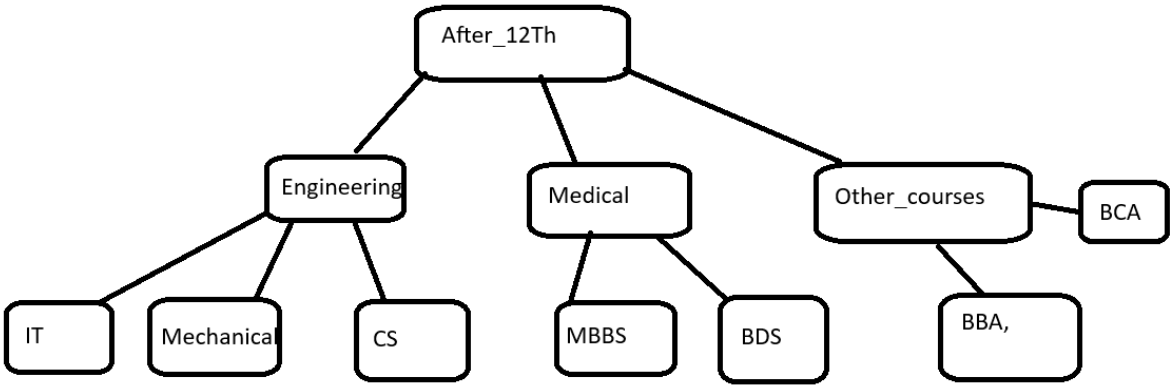
```
    Car() {
        super();
        System.out.println("Car constructor called.");
    }

    void displayBrands() {
        System.out.println("Child brand: " + brand);
        System.out.println("Parent brand: " + super.brand);
    }

    @Override
    void start() {
        super.start();
        System.out.println("Car is starting...");
    }
}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        System.out.println();
        car.displayBrands();
        System.out.println();
        car.start();
    }
}
```

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

```
class Hospital {
    String hospitalName = "City Care Hospital";
    String location = "Downtown";
```

```
    void displayHospitalInfo() {
        System.out.println("Hospital: " + hospitalName);
        System.out.println("Location: " + location);
    }
}

class Patient extends Hospital {
    String patientName;
    int patientId;

    Patient(String name, int id) {
        this.patientName = name;
        this.patientId = id;
    }

    void displayPatientInfo() {
        System.out.println("Patient Name: " + patientName);
        System.out.println("Patient ID: " + patientId);
        System.out.println("Admitted to:");
        super.displayHospitalInfo();
    }
}

public class Main {
    public static void main(String[] args) {
        Patient p = new Patient("Amit Sharma", 101);
        p.displayPatientInfo();
    }
}
```

4.Create Hierarchical inheritance




```
class After_12Th {
    void show() {
        System.out.println("Available career paths after 12th:");
    }
}

class Engineering extends After_12Th {
    void showEngineering() {
        System.out.println("Engineering Fields: IT, Mechanical, CS");
    }
}

class IT extends Engineering {
    void showIT() {
        System.out.println("Specialization: IT Engineering");
    }
}

class Mechanical extends Engineering {
    void showMechanical() {
        System.out.println("Specialization: Mechanical Engineering");
    }
}

class CS extends Engineering {
```

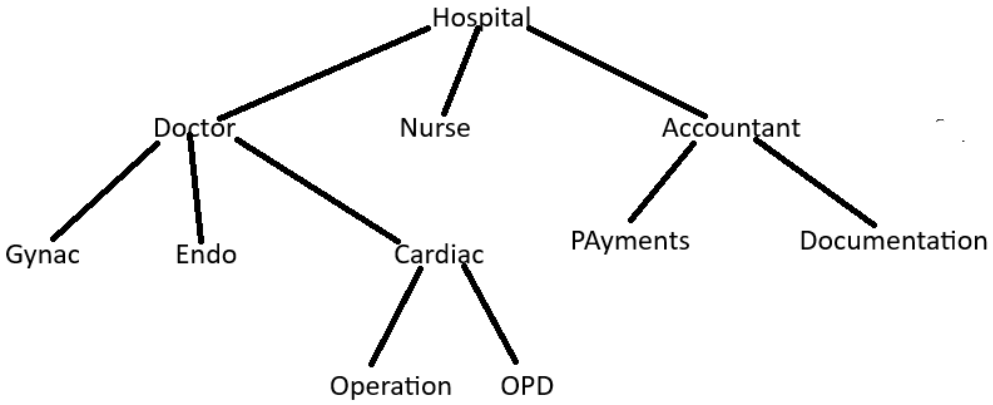
```
    void showCS() {
        System.out.println("Specialization: Computer Science Engineering");
    }
}

class Medical extends After_12Th {
    void showMedical() {
        System.out.println("Medical Fields: MBBS, BDS");
    }
}

class MBBS extends Medical {
    void showMBBS() {
        System.out.println("Specialization: MBBS");
    }
}

class BDS extends Medical {
    void showBDS() {
        System.out.println("Specialization: BDS");
    }
}

class OtherCourses extends After_12Th {
    void showOtherCourses() {
        System.out.println("Other Courses: BBA, BCA");
    }
}

class BBA extends OtherCourses {
    void showBBA() {
        System.out.println("Specialization: BBA");
    }
}
```

```
class BCA extends OtherCourses {
    void showBCA() {
        System.out.println("Specialization: BCA");
    }
}
```

// Main Method to Demonstrate the Structure

```
public class Main {
    public static void main(String[] args) {
        IT it = new IT();
        it.show();
        it.showEngineering();
        it.showIT();

        System.out.println();

        MBBS mbbs = new MBBS();
        mbbs.show();
        mbbs.showMedical();
        mbbs.showMBBS();

        System.out.println();

        BCA bca = new BCA();
        bca.show();
        bca.showOtherCourses();
        bca.showBCA();
    }
}
```

5.Create practice on this




```
class Hospital {
    void showHospital() {
        System.out.println("This is the Hospital.");
    }
}

class Doctor extends Hospital {
    void showDoctor() {
        System.out.println("Doctor Department");
    }
}

class Gynac extends Doctor {
    void showGynac() {
        System.out.println("Gynacologist Section");
    }
}

class Endo extends Doctor {
    void showEndo() {
        System.out.println("Endocrinologist Section");
    }
}
```

```
class Cardiac extends Doctor {
    void showCardiac() {
        System.out.println("Cardiology Department");
    }
}

class Operation extends Cardiac {
    void showOperation() {
        System.out.println("Cardiac Surgery Operation Room");
    }
}

class OPD extends Cardiac {
    void showOPD() {
        System.out.println("Cardiac OPD Section");
    }
}

class Nurse extends Hospital {
    void showNurse() {
        System.out.println("Nursing Staff");
    }
}

class Accountant extends Hospital {
    void showAccountant() {
        System.out.println("Accounts Department");
    }
}

class Payments extends Accountant {
    void showPayments() {
        System.out.println("Payments Section");
    }
```

```
}

class Documentation extends Accountant {
    void showDocumentation() {
        System.out.println("Documentation Section");
    }
}
// Main Method to Test All Classes
public class Main {
    public static void main(String[] args) {
        Gynac gynac = new Gynac();
        gynac.showHospital();
        gynac.showDoctor();
        gynac.showGynac();

        System.out.println();

        Operation operation = new Operation();
        operation.showHospital();
        operation.showDoctor();
        operation.showCardiac();
        operation.showOperation();

        System.out.println();

        Nurse nurse = new Nurse();
        nurse.showHospital();
        nurse.showNurse();

        System.out.println();

        Payments payment = new Payments();
        payment.showHospital();
        payment.showAccountant();
```

```
        payment.showPayments();
    }
}
```

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)

```
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 Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("add(int, int): " + calc.add(10, 20));
        System.out.println("add(int, int, int): " + calc.add(5, 10, 15));
        System.out.println("add(double, double): " + calc.add(3.5, 4.5));
    }
}
```

2.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

```
class Shape {
    void area() {
        System.out.println("Calculating area...");
    }
}

class Circle extends Shape {
    double radius;

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

    @Override
    void area() {
        double result = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + result);
    }
}

class Rectangle extends Shape {
    double length, width;

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

    @Override
    void area() {
        double result = length * width;
```

```
        System.out.println("Area of Rectangle: " + result);
    }
}

public class Main {
    public static void main(String[] args) {
        Shape s1 = new Circle(5);
        s1.area();

        Shape s2 = new Rectangle(4, 6);
        s2.area();
    }
}
```

3.Create a Bank class with a method getInterestRate()

create subclasses:

SBI→return 6.7%

ICICI→return 7.0%

HDFC→return 7.5%

```
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 Main {
    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() + "%");
    }
}
```

4.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

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

```
class Vehicle {
    Vehicle() {
        System.out.println("Vehicle Created");
    }

    void run() {
        System.out.println("Vehicle is running");
    }
}

class Bike extends Vehicle {
    Bike() {
        super();
        System.out.println("Bike Created");
    }

    @Override
    void run() {
        System.out.println("Bike is running safely");
    }
}

abstract class SmartDevice {
    abstract void turnOn();
    abstract void turnOff();
    abstract void performFunction();
}

class SmartPhone extends SmartDevice {
    void turnOn() {
        System.out.println("SmartPhone is turned on");
```

```
    }

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

    void performFunction() {
        System.out.println("SmartPhone is making calls and browsing the internet");
    }
}

class SmartWatch extends SmartDevice {
    void turnOn() {
        System.out.println("SmartWatch is turned on");
    }

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

    void performFunction() {
        System.out.println("SmartWatch is tracking fitness and time");
    }
}

class SmartSpeaker extends SmartDevice {
    void turnOn() {
        System.out.println("SmartSpeaker is turned on");
    }

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

    void performFunction() {
```

```
        System.out.println("SmartSpeaker is playing music and responding to voice commands");
    }
}

public class Main {
    public static void main(String[] args) {
        Bike bike = new Bike();
        bike.run();

        System.out.println();

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

        for (SmartDevice device : devices) {
            device.turnOn();
            device.performFunction();
            device.turnOff();
            System.out.println();
        }
    }
}
```

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) {
        this.balance = initialBalance;
    }
}

class SavingsAccount extends Account implements Bank {
    private final double minimumBalance = 500;

    SavingsAccount(double initialBalance) {
        super(initialBalance);
    }

    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited to Savings: " + amount);
    }

    public void withdraw(double amount) {
        if (balance - amount >= minimumBalance) {
            balance -= amount;
            System.out.println("Withdrawn from Savings: " + 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;
        System.out.println("Deposited to Current: " + amount);
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn from Current: " + amount);
        } else {
            System.out.println("Withdrawal denied: Insufficient funds.");
        }
    }

    public double getBalance() {
        return balance;
    }
}

public class Main {
    public static void main(String[] args) {
        Bank savings = new SavingsAccount(1000);
        savings.deposit(500);
        savings.withdraw(900);
        System.out.println("Savings Balance: " + savings.getBalance());
```

```
        System.out.println();

        Bank current = new CurrentAccount(2000);
        current.deposit(1000);
        current.withdraw(2500);
        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 Vehicle {
    void start() {
        System.out.println("Vehicle is starting");
    }
}

class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("Car is starting");
    }
}

class Bike extends Vehicle {
    @Override
    void start() {
        System.out.println("Bike is starting");
    }
}
```

```
class Truck extends Vehicle {
    @Override
    void start() {
        System.out.println("Truck is starting");
    }
}

public class Main {
    static void startVehicle(Vehicle v) {
        v.start();
    }

    public static void main(String[] args) {
        Vehicle car = new Car();
        Vehicle bike = new Bike();
        Vehicle truck = new Truck();

        startVehicle(car);
        startVehicle(bike);
        startVehicle(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: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("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;
    }

    void getRoleInfo() {
        System.out.println("Professor Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Subject: " + subject);
        System.out.println("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: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Assisting in teaching responsibilities.");
    }
}

public class Main {
    public static void main(String[] args) {
        Person student = new Student("Alice", 20, "B.Sc", 101);
        Person professor = new Professor("Dr. Smith", 45, "Physics", 75000);
        Person ta = new TeachingAssistant("Bob", 22, "M.Sc", 201);
```

```
        student.getRoleInfo();
        System.out.println();
        professor.getRoleInfo();
        System.out.println();
        ta.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 {
    abstract double area();
}

class Circle extends Shape implements Drawable {
    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 implements Drawable {
    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 implements Drawable {
    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 Main {
    public static void main(String[] args) {
        Drawable[] drawables = {
            new Circle(5),
            new Rectangle(4, 6),
            new Triangle(3, 7)
        };

        for (Drawable d : drawables) {
            d.draw();
            Shape s = (Shape) d;
            System.out.println("Area: " + s.area());
            System.out.println();
        }
    }
}
```