Java Assignment – 1

Theory Questions

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

Ans - In Java, data types are categorized into two main groups: primitive data types and reference data types. Here's a detailed explanation of the differences between them, along with examples:

1. Primitive Data Types

Primitive data types are the most basic data types in Java. They represent simple values and are predefined by the language. They are stored directly in memory and have a fixed size.

Characteristics of Primitive Data Types:

- They store actual values (not references).
- They are stored in the stack memory.
- They have a fixed size (e.g., `int` is always 4 bytes).
- They are not objects and do not have methods.
- There are 8 primitive data types in Java.

Example of Primitive Data Types:

```
int age = 25;              // Integer
double salary = 50000.75;  // Floating-point number
char grade = 'A';          // Character
boolean isJavaFun = true;  // Boolean
```

2. Reference Data Types

Reference data types are used to store references (memory addresses) to objects. They do not store the actual data but point to the location in memory where the data is stored.

Characteristics of Reference Data Types:

- They store references to objects (not the actual values).
- They are stored in the heap memory.
- They have variable sizes depending on the object.
- They are objects and have methods (e.g., `String` has methods like `length()`, `substring()`, etc.).
- Examples include classes, arrays, interfaces, and strings.

Common Reference Data Types in Java:

Classes: `String`, `Scanner`, `ArrayList`, etc.

Arrays: `int[]`, `String[]`, etc.

Interfaces: `List`, `Map`, etc.

Example of Reference Data Types:

```
String name = "Arun Parihar";         // String object
int[] numbers = {4, 5, 24, 55};  // Array of integers
Scanner scanner = new Scanner(System.in); // Scanner object
```

Key Differences Between Primitive and Reference Data Types

Features	Primitive Data Types	Reference Data Types
Storage	Stored in stack memory	Stored in heap memory
Size	Fixed size	Variable size (depends on the object)
Value	Stores actual values	Stores references (memory addresses)
Default Value	Has default values (e.g. int is 0)	Default value is null
Methods	No methods	Has methods (e.g. String.length())
Examples	Int, double, char, Boolean etc.	

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

Ans - Encapsulation in Java

Encapsulation is one of the four fundamental principles of Object-Oriented Programming (OOP). It refers to the bundling of data (attributes) and methods (behaviors) that operate on the data into a single unit, typically a class. Encapsulation also involves restricting direct access to some of an object's components, which is achieved using access modifiers like `private`, `public`, `protected`, and `default`.

Key Concepts of Encapsulation:

1. Data Hiding: Restricting direct access to the internal state of an object.

2. Controlled Access: Providing controlled access to the data through public methods (getters and setters).

3. Bundling: Combining data and methods that operate on the data into a single unit (class).

Example: Example is in the code file name – Encapsulation.java

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

Ans - Interfaces and Abstract Classes in Java

Both interfaces and abstract classes are used to achieve abstraction in Java. They allow you to define a blueprint for other classes to follow. However, they have distinct characteristics and use cases. Let's explore both concepts in detail with examples.

1. Abstract Classes

An abstract class is a class that cannot be instantiated (you cannot create objects of an abstract class). It is used as a base class for other classes. Abstract classes can have:

Abstract methods: Methods without a body (must be overridden by subclasses).

Concrete methods: Methods with a body (can be inherited as-is or overridden).

Example of an Abstract Class:

AbstractClass.java > AbstractClass

```java
// Abstract class
abstract class Animal {
    Tabnine | Edit | Test | Explain | Document
    // Abstract method (no body)
    public abstract void makeSound();

    Tabnine | Edit | Test | Explain | Document
    // Concrete method
    public void sleep() {
        System.out.println(x:"The animal is sleeping.");
    }
}

// Subclass (inherits from Animal)
class Dog extends Animal {
    Tabnine | Edit | Test | Explain | Document
    // Override the abstract method
    @Override
    public void makeSound() {
        System.out.println(x:"The dog barks.");
    }
}

```

```java

public class AbstractClass {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); // Calls the overridden method
        dog.sleep();     // Calls the inherited method
    }
}
```

```
E:\NucleusTeq\Git\Git Assignments files\Java Assignment-1> cmd /C ""C:\Users\ARUN PARIHAR\AppData\Roaming\Code\User\globalStorage\pleiades.java-extension-pack-jdk\java\latest\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\ARUN PARIHAR\AppData\Roaming\Code\User\workspaceStorage\250393cec90bde8224fdbdc3aad31fe3\redhat.java\jdt_ws\Java Assignment-1_cb9b3eaa\bin" AbstractClass "
The dog barks.
The animal is sleeping.
```

2. Interfaces

An interface is a completely abstract class that can only contain:

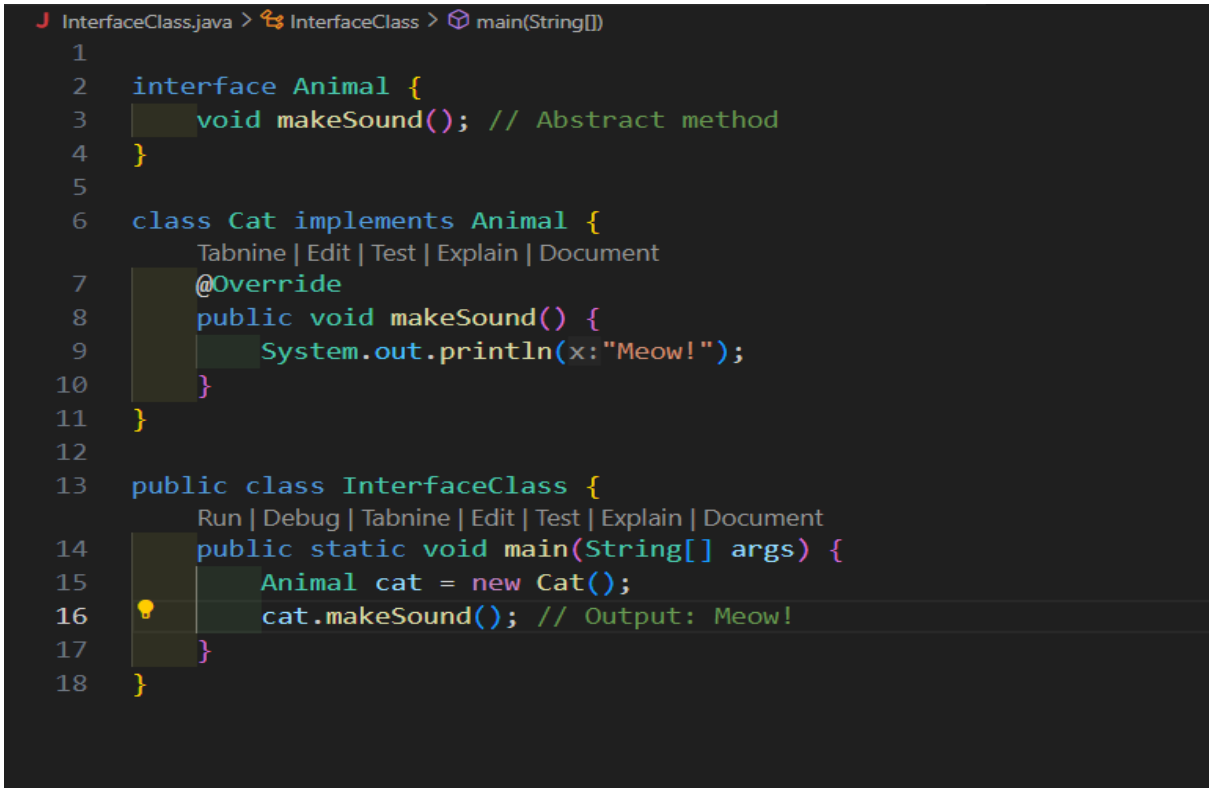
Abstract methods (methods without a body).

Default methods (methods with a body, introduced in Java 8).

Static methods (methods with a body, introduced in Java 8).

Constant fields (`public static final` variables).

Example of an Interface:



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

Ans - Multithreading in Java

Multithreading is a feature in Java that allows concurrent execution of two or more threads to maximize the utilization of the CPU. A thread is the smallest unit of a process that can execute independently. Multithreading is used to perform multiple tasks concurrently, improving the performance and responsiveness of applications.

Key Concepts of Multithreading

1. Thread:

A thread is a lightweight subprocess that executes a set of instructions independently.

Threads share the same memory space of the process they belong to, which allows them to communicate with each other easily

2. Main Thread:

When a Java program starts, the JVM creates a main thread that executes the `main()` method.

3. Thread Lifecycle:

A thread goes through various states during its lifecycle:

New: The thread is created but not yet started.

Runnable: The thread is ready to run and waiting for the CPU.

Running: The thread is executing its task.

Blocked/Waiting: The thread is waiting for a resource or another thread.

Terminated: The thread has completed its execution.

4. Concurrency vs Parallelism:

Concurrency: Multiple tasks make progress simultaneously (not necessarily at the same time).

Parallelism: Multiple tasks execute at the same time (requires multiple CPU cores).

Advantages of Multithreading

1. Improved Performance:

Multithreading allows tasks to run concurrently, making better use of CPU resources.

2. Responsiveness:

Applications remain responsive even when performing long-running tasks.

3. Resource Sharing:

Threads share the same memory space, making it easier to share data between threads.

4. Simplified Modeling:

Multithreading simplifies the design of complex applications by breaking tasks into smaller units.