TRENT UNIVERSITY

COIS 2240H – Software Design & Modelling

Assignment 1 — Object Oriented Programming in Java

Total Points: 100 – Due February 9th, 2025

Note: The assignment should be completed through individual efforts (i.e., group submissions are not accepted). Any issues related to academic integrity, such as plagiarism, copying from another student, or unauthorized use of generative AI tools will be handled in accordance with the university's academic integrity policies.

Objectives:

Develop a simple software system that follows Object-Oriented Programming (OOP) principles in Java.

Introduction:

For this assignment, you will demonstrate your understanding of object-oriented principles in Java. Specifically, you are expected to:

1. Design a software system (Vehicle Rental System) to manage the interactions required for vehicle rentals, following OOP principles.
2. Develop a console interactive app with a menu to allow users to interact with the system.

This assignment is designed to evaluate your ability to use proper software design practices with practical Java implementation, focusing on object-oriented principles. To successfully complete this assessment, you will need to undertake the following tasks.

Deadline: 10% will be deducted for every day for late submission, and no more than 5 days late. If there is a late second attempt, the submission will be considered late.

Task 1: System Structure (70 Points)

Objective:

Develop a Java system with five classes and one interface, to manage vehicle rentals (cars and motorcycles) using OOP principles. The system must handle adding vehicles, renting them out, returning them, and displaying their availability. **Note** that you have to write each class and interface in a separate *.java* file. In this assignment, you should have **6** Java files: 5 for the classes (four in Task 1 and one in Task 2), and one for the interface (in Task 1).

Class Requirements

1. **Vehicle** class (abstract):
   - **Attributes** (all private):
     - licensePlate (String): Unique identifier (e.g., *CAR-001 or MOTO-999*) – all characters in uppercase
     - make (String): Manufacturer (e.g., *Toyota*) – First character in uppercase, and remaining characters in lowercase
     - model (String): Model name (e.g., *Camry*) – First character in uppercase, and remaining characters in lowercase
     - year (int): Year of manufacturing (e.g., *2025*)
     - status (enum): The current status of a vehicle (*Available*, *Reserved*, *Rented*, *Maintenance*, *OutOfService*)

- **Constructors** (public):
  - Another constructor is to initialize all attributes, where make, model, and year are received as parameters, whereas the status is initialized as *Available* and the licensePlate is initialized as an empty string. The required formatting for make and model must be set in this constructor
  - A default constructor that calls the above constructor with proper default values
- **Methods** (all public):
  - **Setters** (for licensePlate and status) **and Getters** (for all attributes)**:** Example setter and getter for licensePlate, which needs to convert the input to uppercase

```java
public void setLicensePlate(String plate) {
     // Convert to uppercase here
     this.licensePlate = plate;
}

public String getLicensePlate() {
     return licensePlate;
}
```

  - **Method** (public):
    - String getInfo(): Returns all vehicle details, with the below header columns, separated by \t instead of a whitespace to make out organized
      | License Plate | Make | Model | Year | Status |

2. **Concrete Subclasses** (inherit the **Vehicle** class)
   - **Car** class:
     - **Additional attributes** (private):
       - numSeats (int): Number of seats in the car (must be greater than zero)
     - **Constructor** (public):
       - Initializes all attributes and passes values of the inherited attributes to the superclass
     - **Method** (public):
       - Override getInfo() to get the details of the parent class and extend them to include the number of seats
   - **Motorcycle** class**:**
     - **Additional attributes** (private):
       - hasSidecar (boolean): Indicates if the motorcycle has a sidecar
     - **Constructor** (public):
       - Initializes all attributes, and passes values to inherited attributes to the super class
     - **Method** (public):
       - Override getInfo() to get the details of the parent class and extend them to include the sidecar status
3. **RentalSystem** class **(**Association**)**
   - **Attributes** (private):
     - vehicles: an *ArrayList* of Vehicle objects
     - rentalRecords: a *HashMap* for preserving vehicles and customer names

```java
// Import the below classes using separate import statements
private List<Vehicle> vehicles = new ArrayList<>();
private Map<String, String> rentalRecords = new HashMap<>();
```

- **Methods** (public):
  - boolean addVehicle(Vehicle *vehicle*): Adds a vehicle to the list
  - boolean rentVehicle(String *licensePlate*, String *customerName*): Checks if the car is *Available*, adds it the hashmap, and marks a vehicle as *Rented* if so. It also converts the first letter of customerName to uppercase and remaining letters to lowercase
  - boolean returnVehicle(String *licensePlate*): Checks if the car is *Rented*, marks the specified vehicle as *Available*, and removes it from the hashmap if so
  - void displayAvailableVehicles(): Shows details of *Available* vehicles only, showing the following columns as a header, followed by the details of the vehicles. Use \t to separate between the columns and values to make the output organized

```
| Type |    Plate    | Make | Model | Year |
-----------------------------------------------------------------
|  CAR | CAR-001 | Toyota | Camry | 2025 |
```

4. **Rentable** interface (Polymorphism)
   - **Methods**:

```java
void rentVehicle();
void returnVehicle();
```

   - **Implementation:**
     - The concrete subclasses (Car and Motorcycle) implement the Rentable interface
     - Override the interface methods to update the status and print messages (e.g., "*Motorcycle MOTO-999 has been rented.*")

Task 2: User Interaction (30 Points)

Objective:

Develop a **VehicleRentalApp** class that implements a console application, in which a main method shows a menu to allow users to interact with the Rental System. The menu options are as follows:

1: Add Vehicle
2: Rent Vehicle
3: Return Vehicle
4: Display Available Vehicles
5: Exit

Functionality Requirements

- **1: Add Vehicle:**
  - Prompt the user (using ***Scanner***) to select the vehicle type:
    - **1:** Car
    - **2:** Motorcycle
  - Based on the user selection, prompt the user to enter the necessary attributes:
    - For **Car**: licensePlate, make, model, year, numSeats
    - For **Motorcycle**: licensePlate, make, model, year, hasSidecar
  - The make, model, year, numSeats, hasSidecar must be passed through the constructor
  - The licensePlate must be set using the setter method, not through the constructor
  - Objects should be declared as Vehicle, then instantiated using the selected vehicle type, and then added to the *ArrayList* in RentalSystem
  - Show a message indicating that the selected vehicle has been added successfully
- **2: Rent Vehicle:**

  - Prompt for licensePlate and customerName
  - The renting process should be done through RentalSystem
  - Display a message for whether renting is successful or not, depending on the feedback received from RentalSystem
- **3: Return Vehicle:**
  - Prompt for licensePlate
  - Check if the vehicle is currently *Rented*
  - The returning process should be done through RentalSystem
  - Display a message for whether returning is successful or not, depending on the feedback received from RentalSystem
- **4: Display Available Vehicles**
  - Call a dedicated RentalSystem method showing a list of available vehicles
- **5: Exit**
  - Terminate the application: System.exit(0);

OOP principles to be followed

- **Encapsulation**
  - All class attributes should be private
  - All class methods should be public
  - Use setters and getters to access attributes
- **Abstraction & Inheritance:**
  - Vehicle is an abstract class
  - Subclasses Car and Motorcycle inherit Vehicle and implement specific details
  - Rentable interface enforces rental behavior, and is implemented by Car and Motorcycle
  - Proper use of *this* and *super* keywords
- **Polymorphism:**
  - getInfo() method should be overridden in subclasses to provide specific information
  - rentVehicle() and returnVehicle() methods should also be overridden in subclasses
  - The Vehicle constructor should be overloaded
- **Composition:**
  - RentalSystem class contains a list of Vehicle objects and manages their interactions

---

Submission

- Submit the **six** Java source files on Blackboard, containing the five classes and one interface you have developed for the system. Ensure that your code follows all the instructions above and is well-structured. Comments (including Javadocs) are strongly recommended.
- Do not *zip* your source files.
- Make sure you submit .java files, NOT .class files.
- Ensure your code does not use any extra, unnecessary packages, classes, interfaces, variables, or methods not mentioned in the instructions or unrelated to the system described.
- Your code will be tested using a separate script and manual inspection. Ensure your classes integrate correctly with each other and follow directions as instructed.
- Submit your work even if your code is not working, to allow grading your work based on the structure and logic.