ADVANCED

1) PROGRAM – CLASS WITH METHOD OVERLOADING

AIM: To implement the concept of class with method overloading

THEORY

- In java, a class serves as a blueprint or a template for creating objects.
- It's a fundamental concept in Object-Oriented Programming, defines the structure and behavior that objects of that class will possess.
- Method Overloading is a feature in Object Oriented Programming where a class can have multiple methods with the same name but different parameter lists (different number of parameters, datatypes or both)
- This allows a single method name to be used for operations that are similar but require different inputs, improving code readability and reusability.

PROGRAM CODE

```
import java.util.Scanner;

class mov

{int a,b,c;

public void add()

{int result =a+b;

System.out.println("Sum 1:"+result);}

public void add(int a, int b)

{int result =a+b;

System.out.println("Sum 2:"+result);}

public int add(int a, int b, int c)

{int result =a+b+c;

return result;}

public static void main(String[] args)

{Scanner sc=new Scanner(System.in);

System.out.print("Enter Value of a:");

int a=sc.nextInt();

System.out.print("Enter Value of b:");

int b=sc.nextInt();

System.out.print("Enter Value of c:");
```

```
int c=sc.nextInt();
mov mv=new mov();
mv.add();
mv.add(a,b);
System.out.println("Sum 3:"+mv.add(a,b,c));}}
```

COMPILATION STEPS

```
javac mov.java
java mov
```

OUTPUT

```
Enter Value of a:20
Enter Value of b:10
Enter Value of c:30
Sum 1:0
Sum 2:30
Sum 3:60
```

2) PROGRAM – SINGLE & MULTI LEVEL INHERITANCE

AIM: To implement the concept of single level and multi-level inheritance.

THEORY

- Inheritance in Java is an **Object-Oriented Programming (OOP)** concept that allows one class to acquire the **properties and behaviours** of another class.
- The class that is inherited from is called the **Parent class** or **Superclass**.
- The class that inherits is called the **Child class** or **Subclass**.
- It helps in **code reusability**, reducing duplication.
- It supports **method overriding** to achieve runtime polymorphism.
- Method Overriding is defined as a feature that allows a **child class (subclass)** to provide its **own specific implementation** of a method that is already defined in its **parent class (superclass).**

PROGRAM CODE

```
class Person
{String name;
void person_details(String name)
{this.name = name; }
```

```
void show_p_details()
{System.out.println("Name: " + name);}}
// Single Inheritance (Person → Student)
class Student extends Person
{int rollNo;
void student_details(int rollNo)
{this.rollNo = rollNo;}
void show_s_details()
{show_p_details();
System.out.println("Roll No: " + rollNo);}}
// Multilevel Inheritance (Person → Student → CollegeStudent)
class CollegeStudent extends Student
{String course;
void setCourse(String course)
{this.course = course;}
void showCourse()
{System.out.println("Course: " + course);}}
public class I_Demo
{public static void main(String[] args)
{System.out.println("=== Single Inheritance ===");
Student s = new Student();
s.person_details("Ayesha");
s.student_details(101); s.show_s_details();
 System.out.println("\n=== Multilevel Inheritance ===");
CollegeStudent cs = new CollegeStudent();
```

```
cs.person_details("Soumya");

cs.student_details(202);

cs.setCourse("B.Tech CSE");

cs.show_p_details(); cs.showCourse();}}
```

COMPILATION STEPS

```
javac I_Demo.java

java I_Demo
```

OUTPUT

```
=== Single Inheritance ===

Name: Ayesha

Roll No: 101


=== Multilevel Inheritance ===

Name: Soumya

Course: B.Tech CSE
```

3) PROGRAM – INTERFACE

AIM: To create an interface Shape with the getArea() method. Create three classes Rectangle, Circle, and Triangle that implement the Shape interface. Implement the getArea() method for each of the three classes. (Using the concept of Interfaces).

THEORY

- An **interface** in Java is a **blueprint of a class** that contains **abstract methods** (methods without body) and **constants**.
  It represents **what a class must do**, but **not how** to do it.
- It is used to achieve:
  - ✔ **Abstraction**
  - ✔ **Multiple inheritance** (which Java classes do not support directly)
  - ✔ **Loose coupling**
- All methods are **public and abstract** by default and all variables are **public, static, and final**
- A class can **implement multiple interfaces**
- Interfaces help in achieving **100% abstraction**

PROGRAM CODE

```
interface Shape
{double getArea();}  // abstract method
class Rectangle implements Shape
{double l,b;
Rectangle(double l, double b)
{this.l=l; this.b=b;}
public double getArea()
{return l * b;}}
class Circle implements Shape
{double r;
Circle(double r)
{this.r = r;}
public double getArea()
{return 3.14 *r*r;}}
class Triangle implements Shape
{double bs, h;
Triangle(double bs, double h)
{this.bs=bs; this.h=h;}
 public double getArea()
{return 0.5 * bs*h;}}
public class ShapeDemo
{public static void main(String[] args)
{Shape rect = new Rectangle(10, 5);
  Shape circle = new Circle(7);
```

```
  Shape triangle = new Triangle(8, 6);

  System.out.println("Area of Rectangle: " + rect.getArea());

  System.out.println("Area of Circle: " + circle.getArea());

  System.out.println("Area of Triangle: " + triangle.getArea());}}
```

COMPILATION STEPS

```
javac ShapeDemo.java

java ShapeDemo
```

OUTPUT

```
Area of Rectangle: 50.0

Area of Circle: 153.86

Area of Triangle: 24.0
```

4) PROGRAM – ABSTRACT CLASS

AIM: To create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape. (Abstract Class concept).

THEORY

- An **abstract class** in Java is a class that **cannot be instantiated** (objects cannot be created from it).
  It is used to provide **partial abstraction** and **common structure** for subclasses.
- Declared using the **abstract** keyword
- Can have **abstract and non-abstract methods**
- Can have **constructors, variables, and static methods**
- Helps in **code reusability** and **method overriding**
- **Must be inherited** and **its abstract methods must be implemented** by the first concrete subclass

PROGRAM CODE

```
abstract class Shape

{abstract double calculateArea();abstract double calculatePerimeter();}

class Circle extends Shape

{double r;
```

```
Circle(double r)
{this.r=r;}
double calculateArea()
{return 3.14*r*r;}
double calculatePerimeter()
{return 2*3.14*r;}}
class Triangle extends Shape
{double s1,s2,s3,b,h;
 Triangle(double s1,double s2,double s3,double b,double h)
{this.s1=s1; this.s2=s2; this.s3=s3; this.b=b; this.h=h;}
double calculateArea()
{return 0.5*b*h;}
 double calculatePerimeter()
 {return s1+s2+s3;}}
public class A_Demo
{public static void main(String[] args)
{Shape c=new Circle(7);
  Shape t=new Triangle(3,4,5,4,6);
  System.out.println("Circle Area: "+c.calculateArea());
  System.out.println("Circle Perimeter: "+c.calculatePerimeter());
  System.out.println("Triangle Area: "+t.calculateArea());
  System.out.println("Triangle Perimeter: "+t.calculatePerimeter());}}
```

COMPILATION STEPS

```
javac A_Demo.java
java A_Demo
```

OUTPUT

Circle Area: 153.86

Circle Perimeter: 43.96

Triangle Area: 12.0

Triangle Perimeter: 12.0

5) PROGRAM – USER DEFINED EXCEPTION HANDLING (FROM NB)
6) PROGRAM – MULTI THREADING CONCEPT

AIM: To create and start multiple threads that increment a shared counter variable concurrently. (Multithreading Concept)

THEORY

- A thread represents a single, independent path of execution within a program.
- It is the smallest unit of execution that can be managed by the operating system's scheduler.
- Multithreading is a feature in Java that allows **multiple tasks (threads)** to run **concurrently** within a single program.
- Multithreading helps improve performance by using **CPU time efficiently**.
- Faster execution & better CPU utilization
- Cost-effective (threads share same memory)
- Background tasks can run alongside main tasks
- Useful in animations, games, networking, servers, etc.

PROGRAM CODE

```
//Counter class is created with synchronized increment()

class counter

{private int c=0;

public synchronized void increment()

{c++;}

public int getCount()

{return c;}}

public class execute

{public static void main(String[] args)

{counter c=new counter();

Thread t1=new Thread();
```

```
for(int i=0;i<3;i++)
{c.increment();}
Thread t2=new Thread();
{for(int i=0;i<3;i++)
{c.increment();}}
t1.start();t2.start();
try
{t1.join();t2.join();}
catch (InterruptedException e)
{e.printStackTrace();}
System.out.println("Final Count:"+c.getCount());}}
```

COMPILATION STEPS

```
javac execute.java
java execute
```

OUTPUT

```
Final Count:6
```

7) PROGRAM – FILE PROPERTIES (IN RECORD)

THEORY -

- Java provides the concept of **File Handling** to perform operations such as:

1) Create a file
2) Read data from a file
3) Write data to a file
4) Delete a file

- File handling allows storing data **permanently** (secondary storage), unlike variables which store data temporarily in RAM.
- File handling in Java involves interacting with the computer's file system to perform operations like creating, reading, writing, and deleting files and directories.
- Java provides robust support for file handling

8) PROGRAM – COPYING CONTENTS OF FILES (FROM RECORD)

9) PROGRAM – SERIALIZATION & DESERIALIZATION

(FROM RECORD)

10) PROGRAM – GUI APPLICATION

(FROM RECORD)

11) PROGRAM – PACKAGES

(FROM RECORD – CHECK COMPILATION STEPS)

12) PROGRAM – CONSTRUCTOR

PROGRAM CODE

```
import java.util.Scanner;
class Student
{int r; String name;
 Student()
 {r=0;
  name="Unknown";
  System.out.println("Default Constructor Called");}
  Student(int rn, String n)
 {r=rn; name=n;
  System.out.println("Parameterized Constructor Called");}
 void display()
  {System.out.println("Roll No:"+r+", Name:"+name);}
  public static void main(String[] args)
  {Scanner sc=new Scanner(System.in);
    System.out.print("Enter Roll Number:");
    int r=sc.nextInt();
    sc.nextLine();
    System.out.print("Enter Name:");
```

```
String n=sc.nextLine();
 Student s1=new Student();
 s1.display();
 Student s2=new Student(r,n);
 s2.display();}}
```

COMPILATION STEPS

```
javac Student.java
java Student
```

OUTPUT

```
Enter Roll Number:120
Enter Name:Ray
Default Constructor Called
Roll No:0, Name:Unknown
Parameterized Constructor Called
Roll No:120, Name:Ray
```