Hong Kong Institute of Vocational Education (LWL)

Department of Information Technology Programme:

Higher Diploma in Software Engineering (IT114105)  
Module: Contemporary Topics in Software Engineering (ITP4507)

Li Yim Fan (220120531)

**1.Assumptions regarding the problem context**

**User Familiarity**:

The system assumes that the users possess a sufficient level of familiarity with the program and its input requirements. It is expected that users will have a good understanding of how to interact with the system and provide accurate and valid input without making mistakes.

**Error-Free Input:**

Building upon the user familiarity assumption, it is further assumed that users will consistently provide error-free input. While some basic input validation may be incorporated, the system does not extensively handle input errors or employ advanced error handling mechanisms.

**Addition of New Sports Teams:**

The system anticipates the future addition of new sports teams. This implies that the system should be designed in a way that allows for seamless integration of new team types without requiring extensive modifications to the existing codebase. The system's architecture should facilitate easy extension to accommodate new sports teams.

**Addition of Other Functions:**

It is expected that the system will undergo further development, leading to the addition of new functions or features. The system's design should prioritize modularity and flexibility to allow for the smooth integration of new functions without disrupting the existing functionality. The system should adhere to principles such as the Open-Closed Principle to facilitate easy extension without the need for significant code modifications.

**2.Application design with class diagram**Open Closed Principle:

一張含有 文字, 圖表, 字型, 筆跡 的圖片

自動產生的描述

Factory pattern:

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

The TeamFactory class creates a Team object based on the specified type, prompts the user for ID and name, and returns the created team.

The PlayerFactory class adds a player to a team by taking player information and position from the user, creating a Player object, and returning it.

Command Pattern: 一張含有 文字, 圖表, 平行, 字型 的圖片

自動產生的描述

The Command interface is implemented by several classes: CreateTeamCommand, SetCurrentTeamCommand, ShowTeamCommand, DisplayAllTeamCommand, and ChangeTeamNameCommand. These classes extend the Command interface and are related to the Team class.

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述

The Command interface is implemented by several classes: AddPlayerCommand, DeletePlayerCommand, and ModifyPlayersPositionCommand. These classes extend the Command interface and are related to the Player class.

一張含有 文字, 圖表, 行, 平行 的圖片

自動產生的描述

The Command interface is implemented by several classes: UndoCommand, RedoCommand, and UndoRedoListCommand.

一張含有 文字, 圖表, 行, 收據 的圖片

自動產生的描述

Associations between the Team and Player classes, implementation of the Command interface by ModifyPlayersPositionCommand, and associations between ModifyPlayersPositionCommand and ModifyPlayerPositionMemento. These classes form a structure for managing teams, players, and executing commands related to player positions.

一張含有 文字, 圖表, 收據, 行 的圖片

自動產生的描述

The Player class is associated with the Team class. The Team class implements the Command interface and has dependencies on the ChangeteamNameMemento, Enumeration, and Vector classes. The ChangeTeamsNameCommand class implements the Command interface and is associated with the Team class. The ChangeTeamsNameMemento class is associated with the Team class. The Command interface defines the common methods that are implemented by the Command classes.

Memento Pattern:

一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

自動產生的描述

The ChangeteamNameMemento class follows the Memento design pattern. It stores the state of a team's name in a memento object. The restore() method restores the team's name to its original state, while the getOrig() method retrieves the original team object. This pattern enables undoing and redoing changes to the team's name.

一張含有 文字, 螢幕擷取畫面, 字型, 筆跡 的圖片

自動產生的描述

The ModifyPlayerPositionMemento class follows the Memento pattern. It stores the state of a player's position in a memento object. The restore() method restores the player's position to its original state, while the getOrig() method retrieves the original player object. This pattern enables undoing and redoing changes to the player's position.

**3. Discussion and explanation on each of the design patterns applied to the application**

1. Command Pattern:

The Command pattern provides a flexible and extensible way to handle commands in the application. By encapsulating each command as an object that implements the `Command` interface, new commands can be easily added without modifying existing code. This makes it convenient to support new sports by introducing additional commands specific to those sports. For example, if a new sport like badminton is added, a "CreateBadmintonTeamCommand" can be implemented without impacting the existing codebase. The Command pattern promotes code modularity and separation of concerns, making it easier to extend the system with new functionality.

2. Factory Pattern:

The Factory pattern provides a way to create objects without exposing the instantiation logic to the client code. By utilizing a factory, new types of team objects (e.g., BadmintonTeam, 4x100mTeam) can be added by implementing the necessary factory methods without modifying existing code. This promotes the Open Closed Principle by allowing the system to be easily extended to support new sports. Additionally, the Factory pattern allows for centralized object creation, making it easier to manage and maintain object creation logic in one place.

3. Memento Pattern:

The Memento pattern allows the application to provide undo and redo functionality by storing the state of objects. This pattern is beneficial when adding new commands or actions that require undo/redo capabilities. For example, when modifying a player's position or changing a team's name, the Memento pattern can capture the previous state of the affected objects, enabling the application to restore them if needed. This promotes the Open Closed Principle by providing a mechanism to support new commands or actions while preserving data integrity. It allows for easy extension without modifying existing code that deals with undo/redo operations.

By applying these design patterns, the system design becomes modular, extensible, and adheres to the Open Closed Principle. It enables the addition of new sports by introducing new commands and objects specific to those sports. The patterns promote code reusability, maintainability, and separation of concerns, making it easier to enhance and extend the application's functionality without negatively impacting existing code.