VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED PROGRAMMING

Report

Assignment 3

Lecturer(s): Dr. Phan Duy Han
Dr. Truong Tuan Anh
Student: Dai Ngoc Quoc Trung 2053537

HO CHI MINH CITY, APRIL 2024

Contents

1.1 UML Diagram

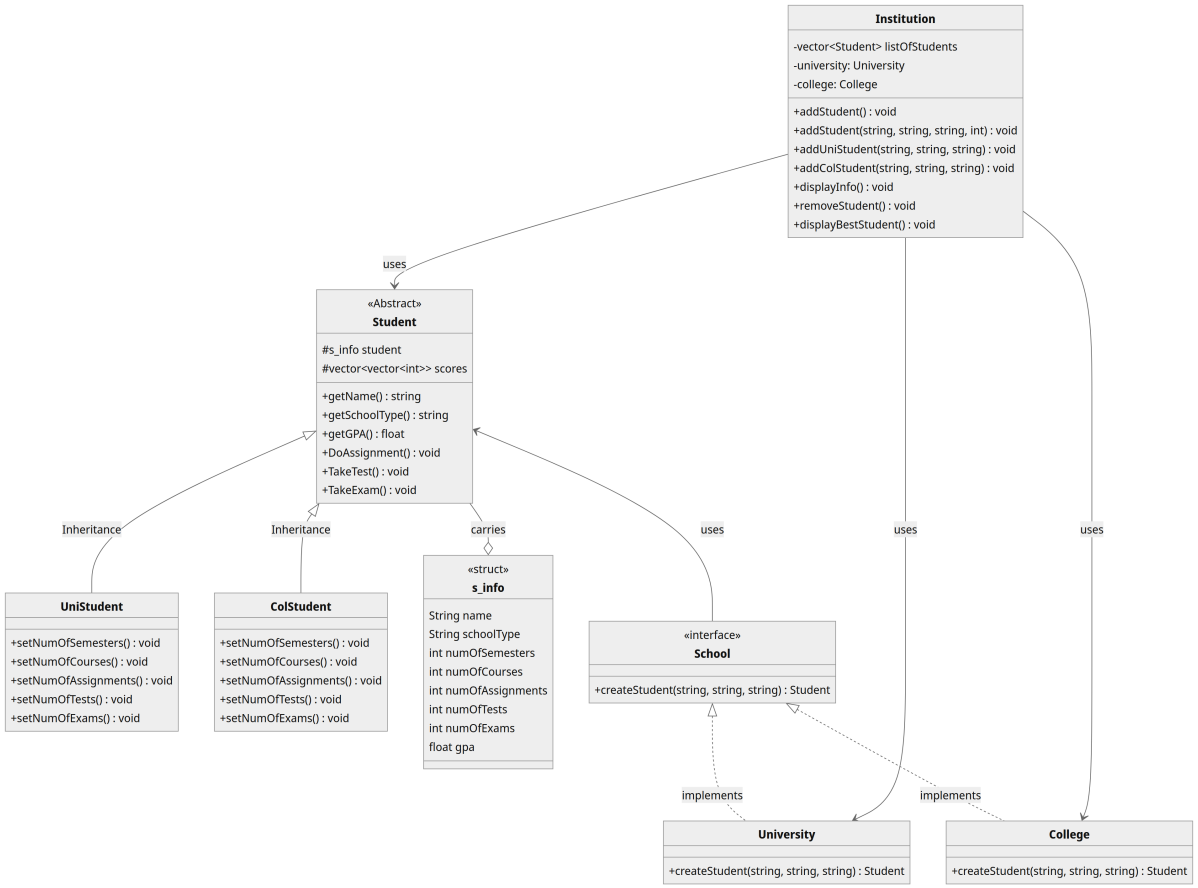

Figure 1.1: Student management system

1.2 Improvement in Part 1

Introducing *struct* and *vector* into the management system have offer several advantages:

- **Simplicity and Ease of Use**: Using a *struct* to encapsulate related data fields of a student (such as name, age, course, etc.) makes it easier to manage and access the data. It simplifies the code by grouping related data together.
- **Memory Management**: Using *vector* for managing collections of students provides automatic memory management. The *vector* handles memory allocation, resizing, and deallocation internally, making it reducing the risk of memory leaks or segmentation faults.
- **Dynamic Size Management**: vector dynamically resizes itself as needed to accommodate new elements. This allows me to simply add or remove students from the collection without worrying about managing the memory.

1.3 Improvement in Part 2

The advantages of Factory Design Pattern are:

- **Loose Coupling**: Factory design pattern removes the instantiation of actual implementation UniStudent and ColStudent classes from the client code. This make the code more flexible and maintainable since the changes on creation logic do not affect the client code.
- **Exensibility**: It's easy to add new student types without breaking existing client code.
- **Code reusability**: The factory method can be reused in different parts of application where object creation is need. This promotes centralizing and reusing object creation logic.

1.4 Improvement in Part 3

In part 3, I have used *modern auto* and *modern range for loop* features to improve my student management system. There are several benefits of them:

- **Concise code**: Using *auto* allows me to declare variables without explicitly specifying their data types, making the code briefly and easier to read.
- **Flexibility**: *auto* could be a game changer when it comes to complex type of a variable, since it is difficult to determine at glance.
- **Maintenance**: *auto* makes the code more maintainable by automatically adapting to changes in variable types.
- **Simplicity and Readability**: *Range-based for loop* provides a more concise and readable syntax compared to traditional iteration loops. This allows me swiftly get to understand and speed up maintenance process.
- **Ease of Use with Containers**: *Range-based for loop* is particularly well-suited for iterating over containers like vectors, arrays. It accommodates a convenient and intuitive way to iterate over elements of a container without dealing with iterator types or ranges explicitly.
- **Enhanced safety**: *Range-based for loop* automatically controls the bounds of the containers, ensuring that the loop terminates correctly and preventing potential buffer overflows or segmentation faults.

- **Compatible with User-defined Types**: *Range-based for loop* can be used with user-defined types that support iteration via iterators or range-based interfaces. This allows me to design my own classes and data structures, promoting encapsulation and abstraction.