Java Assignment (theory questions)

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

<u>Primitive Data Types:</u>

1.Primitive types are stored in stack memory, which allows quick access and automatic memory management.
2.They store actual values directly, unlike reference types that store memory addresses.
3.Each primitive type has a fixed size in memory, such as int (4 bytes) and double (8 bytes).
4.Primitive types are immutable, meaning if you modify a value, a new copy is created rather than changing the original.
5.They are used for basic operations, including numbers (int, double), characters (char), and Boolean values (Boolean).

Example: int a = 10;

<u>Reference Data Types:</u>

1. Reference types are stored in heap memory, with their references (addresses) stored in stack memory.
2. They hold memory addresses that point to objects, rather than storing actual values directly.
3 .Unlike primitive types, reference types have a dynamic size, meaning they can expand or shrink as needed.
4. Reference types are mutable, so changes made to an object are reflected across all references to it.
5. They are used for complex data structures, such as arrays, strings, and objects created from classes.

Example: String s = "Rishita";

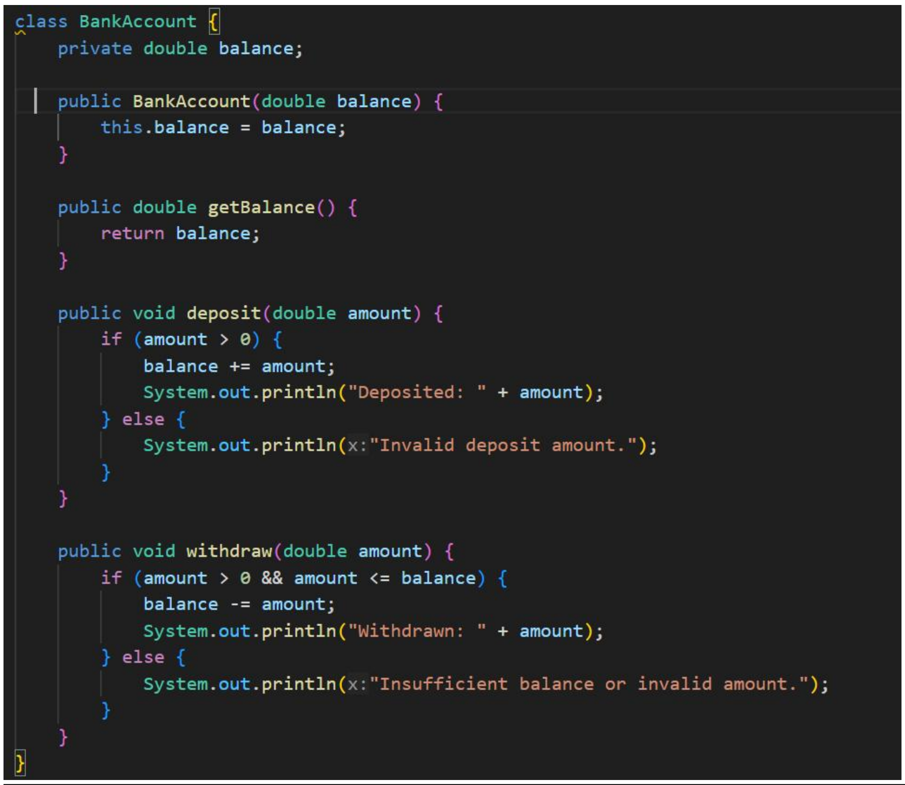
Q. Explain the concept of encapsulation with a suitable example.

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.

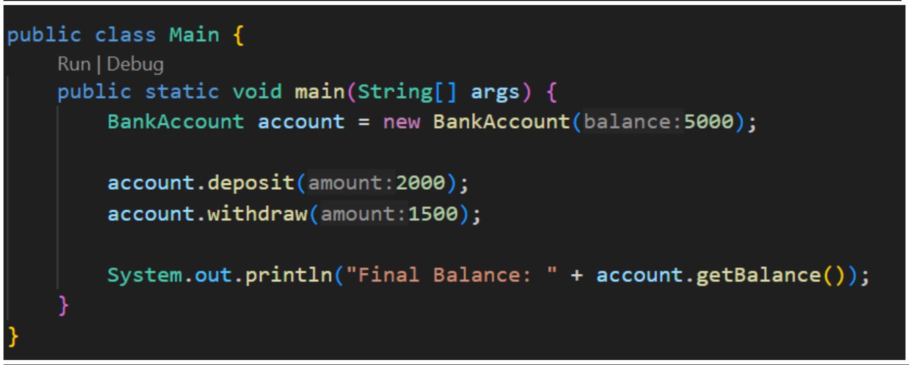
In programming, encapsulation means restricting direct access to certain variables and allowing controlled modifications through methods(like getters and setters). This prevents accidental or unauthorized changes to important data.

Encapsulation helps in maintaining security, data integrity, and better control over how data is accessed and modified.

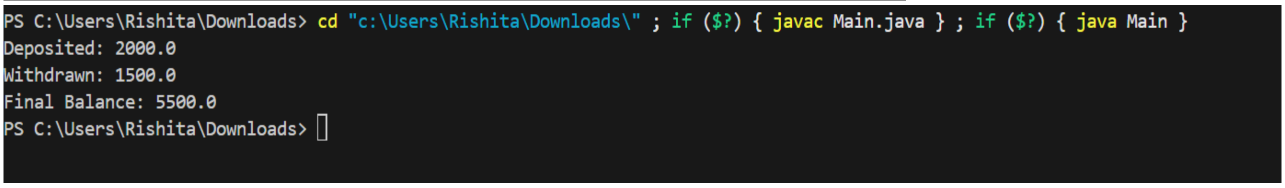
Example code:



```java
class BankAccount {
    private double balance;

    public BankAccount(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        } else {
            System.out.println(x:"Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println(x:"Insufficient balance or invalid amount.");
        }
    }
}
```



```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        BankAccount account = new BankAccount(balance:5000);

        account.deposit(amount:2000);
        account.withdraw(amount:1500);

        System.out.println("Final Balance: " + account.getBalance());
    }
}
```



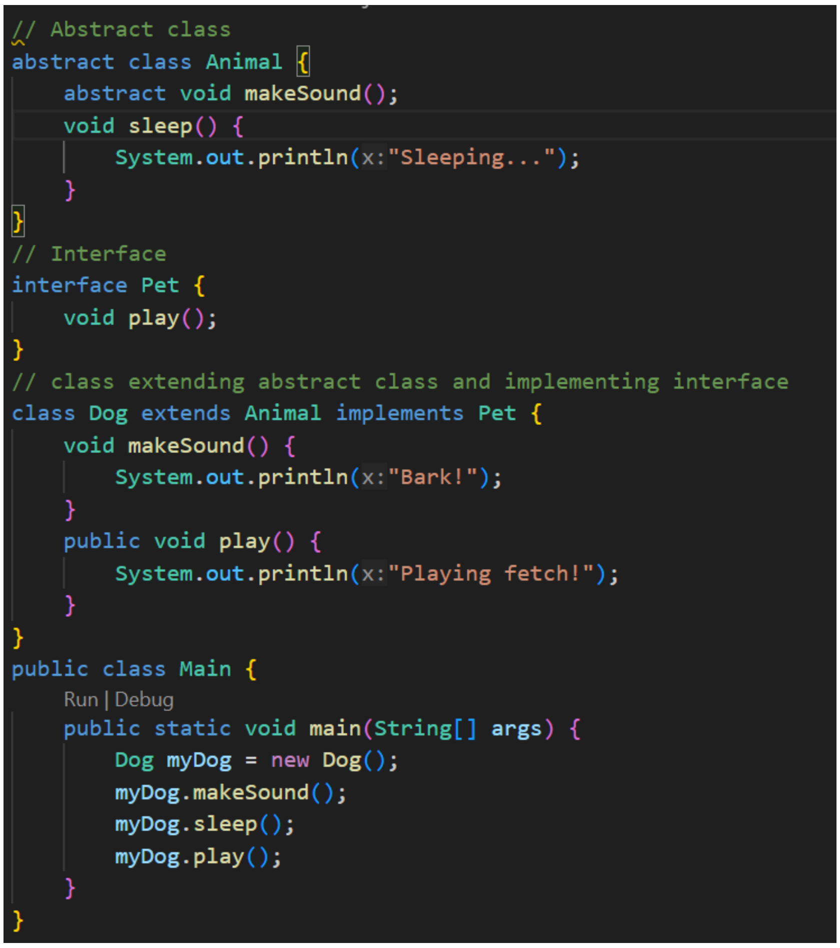
```
PS C:\Users\Rishita\Downloads> cd "c:\Users\Rishita\Downloads\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Deposited: 2000.0
Withdrawn: 1500.0
Final Balance: 5500.0
PS C:\Users\Rishita\Downloads>
```

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

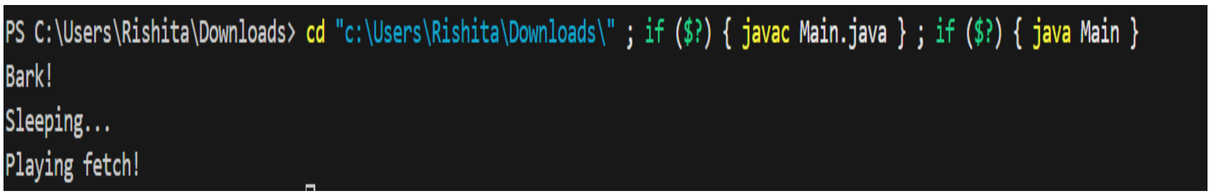
An **abstract class** in Java is a class that cannot be instantiated on its own and may contain both abstract and concrete methods. It is useful when we want to share code among related classes but still we want some of the methods to be defined by subclasses.

An **interface** is like a blueprint that only contains method declarations (without implementations). It is used to enforce a contract on classes that implement it. A class can implement multiple interfaces, unlike abstract classes, which can only be extended by one subclass.

Example code:



```java
// Abstract class
abstract class Animal {
    abstract void makeSound();
    void sleep() {
        System.out.println(x:"Sleeping...");
    }
}
// Interface
interface Pet {
    void play();
}
// class extending abstract class and implementing interface
class Dog extends Animal implements Pet {
    void makeSound() {
        System.out.println(x:"Bark!");
    }
    public void play() {
        System.out.println(x:"Playing fetch!");
    }
}
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound();
        myDog.sleep();
        myDog.play();
    }
}
```



```
PS C:\Users\Rishita\Downloads> cd "c:\Users\Rishita\Downloads\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Bark!
Sleeping...
Playing fetch!
```

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

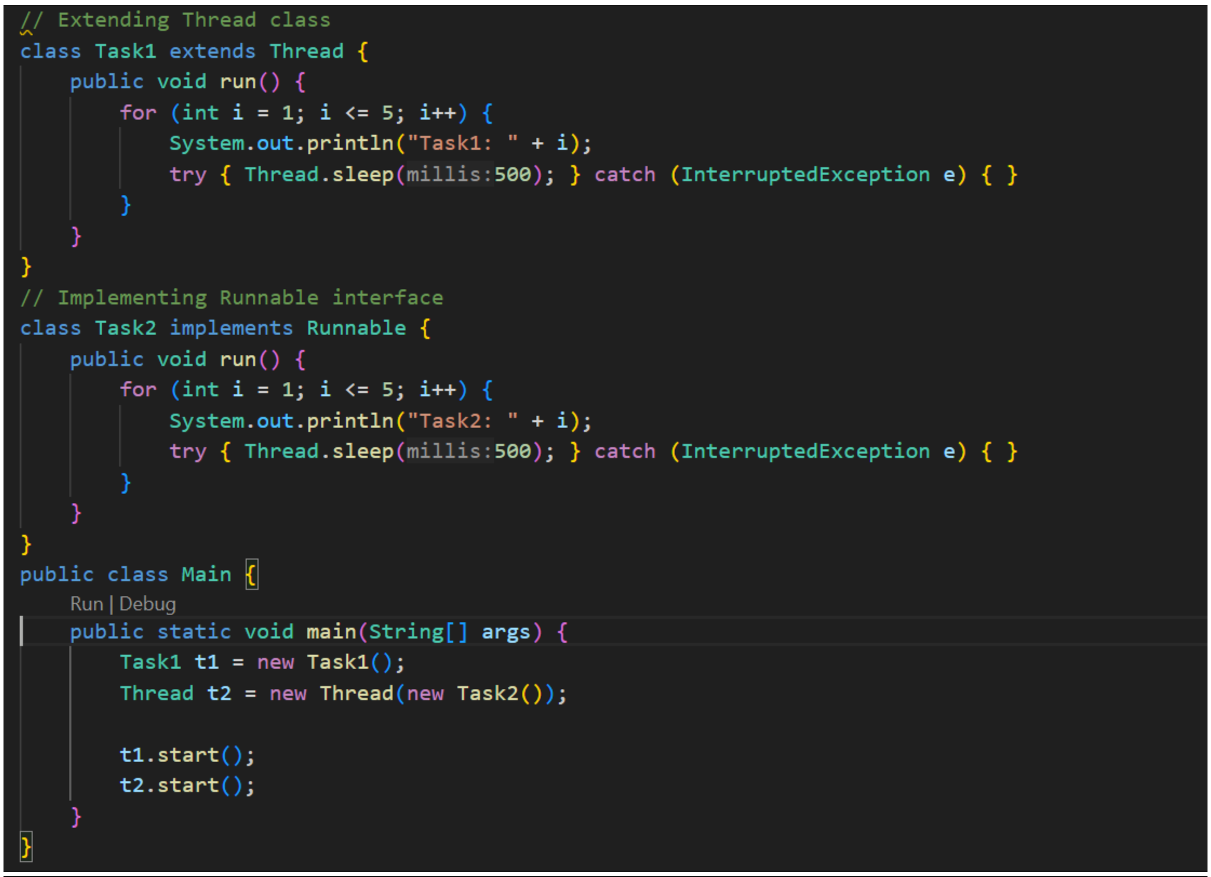
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

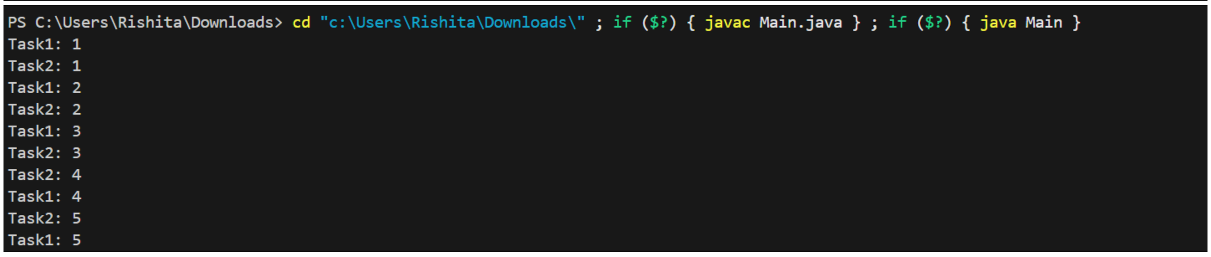
1. Extending the Thread class
2. Implementing the Runnable Interface

Java provides the Thread class and Runnable interface to create and manage threads. Multithreading is useful in scenarios like handling multiple user requests, performing background tasks, and parallel computing.

Example code:



```java
// Extending Thread class
class Task1 extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task1: " + i);
            try { Thread.sleep(millis:500); } catch (InterruptedException e) { }
        }
    }
}
// Implementing Runnable interface
class Task2 implements Runnable {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Task2: " + i);
            try { Thread.sleep(millis:500); } catch (InterruptedException e) { }
        }
    }
}
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Task1 t1 = new Task1();
        Thread t2 = new Thread(new Task2());

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



```
PS C:\Users\Rishita\Downloads> cd "c:\Users\Rishita\Downloads\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Task1: 1
Task2: 1
Task1: 2
Task2: 2
Task1: 3
Task2: 3
Task2: 4
Task1: 4
Task2: 5
Task1: 5
```