Daily Coding Assignment Wipro NGA .NET Cohort

Coding Assignment: SOLID Principles and Design Patterns in .NET

Duration: 45 minutes each

First Assignment: Applying SOLID Principles

Objective: Refactor a given codebase to adhere to SOLID principles and ensure it is well-structured and maintainable.

Problem Statement

You have been provided with a basic implementation of a reporting system. The current code does not adhere to SOLID principles, making it difficult to maintain and extend. Your task is to refactor the code to follow SOLID principles.

User Stories and Expectations

User Story 1: Implement Single Responsibility Principle (SRP)

- **As a developer**, you need to refactor the existing code to ensure that each class has a single responsibility.
- **Acceptance Criteria**:
    - Refactor the `ReportManager` class, which currently handles both report generation and file saving, into separate classes.
    - Create a `ReportGenerator` class responsible for generating reports and a `ReportSaver` class responsible for saving reports.

User Story 2: Apply Open/Closed Principle (OCP)

- **As a developer**, you need to refactor the code to ensure that it is open for extension but closed for modification.
- **Acceptance Criteria**:
    - Refactor the `ReportFormatter` class to use an interface for formatting. Implement different formatting strategies (e.g., PDF, Excel) without modifying the existing `ReportFormatter` code.

User Story 3: Implement Liskov Substitution Principle (LSP)

- **As a developer**, you need to ensure that subclasses can be substituted for their base classes without affecting the correctness of the program.

- **Acceptance Criteria**:
    - Refactor the `Report` class hierarchy to ensure that derived classes correctly override base class methods and can be used interchangeably with the base class.

User Story 4: Apply Interface Segregation Principle (ISP)

- **As a developer**, you need to ensure that clients are not forced to depend on interfaces they do not use.
- **Acceptance Criteria**:
    - Refactor the `IReport` interface to split it into smaller, more focused interfaces. Ensure that classes implementing the interfaces only need to implement methods that are relevant to them.

User Story 5: Implement Dependency Inversion Principle (DIP)

- **As a developer**, you need to refactor the code to ensure that high-level modules are not dependent on low-level modules, but both depend on abstractions.
- **Acceptance Criteria**:
    - Refactor the `ReportService` class to depend on abstractions rather than concrete implementations. Use dependency injection to provide the necessary dependencies.

Challenging Assignment Outcome

By completing this assignment, you should have a well-structured, maintainable, and extensible codebase that adheres to SOLID principles. Your code should be easier to test, extend, and maintain due to the separation of concerns and adherence to best practices.

Best Practices for Submission:

1. **Code Organization**: Ensure that the refactored code is organized into meaningful classes and interfaces.
2. **Documentation**: Provide clear comments explaining the purpose of each class and method, especially where refactoring was done.
3. **Testing**: Include unit tests for the refactored code to verify that it meets the desired functionality and adheres to SOLID principles.
4. **Commit History**: Use meaningful commit messages to describe the changes made during refactoring.

Daily Coding Assignment Wipro NGA .NET Cohort

Second Assignment: Design Patterns Implementation

Objective: Implement a basic example using a common design pattern to demonstrate its application in .NET.

Problem Statement

You are required to implement a simple example using a common design pattern. Choose one of the following patterns: Singleton, Factory, or Observer. Implement the pattern in a .NET application and demonstrate its usage.

User Stories and Expectations

User Story 1: Implement a Singleton Pattern

- **As a developer**, you need to implement the Singleton pattern to ensure that a class has only one instance and provides a global point of access to it.
- **Acceptance Criteria**:
  - Implement a `Logger` class as a Singleton. Ensure that only one instance of `Logger` is created and used throughout the application.
  - Demonstrate the use of this `Logger` class in a sample application.

User Story 2: Implement a Factory Pattern

- **As a developer**, you need to implement the Factory pattern to create objects without specifying the exact class of object that will be created.
- **Acceptance Criteria**:
  - Implement a `DocumentFactory` class that creates different types of documents (e.g., `PDFDocument`, `WordDocument`).
  - Ensure that the factory method returns the correct type of document based on input parameters.

User Story 3: Implement an Observer Pattern

- **As a developer**, you need to implement the Observer pattern to allow objects to notify other objects about changes in their state.
- **Acceptance Criteria**:
  - Implement a `WeatherStation` class that notifies observers (e.g., `WeatherDisplay`) when weather data changes.
  - Ensure that observers can register, unregister, and receive updates from the `WeatherStation`.

Daily Coding Assignment Wipro NGA .NET Cohort

Easy Assignment Outcome

By completing this assignment, you should be able to demonstrate the practical application of a common design pattern in .NET. Your implementation should illustrate the pattern's benefits and usage in a real-world scenario.

Best Practices for Submission:

1. **Code Clarity**: Ensure that the pattern implementation is clean and easy to understand.
2. **Documentation**: Include comments explaining the pattern and its implementation.
3. **Examples**: Provide a brief example or test code demonstrating how the pattern is used.
4. **Code Review**: Ensure the code adheres to standard coding practices and is free of unnecessary complexity.

These assignments will test your understanding of SOLID principles and design patterns while providing practical, real-world scenarios for you to apply your knowledge.