**Examination Answer Book**

**UNIVERSITY EXAMS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REGISTRATION NUMBER | | | | | | | | | VU-DIT-2407-2552-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) | | | | | | | | | | | 1 | | | | |
| Module Code and Name | | | | | | | 1301 ST | | | | | | | | |
| Object Oriented Programming | | | | | | | | | | | | | | | |
| Semester | | | 3 | | | | | | | | | | | | |
| (Copy from the heading to the Examination Paper) | | | | | | | | | | | | | | | |
| Retake: | | Yes | | |  | | | No | |  | | (Tick whichever is applicable) | | | |
| Date of examination | | | | | | Sat Jul 05 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 | | | | | | | | |
|  |  |  |  |  |  |  |  |  |

**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.**

**SECTION A**

**Question 1**

a) Five Key Classes, Responsibilities, and Collaborators (5 Marks)

Company; Its responsibility is to manages vehicles, scheduling pickups, and tracking lost fares while the vehicle is assigned for pickup which collaborate with company.

Vehicle: Its responsibility is to represents a transport unit for example Taxi; tracks availability and location and it collaborate with the driver who operate it.

Passenger; is responsible for representing a passenger needing transport, stores pickup and destination locations this initiates the creation of a passenger request.

Passenger Source; This acts as a client making pickup requests to the company and sends requests to schedule pickups.

Location; This stores geographic coordinates like x and y for pickups and drop-off this location class is use to define travel route.

**b)**

**import java.util.Random;**

**public class PassengerSource {**

**private final Company company;**

**private final Random random = new Random();**

**public PassengerSource(Company company) {**

**this.company = company;**

**}**

**public boolean requestPickup() {**

**Location pickup = generateRandomLocation();**

**Location destination = generateRandomLocation();**

**Passenger passenger = new Passenger(pickup, destination);**

**return company.schedulePickup(passenger);**

**}**

**private Location generateRandomLocation() {**

**int x = random.nextInt(101); // 0 to 100**

**int y = random.nextInt(101);**

**return new Location(x, y);**

**}**

**}**

**Explanation of the code,**

**Company has a method boolean schedulePickup(Passenger passenger)**

**Passenger has a constructor Passenger(Location pickup, Location destination)**

**Location has a constructor Location(int x, int y)**

**c)**

**import org.junit.jupiter.api.Test;**

**import static org.junit.jupiter.api.Assertions.\*;**

**public class CompanyTest {**

**@Test**

**public void testSchedulePickupSuccess() {**

**Company company = new Company();**

**Vehicle taxi = new Taxi(4); // Example taxi with 4 seats**

**company.addVehicle(taxi);**

**Passenger passenger = new Passenger(new Location(10, 20), new Location(30, 40));**

**boolean scheduled = company.schedulePickup(passenger);**

**assertTrue(scheduled, "Pickup should be scheduled successfully when a vehicle is available.");**

**}**

**@Test**

**public void testSchedulePickupFailure() {**

**Company company = new Company(); // No vehicles added**

**Passenger passenger = new Passenger(new Location(10, 20), new Location(30, 40));**

**boolean scheduled = company.schedulePickup(passenger);**

**assertFalse(scheduled, "Pickup should fail when no vehicles are available.");**

**}**

**}**

**d)**

Taking Vehicle as the class to explain the application of Encapsulation:  
Encapsulation is applied by keeping internal fields like isAvailable, currentLocation, assignedDriver as private and exposing only public getter methods or business logic methods for example assignPassenger(), markAvailable(), notifyArrival()).

This is benefitial to

* Prevents external code from directly modifying critical state.
* Allows validation for example cannot assign passengers to unavailable vehicles.
* Enhances maintainability and security by hiding implementation details.

**Question 2**

**a)**

In Java, string immutability means that once a String object is created, its contents cannot be changed. Any operation that seems to modify a string (e.g., concatenation or replacement) actually creates a new String object, leaving the original unchanged.

In relevance to NLU’s Library System.

For Security. Book titles and author names should remain constant after being stored. Since strings are immutable, malicious or accidental code cannot alter these critical identifiers once set.

Efficiency: Java uses a string pool, which allows shared use of string literals. Since many book records may have repeated authors or titles, immutability ensures that only one copy of each unique string is stored in memory, reducing memory usage.

1. **Two Ways to Create Empty String Objects in Java**

**Method 1: String Literal**

String title = ""; // *Preferred for performance, uses String pool*

The empty string "" is stored in the **Java String Pool**, which means it's **reused** if needed. In most cases, especially during initialization of string fields like book titles or authors.

**Method 2: Using Constructor**

String author = new String(); *// Explicitly creates a new empty String object*

**Contextual Example (Book Class Initialization):**

public class Book {

private String title;

private String author;

public Book() {

this.title = ""; *// using string literal*

this.author = new String(); // *using constructor*

}

}

1. **Using equalsIgnoreCase() and toLowerCase() in Improving Book Search Functionality**

When users search for books in a library system, they may type the book title or author name in **different letter cases**. For example:

"Ambayo Emmanuel"

"ambayo emmanuel"

"AMBAYO EMMANUEL"

If the system performs **case-sensitive** comparisons , it may **fail to match** records that only differ in letter casing. This can lead to poor user experience.

METHO 1

**1. equalsIgnoreCase() Method**

This method compares two strings while **ignoring case differences**.

if (userInput.equalsIgnoreCase(bookTitle)) {

*// Match found even if cases differ*

}

**Example:**

String userInput = "ambayo emmanuel";

String bookTitle = "Ambayo Emmanuel";

if (userInput.equalsIgnoreCase(bookTitle)) {

System.out.println("Book found!");

}

**Output:** Book found! — even though casing differs.

**METHOD 2. toLowerCase() Method**

This method **converts a string to lowercase**, allowing both strings to be compared in a **normalized** format.

if (userInput.toLowerCase().equals(bookTitle.toLowerCase())) {

// Match found by comparing lowercase versions

}

**Example:**

String userInput = "AMBAYO emmanuel";

String bookTitle = "Ambayo Emmanuel";

if (userInput.toLowerCase().equals(bookTitle.toLowerCase())) {

System.out.println("Book found!");

}

Output: Book found!

These cases are used to

* Improves flexibility and user-friendliness of the search.
* Prevents missed matches due to inconsistent casing.
* Makes the search engine more robust and intuitive.

**d) Analyze Output and Object Comparison (3 Marks)**

String author1 = "Ainebyoona";

String author2 = "ainebyoona";

String author3 = new String("Ainebyoona");

System.out.println(author1 == author3); // false

System.out.println(author1.equalsIgnoreCase(author2)); // true

**Explanation**

compares **object references**, not content. author1 (from pool) ≠ author3 (new object), so result is false.

equalsIgnoreCase() compares **string content** ignoring case. "Ainebyoona" equals "ainebyoona" ignoring case → true.

**e)**

import java.util.Scanner;

public class UgandaCounter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the book description:");

String description = scanner.nextLine();

String lowerCaseDesc = description.toLowerCase();

String keyword = "uganda";

int count = 0;

int index = lowerCaseDesc.indexOf(keyword);

while (index != -1) {

count++;

index = lowerCaseDesc.indexOf(keyword, index + keyword.length());

}

System.out.println("The word 'Uganda' appears " + count + " times.");

}

}

SECTION B.

Question 3

**a) Concept of Java Exception and Exception Handling**

In Java, **exceptions** are **runtime errors** that occur during the execution of a program and disrupt its normal flow. They are objects that describe an **error or an unusual condition** that a program encounters.

**Key Concepts are;**

1. **Exception:** An event that occurs during the execution of a program that disrupts the normal flow of instructions.
2. **Exception Handling:** A mechanism to **gracefully manage** these errors using try, catch, finally, and throw.

**Benefits of Exception Handling:**

* Prevents program crashes.
* Helps in debugging by providing stack traces.
* Allows alternative flows during errors.
* Ensures resources are closed properly e.g file connection.

**b) Exception Thrown When Dividing by Zero**

The exception thrown is **ArithmeticException**

**This is because**

When the Java Virtual Machine (JVM) encounters a **division by zero** in **integer arithmetic** for example int result = 100 / 0, it throws an ArithmeticException. This is a **runtime unchecked exception**, meaning it's not required to be caught or declared.

**Example**

int sum = 500;

int days = 0;

int average = sum / hours; *// Causes ArithmeticException*

**c) Difference between while and for loops in Java**

**while**

Used when the number of iterations is not known inadvance and to checks condition **before** executing the body.

int day = 1;

while (day <= 30) {

System.out.println("Tracking PM2.5 - Day " + day);

day++;

}

**for loop:**

Used when the number of number of iterations is known in advance. Or when the number of iterations are more concise and readable for counting loops.

for (int day = 1; day <= 30; day++) {

System.out.println("Tracking PM2.5 - Day " + day);

}

Both loops above simulate **30 days of PM2.5 tracking by AirQO**.

**d) Java Program for AQI Monitoring**

import java.util.\*;

public class AirQualityMonitor {

public static void main(String[] args) {

Random rand = new Random();

int[] aqiReadings = new int[30];

***// Generating 30 random AQI readings (1 to 300)***

for (int i = 0; i < 30; i++) {

aqiReadings[i] = rand.nextInt(300) + 1;

}

***// Sort readings to compute the median***

Arrays.sort(aqiReadings);

***// Display all readings***

System.out.println("AQI Readings for 30 Days: " + Arrays.toString(aqiReadings));

/***/ Compute median***

double median;

if (aqiReadings.length % 2 == 0) {

median = (aqiReadings[14] + aqiReadings[15]) / 2.0;

} else {

median = aqiReadings[aqiReadings.length / 2];

}

System.out.println("Median AQI: " + median);

**// Count hazardous days (AQI > 200)**

int hazardousDays = 0;

for (int aqi : aqiReadings) {

if (aqi > 200) {

hazardousDays++;

}

}

System.out.println("Number of Hazardous Days (AQI > 200): " + hazardousDays);

}

}

**On run**

AQI Readings: [12, 23, 45, ..., 299]

Median AQI: 147.5

Number of Hazardous Days: 5

**a) Comparison: Constructors vs Garbage Collection (5 Marks)**

**Constructors** in Java and **Destructors** in C++ serve different roles. Java does **not** have destructor instead, it uses **Garbage Collection (GC)** to manage memory.

|  |  |  |
| --- | --- | --- |
| Feature | Constructor | Garbage Collection (Java) / Destructor (C++) |
| Purpose | Initializes a new object | Cleans up resources when object is no longer needed |
| Trigger | Automatically called when object is created | GC runs automatically; destructors manually called in C++ |
| Manual Definition | You define constructors | You **cannot** define GC; you can use finalize() (deprecated) |

**Example:**

Constructor in Java

class Patient {

String id;

Patient(String id) {

this.id = id;

}

}

***Java Garbage Collector manages memory instead of destructors***

Patient p = new Patient("P001"); ***// Constructor runs***

p = null;

System.gc(); // Suggests GC to run (no guarantee

**b)**

**i)**

The keyword is used to distinguish **class fields** from **constructor parameters** when they have the same names.

**Example:**

class Patient {

String patientID;

String allergyNotes;

Patient(String patientID, String allergyNotes) {

this.patientID = patientID**; // Resolves naming conflict**

this.allergyNotes = allergyNotes;

}

}

ii

a)

A static method in Java programming belongs to the class rather than an instance of the class. This means it can be called directly using the class name, without needing to create an object of the class. Static methods are often used for utility or helper methods that don't require access to instance-specific data.

For example,

class HealthUtil {

public static double calculateBMI(double weightKg, double heightM) {

return weightKg / (heightM \* heightM);

}

}

double bmi = HealthUtil.calculateBMI(70, 1.75);

b)

**Method overloading,**

Also known as function overloading, is a feature in object-oriented programming where a class can have multiple methods with the same name but different method signatures (number, types, or order of parameters). This allows the same method name to perform different actions depending on the context of the method call.

For example;

class ReportGenerator {

void generateReport(String patientID) {

System.out.println("Basic Report for " + patientID);

}

void generateReport(String patientID, boolean includeLabResults) {

System.out.println("Detailed Report for " + patientID + ", Labs: " + includeLabResults);

}

}

**c) Difference between final and finally in Java**

|  |  |  |
| --- | --- | --- |
| Keyword | Difference | Example |
| final | Prevents modification | Class, method, or variable can't be changed |
| finally | Executes after try-catch | Always runs for cleanup tasks like closing files |

d)

import javax.swing.\*;

import java.awt.\*;

import java.io.\*;

public class MedicalLogViewer extends JFrame {

private JTextArea textArea;

public MedicalLogViewer() {

setTitle("Medical Log Viewer");

setSize(600, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null); // Center on screen

textArea = new JTextArea();

textArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(textArea);

add(scrollPane, BorderLayout.CENTER);

// Read file content and display it

readAndDisplayFile();

}

private void readAndDisplayFile() {

// Use absolute or working path to avoid confusion

File medicalLogFile = new File("medical\_log.txt");

if (!medicalLogFile.exists()) {

JOptionPane.showMessageDialog(this,

"File not found at: " + medicalLogFile.getAbsolutePath(),

"Missing File", JOptionPane.ERROR\_MESSAGE);

return;

}

try (BufferedReader reader = new BufferedReader(new FileReader(medicalLogFile))) {

StringBuilder content = new StringBuilder();

String line;

while ((line = reader.readLine()) != null) {

content.append(line).append("\n");

}

// Display content

textArea.setText(content.toString());

} catch (IOException e) {

JOptionPane.showMessageDialog(this,

"Error reading file: " + e.getMessage(),

"Read Error", JOptionPane.ERROR\_MESSAGE);

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new MedicalLogViewer().setVisible(true);

});

}

}