Q. Create multilevel inheritance for
Vehicle
Four_wheeler
Petrol_Four_Wheeler
FiveSeater_Petrol_Four_Wheeler
Baleno_FiveSeater_Petrol_Wheeler

```
class Vehicle

{

   void start()

   {

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

   }

}

class Four_wheeler extends Vehicle

{

   void type()

   {

      System.out.println("Four wheeler");

   }

}

class Petrol_Four_Wheeler extends Four_wheeler

{

   void fuel()

   {

      System.out.println("Petrol vehicle");

   }

}

class FiveSeater_Petrol_Four_Wheeler extends Petrol_Four_Wheeler

{

   void seats()

   {

      System.out.println("Five seater");
```

```
  }
}
class Baleno_FiveSeater_Petrol_Four_Wheeler extends FiveSeater_Petrol_Four_Wheeler
{
  void model()
  {
    System.out.println("Baleno model");
  }
  public static void main(String[] args)
  {
    Baleno_FiveSeater_Petrol_Four_Wheeler b = new Baleno_FiveSeater_Petrol_Four_Wheeler();
    b.start();
    b.type();
    b.fuel();
    b.seats();
    b.model();
  }
}
```

Output:

```
Vehicle started
Four wheeler
Petrol vehicle
Five seater
Baleno model
```

Q. Demonstrate the use of the super keyword

```
class Vehicle
{
  int speed = 70;
}
class Car extends Vehicle
```

```
{
   int speed = 100;
   void showSpeed()
   {
      System.out.println("Car speed = " + speed);
      System.out.println("Vehicle speed = " + super.speed);
   }
   public static void main(String[] args)
   {
      Car c = new Car();
      c.showSpeed();
   }
}
```

Output:

```
Car speed = 100
Vehicle speed = 70
```

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

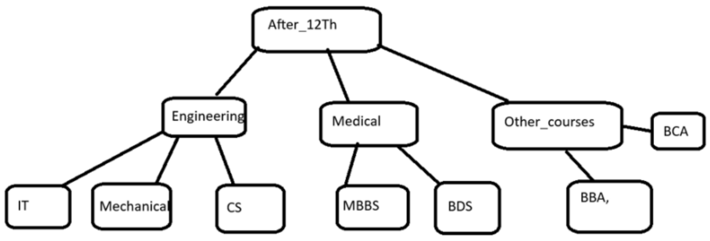
```
class Hospital
{
   String name = "City Hospital";
   void info()
   {
      System.out.println("Hospital name: " + name);
   }
}
class Patient extends Hospital
{
   void show()
   {
      System.out.println("Patient accessing hospital name: " + super.name);
```

```
        super.info();
    }
    public static void main(String[] args)
    {
        Patient p = new Patient();
        p.show();
    }
}
```

Output:

```
Patient accessing hospital name: City Hospital
Hospital name: City Hospital
```

Q. Create Hierarchical inheritance



```
class After12th
{
    void courses()
    {
        System.out.println("After 12th courses");
    }
}
class Engineering extends After12th
{
    void enggCourses()
    {
        System.out.println("Engineering: IT, Mechanical, CS");
    }
}
class Medical extends After12th
```

```
{
   void medicalCourses()
   {
      System.out.println("Medical: MBBS, BDS");
   }
}
class OtherCourses extends After12th
{
   void other()
   {
      System.out.println("Other: BCA, BBA");
   }
}
public class CoursesDemo
{
   public static void main(String[] args)
   {
      Engineering e = new Engineering();
      Medical m = new Medical();
      OtherCourses o = new OtherCourses();
      e.courses();
      e.enggCourses();
      m.courses();
      m.medicalCourses();
      o.courses();
      o.other();
   }
}
```

Output:

```
After 12th courses
Engineering: IT, Mechanical, CS
```

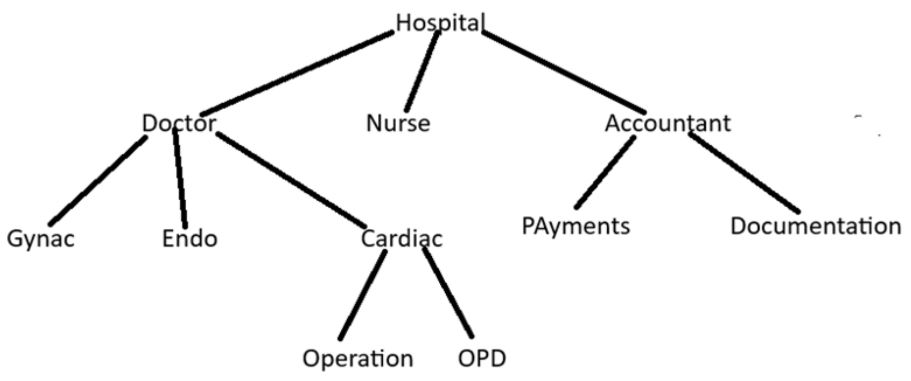
After 12th courses

Medical: MBBS, BDS

After 12th courses

Other: BCA, BBA

Q. Create practice on this Hospital



```
class Hospital
{
   void info()
   {
      System.out.println("Hospital info");
   }
}
class Doctor extends Hospital
{
   void treat()
   {
      System.out.println("Doctor treats patients");
   }
}
class Nurse extends Hospital
{
   void assist()
```

```
    {
        System.out.println("Nurse assists doctor");
    }
}
class Accountant extends Hospital
{
    void managePayments()
    {
        System.out.println("Accountant manages payments");
    }
}
class Gynac extends Doctor
{
    void specialize()
    {
        System.out.println("Gynac specializes in women's health");
    }
}
class Endo extends Doctor
{
    void specialize()
    {
        System.out.println("Endo specializes in endocrine system");
    }
}
class Cardiac extends Doctor
{
    void specialize()
    {
        System.out.println("Cardiac specializes in heart");
    }
```

```
}
class Payments
{
   void paymentProcess()
   {
      System.out.println("Processing payments");
   }
}
class Operation
{
   void operationRoom()
   {
      System.out.println("Operation room ready");
   }
}
class OPD
{
   void opdServices()
   {
      System.out.println("OPD services ongoing");
   }
}
class Documentation
{
   void documents()
   {
      System.out.println("Patient documents maintained");
   }
}
public class HospitalDemo
{
   public static void main(String[] args)
   {
      Doctor d = new Doctor();
      Nurse n = new Nurse();
      Accountant a = new Accountant();
      Gynac g = new Gynac();
      Endo en = new Endo();
      Cardiac c = new Cardiac();
      Payments p = new Payments();
      Operation o = new Operation();
      OPD opd = new OPD();
      Documentation doc = new Documentation();

      d.treat();
      n.assist();
      a.managePayments();
      g.specialize();
      en.specialize();
      c.specialize();
      p.paymentProcess();
      o.operationRoom();
      opd.opdServices();
      doc.documents();
   }
}
```

Output:

```
Doctor treats patients
Nurse assists doctor
Accountant manages payments
Gynac specializes in women's health
Endo specializes in endocrine system
```

```
Cardiac specializes in heart

Processing payments

Operation room ready

OPD services ongoing

Patient documents maintained
```

Q. 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)

```java
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 static void main(String[] args)

  {

    Calculator calc = new Calculator();

    System.out.println(calc.add(5, 10));

    System.out.println(calc.add(5, 10, 15));

    System.out.println(calc.add(5.5, 10.5));

  }

}
```

Output:

```
15

30

16.0
```

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

```
class Shape
{
   void area()
   {
      System.out.println("Area of shape");
   }
}
class Circle extends Shape
{
   double radius = 5;
   void area()
   {
      double area = 3.14 * radius * radius;
      System.out.println("Area of circle: " + area);
   }
}
class Rectangle extends Shape
{
   int length = 10;
   int breadth = 5;
   void area()
   {
      int area = length * breadth;
      System.out.println("Area of rectangle: " + area);
```

```
    }
}
public class ShapeDemo
{
    public static void main(String[] args)
    {
        Shape s = new Shape();
        Circle c = new Circle();
        Rectangle r = new Rectangle();
        s.area();
        c.area();
        r.area();
    }
}
```

Output:

```
Area of shape
Area of circle: 78.5
Area of rectangle: 50
```

Q. Create a Bank class with a method getInterestRate() and create subclasses SBI, ICICI, HDFC with return values 6.7%, 7.0%, 7.5% respectively

```
class Bank
{
    double getInterestRate()
    {
        return 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 BankDemo
{
    public static void main(String[] args)
    {
        SBI s = new SBI();
        ICICI i = new ICICI();
        HDFC h = new HDFC();
        System.out.println("SBI Interest Rate: " + s.getInterestRate() + "%");
        System.out.println("ICICI Interest Rate: " + i.getInterestRate() + "%");
        System.out.println("HDFC Interest Rate: " + h.getInterestRate() + "%");
    }
}
```

Output:

```
SBI Interest Rate: 6.7%
```

```
ICICI Interest Rate: 7.0%

HDFC Interest Rate: 7.5%
```

Q. Runtime Polymorphism with constructor Chaining. Create a class Vehicle with constructor that prints “Vehicle Created”. Create a subclass Bike that overrides 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();
      System.out.println("Bike Created");
   }
   void run()
   {
      System.out.println("Bike is running");
   }
   public static void main(String[] args)
   {
      Bike b = new Bike();
      b.run();
```

```
    }
}
```

Output:

```
Vehicle Created
Bike Created
Bike is running
```

Q. Create an abstract class SmartDevice with methods turnOn(), turnOff(), and performFunction(). Create child classes SmartPhone, SmartWatch, SmartSpeaker. Write code to store all objects in an array and use polymorphism to invoke performFunction()

```
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("Performing calling and browsing");
    }
}
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 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 commands");
   }
}
public class SmartDeviceDemo
{
    public static void main(String[] args)
    {
        SmartDevice[] devices = {new SmartPhone(), new SmartWatch(), new SmartSpeaker()};
        for(SmartDevice d : devices)
        {
            d.turnOn();
            d.performFunction();
            d.turnOff();
            System.out.println();
        }
    }
}
```

Output:

```
SmartPhone turned on
Performing calling and browsing
SmartPhone turned off

SmartWatch turned on
Tracking fitness and time
SmartWatch turned off

SmartSpeaker turned on
Playing music and responding to voice commands
SmartSpeaker turned off
```

Q. 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

```
interface Bank
{
    void deposit(double amt);
    void withdraw(double amt);
    double getBalance();
}
class Account
{
    double balance;
    Account()
    {
        balance = 0;
    }
}
class SavingsAccount extends Account implements Bank
{
    public void deposit(double amt)
    {
        balance += amt;
        System.out.println("Deposited " + amt);
    }
    public void withdraw(double amt)
    {
        if(balance - amt >= 1000)
        {
            balance -= amt;
            System.out.println("Withdrawn " + amt);
        }
        else
        {
            System.out.println("Insufficient balance, minimum 1000 must remain");
        }
    }
    public double getBalance()
    {
        return balance;
    }
}
class CurrentAccount extends Account implements Bank
{
    public void deposit(double amt)
    {
        balance += amt;
        System.out.println("Deposited " + amt);
    }
    public void withdraw(double amt)
    {
        if(balance >= amt)
        {
            balance -= amt;
            System.out.println("Withdrawn " + amt);
        }
        else
        {
            System.out.println("Insufficient balance");
        }
    }
    public double getBalance()
    {
        return balance;
    }
}
public class BankDemo2
{
    public static void main(String[] args)
    {
        SavingsAccount sa = new SavingsAccount();
        sa.deposit(5000);
        sa.withdraw(4500);
        System.out.println("Savings Account balance: " + sa.getBalance());

        CurrentAccount ca = new CurrentAccount();
        ca.deposit(3000);
        ca.withdraw(3500);
        System.out.println("Current Account balance: " + ca.getBalance());
    }
}
```

Output:

```
Deposited 5000.0
Insufficient balance, minimum 1000 must remain
Savings Account balance: 5000.0
Deposited 3000.0
Insufficient balance
Current Account balance: 3000.0
```

Q. Create a base class Vehicle with method start(). Derive Car, Bike, and Truck from it and override 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 started");
    }
}
class Car extends Vehicle
{
    void start()
    {
        System.out.println("Car started");
    }
}
class Bike extends Vehicle
{
    void start()
    {
        System.out.println("Bike started");
    }
}
class Truck extends Vehicle
{
    void start()
    {
        System.out.println("Truck started");
    }
}
public class VehicleDemo
{
    static void testStart(Vehicle v)
    {
        v.start();
    }
    public static void main(String[] args)
    {
        Vehicle v1 = new Car();
        Vehicle v2 = new Bike();
        Vehicle v3 = new Truck();
        testStart(v1);
        testStart(v2);
        testStart(v3);
    }
}
```

Output:

```
Car started
Bike started
Truck started
```

Q. Design an abstract class Person with fields name, age and abstract method getRoleInfo(). Create subclasses Student, Professor, TeachingAssistant. 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 rollNo;
    Student(String name, int age, String course, int rollNo)
    {
        super(name, age);
        this.course = course;
        this.rollNo = rollNo;
    }
    void getRoleInfo()
    {
        System.out.println("Student Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("Roll No: " + rollNo);
    }
}
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 rollNo)
    {
        super(name, age, course, rollNo);
    }
    void getRoleInfo()
    {
        System.out.println("Teaching Assistant Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
        System.out.println("Roll No: " + rollNo);
        System.out.println("Assisting professors and students");
    }
}
public class PersonDemo
{
    public static void main(String[] args)
    {
        Student s = new Student("Ravi", 20, "CS", 101);
        Professor p = new Professor("Dr. Smith", 45, "Physics", 75000);
        TeachingAssistant ta = new TeachingAssistant("Neha", 25, "CS", 102);
        s.getRoleInfo();
        System.out.println();
        p.getRoleInfo();
        System.out.println();
        ta.getRoleInfo();
    }
}
```

Output:

```
Student Name: Ravi
Age: 20
Course: CS
Roll No: 101

Professor Name: Dr. Smith
```

```
Age: 45

Subject: Physics

Salary: 75000.0


Teaching Assistant Name: Neha

Age: 25

Course: CS

Roll No: 102

Assisting professors and students
```

Q. Create Interface Drawable with method draw(). Create abstract class Shape with abstract method area(). Create subclasses Circle, Rectangle, Triangle. Calculate area using appropriate formulas. Demonstrate how interface and abstract class work together

```
interface Drawable
{
   void draw();
}
abstract class Shape implements Drawable
{
   abstract void area();
}
class Circle extends Shape
{
   double radius = 5;
   void area()
   {
      double a = 3.14 * radius * radius;
      System.out.println("Circle area: " + a);
   }
   public void draw()
   {
      System.out.println("Drawing Circle");
```

```
    }
}
class Rectangle extends Shape
{
    int length = 10, breadth = 5;
    void area()
    {
        int a = length * breadth;
        System.out.println("Rectangle area: " + a);
    }
    public void draw()
    {
        System.out.println("Drawing Rectangle");
    }
}
class Triangle extends Shape
{
    double base = 10, height = 5;
    void area()
    {
        double a = 0.5 * base * height;
        System.out.println("Triangle area: " + a);
    }
    public void draw()
    {
        System.out.println("Drawing Triangle");
    }
}
public class ShapeDrawableDemo
{
    public static void main(String[] args)
```

```
  {
     Shape[] shapes = {new Circle(), new Rectangle(), new Triangle()};
     for(Shape s : shapes)
     {
        s.draw();
        s.area();
        System.out.println();
     }
  }
}
```

Output:

```
Drawing Circle
Circle area: 78.5

Drawing Rectangle
Rectangle area: 50

Drawing Triangle
Triangle area: 25.0
```