I - ASSIGNMENT

(Start Writing From Here)

1. Demonstrate Object Oriented Programming concepts.

Object Oriented Programming (OOP's) refers to languages that use objects in programming. It implements real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Oop's Concept :-

(i) Class
(ii) Objects
(iii) Data Abstraction
(iv) Encapsulation
(v) Inheritance
(vi) Polymorphism
(vii) Dynamic Binding
(viii) Message

(i) Class :- It is a user-defined data type. It consists of data members and member function, which can be accessed and used by creating an instance of that class. A class is like a blueprint of an object

(ii) Object :- An object is an instance of a class. When a class is defined, no memory is allocated but when it is instantiated memory is allocated.

An Object has an identity, State and behaviours. For example: "Dog" is a real life object, which has some characteristics like color, Breed, Bark, Sleep and Eats.

(iii) Data Abstraction:- It refers to providing only essential information about the data to the outside world, hiding the background details or implementation. It displays only required information and hiding the details. Consider a real-life example of a man driving a car.

(iv) Encapsulation:- It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. The variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. Here, the data is hidden from other classes, so it is also known as data-hiding.

(v) Inheritance:- It is an important pillar of OOP's. The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes.

So, when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possess it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

-(vi) Polymorphism :-

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, the same person posses different behavior in different situations. This is called polymorphism

-(vii) Dynamic Binding :- The code to be executed in response to the function call is decided at runtime. Dynamic Binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is one of the main advantage of inheritance is that some derived class D has all the members of its base class B

-(viii) Message Passing :- It is a form of communication used in OOP as well as parallel

programming. Objects communicate with one another by sending and receiving information to each other. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Illustrate the use of this keyword in Java.

The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name. If you omit the keyword in below example, the output would be "0" instead of "5"

Eg:-
```
public class Main {
  int x;
  Public Main (int x) {
  this.x = x;
  }
public static void main (String [] args) {
  Main myobj = new Main(5);
  System.out.println ("Value of x = " + myobj.x);
}
}
```

this can also can be used to :
- Invoke current class constructor
- Invoke current class method.
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

3. Distinguish abstract class and interface with suitable examples.

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods.

Abstract class	Interface
• It can have abstract and non-abstract methods	• It can have only abstract methods.
• Abstract class doesn't support multiple inheritance	• Interface supports multiple inheritance.
• This class can have final, non-final, static and non-static variables.	• Interface has only static and final variables.
• It provides the implementation of interface.	• It can't provide the implementation of abstract class.
• The abstract keyword is used to declare abstract class.	• The interface keyword is used to declare interface.

• An abstract class can extend another java class and implement multiple java interfaces	• An interface can extend another java interface only
• A abstract class can be extended using keyword "extends".	• An interface can ~~extend~~ be implemented using keyword "implements".
• A java abstract class can have class member like private, protected, etc.	• Members of java interface are public by default.

• Example :-
public abstract class Shape{
public abstract void draw();
}
	• Example :-
	public interface Drawable{
	void draw();
	}

Example of abstract class and interface in JAVA:-

```
interface A {
void a();
void b();
void c();
void d();
}
abstract class B implements A {
public void c() {System.out.println("I am c");}
```

```
class M extends B {
public void a() {System.out.println("I am a");}
public void b() {System.out.println("I am b");}
public void d() {System.out.println("I am d");}
class Test5 {
public static void main(String args[]) {
A a=new M();
a.a();
b.b();
a.c();
a.d();
}
}.
```

Expected Output:-

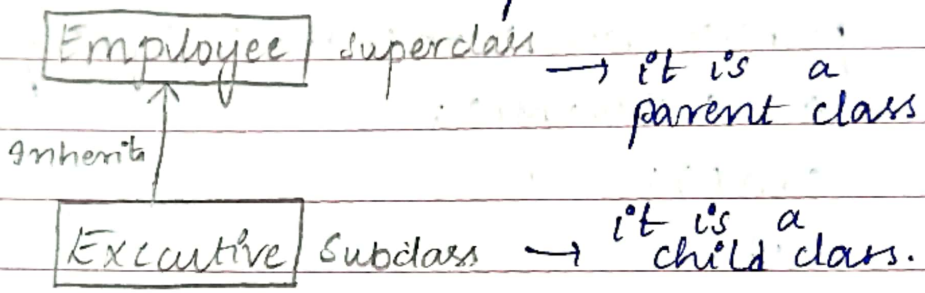
I am a
I am b
I am c
I am d

4. What are the types of Inheritance? Explain with suitable examples.

- (1) Single inheritance
- (2) Multi-level inheritance
- (3) Hierarchical inheritance.
- (4) Hybrid Inheritance.

i) Single inheritance:-

In this, a sub-class is derived from only one super class. It inherits the properties and behaviour of a single-parent class. Sometimes it is also known as simple inheritance.



Eg:-

```
class Employee
{
float salary = 34534 * 12;
}
public class Executive extends Employee
{
float bonus = 3000 * 6;
public static void main (String args[])
{
Executive obj = new Executive();
System.out.println("Total salary credited: "+obj.salary);
System.out.println("Bonus of six months: "+obj.bonus);
}
}
```

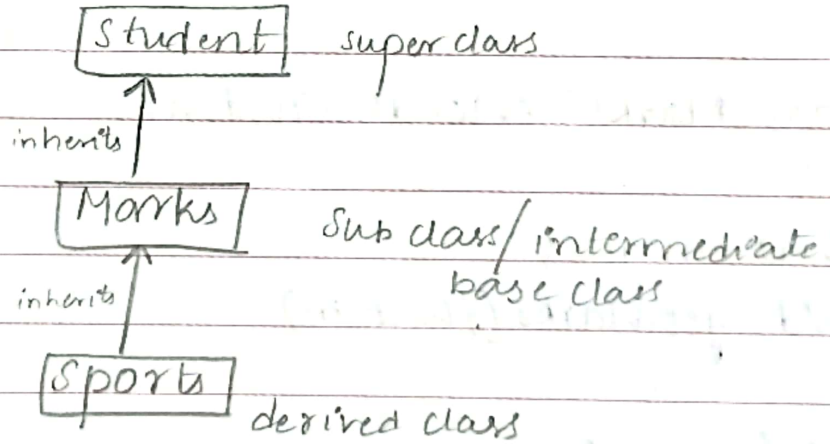
Expected Output :-

Total salary credited : 414408.0

Bonus of six months : 18000.0

(ii) Multi-Level Inheritance :-

A class is derived from a class which is also derived from another class is called multi-level inheritance. A class that has more than one parent class. The classes must be at different levels. there exists a single base class and single derived class but multiple intermediate base classes.



The class marks inherits the members of the class students. The class sports inherits the members of the class Marks. Hence, Student is the parent class of the class marks and the class Marks is the parent of the class Sports. So, the class sports inherits the properties

of the student along with the class Mark.

```
class student
{
int reg-no;
void getNO(int no)
{
reg_no=no;
}
void putNO()
{
System.out.println("registration number="+reg-no);
}
}
class Marks extends student
{
float marks;
void getMarks(float m)
{
marks=m;
}
void putMarks()
{
System.out.println("marks="+marks);
}
}
```

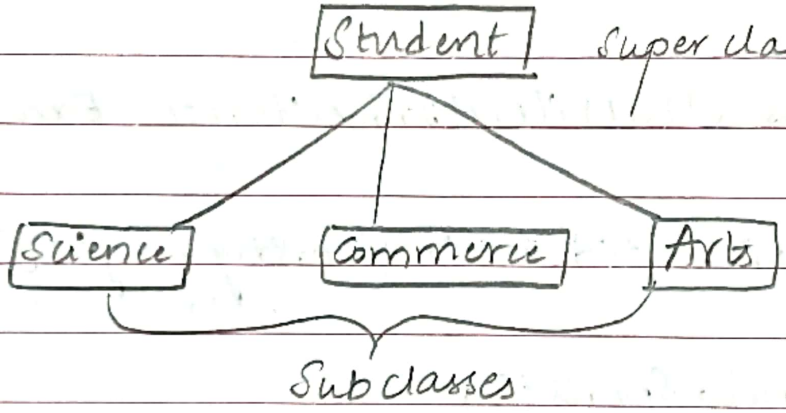
```
class Sports extends Marks
{
float score;
void getScore(float scr)
{
 score = scr;
}
System.out.println("score = "+score);
}
}
public class MultiLevelInheritance Example
 {
public static void main(string args[])
{
Sports ob = new Sports();
ob.getNO(0987);
ob.putNO();
ob.getMarks(78);
ob.putMarks();
ob.getScore(68.7);
ob.putScore();
 }
}
```

Expected Output:-

Registration number = 0987
marks = 78.0
score = 68.7

iii) Hierarchichal Inheritance:-

If a number of classes are derived from a single base class it is called Hierarchical inheritance



The classes science, commerce and arts inherit a single parent class named student.

Eg:-

```
class student
{
public void methodStudent()
{
System.out.println("The method of the class
                    student invoked.");
}
}
```

```
class Science extends Student
{
public void methodScience()
{
System.out.println("The method of the class
                    Science invoked.");
}
}
class Commerce extends Student
{
public void methodCommerce()
{
System.out.println("The method of the class
                    commerce invoked.");
}
}
class Arts extends Student
{
public void methodArts()
{
System.out.println("The method of the class
                    Arts invoked");
}
}
public class HierarchicalInheritanceExample
```

```
{
public static void main (String args[])
{
Science Sci = new Science ();
Commerce comm = new Commerce();
Arts art = new Arts ();
sci.methodStudent();
comm.methodStudent();
art.methodStudent();
}
}
```

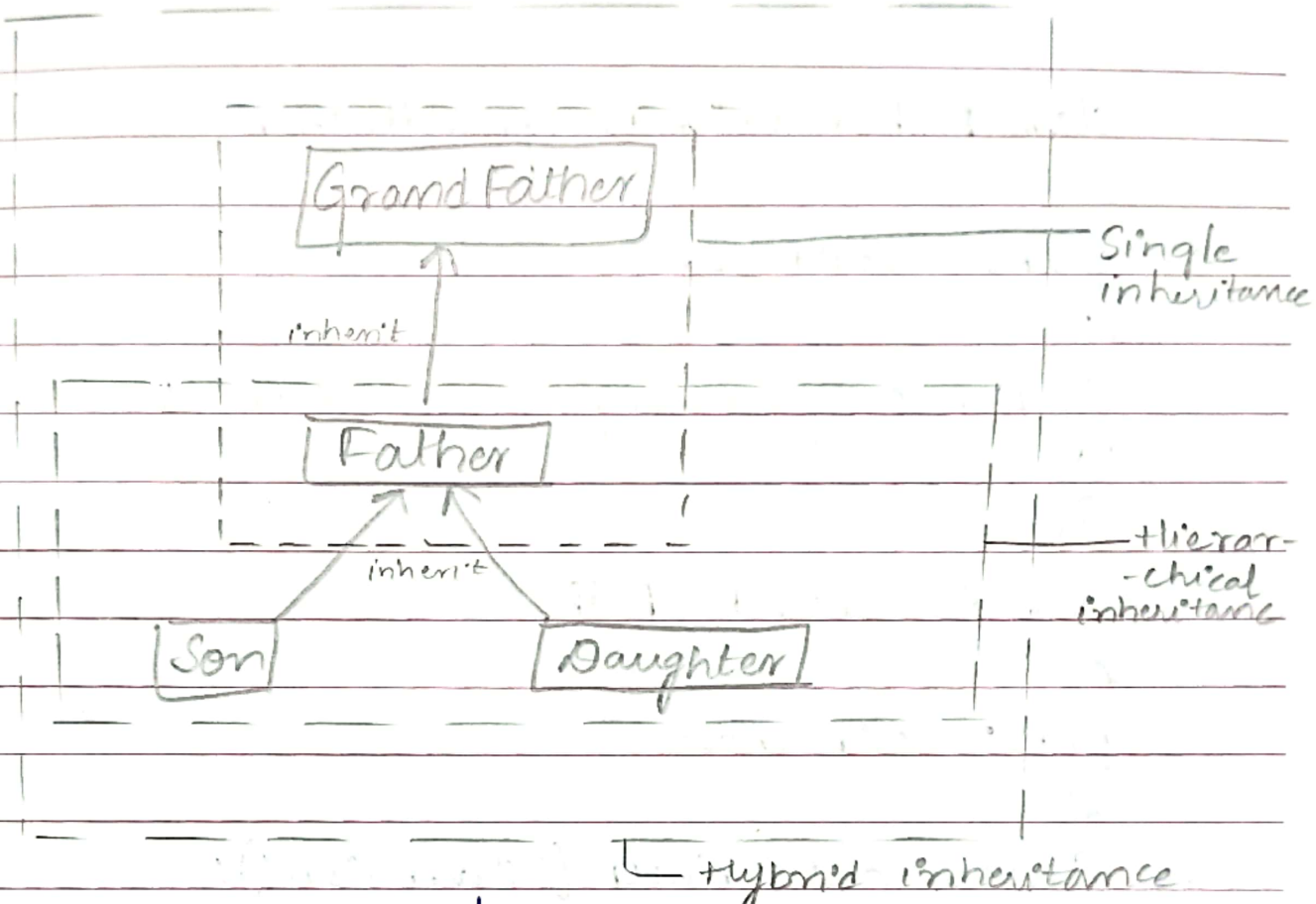
Expected Output :-

The method of the class student invoked
The method of the class student invoked
The method of the class student invoked.

→ Hybrid Inheritance :-

Hybrid means consist of more than one. Hybrid Inheritance is the combination of two or more types of inheritance

In the beside figure, Grandfather is a superclass. The Father class inherits from Grandfather. [∵ Grandfather & Father represents single



inheritance]. The father class is inherited by the Son and Daughter. So, Father class becomes the parent class for Son & Daughter class. These combinedly, it denotes the hybrid inheritance.

Eg:-

```
class Grandfather
{
public void show()
{
System.out.println("I am Grandfather");
}
}
```

```
class Father extends Grandfather
{
public void show()
{
System.out.println("I am father.");
}
}
class Son extends Father
{
public void show()
{
System.out.println("I am son");
}
}
public class Daughter extends Father
{
public void show()
{
System.out.println("I am a daughter.");
}
public static void main(String args[])
{
Daughter obj = new Daughter();
obj.show();
}
}
```

Expected Output:- I am Daughter.

(5) Explain the usage of tray, catch and finally with suitable example for each.

Try:- We specify the block of code that might give rise to the exception in a special block with 'try'

Catch:- When the exception is raised it needs to be caught by the program. It follows the try block that raises an exception. It should always be used with 'try'

Finally:- Sometimes we have an important code in pro that needs to be executed irrespective of whether or not the exception is thrown. This code is placed in a special block starting with the "finally" keyword.

Eg:-
```
class GFG {
    public static void main (String[] args) {
    int[] arr = new int[4];
    try {
    int i = arr[4];
    System.out.println("Inside try block"); }
    Catch (ArrayIndexOutOfBoundsException ex) {
        System.out.println("Exception caught in CatchBlock");
    finally {
        System.out.println("finally block executed"); }
    System.out.println("Outside try-catch-finally clause"); }}
```

Expected o/p :-