UGANDA CHRISTIAN UNIVERSITY

A Center of Excellence in the Heart of Africa

FACULTY OF ENGINEERING DESIGN AND TECHNOLOGY

NAME: WILSON NYUMBE

REG NO: S21B23/038

ACCESS N0: A89145

COURSE: BSCS

COURSE UNIT: CYBER THREAT INTELLIGENCE

LECTURER: Mr. Simon Lubambo

Code Review and Documentation

Upon analyzing the updated code for the application, we observe solid adherence to the SOLID design principles. Here's how the code respects each principle:

Single Responsibility Principle (SRP)

- **Compliance:** Each class has a single responsibility. Employee serves as a base for staff-specific behavior, Manager and Developer implement role-specific logic, ReportGenerator handles reporting, and BonusCalculator manages reward calculation. This separation of concerns exemplifies SRP.

Open/Closed Principle (OCP)

- **Compliance:** The system is designed for extension without modification. New staff roles can be added by simply extending Employee and implementing its abstract methods without altering existing classes, thus adhering to OCP.

Liskov Substitution Principle (LSP)

- **Compliance:** The design ensures that subclasses of Employee can be used interchangeably without affecting the system's behavior, meeting LSP criteria. ReportGenerator and BonusCalculator can operate on any subclass of Employee without needing to know the specific type.

Interface Segregation Principle (ISP)

- **Compliance:** The Employee abstract class mandates only those methods (compute reward and execute tasks) that are necessary for the subclasses, thus adhering to ISP by not forcing subclasses to implement irrelevant methods.

Dependency Inversion Principle (DIP)

- **Compliance:** High-level modules (ReportGenerator and BonusCalculator) depend on the abstract class Employee rather than concrete implementations (Manager, Developer), aligning with DIP.

Refactoring Plan

Given the code's alignment with SOLID principles, the refactoring plan primarily focuses on maintaining and enhancing these principles as the application evolves:

1. **Extend the Abstraction:** As new employee roles are introduced, extend the Employee class to include these roles, ensuring each new class implements the abstract methods.
2. **Enhance Role-Specific Behaviors:** Continue to encapsulate role-specific behaviors within each subclass, ensuring SRP is maintained.

3. **Modularize Reporting and Reward Calculation:** If new types of reports or reward calculations become necessary, consider introducing further abstractions or strategies to handle these variations without altering the existing classes.
4. **Implement Unit Tests:** Ensure comprehensive unit testing for each class and method, focusing on critical scenarios and edge cases to validate the functionality and adherence to SOLID principles.

Implementation and Testing

The implementation strategy involves careful extension of the current architecture to include new employee roles and functionalities while ensuring that unit tests validate the system's behavior and adherence to SOLID principles.

Documentation Update

Updating documentation is crucial to reflect any changes or additions to the system. This includes detailing how new classes conform to SOLID principles and ensuring that the documentation remains a valuable resource for understanding and maintaining the system.

This structured approach ensures the application remains robust, maintainable, and scalable, fully leveraging the benefits of the SOLID design principles.