**Examination Answer Book**

**UNIVERSITY EXAMS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REGISTRATION NUMBER | | | | | | | | | VU-DIT-2311-0262-DAY | | | | | | |
| Title of The Program (eg BBA, BSC, BPH, BSWA) | | | | | | | | | | | | | DIT | | |
| Diploma in Information Technology | | | | | | | | | | | | | | | |
| Department | | | | Other Depts in Faculty of Science and Technology | | | | | | | | | | | |
| Faculty | Faculty of Science and Technology | | | | | | | | | | | | | | |
| Year Of study (YrI , YrII, YrIII, or YrIV) | | | | | | | | | | | 2 | | | | |
| Module Code and Name | | | | | | | 1301 ST | | | | | | | | |
| Object Oriented Programming | | | | | | | | | | | | | | | |
| Semester | | | 1 | | | | | | | | | | | | |
| (Copy from the heading to the Examination Paper) | | | | | | | | | | | | | | | |
| Retake: | | Yes | | | ☐ | | | No | | ☐ | | (Tick whichever is applicable) | | | |
| Date of examination | | | | | | Sun Jan 19 2025 09:00:00 GMT+0300 (East Africa Time) | | | | | | | | | |
|  | | | | | |  | | | | | | | | | |
| ***DIRECTIONS TO CANDIDATES (Turn to page ii for more instructions).*** | | | | | | | | | | | | | **FOR USE BY EXAMINERS ONLY** | | |
| **Question Number** | **Internal Examiner** | **External Examiner** |
| 1. Leave margin blank. 2. Begin each answer on a fresh page. 3. Write the number of each question and theCandidate's Number at the top of each page. 4. Write the numbers of the questionswhich you have attempted, with subsections where necessary, in the spacesprovided below | | | | | | | | | | | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **TOTAL** |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NUMBER OF QUESTIONS** you have answered in the order in which you have written them | | | | | | | | |
| 2 | 6 | 5 | 1 |  |  |  |  |  |

**How and where should I submit my examination script?**

Every student will be required to attend their examination via [VClass Students Portal](https://vclass.ac/) E.g. you go to [www.vclass.ac](http://www.vclass.ac) and login, to your account, then on the left sidebar menu **click on Examinations**.

Under examinations you will see the following: -

1. Instructions for that particular examination with time required to finish your examination as per instructions,
2. A student will be required to download the question paper and the answer sheet provided by the university within the same module examination, or a student can be required to attempt structured questions within the system depending on how the examination was set.
3. Submission of answered questions is done,
4. Student is required to click to **consent** to show that the answered exam belongs to them.
5. **Note** that if an examination is for download, a student will be required to download the question paper and answer sheet, write their examination within the given stipulated time.
6. Required to scan and upload back the answered booklet through the same portal as per format available.
7. Examinations uploaded will directly be received by the Registry department.
8. Students here are required to use [VClass e-Learning system](https://vclass.ac)for all examinations and for any failure they can contact the Registry department for guidance.
9. No late submission will be accepted.

**Avoid any examination malpractice because this will attract severe penalties such as invalidating the exams answered script whose consequences will attract retakes.**

**QUESTION TWO**

**a) Distinctions in Object-Oriented Programming:**

(i) **Base Class and Derived Class**

* **Base Class**: A class that provides common properties and methods for derived classes. It is also known as the parent or super class.
* **Derived Class**: A class that inherits properties and methods from a base class. It is also known as a child or subclass.

Example:

class Animal {

void sound() {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Dog barks");

}

}

In this example, Animal is the base class and Dog is the derived class that inherits the sound() method from Animal and overrides it.

(ii) **Global Scope and Local Scope**

* **Global Scope**: Refers to variables or functions that are accessible throughout the entire program, typically defined outside functions or methods.
* **Local Scope**: Refers to variables or functions that are accessible only within the function or block in which they are defined.

Example:

class ScopeExample {

int globalVar = 10;

void localMethod() {

int localVar = 20;

System.out.println(globalVar);

System.out.println(localVar);

}

}

In this example, globalVar has global scope, and localVar has local scope within the localMethod.

(iii) **Source Code and Object Code**

* **Source Code**: The code written by a programmer in a high-level programming language (like Java). It is human-readable.
* **Object Code**: The machine-readable code that results after compiling the source code. It is in binary form and can be executed by a computer.

Example:

* **Source Code**: Java program with .java extension.
* **Object Code**: The compiled .class file that Java Virtual Machine (JVM) can execute.

(iv) **While Loop and Do-While Loop**

* **While Loop**: A loop that checks the condition before executing the loop body. It may not execute even once if the condition is false initially.
* **Do-While Loop**: A loop that checks the condition after executing the loop body. It will always execute at least once.

Example

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

int j = 0;

do {

System.out.println(j);

j++;

} while (j < 5);

In this example, both loops will print numbers from 0 to 4, but the do-while loop ensures that the body executes at least once even if the condition is false initially.

**b) Java Program for Grade Calculation**

import java.util.Scanner;

public class GradeCalculator {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the marks for 6 units:");

int totalMarks = 0;

for (int i = 1; i <= 6; i++) {

System.out.print("Unit " + i + ": ");

totalMarks += sc.nextInt();

}

double average = totalMarks / 6.0;

char grade;

if (average >= 70) {

grade = 'A';

} else if (average >= 60) {

grade = 'B';

} else if (average >= 50) {

grade = 'C';

} else if (average >= 40) {

grade = 'D';

} else {

grade = 'E';

}

System.out.println("Average Marks: " + average);

System.out.println("Grade: " + grade);

sc.close();

}

}

This program allows the user to input the marks for six units, calculates the average, and then assigns a grade based on the average score.

**c) Java Program for Odd or Even Number Check**

import java.util.Scanner;

class Odd {

int x;

void read() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter an integer: ");

x = sc.nextInt();

if (x % 2 == 0) {

System.out.println(x + " is an even number.");

} else {

System.out.println(x + " is an odd number.");

}

sc.close();

}

public static void main(String[] args) {

Odd oddObj = new Odd();

oddObj.read();

}

}

In this program, the class Odd contains a method read() that accepts an integer, checks if it's odd or even using the if statement, and displays the appropriate message.

**QUESTION SIX**

**a) Main Features and Concepts in Object-Oriented Programming**

(i) **Objects**  
An object is an instance of a class. It is a collection of data (attributes or properties) and methods (functions or behaviors) that operate on the data. Objects are the real-world entities modeled in the program.

Example:

class Car {

String color;

void start() {

System.out.println("Car is starting");

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.color = "Red";

myCar.start();

}

}

In this example, myCar is an object of the Car class.

(ii) **Classes**  
A class is a blueprint or template for creating objects. It defines the properties (data members) and behaviors (methods) that the objects created from it will have.

Example:

class Car {

String color;

void start() {

System.out.println("Car is starting");

}

}

In this example, Car is the class, which defines the properties and methods for its objects.

(iii) **Data Abstraction**  
Data abstraction is the concept of hiding the internal workings or implementation details of a class and exposing only the necessary parts to the user. This is achieved using access modifiers like public, private, and protected. It allows users to interact with the object at a higher level without needing to understand its internal implementation.

Example:

class Employee {

private int salary;

public void setSalary(int salary) {

this.salary = salary;

}

public int getSalary() {

return salary;

}

}

Here, salary is hidden from outside access, but can be manipulated using the setSalary() and getSalary() methods.

(iv) **Encapsulation**  
Encapsulation is the process of bundling the data (attributes) and methods (functions) that operate on the data into a single unit called a class. It is also about restricting access to some of the object's components using access control modifiers. This helps protect the internal state of an object from unintended modifications.

Example:

class BankAccount {

private double balance;

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public double getBalance() {

return balance;

}

}

Here, the balance is encapsulated inside the BankAccount class and can only be accessed or modified through the public methods deposit() and getBalance().

(v) **Inheritance**  
Inheritance is a mechanism where a new class (called a subclass or derived class) acquires the properties and behaviors (methods) of an existing class (called a superclass or base class). This allows for code reuse and logical hierarchical structures.

Example:

class Animal {

void eat() {

System.out.println("This animal eats food");

}

}

class Dog extends Animal {

void bark() {

System.out.println("The dog barks");

}

}

In this example, the Dog class inherits the eat() method from the Animal class.

**b) Java Class Student with Specifications**

import java.util.Scanner;

class Student {

int Adm\_no;

String Sname;

float OOP, CALCULUS, SAD;

float Total;

public void calctotal() {

Total = OOP + CALCULUS + SAD;

}

public void getdata() {

Scanner sc = new Scanner(System.in);

System.out.print("Enter Admission Number: ");

Adm\_no = sc.nextInt();

System.out.print("Enter Student Name: ");

sc.nextLine();

Sname = sc.nextLine();

System.out.print("Enter Marks in OOP: ");

OOP = sc.nextFloat();

System.out.print("Enter Marks in CALCULUS: ");

CALCULUS = sc.nextFloat();

System.out.print("Enter Marks in SAD: ");

SAD = sc.nextFloat();

calctotal();

}

public void displaydata() {

System.out.println("\nStudent Information:");

System.out.println("Admission Number: " + Adm\_no);

System.out.println("Name: " + Sname);

System.out.println("Marks in OOP: " + OOP);

System.out.println("Marks in CALCULUS: " + CALCULUS);

System.out.println("Marks in SAD: " + SAD);

System.out.println("Total Marks: " + Total);

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student();

student.getdata();

student.displaydata();

}

}

In this program:

* The Student class contains data members for Adm\_no, Sname, marks in OOP, CALCULUS, SAD, and Total.
* The calctotal() method calculates the total marks.
* The getdata() method accepts data for all the fields and invokes calctotal().
* The displaydata() method displays all the student data.

**c) Explanation of Variable Visibility and Lifetime**

In the given Java program:

1. **Variable x (Local to main() method)**
   * **Visibility**: The variable x is visible only within the **main()** method.
   * **Lifetime**: It lasts only for the execution of the **main()** method. When main() exits, x is no longer accessible.
2. **Variable y (Local to convert() method)**
   * **Visibility**: The variable y is visible only within the **convert()** method.
   * **Lifetime**: It lasts only for the execution of the **convert()** method. Once the method execution is completed, y is destroyed and cannot be accessed again.
3. **Variable z (Static variable)**
   * **Visibility**: The variable z is visible to all methods in the class that defines it. Static variables are shared among all instances of the class.
   * **Lifetime**: The lifetime of a static variable is the entire lifetime of the **program**. It persists even after the method execution is completed and is not tied to any specific instance of the class.
4. **Variable m (Static variable at the class level)**
   * **Visibility**: The variable m is accessible to all methods in the class, including the main() method and the convert() method.
   * **Lifetime**: Similar to z, m also has a lifetime equal to the program's execution. It exists for the entire lifetime of the program and retains its value across method calls and instances.

Thus, in summary:

* Local variables like x and y are limited in both **visibility** and **lifetime** to the methods they are defined in.
* Static variables like z and m are visible to all methods in the class and have a **program-wide lifetime**, persisting for the entire duration of the program’s execution.

**QUESTION FIVE**

### a) ****Difference Between Single Dimensional and Multi-Dimensional Arrays****

**Single Dimensional Array**:

* A single-dimensional array is essentially a list or a collection of elements, where each element is accessed by a single index.
* It is like a row of elements, with one index used to access each element.

Example

int[] prices = {100, 200, 300, 400, 500};

System.out.println(prices[2]); (Access element at index 2 -> 300)

**Multi-Dimensional Array**:

* A multi-dimensional array is an array of arrays. It can be visualized as a table or matrix with rows and columns.
* It can be two-dimensional, three-dimensional, or higher. Typically, the most common is a 2D array (array of arrays).

Example:

int[][] sales = {

{100, 200, 300},

{150, 250, 350},

{200, 300, 400}

};

System.out.println(sales[1][2]);

### b) ****Java Program for Retail Store Inventory and Sales Management****

#### (i) ****Declare and Initialize an Array to Store Product Prices****

Here’s a simple Java program that initializes an array with the prices of five products and prints their prices:

public class Store {

public static void main(String[] args) {

double[] prices = {19.99, 29.99, 49.99, 99.99, 149.99};

// Print the price of each product

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

System.out.println("Price of Product " + (i + 1) + ": $" + prices[i]);

}

}

}

**Explanation**:

* We declare a prices array to store the prices of five products.
* We use a for loop to print the price of each product.

#### (ii) ****Calculate Total Weekly Sales for Each Product Using a 2D Array****

Here is the program that uses a 2D array to store the sales data for each product for seven days (one week), and then calculates and prints the total weekly sales for each product:

public class Store {

public static void main(String[] args) {

int[][] sales = {

{10, 12, 9, 15, 11, 13, 14}, // Sales for Product 1 (7 days)

{8, 9, 7, 6, 8, 7, 9}, // Sales for Product 2

{13, 14, 12, 16, 14, 15, 17}, // Sales for Product 3

{5, 4, 6, 7, 5, 6, 4}, // Sales for Product 4

{3, 2, 4, 3, 3, 2, 5} // Sales for Product 5

};

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

int totalSales = 0;

for (int j = 0; j < sales[i].length; j++) {

totalSales += sales[i][j];

}

System.out.println("Total sales for Product " + (i + 1) + ": " + totalSales);

}

}

}

**Explanation**:

* We use a 2D array sales where each row represents a product, and each column represents the sales for a specific day.
* A nested loop calculates the total sales for each product by summing the sales for all 7 days.
* The result is printed for each product.

### c) ****Re-writing the Program with Comments and Output****

#### (i) ****Rewriting the Program with Comments****

import java.io.IOException;

public class Main {

public static void fact(int z) {

try {

fact2(z); // Call 'fact2' method

System.out.println(15);

} catch (ArithmeticException e) {

System.out.println(20);

} catch (Exception e) {

System.out.println(25);

}

}

public static void fact2(int z) throws IOException {

System.out.println(30);

if (z == 1) {

throw new IOException();

}

if (z == 0) {

throw new ArithmeticException();

}

System.out.println(35); // Print 35 if no exception is thrown

}

public static void main(String[] args) {

fact(1);

fact(2);

}

}

#### (ii) ****What is Printed to the Console by**** fact(1) ****and**** fact(2)

* **Calling fact(1)**:
  + The fact2(1) method is called, printing 30.
  + An IOException is thrown, so the catch (IOException) block handles it, and 20 is printed.
  + The System.out.println(15) in the fact method is not executed because the exception was caught.

**Output for fact(1)**:

30

20

* **Calling fact(2)**:
  + The fact2(2) method is called, printing 30.
  + No exception is thrown, so the program proceeds to print 35.
  + Then, 15 is printed from the fact method.

**Output for fact(2)**:

CopyEdit

30

35

15

### Summary of Output:

* **fact(1) prints:**

30

20

* **fact(2) prints:**

30

35

15

**(d)** **Explanation of the Program's Functionality**:

The program demonstrates the use of an array in Java. Here's the breakdown:

Array Initialization: An integer array marks is declared and initialized with the values {16, 22, 77, 40, 75}.

Printing Array Elements: A for loop iterates through the array, printing each element. The output will be:

16 22 77 40 75

Calculating Total: Another for loop calculates the sum of all elements in the array. The total is stored in the variable total, and the program prints:

Total is 230

Finding Maximum Value: A for loop finds the maximum value in the array by comparing each element with the current maximum (max). Initially, max is set to the first element (marks[0]). The program prints:

Max is 77

Example: Passing an Array as a Parameter in Java

Here’s a simple Java program demonstrating how an array can be passed as a method parameter:

public class ArrayAsParameter {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40, 50};

printArray(numbers);

int sum = calculateSum(numbers);

System.out.println("Sum is: " + sum);

}

public static void printArray(int[] arr) {

System.out.print("Array elements: ");

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

public static int calculateSum(int[] arr) {

int sum = 0;

for (int num : arr) {

sum += num;

}

return sum;

}

}

Output:

Array elements: 10 20 30 40 50 Sum is: 150