Java Assignment-1

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

Primitive Data Types:

In Java, data types are divided into two categories: primitive data types and reference data types. Primitive data types, such as int, float, char, and boolean, represent simple values and are not objects. Primitive types are stored in stack memory, which allows quick access and automatic memory management. These types are more efficient in terms of memory and performance because they do not have the overhead associated with objects.

EG: int num=10;

Reference Data Types:

Reference types are stored in heap memory, with their references (addresses) stored in stack memory. They hold memory addresses that point to objects, rather than storing actual values directly. These include objects created from classes, arrays, and strings. Since reference types store the location of an object in memory, they allow more complex data structures but are less memory efficient compared to primitive types. The default value for reference types is null, indicating that the reference does not point to any object initially. Reference types are mutable, so changes made to an object are reflected across all references to it.

EG: string name=”Rohit”;

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

Encapsulation is a fundamental concept in object-oriented programming that helps protect data and ensure controlled access. It is similar to how we see a medicine capsule which have various medicine all together just like variables and methods in our code.

In other words, Encapsulation is a concept where the details of an object's data are hidden from the outside world. It is achieved by making the fields (variables) private and providing public methods (getters and setters) to access and modify those fields.

We make private fields to avoid direct access to them.

Example:

```java
class Student
{
    private String name;
    private int rollNumber;
    private double marks;

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

    // Getter and Setter methods (Encapsulation)
    public String getName()
    {
        return name;
    }
    public int getRollNumber()
    {
        return rollNumber;
    }
    public double getMarks()
    {
        return marks;
    }

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

    public void setRollno(int rollno)
    {
        this.rollNumber=rollno;
    }
```

Creating getter setter methods as the private members cannot be directly accessed for security purpose.

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

An **interface** is a blueprint that defines a set of abstract methods. It cannot have method implementations. It is used to enforce a contract on classes that implement it. A class that implements an interface must provide implementations for all its methods.

An **abstract class** in Java is a class that cannot be instantiated on its own and may contain both abstract and concrete methods. It is useful when we want to share code among related classes but still we want some of the methods to be defined by subclasses.

Example:

```java
interface Vehicle
{
    // Abstract method (must be implemented)
    void start();
    void stop();
}

abstract class Car implements Vehicle {
    String brand;

    // Constructor
    public Car(String brand) {
        this.brand = brand;
    }

    // Concrete method
    public void showBrand() {
        System.out.println("Car Brand: " + brand);
    }

    // Abstract method
    abstract void fuelType();
}
```

Now the class implementing vehicle has to define the 2 methods inside the interface.

But the class that extends Car only needs to define the fuelType method.

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

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process. Threads can be created by using two mechanisms :

1. Extending the Thread class

2. Implementing the Runnable Interface

Well the main difference between the 2 methods is that when we are implementing multi-threading with runnable interface we can still extend more class to it (Multiple Inheritance) but that cannot be achieved with Thread class.

Example

Multithreading.java

```java
// Thread using Thread class
class MyThread extends Thread
{
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Thread Class: " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}

// Thread using Runnable interface
class MyRunnable implements Runnable
{
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Runnable Interface: " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}

// Main class
public class Multithreading
{
    public static void main(String[] args) {

        MyThread t1 = new MyThread();


        Thread t2 = new Thread(new MyRunnable());

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

```