Applied Programming and Design Principles

BTEC HND| Software Engineering

Department of Computing

Faculty of Computing and Technology

Saegis Campus

Nugegoda

Instructions:

- Complete all sections of the provided Word template. Areas highlighted in red are placeholders that need to be filled by you.
- Ensure that all Java code snippets are included in the Word document. Capture and insert screenshots of the output as required by the tasks.
- Once you have completed all tasks, save your document as a PDF file.
- Submit the PDF file by **15th October 2024**.

Assignment

Question 01:

You have been hired by a company to develop a Vehicle Management System for their fleet of vehicles. The system needs to manage different types of vehicles like cars, motorcycles, and trucks each with its own specific characteristics and behaviors. The company wants the system to be flexible enough to accommodate new types of vehicles in the future. You have to design and implement a Vehicle Management System based on the following requirements:

- The company requires that every vehicle in the system should be able to start and stop its engine. You need to ensure that this behavior is consistent across all vehicle types.
- All vehicles in the company's fleet share common attributes such as **make**, **model**, and **year**. The system should manage these attributes efficiently, ensuring that they are securely encapsulated.
- The company operates various types of vehicles, including:
  - **Cars**: Each car has a specific number of doors.
  - **Motorcycles**: Each motorcycle has an engine capacity measured in cubic centimeters (cc).
  - **Trucks**: Each truck has a payload capacity indicating how much weight it can carry.

        - These vehicle types should exhibit unique behaviors when starting and stopping their engines.
- The company wants to store and manage all vehicles in a centralized system. The system should be able to handle different types of vehicles seamlessly, allowing for easy retrieval and manipulation of vehicle information. The company needs the system to demonstrate flexibility by showing that it can handle different types of vehicles using the same interface, showcasing the system's capability to support polymorphism.

- **Tasks:**
    1. Design an interface that defines the operations common to all vehicles.
    2. Develop an abstract class that manages the common attributes of all vehicles and ensures that these attributes are encapsulated.
    3. Implement the specific vehicle types (Car, Motorcycle, Truck), ensuring that each class handles its specific attributes and behaviors appropriately.
    4. Design the main class that will instantiate and manage the various vehicles. This class should demonstrate how different types of vehicles can be handled in a uniform way.

- Draw a class diagram representing the structure of your Vehicle Management System, showing the relationships between Vehicle, AbstractVehicle, Car, Motorcycle, Truck, and VehicleManagementSystem.
- Write a brief explanation of how each OOP concept is applied in your Vehicle Management System.
- Submit the complete Java code for the Vehicle Management System with screenshots of the outcomes.

Question 02

Below are five Java code snippets. Each snippet violates one or more SOLID principles. Your task is to identify the specific SOLID principle(s) that are not followed in each snippet. After identifying the issue, discuss why the code violates the principle and suggest how the code can be refactored to adhere to the appropriate SOLID principle(s).

Code Snippet 1

```
class Invoice {
    private double amount;
    private String customer;
    public Invoice(double amount, String customer) {
        this.amount = amount;
        this.customer = customer;
    }
    public void printInvoice() {
        // Logic to print the invoice
        System.out.println("Invoice details: " + amount + " for " + customer);
    }

    public void saveToDatabase() {
        // Logic to save the invoice to a database
        System.out.println("Invoice saved to database.");
    }
}
```

Code Snippet 2

```
class Bird {
    public void fly() {
        System.out.println("Flying...");
    }
}

class Ostrich extends Bird {
    @Override
    public void fly() {
        System.out.println("Ostriches can't fly.");
    }
}
```

Code Snippet 3

```
class Order {
    private String item;
    private int quantity;

    public Order(String item, int quantity) {
        this.item = item;
        this.quantity = quantity;
    }

    public int calculateTotal() {
        // Hardcoded price calculation
        int price = 100;
        return price * quantity;
    }
}
```

Code Snippet 4

```
class Rectangle {
    protected int width;
    protected int height;

    public void setWidth(int width) {
        this.width = width;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getArea() {
        return width * height;
    }
}

class Square extends Rectangle {
    @Override
    public void setWidth(int width) {
        this.width = width;
        this.height = width; // A square's width and height must be the same
    }

    @Override
    public void setHeight(int height) {
        this.width = height;
        this.height = height; // A square's width and height must be the same
    }
}
```

Code Snippet 5

```
class User {
    private String username;
    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public void login() {
        // Logic to log in the user
        System.out.println("User logged in.");
    }

    public void sendEmail() {
        // Logic to send an email
        System.out.println("Email sent to user.");
    }
}
```

- For each code snippet, identify which SOLID principle is violated.
- Explain why the code violates the identified principle.
- Propose a refactored solution or suggest a suitable SOLID principle that should be applied to fix the issue.
- Provide your analysis for each code snippet in a document. Suggest code modifications or refactored versions where necessary. Ensure your explanations are clear and concise.

Question 03

Below are four scenarios. For each scenario, apply the appropriate design pattern (Singleton, Factory Method, Adapter, or Proxy) and describe how the pattern can be used to solve the problem. Explain your suggestions and how the pattern improves the design.

1. You are developing a logging service for an application. The logging service should ensure that there is only one instance of the logger throughout the application to avoid multiple log files and inconsistent logging behavior. Apply the Singleton pattern to ensure that the logging service has only one instance. Describe how the Singleton pattern ensures that only one instance of the logger is created and how it provides a global point of access to that instance.

2. You are developing a notification system that needs to send different types of notifications, such as email, SMS, and push notifications. The system should be flexible enough to handle new types of notifications in the future. Apply the Factory Method pattern to create different types of notification objects. Describe how the Factory Method pattern allows the system to create instances of various notification types without knowing the exact class of the object that will be created.

3. You have a third-party library for payment processing that uses a different interface than the one used by your application. You need to integrate this library into your application without modifying the existing code that depends on your application's interface. Apply the Adapter pattern to adapt the third-party payment processing library to your application's interface. Describe how the Adapter pattern enables your code to work with the third-party library while keeping the existing codebase unchanged.

4. You are developing an image viewer application that loads images from the network. To improve performance and reduce network usage, you want to implement a mechanism that delays the actual image loading until it is needed. Apply the Proxy pattern to manage the image loading process. Describe how the Proxy pattern can be used to provide a

placeholder for the image and only load the actual image from the network when it is required.

- For each scenario, you should use the design pattern that best fits the scenario.
- Explain how the design pattern solves the problem and improves the design.

Offer implementation suggestions and describe the benefits of using the pattern in the given scenario.

Question 04

1. You are tasked with designing a customer support ticketing system for a tech company. Tickets are submitted by customers and need to be handled in the order they are received. This ensures fairness and systematic processing of requests.
    - Design a Queue to manage customer support tickets. Each ticket has a unique ID, customer name, and issue description.
    - Implement the following functionalities:
        - Add a new ticket to the end of the queue.
        - Remove and handle the ticket at the front of the queue. Mark the ticket as "processed" once handled.
        - Display all tickets currently in the queue, showing their ID, customer name, and issue description.
   - Write a Java program to simulate this ticket system. Create a Ticket class with fields for ticketID, customerName, and issueDescription. Use Java's LinkedList to implement the queue.
   - Example Scenario:
        - Customer A submits a ticket: ID=101, Name=John Doe, Issue="Login problem".
        - Customer B submits a ticket: ID=102, Name=Jane Smith, Issue="Payment issue".
        - Customer C submits a ticket: ID=103, Name=Emily Davis, Issue="Shipping delay".
        - Handle tickets in the order they were received, starting with ID=101.
   - Expected Output:
        - When submitting tickets, they should appear in the queue in the order they were received.
        - When handling tickets, they should be removed from the front of the queue.
        - Display the list of pending tickets at any point, showing remaining tickets in the queue.
2. You are developing an emergency task handling system for a hospital. Urgent tasks that require immediate attention are pushed onto a stack. The most recent urgent task is handled first, ensuring that the latest emergency is addressed before others.

- Design a Stack to manage urgent tasks. Each task has a unique ID, task name, and priority level.
- Implement the following functionalities:
  - Push a new task onto the stack with its details.
  - Pop and complete the most recent task from the stack. Mark the task as "completed" once handled.
  - Display all tasks currently in the stack, showing their ID, task name, and priority level.
- Example Implementation
  - Write a Java program to simulate this task management system.
  - Create a Task class with fields for taskID, taskName, and priorityLevel.
  - Use Java’s Stack class to implement the stack.
- Example Scenario
  - An urgent task with ID=201, Name="Heart surgery", Priority=1 is added.
  - An urgent task with ID=202, Name="Emergency room cleanup", Priority=2 is added.
  - An urgent task with ID=203, Name="Critical lab results", Priority=3 is added.
  - Complete tasks starting with the most recent, which is ID=203.
- Expected Output:
  - When adding tasks, they should be pushed onto the top of the stack.
  - When completing tasks, they should be popped from the top of the stack.
  - Display the list of pending tasks at any point, showing remaining tasks in the stack.

Question 05

You are working on a software development project and need to focus on automatic testing for a simple Java application. Your task involves examining various methods of implementing automatic testing, implementing tests on a sample Java application, comparing developer-produced and vendor-provided testing tools, and analyzing the benefits and drawbacks of automatic testing.

1. Research and describe various methods for implementing automatic testing as outlined in a test plan. Include details about unit testing, integration testing, and basic system testing. Explain the advantages and potential limitations of each method.

2. Implement automatic testing for a simple Java application. Provide Java code examples and corresponding test cases.

3. Compare developer-produced automatic testing tools (e.g., JUnit) with vendor-provided automatic testing tools (e.g., Selenium). Discuss their functionalities, ease of use, and integration capabilities.

4. Analyze the benefits and drawbacks of different forms of automatic testing using examples from the simple Java application. Discuss aspects such as test coverage, reliability, and maintenance.