1) What is inheritance and explain types of inheritance with examples

Inheritance:

It is the mechanism in java by which one class is allowed to acquire features of other class

Types of Inheritance:

1)Single Inheritance

2)Multiple Inheritance

3)Multilevel Inheritance

4)Hierarchical Inheritance

5)Hybrid Inheritance

1)Single Inheritance: In Single Inheritance subclasses inherits the features of one super class

Example:

```
class Calculation{
   Calculation(){
          System.out.println("Super class constructor is called :");
   }
   Calculation(String str){
          System.out.println(str+" constructor is called:");
   }
   public int addition(int a, int b){
          return a+b;
   }
   public void display(){
          System.out.println("Invoked super class display() method :");
   }

}
public class SingleInheritance extends Calculation {
   SingleInheritance(){
          super("parameterized ");
          System.out.println("SubClass constructor is called :");
```

```
    }
    public int multiplication(int a,int b){
            System.out.println("Invoking sub-class method display() method:");
            display();
            System.out.println("Invoking superclass display() method  :");
            super.display();
            System.out.print("Multiplication of 2 and 3 is :");
            return a*b;

    }
    public void display(){
            System.out.println("Invoked sub-class display() method:");
    }


    public static void main(String args []){
            SingleInheritance inheritance=new SingleInheritance();
            System.out.println(inheritance.multiplication(2,3));
            System.out.println("Addition of 2 and 3 is :"+inheritance.addition(2, 3));
    }

}
```

Output:

```
parameterized  constructor is called:
SubClass constructor is called :
Invoking sub-class method display() method:
Invoked sub-class display() method:
Invoking superclass display() method  :
Invoked super class display() method :
Multiplication of 2 and 3 is :6
Addition of 2 and 3 is :5
```

In the above program when the object of SingleInheritance is created a copy of the content of the super class is created with in the object, this is why using the object of the subclass we can access the members of superclass

The superclass reference variable can hold the subclass objects, but can only access the members which are present in the superclass ,it cannot access the members which are not present in the super class, so it is recommended to create object for subclass

Note: A subclass inherits all the members from its superclass ,but not constructor , because constructor is not a member, but invokes the constructor from subclass

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class, but if we want to call a parameterized constructor of the super class we use super keyword as below example

Eg:Super(parameter);

2) Multiple Inheritance : In multiple Inheritance one class can have more than one super class and inherit features from all the parent classes,java doesn't support multiple inheritance, we achieve multiple inheritance through interface

Example:

```
interface Base1{
   public byte add(byte a, byte b);
}
interface Base2{
   public byte subtract(byte a, byte b);
}
public class MultipleInheritance implements Base1,Base2{
   byte sum=0;
   public byte add(byte a, byte b){
          this.sum=(byte) (a+b);
          return this.sum;
   }
   public byte subtract(byte a, byte b){
          this.sum=(byte) (a-b);
          return this.sum;
   }
   public static void main(String args[]){
          MultipleInheritance multi=new MultipleInheritance();
          byte add=multi.add((byte)2,(byte) 3);
```

```
            byte subtract=multi.subtract((byte)3,(byte) 2);
            System.out.println("Addition of two numbers 2 and 3 is :"+add);
            System.out.println("Subtraction of two numbers 3 and 2 is :"+subtract);

   }

}
```

Output:

```
Addition of two numbers 2 and 3 is :5
Subtraction of two numbers 3 and 2 is :1
```

3) Multilevel Inheritance : In Multilevel inheritance a derived class will be inheriting a base class and that derived class will be acting as a base class for another class

Example :

```
class GrandParent{
   public void display(){
         System.out.println("You accessed grand parent class method :");
   }
}
class Parent extends GrandParent{
   public void display(){
         super.display();
         System.out.println("You accessed parent class method :");
   }
}

public class MultiLevelInheritence extends Parent {
   public void display(){
         super.display();
         System.out.println("You are accessing the child class :");
   }
   public static void main(String args[]){
         MultiLevelInheritence m1=new MultiLevelInheritence();
         m1.display();
```

```
    }

}
```

Output :

```
You accessed grand parent class method :
You accessed parent class method :
You are accessing the child class :
```

4) Hierarchical Inheritance : In hierarchical inheritance one class serves as super class for more than one subClass

Example:

```
class Base{
    public void methodBase(){
            System.out.println("You are in Base class method :");
    }
}
class DerivedA extends Base{
    public void methodDerivedA(){
            System.out.println("You are in DerivedA class method :");
    }

}
class DerivedB extends Base{
    public void methodDerivedB(){
            System.out.println("You are in DerivedB class method :");
    }

}
class DerivedC extends Base{
    public void methodDerivedC(){
            System.out.println("You are in DerivedC class method :");
    }

}
public class HierarchicalInheritance {
    public static void main(String args []){
            DerivedA derivedA=new DerivedA();
            derivedA.methodBase();
            DerivedB derivedB=new DerivedB();
            derivedB.methodBase();
            DerivedC derivedC=new DerivedC();
            derivedC.methodBase();
    }

}
```

Output:

```
You are in Base class method :
You are in Base class method :
You are in Base class method :
```

5) Hybrid Inheritance : In java Hybrid inheritance is a mix of two or more than two types of inheritance.since java does not support multiple inheritance hybrid inheritance is also not possible with java, It can be achieved by interface

Example:

```
/*          Base
            ↑
            |
  ---------------
  ↑                 ↑
  |                 |
  Derived1        Derived2
  ↑
  |
  Child
 *
 *
 *
 * */
class Base{
    public void display(){
            System.out.println("You are in base class display() method :");
    }
}
```

```
class Derived1 extends Base{
   public void display(){
          super.display();
          System.out.println("You are in Derived1 class display() method :");
   }
}
class Derived2 extends Base{
   public void display(){
          System.out.println("You are in Derived2 class display() method :");
   }
}
class Child extends Derived1{
   public void display(){
          super.display();
          System.out.println("You are in Child class display() method :");
   }
}
public class HybridInheritence {
   public static void main(String args []){
          Child child=new Child();
          child.display();
   }

}
```

Output:

```
You are in base class display() method :
You are in Derived1 class display() method :
You are in Child class display() method :
```

2) Let’s consider the example of vehicles like car, bike and bus they have common functionalities. So we make an interface and put all these common functionalities. And lets Bike, car and bus implement all these functionalities in their own class in their own way.

Example :

```
/*
 * Below interface vehicle has some functionalities classes that implement vehicle
 * interface must use all the methods of vehicle interface
 */
```

```
interface Vehicle{
    public void setGear(byte gear);
    public void setBreaks(boolean isOn);
    public void setSpeed(short speed);
}
/*
 * Bike class implements vehicle interface and Bike class uses all the methods of vehicle
interface
 */
class Bike implements Vehicle{
    private byte gear;
    private boolean isOn;
    private short speed;
    Bike(){
            this.speed=10;
    }

    public void setGear(byte gear){
            this.gear=gear;
    }
    public void  setBreaks(boolean isOn){
            this.isOn=isOn;
    }
    public void setSpeed(short speed){
            this.speed=(short) ((short)this.speed+(short)speed);
    }
    public int getGear(){
            return this.gear;
    }
    public String getBreaks(){
            if(isOn)
                    return "are ON";
            else
                     return "are OF";
    }
    public int getSpeed(){
            return speed;
    }

}
/*
 * Car class implements vehicle interface and Car class uses all the methods of vehicle interface
 */
class Car implements Vehicle{
    private byte gear;
    private boolean isOn;
    private short speed;

    public void setGear(byte gear){
            this.gear=gear;
    }
    public void  setBreaks(boolean isOn){
            this.isOn=isOn;
    }
    public void setSpeed(short speed){
            this.speed=(short) (this.speed+speed);
    }
    public int getGear(){
            return this.gear;
    }
    public String getBreaks(){
            if(isOn)
                    return "are ON";
            else
                     return "are OF";
    }
    public int getSpeed(){
            return speed;
    }

}
/*
 * Bus class implements vehicle interface and bus class uses all the methods of vehicle
interface
 */

class Bus implements Vehicle{
    private byte gear;
    private boolean isOn;
    private short speed;

    public void setGear(byte gear){
            this.gear=gear;
    }
    public void  setBreaks(boolean isOn){
            this.isOn=isOn;
    }
    public void setSpeed(int speed){
            this.speed=(short) (this.speed+speed);
    }
    public int getGear(){
            return this.gear;
    }
    public String getBreaks(){
            if(isOn)
                    return "are ON";
            else
                     return "are OF";
    }
    public int getSpeed(){
            return speed;
    }
}
/*
 * Here in Main class I created the objects of all the classes and called the
 * appropriate functions of the implemented interface methods which are present
 * in the class
 */
public class Main {
    public static void main(String args[]){
            Bike bike=new Bike();
            bike.setGear((byte)2);
            bike.setBreaks(true);
            bike.setSpeed((short)50);
            System.out.println("Bike is in gear no :"+bike.getGear()+" and breaks
"+bike.getBreaks()+" and speed of the bike is :"+bike.getSpeed()+" kmph");
            Car car=new Car();
            car.setGear((byte)4);
            car.setBreaks(false);
            car.setSpeed((short)80);
            System.out.println("car is in gear no :"+car.getGear()+" and breaks "+car.getBreaks()+"
and speed of the car is :"+car.getSpeed()+" kmph");
            Bus bus=new Bus();
            bus.setGear((byte)3);
            bus.setBreaks(true);
            bus.setSpeed((short)30);
            System.out.println("Bus is in gear no :"+bus.getGear()+" and breaks
"+bus.getBreaks()+" and speed of the bus is :"+bus.getSpeed()+" kmph");
```

```
    }

}
```

Output :

```
Bike is in gear no :2 and breaks are ON and speed of the bike is :60 kmph
car is in gear no :4 and breaks are OF and speed of the car is :80 kmph
Bus is in gear no :3 and breaks are ON and speed of the bus is :30 kmph
```

3) Consider a base class is "shape" and each shape has a color, size and ares. From this, specific types of shapes are derived(inherited)-circle, square, triangle, each of which may have additional characteristics and behaviours. For example, certain shapes can be flipped. Some behaviours may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.

Example :

```
class Shape{
    String color;
    float area;
    Shape(){
            color="red";
    }
    public float size(float r){
            return 0;
    }
    public float area(float r){
            return 0;
    }

}
```

```
class Circle extends Shape{

    public float area(float r){
            float area=3.14f*r*r;
            return area;
    }
    public float size(float r){
             float size=2*3.14f*r;
             return size;


    }
}

class Square extends Shape{

    public float size(float r){
             return 4*r;
    }
    public float area(float r){
             return r*r;
    }
}

class Triangle extends Shape{

    public float area(float base,float height){
             return (base*height)/2;
    }
    public float size(float base,float height,float width){
             return (base+height+width)/2;
    }

}

public class Operations {
    public static void main(String args []){
             Circle circle=new Circle();
             System.out.println("area of circle of radius 5 is :"+circle.area(5)+" and size of circle of
radius 5 is :"+circle.size(5));
             Square square=new Square();
             System.out.println("area of square of radius 5 is :"+square.area(5)+" and size of square
of radius 5 is :"+square.size(5));
```

```
            Triangle triangle=new Triangle();
            System.out.println("area of triangle of base 5.0 and height 5.0 is :"+triangle.area(10.0f,
10.0f)+" and size of triangle of base,width,height  5 is :"+triangle.size(5.0f,5.0f,5.0f));


    }

}
```

Output :

```
area of circle of radius 5 is :78.5 and size of circle of radius 5 is :31.400002
area of square of radius 5 is :25.0 and size of square of radius 5 is :20.0
area of triangle of base 5.0 and height 5.0 is :50.0 and size of triangle of base,width,height  5 is
:7.5
```

4) What is encapsulation, mention its advantages? explain with an example

Encapsulation: It means wrapping up of data into single unit, In other words, here all the variables are declared private and can be accessed only by public methods, by using getters and setters we manipulate the data

Advantages :
1) It is a way to achieve data hiding because other class members cannot access private members
2) By providing only a setters or getters method we can make the class read only or write only by giving access to setter or getter method
3)Encapsulation improves re-usability and easy to change with new requirements
4)Encapsulated code is easy to test for unit testing

Example :

```
public class Encapsulation {
    private String name;
    private byte age;
    private String rollNo;
    /*
     * getXXX() methods are used for accessing the values of the private variables
     */
    public byte getAge(){
            return age;
    }
    public String getName(){
            return name;
    }
    public String getRollNo(){
            return rollNo;
    }
    /*
     * setXXX() methods are used for setting the values to the private variables
     */
    public void setName(String name){
            this.name=name;
    }
    public void setAge(byte age){
            this.age=age;
    }
    public void setRollNo(String rollNo){
            this.rollNo=rollNo;
    }

    public static void main(String args []){
            Encapsulation bindData=new Encapsulation();
            bindData.setName("Vamsi");
            bindData.setAge((byte)23);
            bindData.setRollNo("11311896");
            System.out.println("Name is :"+bindData.getName()+" roll no is : "+bindData.rollNo+"
Age is : "+bindData.age);
    }

}
```

Output :

```
Name is :Vamsi roll no is : 11311896 Age is : 23
```

5) What is method overloading and method overriding? explain with an example

Method Overloading:

If a class has multiple methods with same name but different parameters then it is called method overloading,overloading is related to compile time polymorphism

Error case :

```
int mymethod(int a, int b)
float mymethod(int var1, int var2)
```

Result: Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn't matter while overloading a method.

Example:

```
public class MethodOverloading {
    public int add(int a, int b){
            return a+b;
    }
          public int add(int a, int b,int c){
            return a+b+c;
    }
          public int add(int a, int b,int c,int d){
            return a+b+c+d;
    }
    public static void main(String args[]){
            MethodOverloading overloading= new MethodOverloading();
            System.out.println("Addition of two numbers 2,3 is :"+overloading.add(2, 3));
            System.out.println("Addition of three numbers 2,3,4 is :"+overloading.add(2, 3,4));
            System.out.println("Addition of four numbers 2,3,4,5 is :"+overloading.add(2, 3,4,5));
    }

}
```

output:

```
Addition of two numbers 2,3 is :5
Addition of three numbers 2,3,4 is :9.0
Addition of four numbers 2,3,4,5 is :14
```

Method Overloading and type promotion :
When data type of smaller size is promoted to data type of bigger size then it is called type promotion for example byte type is converted to short type etc..

The data type of the left side is converted to any of the data type of the right side

byte → short → int → long
short → int → long
int → long → float → double
float → double
long → float → double

Example 1 :

```
class Sample{
  void display(int a, double b){
        System.out.println("Method A");
  }
  void disp(int a, double b, double c){
        System.out.println("Method B");
  }
  public static void main(String args[]){
        Sample samp = new Sample();
        /*
         In the below display method I have given an float type as parameter it gets promoted to
double type and display(int a, double b) is printed
        */
        samp.display(100, 20.67f);
  }
}
```

Output :

```
Method A
```

Example 2 :

```
class Sample{
  void display(int a, double b){
        System.out.println("Method A");
  }
  void disp(int a, double b, double c){
        System.out.println("Method B");
  }
void display(int a, float b){
        System.out.println("Method C");
  }
  public static void main(String args[]){
        Sample samp = new Sample();
        /*
        In the previous program float is converted into double but here we have float type
parameter in the method void display(int a, float b) so prints Method C
        */
        samp.display(100, 20.67f);
  }
}
```

Output :

```
Method C
```

Method Overriding :

Declaring a method in child class which is already present in parent class is known as method overriding

Example :

```
class Human{
   public void eat(){
         System.out.println("Human can eat :");
   }
}
public class Boy extends Human{
   public void eat(){
         System.out.println("Boy is human so he can eat :");
   }
   public static void main(String args []){
         Boy boy=new Boy();
         /*
          * below boy.eat() method overrides the parent class human eat() method
          * and prints Boy class eat() method
          */
         boy.eat();
   }
}
```

Output :

```
Boy is human so he can eat :
```

6.)Consider a class Rectangle, it should contains 5 printArea methods accepting different parameter

Example :

```
public class Rectangle {

    public void printArea(int length, int breadth){
            System.out.println(2*(length+breadth));
    }
    public void printArea(float length, int breadth){
            System.out.println(2*(length+breadth));
    }
    public void printArea(int length, float breadth){
            System.out.println(2*(length+breadth));
    }
    public void printArea(byte length, byte breadth){
            System.out.println(2*(length+breadth));
    }
    public void printArea(int length, byte breadth){
            System.out.println(2*(length+breadth));
    }
    public static void main(String args []){

            Rectangle rectangle=new Rectangle();
            /*
             * below methods with different parameters will call 5 printArea() methods
             * and prints the output
             */
            rectangle.printArea(10,15);
            rectangle.printArea(10.0f,15 );
            rectangle.printArea(11,15.0f );
            rectangle.printArea((byte)11,(byte)15);
            rectangle.printArea(10,(byte)15);
    }

}
```

Output :

```
50
50.0
52.0
52
50
```

7) Consider a class Plants, it should contain the methods releases oxygen and accepts corbondioxide and this class should be extends by different plants(for example SunflowerPlant extends Plants) and those classes should contain same methods (extends by atleast 3 classes).

Example :

```
class Plants{
    public void releasesOxygen(){
            System.out.println("Plants releases oxygen :");
    }
    public void acceptsCarbondioxide(){
            System.out.println("Plants accepts oxygen :");
    }
}
/*
 * Here below three classes SunFlowerPlant, SeedPlant,Moss  extends the three methods
 * of Plant class and overrides the methods and prints the overridden methods output
 */
class SunFlowerPlant extends Plants{
    public void releasesOxygen(){
            System.out.println("SunFlower plant releases oxygen :");
    }
    public void acceptsCarbondioxide(){
            System.out.println("SunFlower plant accepts oxygen :");
    }
}
```

```
}


class SeedPlant extends Plants{
   public void releasesOxygen(){
         System.out.println("Seed plant releases oxygen :");
   }
   public void acceptsCarbondioxide(){
         System.out.println("Seed plant accepts oxygen :");
   }
}

class Moss extends Plants{
   public void releasesOxygen(){
         System.out.println("Moss plants releases oxygen :");
   }
   public void acceptsCarbondioxide(){
         System.out.println("Moss plants accepts oxygen :");
   }
}
class ButterCup extends Plants{
   public void releasesOxygen(){
         System.out.println("ButterCup pants releases oxygen :");
   }
   public void acceptsCarbondioxide(){
         System.out.println("ButterCup plants accepts oxygen :");
   }
}
public class Food {
   public static void main(String args []){
         SunFlowerPlant sunFlowerPlant =new SunFlowerPlant();
         sunFlowerPlant.releasesOxygen();
         sunFlowerPlant.acceptsCarbondioxide();
         SeedPlant seedPlant=new SeedPlant();
         seedPlant.releasesOxygen();
         seedPlant.acceptsCarbondioxide();
         Moss mossPlant=new Moss();
         mossPlant.releasesOxygen();
         mossPlant.acceptsCarbondioxide();
         ButterCup butterCupPlant = new ButterCup();
         butterCupPlant.releasesOxygen();
         butterCupPlant.acceptsCarbondioxide();

   }

}
```

Output :

```
Seed plant releases oxygen :
Seed plant accepts oxygen :
Moss plants releases oxygen :
Moss plants accepts oxygen :
ButterCup pants releases oxygen :
ButterCup plants accepts oxygen :
```