Unit 1: Introduction to Java and OOP

This unit covers the fundamental concepts of Object-Oriented Programming, the features of Java, and its basic architecture.

Question 1: Differentiate between Procedure-Oriented Programming (POP) and Object-Oriented Programming (OOP). (4 Marks)

Step-by-Step Explanation

Procedure-Oriented Programming (POP) is a programming paradigm where the main focus is on procedures or functions. Large programs are divided into smaller functions, and data is often shared globally between them. The primary approach is top-down. Examples include C and Pascal.

Object-Oriented Programming (OOP) is a paradigm based on the concept of "objects," which bundle data and the methods that operate on that data. It emphasizes data security through concepts like encapsulation and uses a bottom-up approach. Examples include Java and C++.

Key Differences:

Feature	Procedure-Oriented Programming (POP)	Object-Oriented Programming (OOP)		
Approach	Follows a	**top-down** approach.	Follows a	**bottom-up** approach.
Focus	Emphasis is on	**functions** or procedures, not data.	Emphasis is on	**data** and objects.

Data Security	Lacks data hiding; data is often global and can be accessed by any function, making it less secure.	Provides data hiding through	**encapsulation**; data is secured and can only be accessed by an object's own methods.
Inheritance	Does not support inheritance.	Supports	**inheritance**, which allows for code reusability.

Export to Sheets

MCQs for Practice

1. **Which programming paradigm uses a top-down approach for program design?** a) OOP b) POP c) Both OOP and POP d) Neither
   Answer: b) POP
2. **Encapsulation and data hiding are core features of which paradigm?** a) Procedure-Oriented b) Object-Oriented c) Structure-Oriented d) Function-Oriented
   Answer: b) Object-Oriented

Question 2: Explain four core concepts of OOP with examples. (4 Marks)

Step-by-Step Explanation

Object-Oriented Programming is built on several key concepts that allow for the creation of modular and reusable code.

1. **Encapsulation** 💊
   - **What it is:** The process of wrapping code (methods) and data (variables) together into a single unit, like a class. It hides the internal complexity of an object from the outside world.
   - **Example:** A medical store is a form of encapsulation. You (the user) only interact with the chemist (the public method), while the medicines (the private data) are hidden and protected inside the store. You cannot access the medicines directly.
2. **Inheritance** 👪
   - **What it is:** The mechanism by which one class (the child or subclass) acquires the properties and methods of another class (the parent or base class). It promotes code reusability.

   - **Example:** A `Car` class can inherit common properties like wheels and an engine from a `Vehicle` class, while also having its own unique features.
3. **Polymorphism** 🎭
   - **What it is:** A Greek term meaning "many forms." It is the ability of a method or object to take on many forms. This allows a single action to be performed in different ways.
   - **Example:** The `+` operator in Java exhibits polymorphism. When used with numbers, it performs addition (`2 + 3 = 5`), but when used with strings, it performs concatenation (`"Hello" + "World" = "HelloWorld"`).
4. **Abstraction** 📺
   - **What it is:** The concept of hiding implementation details and showing only the necessary features (functionality) to the user.
   - **Example:** When you use a TV remote, you only see and use the buttons to change the channel or volume. You don't need to know the complex electronics working inside the remote. This is abstraction.

MCQs for Practice

The ability of an operator or method to behave differently in different situations is known as?

a) Inheritance

b) Encapsulation

c) Polymorphism

d) Abstraction
Answer: c) Polymorphism

Which OOP concept is primarily used for achieving code reusability?

a) Abstraction

b) Inheritance

c) Encapsulation

d) Polymorphism
Answer: b) Inheritance

Question 3: Explain the architecture of Java (JDK, JRE, JVM). (4 Marks)

Step-by-Step Explanation

The Java architecture consists of three core components that enable the development and execution of Java applications.

1. **JVM (Java Virtual Machine)**
    - **Role:** The JVM is the heart of the Java architecture. It's an abstract machine that provides a runtime environment to execute Java bytecode.
    - **Function:** It loads code, verifies it for security, executes it, and manages memory. The JVM is what makes Java platform-independent.
2. **JRE (Java Runtime Environment)**
    - **Role:** The JRE is the on-disk implementation of the JVM. It contains the JVM, core libraries (like `rt.jar`), and other files needed to *run* Java applications.
    - **Function:** If you only want to run a Java program, you only need to install the JRE.
3. **JDK (Java Development Kit)**
    - **Role:** The JDK is the complete software package for Java developers. It contains everything in the JRE, plus development tools needed to *create* Java applications.
    - **Function:** Key tools in the JDK include `javac` (the compiler), `java` (the launcher), and `jar` (the archiver).

Relationship: The components are hierarchical: The **JDK** includes the **JRE**, and the **JRE** includes the **JVM**.

MCQs for Practice

Which component is responsible for making Java platform-independent?

a) JDK

b) JRE

c) JVM

d) Javac
Answer: c) JVM

To develop a Java application, which of the following must be installed?

a) Only JVM

b) Only JRE

c) JDK

d) A text editor
Answer: c) JDK

Unit 2: Literals, Data Types, Variables, Operators & Control Statements

This unit covers the building blocks of Java programming, including how to store data, perform operations, and control the flow of a program.

Question 1: Explain the different types of variables in Java (local, instance, and static) with an example. (4 Marks)

Step-by-Step Explanation

In Java, variables are containers for storing data values. They are categorized based on where they are declared.

1. **Local Variables:**
   - **Definition:** Declared *inside* a method, constructor, or block.
   - **Scope:** Their scope is limited to the method they are declared in. They are created when the method starts and destroyed when it ends.
   - **Memory:** Stored in the stack memory.
2. **Instance Variables:**
   - **Definition:** Declared *inside* a class but *outside* any method. They are also known as non-static fields.
   - **Scope:** They belong to an object (an instance) of the class. Each object has its own copy of instance variables.
   - **Memory:** Stored in the heap memory as part of the object.
3. **Static Variables:**
   - **Definition:** Declared with the `static` keyword, inside a class but outside any method.
   - **Scope:** They belong to the class, not to any individual object. There is only one copy of a static variable, shared among all objects of that class.
   - **Memory:** Stored in a special area of memory called the static memory area.

Example Program

Java
```
class Student {
   // Instance variable: each student object will have its own name
   String name;
```

```
    // Static variable: school name is shared by all students
    static String schoolName = "Ganpat University";

    void display() {
        // Local variable: age is only accessible within this method
        int age = 20;
        System.out.println("Name: " + name + ", Age: " + age + ", School: " + schoolName);
    }

    public static void main(String[] args) {
        Student s1 = new Student();
        s1.name = "Karan";

        Student s2 = new Student();
        s2.name = "Aryan";

        s1.display();
        s2.display();
    }
}
```

MCQs for Practice

Which type of variable is shared among all objects of a class?

a) Local variable

b) Instance variable

c) Static variable

d) Final variable
Answer: c) Static variable

Where is a local variable declared?

a) Inside a class, outside a method

b) Inside a method or block

c) Globally

d) Using the `static` keyword
Answer: b) Inside a method or block

Question 2: Differentiate between `while` and `do-while` loops. Explain with a suitable example. (4 Marks)

Step-by-Step Explanation

Both `while` and `do-while` loops are used to execute a block of code repeatedly as long as a condition is true. The key difference lies in *when* the condition is checked.

`while` Loop (Entry-Controlled Loop):

1. **Condition Check:** The condition is checked at the *beginning* of the loop.
2. **Execution:** The loop body executes only if the condition is true. If the condition is false initially, the loop body will not execute even once.

Syntax:
Java
```
while (condition) {
   // statements
}
```

3.

`do-while` Loop (Exit-Controlled Loop):

1. **Condition Check:** The condition is checked at the *end* of the loop.
2. **Execution:** The loop body is guaranteed to execute at least once, regardless of whether the condition is true or false.

Syntax:
Java
```
do {
   // statements
} while (condition);
```

3.

Example Program
Java
```
public class LoopTest {
   public static void main(String[] args) {
      int i = 5;

      System.out.println("--- While Loop ---");
```

```
        // This loop will not execute because i is not less than 5
        while (i < 5) {
            System.out.println("Value of i: " + i);
            i++;
        }
        System.out.println("While loop finished.");


        System.out.println("\n--- Do-While Loop ---");
        // This loop will execute once because the condition is checked at the end
        do {
            System.out.println("Value of i: " + i);
            i++;
        } while (i < 5);
        System.out.println("Do-while loop finished.");
    }
}
```

Output:

```
--- While Loop ---
While loop finished.

--- Do-While Loop ---
Value of i: 5
Do-while loop finished.
```

MCQs for Practice

Which loop guarantees execution of its body at least once?

a) for loop

b) while loop

c) do-while loop

d) enhanced for loop

Answer: c) do-while loop

A `while` loop is also known as an:

a) Exit-controlled loop

b) Entry-controlled loop

c) Fixed iteration loop

d) Uncontrolled loop
Answer: b) Entry-controlled loop

Unit 3: Arrays and Strings

This unit focuses on handling collections of data using arrays and manipulating text using Java's `String` and `StringBuffer` classes.

Question 1: What is a one-dimensional array? Explain its declaration, instantiation, and initialization with an example. (4 Marks)

Step-by-Step Explanation

An **array** is a data structure that stores a fixed-size collection of elements of the same data type in contiguous memory locations. A **one-dimensional array** organizes these elements in a single row.

1. **Declaration:** This step declares a variable that will hold the array. It defines the variable's name and the type of data it will store, but it doesn't allocate memory yet.
    - **Syntax:** `dataType[] arrayName;` (e.g., `int[] age;`)
2. **Instantiation (Memory Allocation):** This step uses the `new` keyword to create the array object in memory and specifies its size.
    - **Syntax:** `arrayName = new dataType[size];` (e.g., `age = new int[5];`)
3. **Initialization:** This step involves assigning values to the elements of the array. Array indices in Java start from 0.
    - **Syntax:** `arrayName[index] = value;` (e.g., `age[0] = 12;`)

These three steps can be combined into a single line for convenience.

Example Program

Java
```java
public class ArrayExample {
    public static void main(String[] args) {
        // Declaration, Instantiation, and Initialization in one line
        int[] age = {12, 4, 5, 2, 5};

        System.out.println("--- Accessing Array Elements ---");
        // Accessing the first element (at index 0)
        System.out.println("First element: " + age[0]);

        // Using a loop to access all elements
        for (int i = 0; i < age.length; i++) {
            System.out.println("Element at index " + i + ": " + age[i]);
        }
    }
}
```

MCQs for Practice

In Java, array indices start from:

a) 1

b) 0

c) -1

d) It can be defined by the user
Answer: b) 0

Which keyword is used to allocate memory for an array?

a) `alloc` b) `create` c) `new` d) `array`
Answer: c) `new`

Question 2: Differentiate between `String` and `StringBuffer`. (4 Marks)

Step-by-Step Explanation

`String` and `StringBuffer` are both used to work with sequences of characters, but they have a fundamental difference in how they handle memory and mutability.

Feature	`String`	`StringBuffer`
1. Mutability	**Immutable:** Once a `String` object is created, its value cannot be changed. Any modification (like concatenation) creates a *new* `String` object in memory.	**Mutable:** A `StringBuffer` object can be modified after it is created. Methods like `append()` or

		`insert()` change the existing object without creating a new one.
2. Performance	Slower for frequent modifications, as creating new objects for each change consumes more memory and processing time.	Faster for frequent modifications, as it modifies the same object in memory. This is more memory-efficient.
3. Memory Location	Stored in the "String Constant Pool" in the heap, which allows for sharing of identical string literals.	Stored in the heap memory like regular objects.
4. When to Use	Use when the string value will not change, or will change rarely (e.g., for storing names, passwords).	Use when you need to perform many modifications to a string, such as building a long string in a loop.

Export to Sheets

Example

Java
```
// String is immutable
String s = "Hello";
s.concat(" World"); // This creates a new object, but 's' still refers to "Hello"
System.out.println(s); // Output: Hello

// StringBuffer is mutable
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World"); // This modifies the existing object
System.out.println(sb); // Output: Hello World
```

MCQs for Practice

Which of the following is true about the `String` class?

a) It is mutable b) It is immutable

c) It is a primitive data type

d) Its size can be changed
Answer: b) It is immutable

For building a string with many modifications, which class is more efficient?

a) `String`

b) `StringBuffer`

c) `StringArray`

d) `char[]`

Answer: b) `StringBuffer`

Unit 4: Classes and Objects

This unit introduces the core concepts of object-oriented programming in practice, including how to define classes, create objects, and use constructors.

Question 1: What is a constructor? Explain default and parameterized constructors with an example. (4 Marks)

Step-by-Step Explanation

A **constructor** is a special method in a class that is automatically called when an object of that class is created. Its primary purpose is to initialize the object's state (its instance variables).

Key Properties of a Constructor:

- Its name must be the *same* as the class name.
- It does not have a return type, not even `void`.

1. **Default Constructor:**
   - **Definition:** A constructor that takes no arguments (parameters).
   - **Creation:** If you do not define any constructor in your class, the Java compiler automatically provides a default constructor. This constructor initializes instance variables to their default values (e.g., 0 for `int`, `null` for objects).
2. **Parameterized Constructor:**
   - **Definition:** A constructor that accepts one or more arguments.
   - **Purpose:** It allows you to initialize each object with custom values at the time of its creation.

Example Program

```java
class Bike {
   String name;
   int speed;

   // Default Constructor
   Bike() {
       System.out.println("Default constructor called: Bike is created.");
       name = "Default Bike";
   }

   // Parameterized Constructor
   Bike(String n, int s) {
       System.out.println("Parameterized constructor called.");
       name = n;
       speed = s;
   }

   void display() {
       System.out.println("Name: " + name + ", Speed: " + speed + "km/h");
   }

   public static void main(String[] args) {
       // Calls the default constructor
       Bike b1 = new Bike();
       b1.display();

       // Calls the parameterized constructor
       Bike b2 = new Bike("Pulsar", 120);
       b2.display();
   }
}
```

MCQs for Practice

Which statement about constructors is true?

a) A constructor must have a `void` return type.

b) A constructor's name can be different from the class name.

c) A constructor is called automatically when an object is created.

d) A class can have only one constructor.
Answer: c) A constructor is called automatically when an object is created.

What happens if a class has no constructor defined?

a) The program will not compile.

b) The Java compiler provides a default constructor.

c) You cannot create an object of that class.

d) The constructor of the parent class is used.
Answer: b) The Java compiler provides a default constructor.

Question 2: Explain method overloading with a suitable example. (4 Marks)

Step-by-Step Explanation

Method overloading is a feature in Java that allows a class to have more than one method with the same name, as long as their parameter lists are different. This allows you to perform similar operations with different types or numbers of inputs.

How to Overload a Method: You can overload a method by changing:

1. **The number of arguments:** One method can take two arguments, while another with the same name takes three.
2. **The data type of arguments:** One method can take two `int` arguments, while another with the same name takes two `double` arguments.
3. **The sequence of data types of arguments.**

Note: Method overloading cannot be achieved by changing only the return type of the method.

Example Program

Java
```
class Adder {
   // Overloaded method 1: adds two integers
   static int add(int a, int b) {
      return a + b;
   }

   // Overloaded method 2: adds three integers (different number of arguments)
   static int add(int a, int b, int c) {
      return a + b + c;
   }

   // Overloaded method 3: adds two doubles (different data type of arguments)
   static double add(double a, double b) {
      return a + b;
   }

   public static void main(String[] args) {
      System.out.println("Sum of 2 integers: " + Adder.add(11, 11));
```

```
        System.out.println("Sum of 3 integers: " + Adder.add(11, 11, 11));
        System.out.println("Sum of 2 doubles: " + Adder.add(12.5, 12.5));
    }
}
```

In this example, the method `add` is overloaded to handle different types and numbers of inputs, making the code more intuitive.

MCQs for Practice

Which of the following is a valid way to overload a method?

a) Changing the return type only.

b) Changing the method name.

c) Changing the number of arguments.

d) Changing the access modifier.
Answer: c) Changing the number of arguments.

Method overloading is an example of which OOP concept?

a) Inheritance

b) Polymorphism (Compile-time)

c) Encapsulation

d) Abstraction
Answer: b) Polymorphism (Compile-time)