**City University of Hong Kong**

CS3343 Software Engineering Practice

Analysis and Design Report

Project Title: Club Management System

**Group**: 2 **Tutorial** **Session**: L01

|  |  |
| --- | --- |
| **Name** | **Student Number** |
| WANG KA CHUN KEVIN | 55229123 |
| WONG SIU FAI | 55223247 |
| HO KWAN TO | 55233726 |

Contents

[Project Scope 2](#_Toc26823071)

[Design Constraints 2](#_Toc26823072)

[Functional Requirements 3](#_Toc26823073)

[Non-Functional Requirements 6](#_Toc26823074)

[Class Diagram 7](#_Toc26823075)

[Sequence Diagram 8](#_Toc26823076)

[Design Principles 11](#_Toc26823077)

[Design Patterns 13](#_Toc26823078)

[Code Refactoring 15](#_Toc26823079)

Project Scope

Nowadays, having a management system in a large company can highly increase its production efficiency. Therefore, our company, Nexus, mainly design management software like library management system, and staff management system.

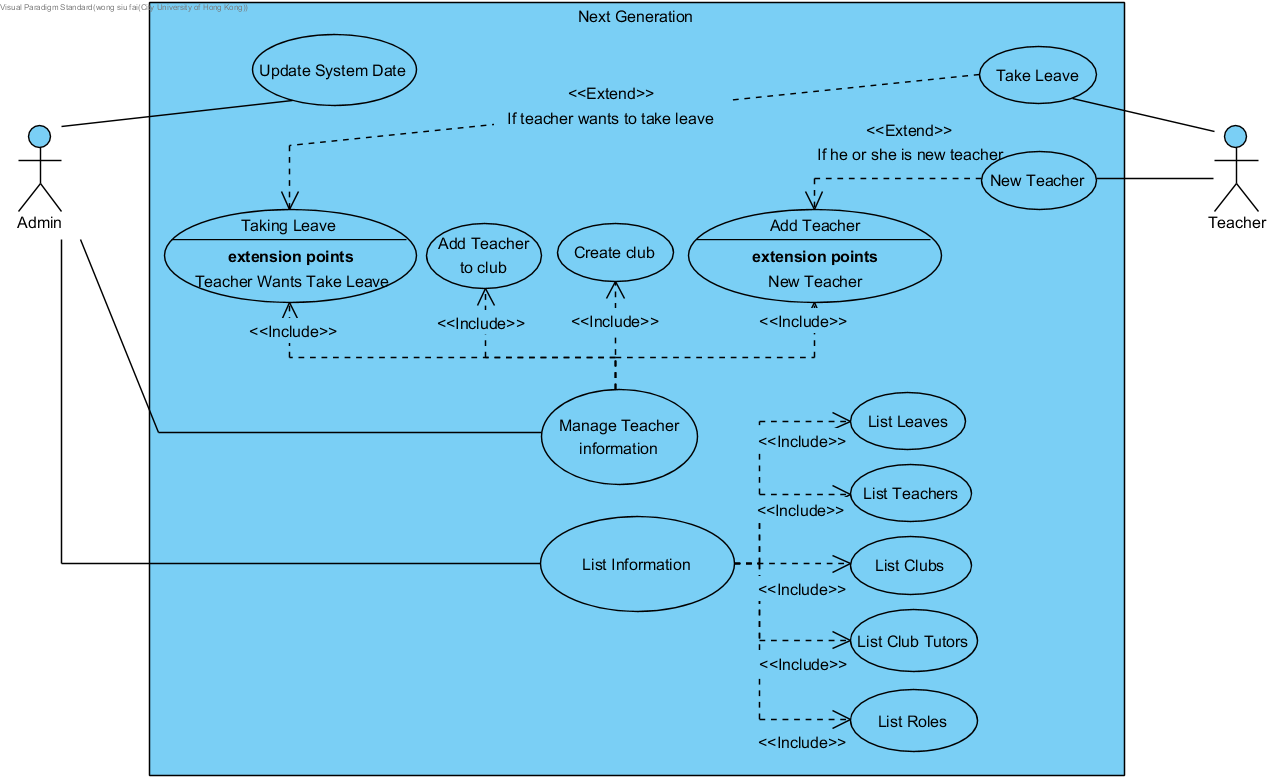
Extra curriculum activities are essential aspects of the school, but it is hard to manage various clubs at one time. So, we have designed a clubs management system with hiring teacher functions in it, which reduces human errors and highly increases process efficiency.

NextGeneration, an intelligent club management system, accepts, stores, and updates the teachers' and clubs’ information. The software provides four primary functions, hiring teachers, adding new clubs, taking leaves, and display the data.

Design Constraints

1. Cross-Platform  
   JAVA is a cross-platform program language since it compiled the JAVA program runs on all major platforms in which Java virtual machine exists. Therefore, the program can run on all major operating systems like Windows, MacOS, and Linux. Also, the user can run the application by running the executable file (.jar).
2. Graphical User Interface  
   The software does not require a fancy design on the GUI. Therefore, we maintain a clean and straightforward design on the GUI to let the user input and view the information quickly and achieve a user-friendly system. However, in the future, we would like to design an interactive GUI, for example, by clicking buttons.
3. Technical Feasibility  
   To achieve this, we will confirm the clients’ requirements and goals, then design the use case diagram for it. Then, we create the best software architecture by put on different design principles, design patterns, and using an object-oriented approach to it. Also, we have compared our pros and cons with the existing product on the market, which optimizes our product quality.
4. Logging  
   To trace the changes made by the users, we should have a logging system for the software, which can record the significant changes in the system. When it is needed, the administrator can generate a report for future use.

Functional Requirements



|  |  |  |
| --- | --- | --- |
| Use Case Name: | Take Leave | |
| Actor(s): | Teacher, admin(user) | |
| Description: | This use case describes the process of a teacher taking leaves. On completion, the system will record the leaves and display a confirmation message. | |
| Typical Course of Events: | Actor Action | System Response |
|  | **Step 1:** The users input the command with the leave type, name, and duration. | **Step 2:** The system will receive the command and display a successful message |
| Precondition: | TakeLeave can only be made if the teacher is hired in the system. | |
| Postcondition: | The system will record the leave. The teacher will be notified of the success of the operation. | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Add Teacher | |
| Actor(s): | Teacher, admin(user) | |
| Description: | This use case describes the process of hiring a new teacher. On completion, the system will add the teacher’s record to the system and display a confirmation message. | |
| Typical Course of Events: | Actor Action | System Response |
|  | **Step 1:** The users input the command with the name and number of annual leaves. | **Step 2:** The system will receive the command and display a successful message |
| Precondition: | AddTeacher can only be made if the teacher is new in the system. | |
| Postcondition: | The system will record the teacher information. The teacher and administrator will be notified of the success of the operation. | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: | Create Club | |
| Actor(s): | Admin(user) | |
| Description: | This use case describes the process of creating a club. On completion, the system will add a new club, and it’s the main tutor to the system and display a confirmation message. | |
| Typical Course of Events: | Actor Action | System Response |
|  | **Step 1:** The users input the command with the name of the club and its main tutor. | **Step 2:** The system will receive the command and display a successful message |
| Precondition: | CreateClub can only be made if the club is new in the system. | |
| Postcondition: | The system will record the club information. The teacher and administrator will be notified of the success of the operation. | |

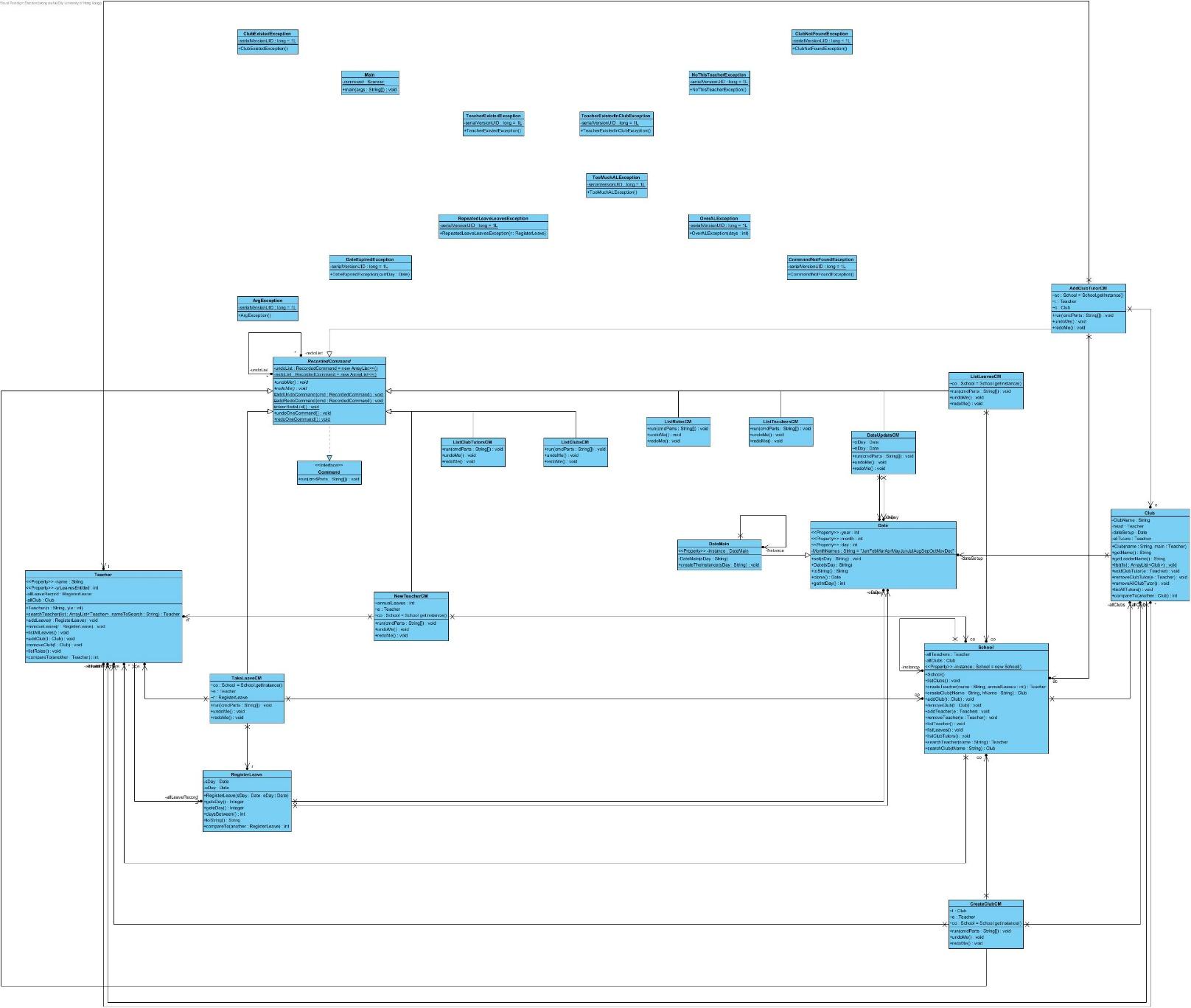
|  |  |  |
| --- | --- | --- |
| Use Case Name: | Add Teacher to Club | |
| Actor(s): | Admin(user) | |
| Description: | This use case describes the process of adding a new teacher to the club. On completion, the system will add the new teacher as the main tutor of a club to the system and display a confirmation message. | |
| Typical Course of Events: | Actor Action | System Response |
|  | **Step 1:** The users input the command with the name of the club and teacher, which will be the club’s main tutor. | **Step 2:** The system will receive the command and display a successful message |
| Precondition: | AddTeacherToClub can only be made if the teacher is not the existing tutor. | |
| Postcondition: | The system will record the new tutor information. The teacher and administrator will be notified of the success of the operation. | |

|  |  |  |
| --- | --- | --- |
| Use Case Name: | List Information | |
| Actor(s): | Admin(user) | |
| Description: | This use case describes the process of listing information. On completion, the system will display the information in the message. | |
| Typical Course of Events: | Actor Action | System Response |
|  | **Step 1:** The users input the command. | **Step 2:** The system will receive the command and display the information. |
| Precondition: | N/A | |
| Postcondition: | The system will display the information requested. | |

Non-Functional Requirements

1. Security  
   Data security is vital for end-users and clients, and data leakage can lead to a colossal disaster. To enhance the data security of the software, we require two-factor authentication to access the logs and records in the system. To ensure only authorized persons can enter the server room, and we will install CCTVs and biometric scanners for the server room.
2. User-friendly  
   Since our program is a command-line based in the current stage, so we need to keep a clean design of the user interface. So we have clear instructions to let the user input parameters and straightforward commands to access each function. Also, we remove unnecessary information, which can improve the users’ experience.

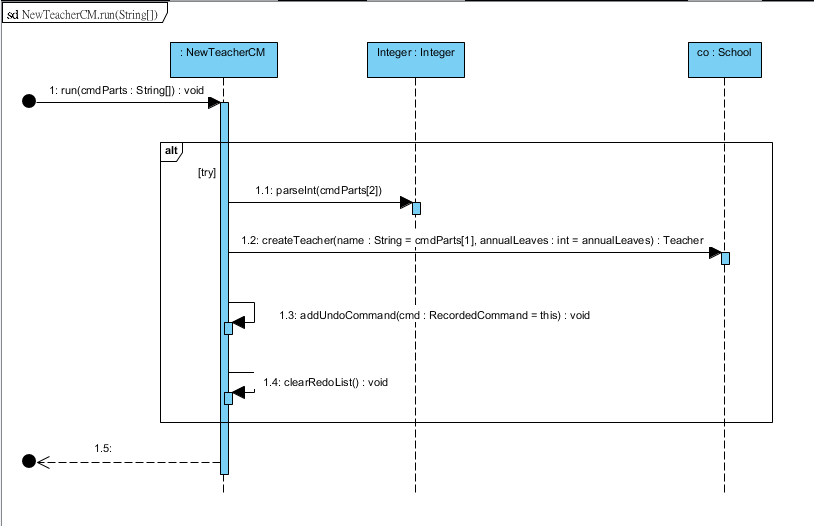
Class Diagram

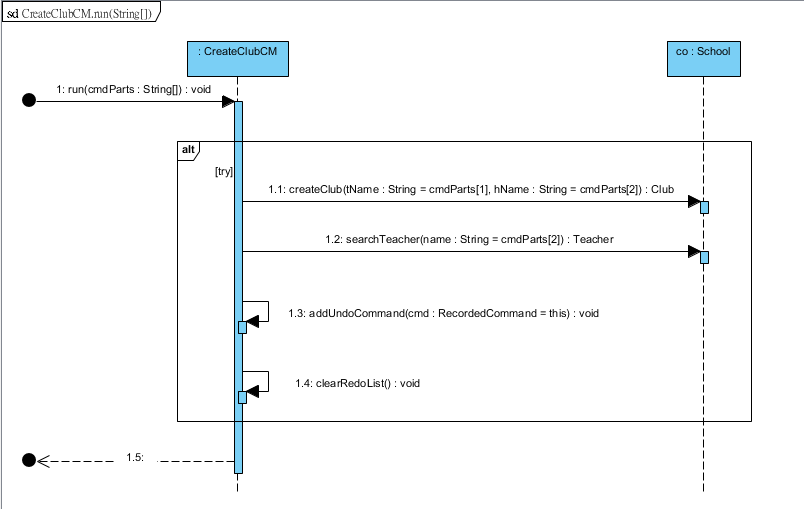


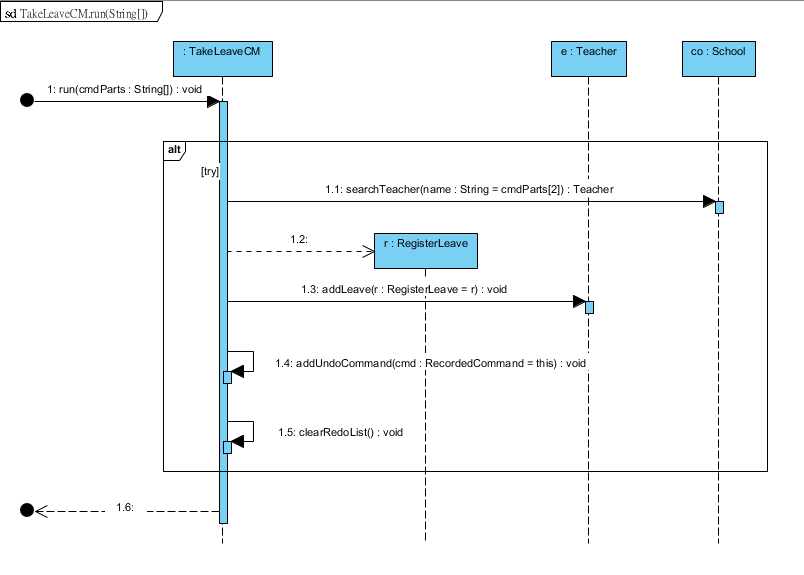
The original diagram will be located in the Diagram folder.

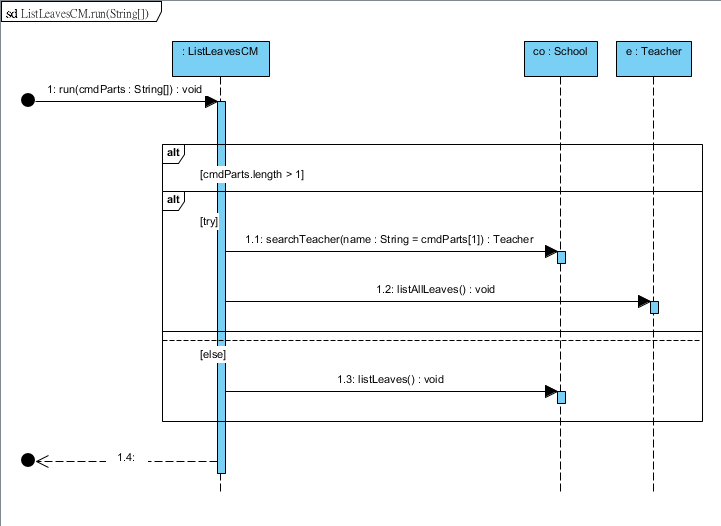
Sequence Diagram

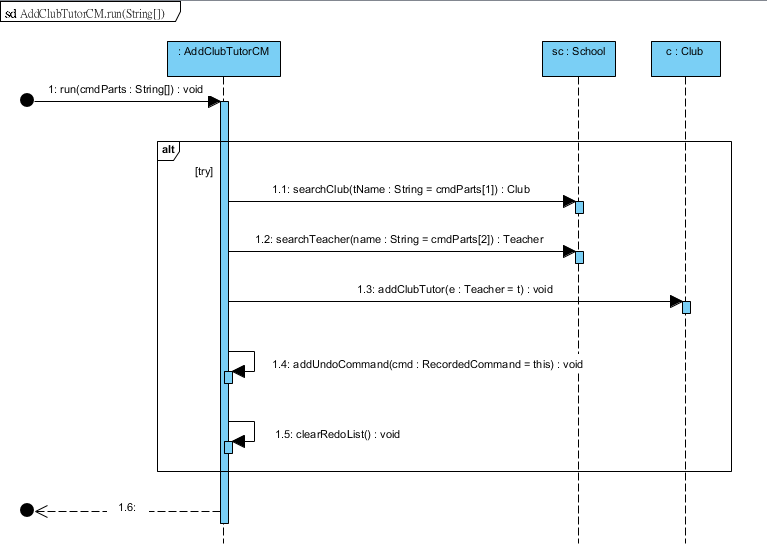
We have the following sequence diagram for the main functions of our software.

NewTeacherCM – run()  


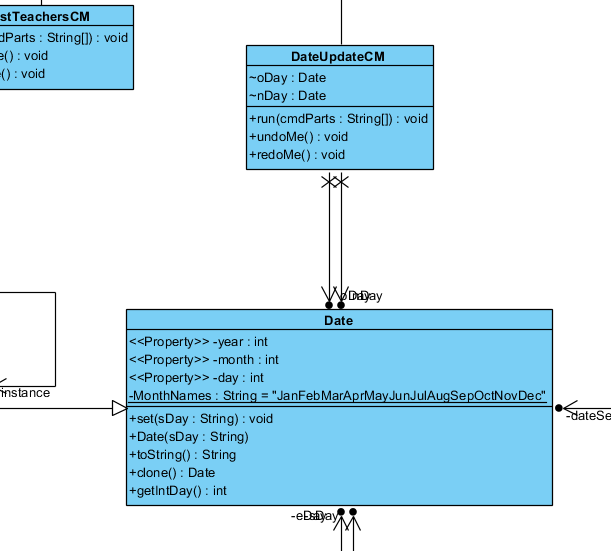
CreateClubCM – run()  


TakeLeaveCM – run()  


ListLeavesCM – run()  


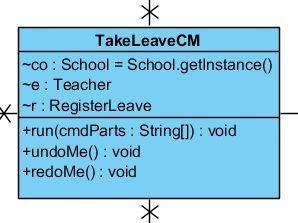
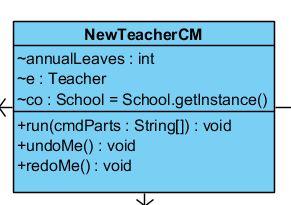
AddClubTutorCM – run()  


Design Principles

Open-Closed Principle (OCP)  


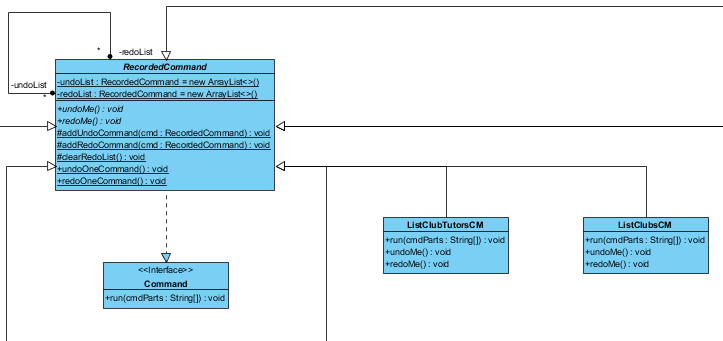
From the above diagram, all the variables in the Date class are private and cannot access or modify. If the users need to access them, they can access it by the function provided.

Also, the class Date can extend easily by other new classes without any modification of its methods and attributes.

Single Responsibility Principle (SRP)  
 

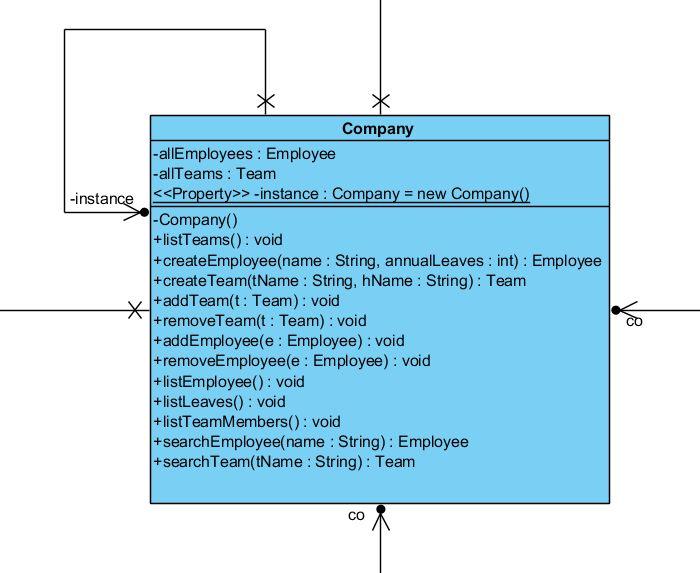
**TakeLeaveCM**, **NewTeacherCM**, and other **CMs** are following the Single Responsibility Principle, which only responsible for only one task in each class. As we mentioned, **TakeLeaveCM** solely accountable for taking leaves requests, and **NewTeacherCM** only responsible for hiring new teachers.

Design Patterns

Command Pattern  


As the command pattern used, our software provides an undo and redo function for the user. Also, it encapsulates the request as an object which achieves the goal of the rollback system.

The software can also add new commands easily without changing the existing code, and this will be a massive advantage for the future development of the system.

Singleton Pattern  


Since there exists one and only one company among all classes. Therefore, when the admin requires to hire new teachers or adding new clubs, no further objects of the **Company** will create. Also, all the teachers and admins are accessing the same instance of the **Company**.

On the other hand, the singleton pattern provides a global point of access, and it results in saving memory usage because of instantiating only once the Company during the whole running time.

Code Refactoring

Code before refactoring:

A screenshot of a social media post

Description automatically generated

Code after refactoring – *Extract method*:

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

Improvement:  
It achieves code reuse after extract method as the multiple lines code is converted into a technique. Some other class or method can then use addClubTutor instead of using numerous lines of code.