JAVA ASSIGNMENT – ANSHIMA SHARMA

Data Types

Explain the difference between primitive and reference data types with examples.

1. Location in Memory

Primitive Types are kept in the stack memory.

Reference Types are kept in the heap memory (but reference itself is kept in the stack).

2. Default Values

Primitive Types contain default values:

0 for the numeric types (int, float, etc.)

false for the boolean

\u0000 (null character) for char

Reference Types default to null.

3. Data Storage

Primitive Types keep the value directly in memory.

Reference Types keep the address in memory (reference) and not the data.

4. Mutability

Primitive Types are immutable (you can't alter the value, but it overwrites the previous one).

Reference Types may be mutable (e.g., an array or object is modified and alters the original data).

5. Performance

Primitive Types are quicker since they hold direct values.

Reference Types are a bit slower since they must refer to the actual data via memory addresses.

6. Example Data Types

Primitive Types: int, double, char, boolean, etc.

Reference Types: String, Array, Class Objects, etc.

Primitive and References Data Types Example:-

```
public class Program6 {
    public static void main(String[] args) {
        //Primitive Data Types
        int number=55;
        boolean isValid=true;
        char grade='A';
        double value=999.896;

        System.out.println("Integer value: "+number);
        System.out.println("Boolean value: "+isValid);
        System.out.println("Character value: "+grade);
        System.out.println("Double value: "+value);

        //Reference Data Types
        String name="Anshima Sharma";
        int marks[]={98,89,78,99,93};

        System.out.println("String: "+name);
        System.out.println("Array first element: "+marks[0]);
```

Object Oriented Programming (OOP)

1. Explain the concept of encapsulation with a suitable example.

Encapsulation in Java

Encapsulation is the process of binding data and methods together into a single unit and restricting data access to the data by making it private. It is one of the fundamental principles of Object-Oriented Programming. Instead access to the data is provided through public getter and setter methods.

Features of Encapsulation:-

1. Enhances reusability:- Encapsulated classes can be reused in difference programs.

2. Controlled access:- Only specific methods can specify the data , maintaining data integrity.

3. Data hiding:- Prevent data access to data members and methods, ensuring better security.

4. Improves maintainability:- Changes to fields can be managed without affecting the rest of the program.

Program:-

```
class Student{
   private String name;
   private int rollNo;
   private double marks;

   public Student(String name,int rollNo, double marks){
      this.name=name;
      this.rollNo=rollNo;
      this.marks=marks;
   }
```

```
public String getName(){
   return name;
}

public void setName(String name){
   this.name=name;
}

public int getRollNo(){
   return rollNo;
}
public void setRollNo(int rollNo){
   this.rollNo=rollNo;
}

public double getMarks(){
   return marks;

}
public void setMarks(double marks){
   if (marks >= 0 && marks <= 100) {
      this.marks = marks;
   } else {
      System.out.println("Invalid marks! Please enter between 0 and 100.");
   }
}

public void display(){
```

```
        System.out.println("Student Name: "+name);
        System.out.println("Roll Number: "+rollNo);
        System.out.println("Marks: "+marks);
    }



}

public class Program19 {

    public static void main(String[] args) {
        Student s1=new Student("Anshima",11 , 98);
        s1.display();
        s1.setMarks(95.5);
        System.out.println("Updates details: ");
        s1.display();

    }
}
```

Advanced Concepts (OOP)

2. Explain the concept of interfaces and abstract classes with examples.

Interfaces

An **interface** is like a blueprint of a class that only contains **abstract methods** (before Java 8) or default/static methods (after Java 8) that must be implemented by any class that uses it. Interfaces help achieve **multiple inheritance** and ensure that all implementing classes follow the same structure.

Features:-

Can only have abstract methods. (before Java 8)

Can have default and static methods. (after Java 8)

All variables are public abstract and final by default.

Does not have constructors.

Abstract Classes

An abstract class is a class that cannot be instantiated and may contain both abstract methods (without implementation) and concrete methods (with implementation) methods. It is used when classes share common behavior but also need some method implementation.

Features:-

Can have both abstract methods and concrete methods.

Can have constructors.

Can declare instance variables and methods with any access modifiers.

Can extend only one abstract class.

Program:-

```
abstract class Animal {

   abstract void makeSound();

}
```

```
interface Pet {
    void play();
}

class Dog extends Animal implements Pet {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }

    @Override
    public void play() {
        System.out.println("Dog is playing");
    }
}

public class Program23 {
    public static void main(String[] args) {
        Dog d=new Dog();
        d.makeSound();
        d.play();
    }
}
```

3. Explore multithreading in Java to perform multiple tasks concurrently.

Multithreading in Java

Multithreading is a feature in Java that allows multiple threads to run concurrently, enabling efficient CPU utilization and improved performance for tasks such as background computations, GUI applications, and handling multiple client requests.

Important Terms in Multithreading

- Thread – A lightweight process that executes independently.
- Concurrency – Multiple threads executing at the same time.
- Synchronization – Coordinating access to shared resources to prevent conflicts.
- Thread Life Cycle:-
  New – Thread is created but not started.
  Runnable – Thread is ready to run but waiting for CPU.
  Running – Thread is executing.
  Blocked/Waiting – Thread is waiting for a source.
  Terminated – Thread execution is finished.

Program:-

Extending thread class

```
class First extends Thread{
  public void run(){
    System.out.println("Priority of "+Thread.currentThread().getName()+" is:
  "+Thread.currentThread().getPriority());
  }
}

public class Program26 {
  public static void main(String[] args) {
    First t1=new First();
    First t2=new First();

    t1.start();
    t2.start();
  }
}
```

Implementing Runnable Interface

```
class Second implements Runnable {
    public void run() {
            System.out.println("Priority of "+Thread.currentThread().getName()+" is:
"+Thread.currentThread().getPriority());
    }
}

public class Program26 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Second ());
        Thread t2 = new Thread(new Second());
        t1.start();
        t2.start();
    }
}
```