

# Declaration of Original Work for CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Course  (CE2002 or CZ2002) | Lab Group | Signature /Date |
| Audrey Ho Hai Yi | CZ2002 | FSP2 | C:\Users\AUDREY\Documents\GitHub\oodp.ass\Audrey.png |
| Ong Wei Feng Kelvin | CZ2002 | FSP2 | Kelvin.png |
| Wong Kin Sum | CZ2002 | FSP2 | KinSum.png |
| Toh Jian Hao | CZ2002 | FSP2 | JianHao.png |

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.

# Design Considerations

## System Conceptual Design

The entire purpose of the system is to allow students to register for the courses of their choice. It to attempts to streamline the process by automating tasks that would have to be done manually by the user and prevent them from making mistakes during the process. There has also been several assumptions and or unresolved design issues that we have made or discovered respectively during development and has shaped or program into what it is. All of these will be stated below in point form.

## Issues that have been considered

1. Keeps track of Courses that they are currently registered into or in waiting list for.
2. Detect and alert them of clashes in time table.
3. Prevents them from registering into the same Course twice.
4. Alerts the user about if they have managed to successfully been registered or they have been put into the waiting list.
5. Only allows shifting of Course Group if they are registered under a Course already and if only there is a place of them or if there is student who has agreed to shift with them.

## Assumptions and Unsolvable Design Issues

1. As based on the assignment document these are the issue that have been considered of
   1. Need not consider multi-users concurrent login.
   2. Need not consider pre-requisite conditions when registering course.
   3. The passwords for login will be stored a flat file in hashed format and not clear text.
   4. External source implies pre-existing records and can be loaded from the file/s.
   5. Course and Students are to be stored in files. The format of storage is up to individual group’s considerations.
2. SMS alert has not been implemented and instead a place holder method giving the message that SMS has been sent has been use to simulate the affect.
3. It is assumed that users know or can find out the Group Index number of the Course of their choice and will not be provided on the screen.
   1. Since it is a Console Application it will be unwieldy to print out all of the possible Groups that a user can join and provide a way for the student to select them.
   2. The growth of the number of possible groups available is exponential to the number of Courses that can be added to the system, as each Course has on average seven groups, therefore making it more improbable to print out every Course Group a student can join.
   3. The user is assume to already know the Course Group Index Code of their choice and done prior research before the application date and therefore manual insertion should not prove to be an issue.
4. As concurrent login has no need to be considered the system will only accommodate one person at a time and has no need to provide real time alerts to certain events, such a requests to swap courses, and management of updates to the flat file system from multiple sources.

## Model, View and Control (MVC)

The system design uses the MVC Framework in order to implement Object Oriented Programming principal of SRP. How this is achieved is that it establishes the domain of interest that each class works on and how are they supposed to work with each other. In the MVC framework, the components of a program can be categorised as Model, View and Controller based on their function in respect to the system’s purpose.

The Model represents the real-life entities that are part of the process of Course Registration. They hold attributes that describe these entities and provide external interfaces to allow processing of information.

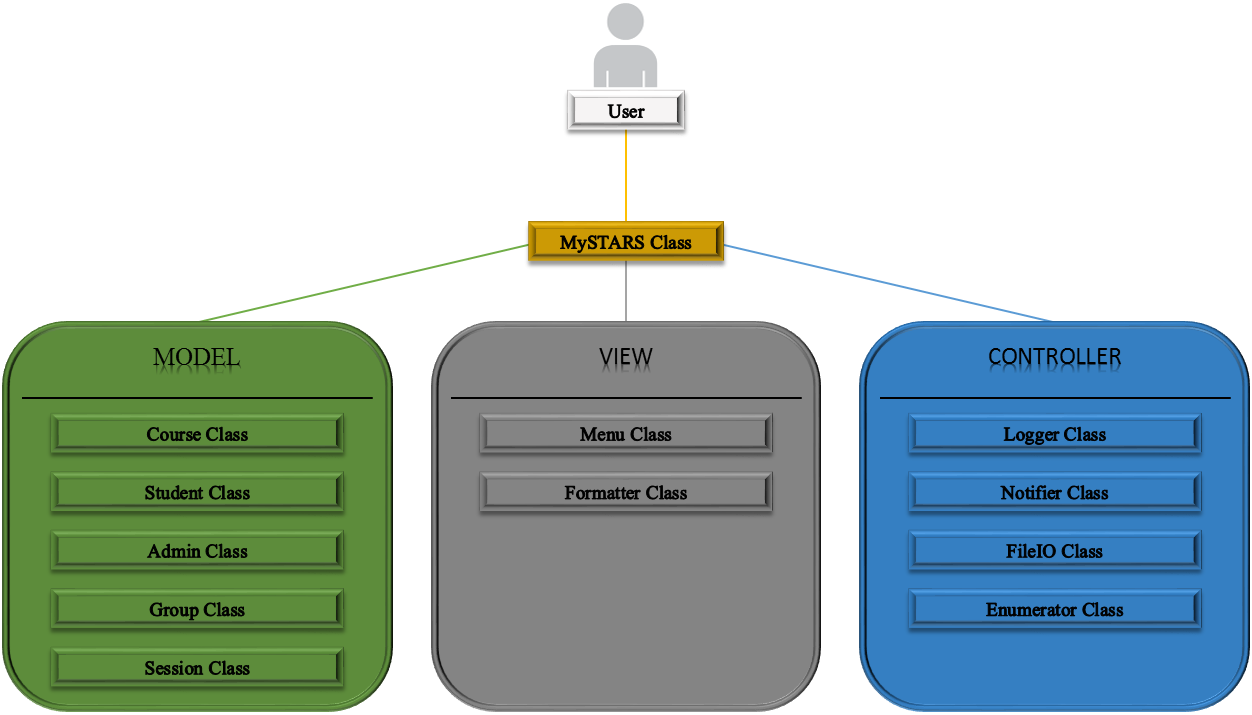
The View formats the data before displaying it to the user such as controlling the form and content of the User Interface (UI) displayed onto the console.

The Controller processes any of the user’s commands and translate it down to working on information or outputting the proper response to the user.

This segregation of duties can be taken further by making classes take more specific roles in those areas. For example the Menu class and Formatter class that fall under View both format information into away that it can be displayed in a meaningful manner to the user but does so in two different areas. The Menu Class focuses on forming UIs for the user to input commands into the system while the Formatter decides how data should be arranged on screen so as to give the user meaningful output.

In this way SRP is easier to establish and maintain. Classes can be categorised as a Model, View or Controller and have their scope narrowed down further into a single purpose. It also provide a mental map that can be used during development to check if a class that we wish to insert is either not well established or we placing too heavy a responsibility on it. If we are unable to determine which category a class should be placed into, likely the above two issues have occurred.

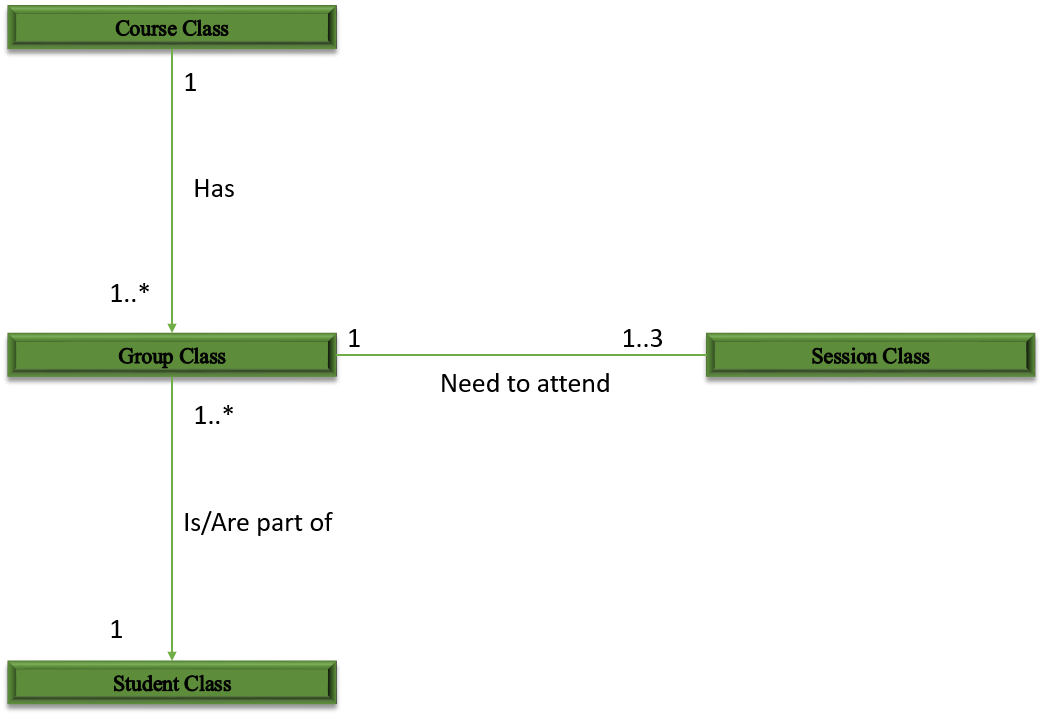
Abstraction of details becomes easier to be applied as the MVC clearly defines what interfaces need to be given to the other classes in order for them to use their services. The Model only has to provide interfaces for the other classes to manipulate data that they store. The View only has to provide an interface for information to be passed into them and returned into their formatted form. Lastly the Controller only has to provides interfaces to be invoked so that the User’s requests can be processed



The diagram above illustrates the architecture of our system in context to the MVC architecture. From the diagram we can see that all User interaction with the system is performed at the MySTARS class. The purpose of the MySTARS class is to translate user input into actions by invoking the services of the other classes as they are needed. Since the inner working of the system are never exposed to the user, changes to the system are transparent to them.

## Student Registration Process

In the system, Courses and the Students who are registered under them have their relationship established by the usage of Course Groups. These groups keep track of the number of students registered to the course and those who are in the waitlist, waiting for a free slot to be inserted into. The Group also establishes the lessons that are required to attend-lectures, tutorials and labs-these are kept track of in terms of Sessions-which state the time and venue of each lesson.



From the diagram above we can see that the Group class establishes the relationship of student with the Course they are taking and the time table slots that they are fulfilling. This relationship can be used to determine information about the Courses they are taking, the time table of the student and the number of AUs each course they are taking is worth without violating the principal of abstraction and Dependency Injection Principal (DIP).

This is because all the classes serve their own distinct function without relying on the others. They only provide information about the entity that they have modelled, nothing more and nothing less. The Student class only needs to take care of the Student attributes-the students name, email address for example, -it does not directly keep track of the courses that they have taken. Instead it relies on the interface of the Group Class to see if it is part of that Group and what Course do they fall under based on this relationship.

Likewise, the Course Class only uses the Group Class to keep track of the students under them and the time slots it has should there be a need to add more slots or change the timetable it all can be done by simply manipulating the Group Class or session class respectively. This design leads to a decoupled system whereby changes to the classes is not only minimized but each class can be reused based on what they are supposed represent as opposed to concrete implementation.

# Detailed UML Class Diagram.

* Further Notes, if needed

# Detailed UML Sequence Diagram of stated function.

* Further Notes, if needed

# Testing.

* Test Cases and Results