1. create multilevel inheritance for

```
//Vehicle
//Four_wheeler
//Petrol_Four_Wheeler
//FiveSeater_Petrol_Four_Wheeler
//Baleno_FiveSeater_Petrol_Four_Wheeler


class Vehiclee {

 void display() {

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

 }

}

class Four extends Vehiclee {

 void fourw() {

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

 }

}

class Petrolfw extends Four {

 void petrol() {

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

 }

}

class Fiveseater extends Petrolfw {

 void seating() {

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

 }

}
```

```
class Balenofiveseater extends Fiveseater {
 void model() {
    System.out.println("This is a Baleno Car");
 }
}
public class Vehicle {
 public static void main(String[] args) {
    Balenofiveseater baleno = new Balenofiveseater();
    baleno.display();
    baleno.fourw();
    baleno.petrol();
    baleno.seating();
    baleno.model();
 }
}
```

Output:

This is a Vehicle

This is a Four Wheeler

This is a Petrol Four Wheeler

This is a Five Seater Petrol Four Wheeler

This is a Baleno Car

2. Demonstrate the use of the super keyword
   It's mainly used to:
   Call the parent class constructor
   Access parent class methods or variables when overridden.

```
class SoftwareTesting{
	void Syllabus() {
		System.out.println("Software testing course details");
		System.out.println("manual testing,core java,selenium web driver,
jira(project management),mysql");
	}
}

class Manual extends SoftwareTesting{
	void manual_testing() {
		super.Syllabus();
		System.out.println("if it is manual testing only \n then it incudes
manual testing concepts,jira");
	}
}

class Automation extends SoftwareTesting{
	void automation_testing() {
		super.Syllabus();
		System.out.println("if it is automation testing only \n then it includes
core java,mysql");
	}
}

public class Keyword_super {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("**manual testing details***");
		Manual mn = new Manual();
		mn.manual_testing();
		System.out.println("**automation testing**");
		Automation at = new Automation();
		at.automation_testing();

	}

}
```

Output:

```
**manual testing details***
Software testing course details
manual testing,core java,selenium web driver, jira(project management),mysql
if it is manual testing only
```

then it incudes manual testing concepts,jira
automation testing
Software testing course details
manual testing,core java,selenium web driver, jira(project management),mysql
if it is automation testing only
then it includes core java,mysql.

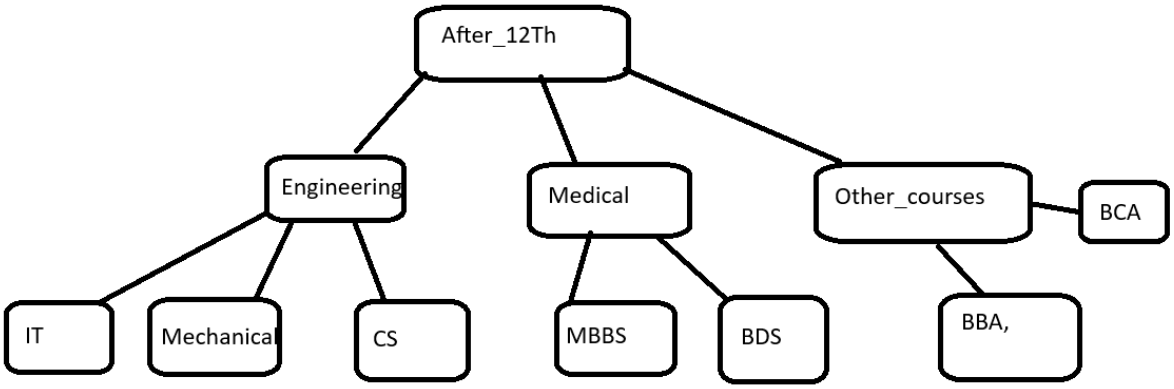
3. Create Hospital super class and access this class inside the patient child class and access properties from Hospital class.

```
class Hospital_info {
 String hospitalName = "Care Hospital";
 String location = "India";

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

class Patient extends Hospital_info {
 String patientname;
 int age;

 void displayDetails(String patientname,int age) {
            super.hospitalInfo();
    System.out.println("Patient Name: " + patientname);
    System.out.println("Age: " + age);
 }
}




public class Hospital {

            public static void main(String[] args) {

                        Patient p1 = new Patient();
      p1.displayDetails("john",70);

            }

}
```

Output:
Hospital Name: Care Hospital
Location: India
Patient Name: john

Age: 70

4. Create Hierarchical inheritance



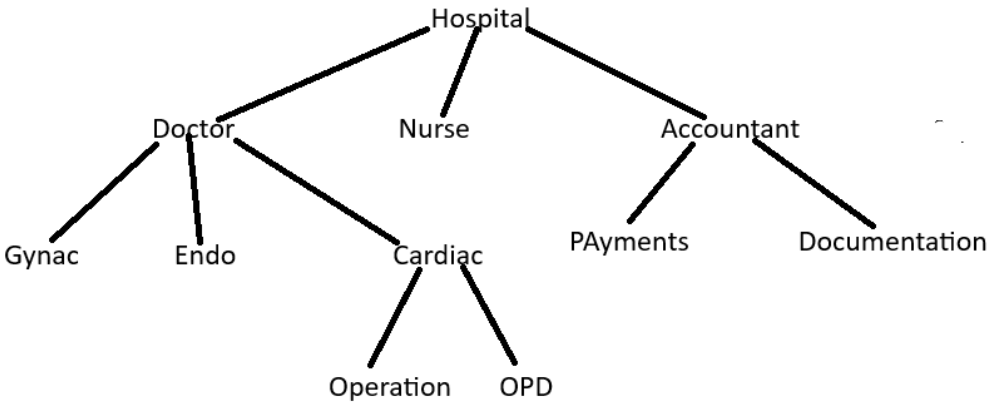

```
class twelth{
	void display() {
		System.out.println("courses after twelth class:
Engineering,medicine");
	}
}

class engineering extends twelth {
	void show() {
		System.out.println("branches in engineering : CS,IT,ECE,EEE");
	}

}

class medical extends twelth {
	void show1() {
		System.out.println("branches in medicine : MBBS , BDS");
	}

}

class infotech extends engineering {
	void display1() {
		System.out.println("subjects in IT : java , python");
	}

}
```

```
class mbbs extends medical {
	void display2() {
		System.out.println("branches in mbbs : medicine");
	}

}

public class Courses {

	public static void main(String[] args) {

		infotech it = new infotech();
		it.display();
		it.show();
		it.display1();
		System.out.println();
		mbbs mb = new mbbs();
		mb.display();
		mb.show1();
		mb.display2();


	}

}
```

Output:

```
courses after twelth class: Engineering,medicine
branches in engineering : CS,IT,ECE,EEE
subjects in IT : java , python

courses after twelth class: Engineering,medicine
branches in medicine : MBBS , BDS
branches in mbbs : medicine
```

5. Create practice on this




```
class hospital_details{
    String name;
    String location;
    void display(String name,String location) {
            System.out.println("hospital name = "+name);
            System.out.println("location name = "+location);
    }
}

class doctor extends hospital_details{
    String name="john";
    String dept="gynaec";
    void display1() {
            System.out.println("doctor name = "+name);
            System.out.println("dept name = "+dept);
    }
}

class nurse extends hospital_details{
```

```
    String name;
    String dept;
    void display2(String name,String dept) {
            System.out.println("nurse name = "+name);
            System.out.println("dept name = "+dept);
    }
}

class accountant extends hospital_details{
    String name="michael";
    String dept="accountant";
    void display3() {
            System.out.println("accountant name = "+name);
            System.out.println("dept name = "+dept);
    }
}

class gynaec extends doctor{
    void display4() {
            super.display1();
    }
}

class payments extends accountant{
    void display5() {
            super.display3();
    }
}
public class Hospitaldatails {
```

```
    public static void main(String[] args) {


            gynaec g = new gynaec();
            g.display("care hospital", "india");
            g.display4();
            payments p = new payments();
            p.display5();


    }


}
```

Output:

hospital name = care hospital

location name = india

doctor name = john

dept name = gynaec

accountant name = michael

dept name = accountant

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 Addition {

      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 Calculator {

        public static void main(String[] args) {


                Addition calc = new Addition();

        int result1 = calc.add(10, 20);
        int result2 = calc.add(5, 15, 25);
        double result3 = calc.add(12.5, 7.3);

        System.out.println(result1);
        System.out.println(result2);
        System.out.println(result3);

            }

}
```

Output:

30

45

19.8

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

```
class shape{
        void area() {
                System.out.println("calculate the area");
        }
}

class rectangle extends shape{
        int l;
        int b;
```

```
        void area(int l,int b) {
                System.out.println("area of rectangle : "+(l*b));
        }
}

class circle extends shape{
        int r;
        void area(int r) {
                double s = (3.14)*r*r;
                System.out.println("area of circle : "+s);
        }
}

public class Areas {

        public static void main(String[] args) {

                rectangle r = new rectangle();
                r.area(2,4);
                circle c = new circle();
                c.area(3);

        }

}
```

Output:
area of rectangle : 8
area of circle : 28.259999999999998

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 {
   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 BankInterest {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Bank sb = new sbi();
    Bank ic = new icici();
    Bank hdf = new hdfc();


    System.out.println(sb.getinterestrate() + "%");
    System.out.println(ic.getinterestrate() + "%");
    System.out.println(hdf.getinterestrate() + "%");

        }

}
```

Output:

6.7%

7.0%

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

   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().

```
abstract class Smart_Device {
   abstract void turnon();
   abstract void turnoff();
   abstract void performfunction();
}

class SmartPhone extends Smart_Device {
   void turnon() {
      System.out.println("SmartPhone is turning on...");
   }
   void turnoff() {
      System.out.println("SmartPhone is turning off...");
   }
   void performfunction() {
      System.out.println("SmartPhone is performing calling and browsing.");
   }
}
class SmartWatch extends Smart_Device {
   void turnon() {
      System.out.println("SmartWatch is turning on...");
   }
   void turnoff() {
      System.out.println("SmartWatch is turning off...");
   }
   void performfunction() {
      System.out.println("SmartWatch is tracking fitness and showing time.");
   }
}

class SmartSpeaker extends Smart_Device {
   void turnon() {
      System.out.println("SmartSpeaker is turning on...");
   }
```

```
    void turnoff() {
        System.out.println("SmartSpeaker is turning off...");
    }
    void performfunction() {
        System.out.println("SmartSpeaker is playing music and responding to voice
commands.");
    }
}
public class Smartdevice {

            public static void main(String[] args) {
                    // TODO Auto-generated method stub
                    SmartPhone phone = new SmartPhone();
        SmartWatch watch = new SmartWatch();
        SmartSpeaker speaker = new SmartSpeaker();

        System.out.println(" SmartPhone ");
        phone.turnon();
        phone.performfunction();
        phone.turnoff();

        System.out.println(" SmartWatch ");
        watch.turnon();
        watch.performfunction();
        watch.turnoff();

        System.out.println(" SmartSpeaker ");
        speaker.turnon();
        speaker.performfunction();
        speaker.turnoff();

            }

}
```

Output:

```
SmartPhone
SmartPhone is turning on...
SmartPhone is performing calling and browsing.
SmartPhone is turning off...
 SmartWatch
SmartWatch is turning on...
SmartWatch is tracking fitness and showing time.
SmartWatch is turning off...
```

SmartSpeaker
SmartSpeaker is turning on...
SmartSpeaker is playing music and responding to voice commands.
SmartSpeaker is turning 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).

```
interface Bank {
   void deposit(double amount);
   void withdraw(double amount);
   double getBalance();
}

class Account {
   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 implements Bank {
   private static final double MIN_BALANCE = 1000;

   public SavingsAccount(double initialBalance) {
      super(initialBalance);
```

```
    }


    public void withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid withdrawal amount.");
        } else if (balance - amount < MIN_BALANCE) {
            System.out.println("Withdrawal denied. Minimum balance of " +
MIN_BALANCE + " must be maintained.");
        } else {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        }
    }
}

class CurrentAccount extends Account implements Bank {
    public CurrentAccount(double initialBalance) {
        super(initialBalance);
    }


    public void withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid withdrawal amount.");
        } else if (amount > balance) {
            System.out.println("Insufficient balance.");
        } else {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        }
    }
}

public class BankDemo {
    public static void main(String[] args) {
        SavingsAccount savings = new SavingsAccount(2000);
        CurrentAccount current = new CurrentAccount(1500);

        System.out.println("--- Savings Account ---");
        savings.deposit(500);
        savings.withdraw(1800);  // Should deny due to min balance
        savings.withdraw(1000);  // Allowed
        System.out.println("Balance: " + savings.getBalance());
```

```
        System.out.println("\n--- Current Account ---");
        current.deposit(300);
        current.withdraw(1800);  // Allowed as no min balance restriction
        current.withdraw(100);   // Allowed
        System.out.println("Balance: " + current.getBalance());
    }
}
```

Output:

```
--- Savings Account ---
Deposited: 500.0
Withdrawal denied. Minimum balance of 1000.0 must be maintained.
Withdrawn: 1000.0
Balance: 1500.0

--- Current Account ---
Deposited: 300.0
Withdrawn: 1800.0
Withdrawn: 100.0
Balance: -100.0
```

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.

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

class Car extends Vehicle {
    public void start() {
        System.out.println("Car is starting with a key ignition");
    }
}

class Bike extends Vehicle {
```

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

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

public class VehicleDemo {

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

    public static void main(String[] args) {
        Vehicle myCar = new Car();
        Vehicle myBike = new Bike();
        Vehicle myTruck = new Truck();

        System.out.println("Testing polymorphism:");

        startVehicle(myCar);
        startVehicle(myBike);
        startVehicle(myTruck);
    }
}
```

Output:

```
Testing polymorphism:
Car is starting with a key ignition
Bike is starting with a kick start
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().

```
abstract class Person {
    protected String name;
    protected int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public abstract String getRoleInfo();


    public String getBasicInfo() {
        return "Name: " + name + ", Age: " + age;
    }
}

class Student extends Person {
    private String course;
    private int rollNumber;

    public Student(String name, int age, String course, int rollNumber) {
        super(name, age);
        this.course = course;
        this.rollNumber = rollNumber;
    }


    public String getRoleInfo() {
        return getBasicInfo() + ", Role: Student, Course: " + course + ", Roll Number: " +
rollNumber;
    }
}

class Professor extends Person {
    private String subject;
    private double salary;

    public Professor(String name, int age, String subject, double salary) {
        super(name, age);
        this.subject = subject;
        this.salary = salary;
    }
```

```
    public String getRoleInfo() {
        return getBasicInfo() + ", Role: Professor, Subject: " + subject + ", Salary: $" +
salary;
    }
}

class TeachingAssistant extends Student {
    private double stipend;

    public TeachingAssistant(String name, int age, String course, int rollNumber, double
stipend) {
        super(name, age, course, rollNumber);
        this.stipend = stipend;
    }

    public String getRoleInfo() {
        return super.getRoleInfo() + ", Role: Teaching Assistant, Stipend: $" + stipend;
    }
}

public class PersonDemo {
    public static void main(String[] args) {
        Student student = new Student("Ajay", 20, "Computer Science", 101);
        Professor professor = new Professor("vijay", 45, "Mathematics", 75000);
        TeachingAssistant ta = new TeachingAssistant("vinay", 24, "Physics", 202, 1500);

        System.out.println(student.getRoleInfo());
        System.out.println(professor.getRoleInfo());
        System.out.println(ta.getRoleInfo());
    }
}
```

Output:

Name: Ajay, Age: 20, Role: Student, Course: Computer Science, Roll Number: 101
Name: vijay, Age: 45, Role: Professor, Subject: Mathematics, Salary: $75000.0
Name: vinay Age: 24, Role: Student, Course: Physics, Roll Number: 202, Role: Teaching Assistant, Stipend: $1500.0

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 {
   public abstract double area();
}

class Circle extends Shape {
   private double radius;

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

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


   public void draw() {
      System.out.println("Drawing a Circle with radius " + radius);
   }
}

class Rectangle extends Shape {
   private double length, width;

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

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

   public void draw() {
      System.out.println("Drawing a Rectangle with length " + length + " and width " +
width);
   }
}
```

```
class Triangle extends Shape {
   private double base, height;

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

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


   public void draw() {
      System.out.println("Drawing a Triangle with base " + base + " and height " +
height);
   }
}

public class ShapeDemo {
   public static void main(String[] args) {
      Shape[] shapes = {
         new Circle(5),
         new Rectangle(4, 6),
         new Triangle(3, 7)
      };

      for (Shape shape : shapes) {
         shape.draw();
         System.out.println("Area: " + shape.area());
         System.out.println();
      }
   }
}
```

Output:

```
Drawing a Circle with radius 5.0
Area: 78.53981633974483

Drawing a Rectangle with length 4.0 and width 6.0
Area: 24.0

Drawing a Triangle with base 3.0 and height 7.0
Area: 10.5
```