1. create multilevel inheritance for
Vehicle
Four_wheeler
Petrol_Four_Wheeler
FiveSeater_Petrol_Four_Wheeler
Baleno_FiveSeater_Petrol_Four_Wheeler

```
package day4_Assessment;

class Vehicle {
    void vehicleType() {
        System.out.println("This is a vehicle");
    }
}

class FourWheeler extends Vehicle {
    void wheels() {
        System.out.println("It has 4 wheels");
    }
}

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

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

class BalenoFiveSeaterPetrolFourWheeler extends FiveSeaterPetrolFourWheeler {
    void modelName() {
        System.out.println("Model: Baleno");
    }
}

public class Multilevel_Inheritance {
    public static void main(String[] args) {
        BalenoFiveSeaterPetrolFourWheeler car = new BalenoFiveSeaterPetrolFourWheeler();
        car.vehicleType();
        car.wheels();
        car.fuelType();
        car.seatingCapacity();
        car.modelName();
    }
}
```

output
This is a vehicle
It has 4 wheels
Runs on petrol
Seating capacity: 5
Model: Baleno

1. Demonstrate the use of the super keyword

```
package day4_Assessment;

        class Person {
            String name;
            int age;

            Person(String name, int age) {
                this.name = name;
                this.age = age;
                System.out.println("Person constructor called");
            }

            void displayInfo() {
                System.out.println("Name: " + name);
                System.out.println("Age: " + age);
            }
        }

        class Student extends Person {
            int rollNumber;

            Student(String name, int age, int rollNumber) {
                super(name, age);
                this.rollNumber = rollNumber;
                System.out.println("Student constructor called");
            }

            void displayDetails() {
                super.displayInfo();
                System.out.println("Roll Number: " + rollNumber);
            }
        }

        public class SuperKeyword {
            public static void main(String[] args) {
                Student s = new Student("Jhon", 21, 101);
                s.displayDetails();
            }
        }
```

output
Person constructor called

Student constructor called
Name: Jhon
Age: 21
Roll Number: 101

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

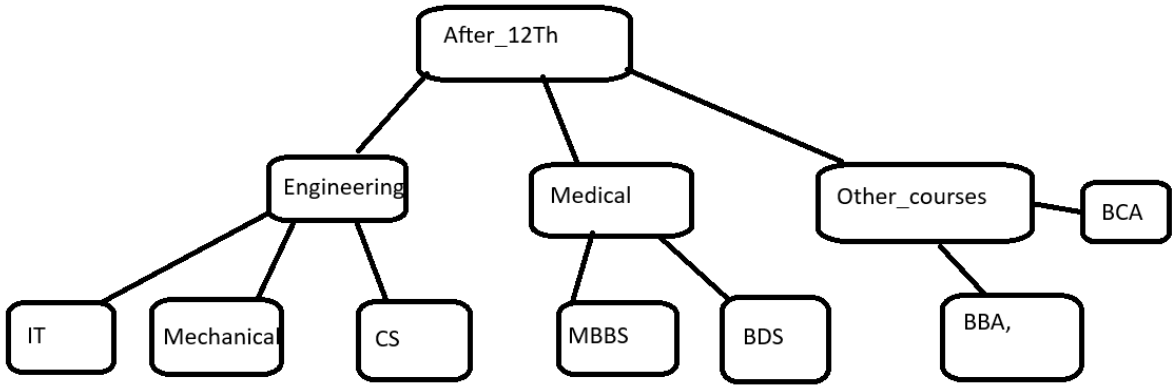
```
package day4_Assessment;
//Superclass: Hospital
class Hospital2 {
 String hospitalName;
 String location;
 public Hospital2(String hospitalName, String location) {
    this.hospitalName = hospitalName;
    this.location = location;
 }
 public void showHospitalDetails() {
    System.out.println("Hospital Name: " + hospitalName);
    System.out.println("Location: " + location);
 }
}
class Patient2 extends Hospital2 {
 String patientName;
 int age;
 public Patient2(String hospitalName, String location, String patientName, int age) {
    super(hospitalName, location);
    this.patientName = patientName;
    this.age = age;
 }
 public void showPatientDetails() {
    System.out.println("Patient Name: " + patientName);
    System.out.println("Age: " + age);
    System.out.println("--- Hospital Info ---");
    super.showHospitalDetails();
 }
}
public class HospitalDemo {
 public static void main(String[] args) {
    Patient2 p = new Patient2("City Care Hospital", "Hyderabad", "Manasa Aduvala", 25);
    p.showPatientDetails();
 }
}
```

Output:

Patient Name: Manasa Aduvala
Age: 25

--- Hospital Info ---
Hospital Name: City Care Hospital
Location: Hyderabad

3. Create Hierarchical inheritance




```
package day4_Assessment;

class After12thCollege {
    After12thCollege() {
        System.out.println("After 12th, you can join college.");
    }

    void showCourses() {
        System.out.println("Courses: Engineering, Medical, Other Courses");
    }
}

class Engineering extends After12thCollege {
    Engineering() {
        super();
        System.out.println("Welcome to Engineering Stream.");
    }

    void showBranches() {
        System.out.println("Engineering has branches: IT, Mechanical, CS");
    }
}

class Medical extends After12thCollege {
    Medical() {
        super();
        System.out.println("Welcome to Medical Stream.");
    }

    void showBranches() {
        System.out.println("Medical has branches: MBBS, BDS");
```

```
    }
}

class OtherCourses extends After12thCollege {
    OtherCourses() {
        super();
        System.out.println("Welcome to Other Courses Stream.");
    }

    void showBranches() {
        System.out.println("Other Courses: BCA, BBA");
    }
}

class IT extends Engineering {
    IT() {
        super();
        System.out.println("You selected IT Branch.");
    }
}

class Mechanical extends Engineering {
    Mechanical() {
        super();
        System.out.println("You selected Mechanical Branch.");
    }
}

class CS extends Engineering {
    CS() {
        super();
        System.out.println("You selected CS Branch.");
    }
}

class MBBS extends Medical {
    MBBS() {
        super();
        System.out.println("You selected MBBS Course.");
    }
}

class BDS extends Medical {
    BDS() {
        super();
        System.out.println("You selected BDS Course.");
    }
}

class BCA extends OtherCourses {
```

```
    BCA() {
        super();
        System.out.println("You selected BCA Course.");
    }
}

class BBA extends OtherCourses {
    BBA() {
        super();
        System.out.println("You selected BBA Course.");
    }
}

public class CollegeHierarchy {
    public static void main(String[] args) {
        System.out.println("---- IT Student ----");
        IT itStudent = new IT();

        System.out.println("\n---- MBBS Student ----");
        MBBS mbbsStudent = new MBBS();

        System.out.println("\n---- BCA Student ----");
        BCA bcaStudent = new BCA();
    }
}
```

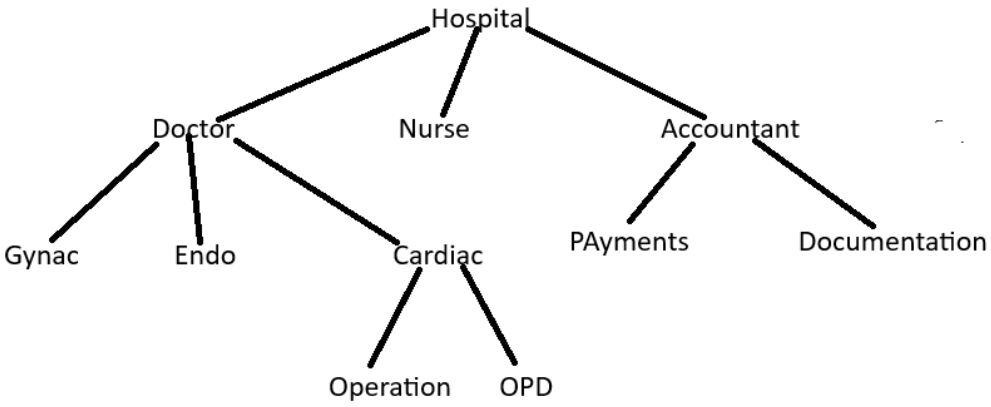
output

```
 ---- IT Student ----
After 12th, you can join college.
Welcome to Engineering Stream.
You selected IT Branch.

---- MBBS Student ----
After 12th, you can join college.
Welcome to Medical Stream.
You selected MBBS Course.

---- BCA Student ----
After 12th, you can join college.
Welcome to Other Courses Stream.
You selected BCA Course.
```

4. Create practice on this




```
package day4_Assessment;

interface Doctor
{
	void operation();
	void OPD();
}
interface Nurse
{
	void dailyCheck();
	void documentation();
}
interface Accountant
{
	void payment();
	void query();
}
class Patient implements Doctor, Nurse, Accountant
{
	public void operation()
	{
		System.out.println("Doctor came to perform operation");
	}
	public void OPD()
	{
		System.out.println("OPD is done");
	}
	public void dailyCheck()
	{
		System.out.println("Nurse came to done the regular checkup");
	}
	public void documentation()
	{
		System.out.println("Documentation is done");
	}
	public void payment()
```

```
	{
		System.out.println("payment is done");
	}
	public void query()
	{
		System.out.println("Accountant verified queries");
	}
}
public class HospitalDemo2 {
	public static void main(String args[])
	{
		Patient p=new Patient();
		p.operation();
		p.OPD();
		p.dailyCheck();
		p.documentation();
		p.payment();
		p.query();
	}

}
```

output

```
Doctor came to perform operation
OPD is done
Nurse came to done the regular checkup
Documentation is done
payment is done
Accountant verified queries
```

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)

```
package day4_Assessment;
public class OverloadingAddMethod {
	public int add(int a, int b) {
	   return a + b;
	}

	public int add(int a, int b, int c) {
	   return a + b + c;
	}

	public double add(double a, double b) {
	   return a + b;
```

```
    }

    public static void main(String[] args) {
        OverloadingAddMethod calc = new OverloadingAddMethod();
       System.out.println("Addition two integers: " + calc.add(10, 20));
       System.out.println("Addition three integers: " + calc.add(10, 20, 30));
       System.out.println("Addition two floating point numbers: " + calc.add(10.5,
20.3));
    }
}
```

output
Addition two integers: 30
Addition three integers: 60
Addition two floating point numbers: 30.8

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

```
package day4_Assessment;
class Shape {
   void area() {
      System.out.println("Area method in Shape class");
   }
}
class Circle extends Shape {
   double radius;
   Circle(double radius) {
      this.radius = radius;
   }
   void area() {
      double result = 3.14 * 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;
   }
   void area() {
      double result = length * width;
      System.out.println("Area of Rectangle: " + result);
   }
}
public class CalculatingArea {
   public static void main(String[] args) {
```

```
        Shape s;
        s = new Circle(7);
        s.area();
        s = new Rectangle(5, 10);
        s.area();
    }
}
```

output:

Area of Circle: 153.86
Area of Rectangle: 50.0

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

```
package day4_Assessment;
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;
    }
}
class HDFC extends Bank {
    double getInterestRate() {
        return 7.5;
    }
}

public class BankInterestRate {
    public static void main(String[] args) {
        Bank bank;

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

        bank = new ICICI();
        System.out.println("ICICI Interest Rate: " + bank.getInterestRate() + "%");

        bank = new HDFC();
```

```
        System.out.println("HDFC Interest Rate: " + bank.getInterestRate() + "%");
    }
}
```

output
SBI Interest Rate: 6.7%
ICICI Interest Rate: 7.0%
HDFC Interest Rate: 7.5%

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

```
package day4_Assessment;
class Run {
    Run() {
        System.out.println("Vehicle Created");
    }
    void start() {
        System.out.println("Vehicle is starting...");
    }
}
class Bike extends Vehicle {
    Bike() {
        super();
        System.out.println("Bike Created");
    }
    void start() {
        System.out.println("Bike is starting with a kick!");
    }
}
public class Polymorphism {
    public static void main(String[] args) {
        Bike v;
        v = new Bike();
        v.start();
    }
}
```

Output:
Bike Created
Bike is starting with a kick!

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().

```
package day4_Assessment;

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 used for calling and browsing.");
 }
}
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 tracks fitness and shows 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 plays music and responds to voice commands.");
 }
}
public class SmartDeviceTest {
 public static void main(String[] args) {
    SmartDevice[] devices = new SmartDevice[3];
    devices[0] = new SmartPhone();
    devices[1] = new SmartWatch();
    devices[2] = new SmartSpeaker();
    for (SmartDevice device : devices) {
       device.turnOn();
       device.performFunction();
       device.turnOff();
    }
 }
}
```

output

```
SmartPhone is turned ON
SmartPhone is used for calling and browsing.
SmartPhone is turned OFF
SmartWatch is turned ON
SmartWatch tracks fitness and shows time.
SmartWatch is turned OFF
SmartSpeaker is turned ON
SmartSpeaker plays music and responds to voice commands.
SmartSpeaker is turned OFF
```

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).

```
package day4_Assessment;
interface Bank1 {
 void deposit(double amount);
 void withdraw(double amount);
 double getBalance();
}
abstract class Account implements Bank1 {
 protected double balance;

 public Account(double initialBalance) {
```

```
    this.balance = initialBalance;
 }
public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    } else {
        System.out.println("Invalid deposit amount!");
    }
 }
 public double getBalance() {
    return balance;
 }
}

class SavingsAccount extends Account {
 private final double MIN_BALANCE = 500; // minimum required balance

 public SavingsAccount(double initialBalance) {
    super(initialBalance);
 }
 public void withdraw(double amount) {
    if (balance - amount >= MIN_BALANCE) {
        balance -= amount;
        System.out.println("Withdrawn from Savings: " + amount);
    } else {
        System.out.println("Cannot withdraw! Minimum balance of " + MIN_BALANCE + "
must be maintained.");
    }
 }
}
class CurrentAccount extends Account {
 public CurrentAccount(double initialBalance) {
    super(initialBalance);
 }
 public void withdraw(double amount) {
    if (amount <= balance) {
        balance -= amount;
        System.out.println("Withdrawn from Current: " + amount);
    } else {
        System.out.println("Insufficient balance in Current Account!");
    }
 }
}
public class BankInterface {
 public static void main(String[] args) {
    Bank1 savings = new SavingsAccount(2000);
    Bank1 current = new CurrentAccount(5000);
    System.out.println("=== Savings Account Transactions ===");
    savings.deposit(1000);
```

```
    savings.withdraw(2300);
    savings.withdraw(3000);
    System.out.println("Savings Balance: " + savings.getBalance());
    System.out.println("\n=== Current Account Transactions ===");
    current.deposit(2000);
    current.withdraw(6000);
    current.withdraw(2000);
    System.out.println("Current Balance: " + current.getBalance());
 }
}
```

Output:

```
=== Savings Account Transactions ===
Deposited: 1000.0
Withdrawn from Savings: 2300.0
Cannot withdraw! Minimum balance of 500.0 must be maintained.
Savings Balance: 700.0

=== Current Account Transactions ===
Deposited: 2000.0
Withdrawn from Current: 6000.0
Insufficient balance in Current Account!
Current Balance: 1000.0
```

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.

```
package day4_Assessment;
class Transport {
 void start() {
    System.out.println("Transport is starting...");
 }
}

class Car extends Transport {
 void start() {
    System.out.println("Car is starting with a key ignition.");
 }
}
class TwoWheeler extends Transport {
 void start() {
    System.out.println("Bike is starting with a kick or self-start.");
 }
}
```

```
class Truck extends Transport {
 void start() {
    System.out.println("Truck is starting with a heavy-duty ignition.");
 }
}

public class TransportTest {
 static void startTransport(Transport t) {
    t.start();
 }

 public static void main(String[] args) {
    Transport car = new Car();
    Transport bike = new TwoWheeler();
    Transport truck = new Truck();
    System.out.println("***Car***");
    startTransport(car);

    System.out.println("\n***Bike***");
    startTransport(bike);

    System.out.println("\n***Truck***");
    startTransport(truck);
 }
}
```

output:

```
***Car***
Car is starting with a key ignition.

***Bike***
Bike is starting with a kick or self-start.

***Truck***
Truck is starting with a heavy-duty ignition.
```

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().

```
package day4_Assessment;
abstract class Persons {
 String name;
 int age;
```

```
 Persons(String name, int age) {
    this.name = name;
    this.age = age;
 }
 abstract void getRoleInfo();
}
class Students extends Persons {
 String course;
 int rollNumber;

 Students(String name, int age, String course, int rollNumber) {
    super(name, age);
    this.course = course;
    this.rollNumber = rollNumber;
 }
 void getRoleInfo() {
    System.out.println("Student Info:");
    System.out.println("Name: " + name + ", Age: " + age);
    System.out.println("Course: " + course + ", Roll No: " + rollNumber);
 }
}
class Professor extends Persons {
 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 Info:");
    System.out.println("Name: " + name + ", Age: " + age);
    System.out.println("Subject: " + subject + ", Salary: ₹" + salary);
 }
}
class TeachingAssistant extends Students {
 String assistSubject;
 TeachingAssistant(String name, int age, String course, int rollNumber, String assistSubject)
{
    super(name, age, course, rollNumber);
    this.assistSubject = assistSubject;
 }
 void getRoleInfo() {
    System.out.println("Teaching Assistant Info:");
    System.out.println("Name: " + name + ", Age: " + age);
    System.out.println("Course: " + course + ", Roll No: " + rollNumber);
    System.out.println("Assisting Subject: " + assistSubject);
 }
}
```

```
public class RoleTest {
 public static void main(String[] args) {
    Persons p1 = new Students("Manasa", 20, "B.Tech CSE", 101);
    Persons p2 = new Professor("Dr. Rao", 45, "Operating Systems", 95000);
    Persons p3 = new TeachingAssistant("Raj", 22, "B.Tech CSE", 202, "Java
Programming");

    System.out.println("**************");
    p1.getRoleInfo();

    System.out.println("\n**************");
    p2.getRoleInfo();

    System.out.println("\n**************");
    p3.getRoleInfo();
 }
}
```

output:

```
**************
Student Info:
Name: Manasa, Age: 20
Course: B.Tech CSE, Roll No: 101

**************
Professor Info:
Name: Dr. Rao, Age: 45
Subject: Operating Systems, Salary: ₹95000.0

**************
Teaching Assistant Info:
Name: Raj, Age: 22
Course: B.Tech CSE, Roll No: 202
Assisting Subject: Java Programming
```

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.

```
package day4_Assessment;
interface Drawable {
   void draw();
}
abstract class Shape1 {
   abstract double area1();
}
```

```
class Circle1 extends Shape1 implements Drawable {
    double radius;
    Circle1(double radius) {
        this.radius = radius;
    }
    public void draw() {
        System.out.println("Drawing Circle");
    }
    double area1() {
        return Math.PI * radius * radius;
    }
}
class Rectangle1 extends Shape1 implements Drawable {
    double length, width;
    Rectangle1(double length, double width) {
        this.length = length;
        this.width = width;
    }
    public void draw() {
        System.out.println("Drawing Rectangle");
    }
    double area1() {
        return length * width;
    }
}
class Triangle extends Shape1 implements Drawable {
    double base, height;
    Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }
    public void draw() {
        System.out.println("Drawing Triangle");
    }
    double area1() {
        return 0.5 * base * height;
    }
}
public class AreaUsingInterface {
    public static void main(String[] args) {
        Drawable d1 = new Circle1(5);
        Shape1 s1 = (Shape1) d1;
        d1.draw();
        System.out.println("Area of Circle: " +s1.area1());
        Drawable d2 = new Rectangle1(4, 6);
        Shape1 s2 = (Shape1) d2;
        d2.draw();
        System.out.println("Area of Rectangle: " +s2.area1());
        Drawable d3 = new Triangle(3, 7);
        Shape1 s3 = (Shape1) d3;
```

```
        d3.draw();
        System.out.println("Area of Triangle: " +s3.area1());
    }
}
```

output:

```
Drawing Circle
Area of Circle: 78.53981633974483
Drawing Rectangle
Area of Rectangle: 24.0
Drawing Triangle
Area of Triangle: 10.5
```
