CS 2340 | SOLID & GRASP Write-Up **TEAM 10**

Aditi T. | Nithyaa B. | Roshni D. | Sumayya M. | Heeba E. | Hala E.

SOLID Principles

Single Responsibility Principle (SRP)

Each class in the system has a single responsibility, making the code easier to maintain and modify. The Task class is responsible for managing task-related attributes such as title, description, due date, and status. The Project class handles project-specific operations like adding/removing tasks and team members, and doesn't concern itself with the internal logic of tasks or members. The TeamMember class focuses on the attributes and actions of team members, such as joining and leaving a project. This ensures that the modification of one responsibility (like changing how a task is managed) doesn't affect unrelated parts of the system (such as project management).

Open/Closed Principle (OCP)

The system is open for extension but closed for modification, which means new functionality can be added without modifying existing classes. A great example is how we created new types of tasks, like RecurringTask and HighPriorityTask, by extending the Task class.

This ensures that the core Task class remains unchanged. New functionalities, like notifying the team for tasks that are high priority can be added by extending the Task class, which makes sure that the Task class stays unchanged.

Liskov Substitution Principle (LSP)

The LSP is concerned that subtypes must be substitutable for their base types. The RecurringTask and HighPriorityTask classes are subclasses of Task. These subclasses can be used in place of a Task object in any operation, ensuring the system behaves consistently. This means that the system will not break when RecurringTask or HighPriorityTask objects are used where a Task object is expected, maintaining the functionality no matter the specific subclass.

Dependency Inversion Principle (DIP)

According to DIP, higher-level modules should not depend on lower-level modules but rather on abstractions. In our project The TaskManagementSystem interacts with abstractions like Task and Project rather than directly interacting with their implementations.

This makes the system easier to extend and maintain. The system is designed to depend on high-level abstractions, allowing for flexibility in how tasks and projects are handled without changing the core system logic. We achieve this by using interfaces or abstract classes, which let us handle different types of tasks and projects in a consistent way all while we keep the main system separate from the specific details of each implementation.

GRASP Principles

Creator

The Project class is the creator of Task and TeamMember objects. This follows the Creator pattern since the Project class aggregates these objects and manages their lifecycle. The Project class adds tasks using addTask and removes tasks using removeTask. Project is also responsible for adding and removing any TeamMember objects.

Low Coupling

The system exhibits low coupling, classes interact with each other through well-defined methods but are not constantly dependent on/using one another. The Project class interacts with Task and TeamMember objects through their public methods, without needing to understand their internal workings. This allows changes in the Task or TeamMember class to be made without impacting the Project class.

High Cohesion

Each class has high cohesion, meaning that the attributes and methods in each class are closely related to its main responsibilities. The Task class is focused only on task management (updating status, marking as complete, etc.), while the TeamMember class is concerned with team-related actions (joining or leaving a project). This tight focus ensures that each class is specialized and its functionality is clear.