**REG NO: SCT221-0962/2022**

**NAME :Felix Kipyator**

**Part A: Object-Oriented Concepts**

1. **Object Modeling Techniques (OMT):** OMT is a method for object-oriented analysis and design. It provides a notation for representing objects, classes, and their relationships. It includes concepts like objects, classes, and associations.
2. **Comparison of OOAD and OOP:**
   * OOAD (Object-Oriented Analysis and Design) involves analyzing and designing a system using object-oriented concepts.
   * OOP (Object-Oriented Programming) is the actual implementation of a system using object-oriented principles.
3. **Main Goals of UML:**
   * UML (Unified Modeling Language) aims to provide a standardized way to visualize, document, construct, and communicate the design of a system.
   * Goals include fostering understanding, capturing structure and behavior, and promoting best practices in system development.
4. **Advantages of Object-Oriented Development:**
   * **Modularity:** Encapsulation allows for modular design.
   * **Reusability:** Objects and classes can be reused in different parts of the system.
   * **Flexibility and Maintainability:** Changes to one part of the system do not affect others due to encapsulation.
5. **Object-Oriented Programming Terms:**
   * **Constructor Method:** Initializes an object. Example:

public class Example {

public Example() {

// Constructor code

}

}

* + **Object:** An instance of a class. Example:

Example myObject = new Example();

* + **Interface:** Defines a contract for classes. Example:

public interface MyInterface {

void myMethod();

}

* + **Polymorphism:** Enables a method to take different forms. Example:

javaCopy code

public void myMethod(ParentClass obj) {

// Code that works with objects of different subclasses

}

* + **Class:** Blueprint for creating objects. Example:

javaCopy code

public class MyClass {

// Class code

}

1. **Types of Associations in OOP:**
   * **Aggregation:** Represents a "whole-part" relationship where parts can exist independently.
   * **Composition:** Implies a strong ownership relationship where the part cannot exist without the whole.
   * **Association:** Represents a general connection between two classes.
2. **Class Diagram:**
   * A class diagram is a visual representation of classes and their relationships in a system.
   * It is used in the design phase to illustrate the structure of the system.
   * Steps :
     1. Identify classes and their attributes.
     2. Define relationships between classes.
     3. Add methods and other details.
     4. Draw the final diagram.

**Part A: Code Modification (CalculateG class):**

javaCopy code

public class CalculateG {

public static double gravity = -9.81;

public static double fallingTime = 30;

public static double initialVelocity = 0.0;

public static double finalVelocity;

public static double initialPosition = 0.0;

public static double finalPosition;

public static double multi(double a, double b) {

return a \* b;

}

public static void outline(double result) {

System.out.println("Result: " + result);

}

public static void main(String[] args) {

// Formulas for position and velocity

finalPosition = 0.5 \* gravity \* Math.pow(fallingTime, 2) + initialVelocity \* fallingTime + initialPosition;

finalVelocity = gravity \* fallingTime + initialVelocity;

// Output position and velocity

System.out.println("The object's position after " + fallingTime + " seconds is " + finalPosition + " m.");

outline(finalVelocity);

}

}

**Part B: Java Programs**

1. **Fibonacci Sequence:**

javaCopy code

public class Fibonacci {

public static int sumOfEvenFibonacci(int limit) {

int a = 1, b = 2, sum = 0;

while (b <= limit) {

if (b % 2 == 0) {

sum += b;

}

int temp = a + b;

a = b;

b = temp;

}

return sum;

}

}

1. **Array Operations:**

javaCopy code

import java.util.Arrays;

import java.util.Scanner;

public class ArrayOperations {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[] values = new int[15];

// Input values from the user

for (int i = 0; i < values.length; i++) {

System.out.print("Enter value #" + (i + 1) + ": ");

values[i] = scanner.nextInt();

}

// Print values in the array

System.out.println("Values stored in the array: " + Arrays.toString(values));

// Check if a number is present in the array

System.out.print("Enter a number to search: ");

int searchNumber = scanner.nextInt();

int index = -1;

for (int i = 0; i < values.length; i++) {

if (values[i] == searchNumber) {

index = i;

break;

}

}

if (index != -1) {

System.out.println("The number found at index " + index);

} else {

System.out.println("Number not found in this array");

}

// Create a new array in reverse order

int[] reversedArray = new int[values.length];

for (int i = 0; i < values.length; i++) {

reversedArray[i] = values[values.length - 1 - i];

}

// Print reversed array

System.out.println("Reversed array: " + Arrays.toString(reversedArray));

// Calculate and print sum and product

int sum = 0;

long product = 1;

for (int value : values) {

sum += value;

product \*= value;

}

System.out.println("Sum of array elements: " + sum);

System.out.println("Product of array elements: " + product);

}

}