N01



|  |  |  |
| --- | --- | --- |
| **Class** | **Primary Responsibility** | **Collaborator Class** |
| Vehicle | Represents taxis and shuttles; holds status, location, and type. | Company |
| Company | Manages scheduling, tracking lost fares, and vehicle coordination. | Passenger Source, Vehicle |
| Passenger | Holds passenger request details like source and destination. | Passenger Source |
| Location | Represents geographical coordinates (x, y). | Passenger, Vehicle |
| Passenger Source | Initiates pickup requests based on external triggers. | Company, Passenger |

D)

**Encapsulation** is the practice of hiding the internal state and requiring all interaction to be performed through an object’s methods.

Example using the **Vehicle class**:

**public class Vehicle {**

**private boolean isAvailable;**

**public boolean isAvailable() {**

**return isAvailable;**

**}**

**public void setAvailable(boolean available) {**

**this.isAvailable = available;**

**}**

**}**

N02



**String immutability** means once a String object is created, its value cannot be changed.

**Benefits in NLU System:**

* **Security**: Prevents unauthorized modifications of book titles/author names.
* **Efficiency**: Java uses **String pooling** to reuse immutable string instances, reducing memory usage.
* **Thread Safety**: Immutable strings are inherently thread-safe and ideal in concurrent environments.



// Method 1: Using String Literal

**String empty1 = "";**

// Method 2: Using String Constructor

**String empty2 = new String();**

**For context in book record initialization:**

**Book book = new Book();**

**book.setTitle(""); // initialize with empty string**

**book.setAuthor(new String()); // alternative constructor form**



we can use equalsIgnoreCase() and toLowerCase() in the search :

**String inputTitle = "THINGS FALL APART";**

**String recordTitle = "Things Fall Apart";**

// Using equalsIgnoreCase

**if (inputTitle.equalsIgnoreCase(recordTitle)) {**

**System.out.println("Match found!");**

**}**

// Using toLowerCase

**if (inputTitle.toLowerCase().equals(recordTitle.toLowerCase())) {**

**System.out.println("Match found!");**

**}**



**String author1 = "Ainebyoona";**

**String author2 = "ainebyoona";**

**String author3 = new String("Ainebyoona");**

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

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

== compares **references**, not values.

equalsIgnoreCase() compares string **values** ignoring case.

N03



**Exception**: An error that occurs during runtime.

**Exception Handling**: Mechanism to catch and manage exceptions gracefully using try, catch, and finally.

**try {**

**int aqi = 300 / 0;**

**} catch (ArithmeticException e) {**

**System.out.println("Cannot divide by zero!");**

**}**



**Exception Thrown**: ArithmeticException

**Cause**: Occurs when dividing AQI sum by 0 (e.g., no days recorded)

**Reason**: Division by zero is mathematically undefined and illegal in Java.



Both loops iterate 30 times but differ in syntax and flexibility as shown in the code:

// Using while loop

**int day = 1;**

**while (day <= 30) {**

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

**day++;**

**}**

// Using for loop

**for (int i = 1; i <= 30; i++) {**

**System.out.println("Tracking PM2.5 for Day " + i);**

**}**

N04



|  |  |  |
| --- | --- | --- |
| **Feature** | **Constructor** | **Garbage Collector (GC)** |
| Purpose | Initializes object | Frees memory of unused objects |
| Called when | Object is created | Object is unreachable |
| Manual/Auto | Programmer defines | JVM handles automatically |

For example;

**class Patient {**

**Patient() {**

**System.out.println("New Patient Registered.");**

**}**

**}**

**// GC: No destructor needed; JVM cleans up unused objects**







**class Patient {**

**private String id;**

**private String allergy;**

**public Patient(String id, String allergy) {**

**this.id = id; // Resolves variable conflict**

**this.allergy = allergy;**

**}**

**}**



static method

**public class HealthUtil {**

**public static double calculateBMI(double weight, double height) {**

**return weight / (height \* height);**

**}**

**}**



Method Overloading

**public class Report {**

**public void printReport(String patientName) { }**

**public void printReport(String patientName, String diagnosis) { }**

**}**



final prevents modification.

finally ensures execution of cleanup code.

**final class Constants {**

**public static final double MAX\_TEMP = 42.0;**

**}**

**// finally block**

**try {**

**// file read**

**} catch (IOException e) {**

**e.printStackTrace();**

**} finally {**

**file.close(); // Always runs**

**}**