EASTERN INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING

AND INFORMATION TECHNOLOGY

Practice Assignment – Quarter 1, 2024-2025

Course Name: OOD

Course Code: CSE 203

Student's Full Name:

Student ID:

Lab 7

1. Singleton Pattern

Objective:

- Implement a Singleton class to ensure only one instance of a resource exists.

Task:

1. Create a `Logger` class:
   - Ensure it has a private constructor.
   - Use a static method `getInstance()` to return the single instance.
   - Add a method `log(String message)` to print messages with timestamps.
2. Write a main program:
   - Retrieve the `Logger` instance multiple times and log different messages.
   - Verify that all messages are logged using the same instance.

Expected Output:

- Logs should show timestamps and be consistent across the application, ensuring one instance of `Logger`.

Hint: *the log method could be:*

```
// Method to log messages with a timestamp
    public void log(String message) {
        String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-
MM-dd HH:mm:ss"));
        System.out.println("[" + timestamp + "] " + message);
    }
```

2. Factory Pattern

Objective:

- Design a Factory to create different types of objects dynamically.

Task:

1. Create an interface `Shape` with a method `draw()`.
2. Implement `Circle`, `Rectangle`, and `Square` classes that implement `Shape`.

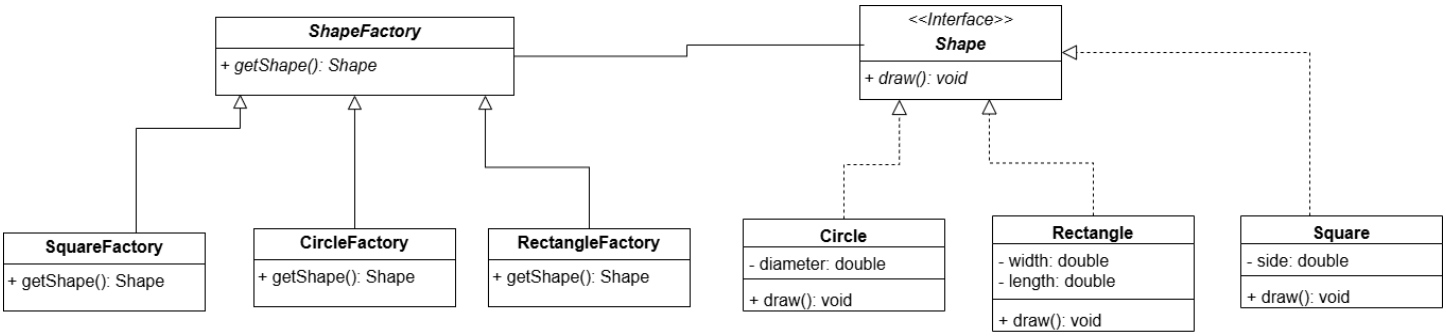
3. Implement a `ShapeFactory` class with an abstract method `getShape()`:
    - Return an instance of a shape.
4. Implement `CircleFactory`, `RectangleFactory`, and `SquareFactory` classes that implement `ShapeFactory` and return appropriate shape objects in the `getShape` method.
5. Write a main program:
    - Use classes to create shapes based on user need and call their `draw()` methods.

Expected Output:

Different shapes should display messages like:

- `"Drawing a Circle"`
- `"Drawing a Rectangle"`

***Hint**: Students can use the following UML:*



3. Decorator Pattern

Objective:

- Extend the functionality of an object dynamically.

Task:

1. Create an interface `Coffee` with a method `getCost()` and `getDescription()`.
2. Implement a basic `SimpleCoffee` class.
3. Create decorators like `MilkDecorator`, `SugarDecorator`, and `WhippedCreamDecorator`:
    - Each decorator adds a specific feature and cost to the coffee.
4. Write a main program:
    - Start with `SimpleCoffee` and dynamically add milk, sugar, or whipped cream.
    - Print the final description and cost.

Expected Output:

For example, ordering coffee with milk and sugar should output:

- `"Description: Simple Coffee + Milk + Sugar"`
- `"Cost: $4.50"`

4. Strategy Pattern

Objective:

- Implement a family of algorithms and make them interchangeable.

Task:

1. Create a `ShoppingItem` class which has fields to store description and cost of shopping items
2. Create an interface `PaymentStrategy` with a method `pay(double amount)`.
3. Implement strategies like `CreditCardPayment`, `PaypalPayment`, and `CashPayment`.
4. Create a `ShoppingCart` class:
    - Hold a list of items and calculate the total price.
    - Hold and accept a `PaymentStrategy` object to process the payment.
5. Write a main program:
    - Add items to the cart and choose different payment strategies to pay.

Expected Output:

For example:

- `"Total amount: $100"`
- `"Paid using Credit Card"`

5. Observer Pattern

Objective:

- Create a publish-subscribe mechanism to notify observers about changes.

Task:

1. Create an interface `Observer` with a method `update(String message)`.
2. Create a `Subject` class:
    - Maintain a list of observers.
    - Methods: `addObserver(Observer observer)`, `removeObserver(Observer observer)`, and `notifyObservers(String message)`.
3. Implement concrete observers like `EmailSubscriber` and `SMSSubscriber`.
4. Create a `NewsAgency` class extending the `Subject` class
5. Write a main program:
    - Simulate a `NewsAgency` that publishes news and notifies all subscribers.

Expected Output:

When the `NewsAgency` publishes news, observers should receive it:

- `"Email Subscriber: Breaking News!"`
- `"SMS Subscriber: Breaking News!"`

Submit: Must include:

- a pdf file containing your information (student id, name), and images of the class diagrams, codes, answers
- and all source code files (if any)

in a zipped (.zip or *.rar) file to Moodle