CS3343 Software Engineering Practice

BSC4 Project Group 6

Analysis and Design Report

Group Member:

CHAN Ho Man 53078369

CHEUK Yik Sum 53012319

CHAN Ho Man 53091240

LEE Man To 53075299

HO Sui Cheong Jonathan 53117178

CHEUNG Chi Ngai 53013494

Tutorial Section: LB2

# **Contents**

[**Contents** 2](#_Toc405556688)

[**Introduction** 3](#_Toc405556689)

[Design 3](#_Toc405556690)

[Class Description 3](#_Toc405556691)

[Work Flow 4](#_Toc405556692)

[Use Case Diagram 5](#_Toc405556693)

[Class Diagram 6](#_Toc405556694)

[Sequence Diagram 7](#_Toc405556695)

[Analysis Class Diagram 8](#_Toc405556696)

[Design Pattern and Design Principle Used: 8](#_Toc405556697)

[1. Open Close Principle (OCP) 8](#_Toc405556698)

[2. Single Responsibility Principle (SRP) 8](#_Toc405556699)

[3. Façade Pattern: 8](#_Toc405556700)

[4. Singleton Pattern 9](#_Toc405556701)

[Test Structure 9](#_Toc405556702)

[Terms 9](#_Toc405556703)

[Conflict Number 9](#_Toc405556704)

[Priority 11](#_Toc405556705)

[Our algorithm 12](#_Toc405556706)

[Handle the Core course 12](#_Toc405556707)

[Handle the first course 12](#_Toc405556708)

[Adding course and section into possibleResult. 12](#_Toc405556709)

[For other course 13](#_Toc405556710)

[The loop of possibleResult is ended… 13](#_Toc405556711)

# **Introduction**

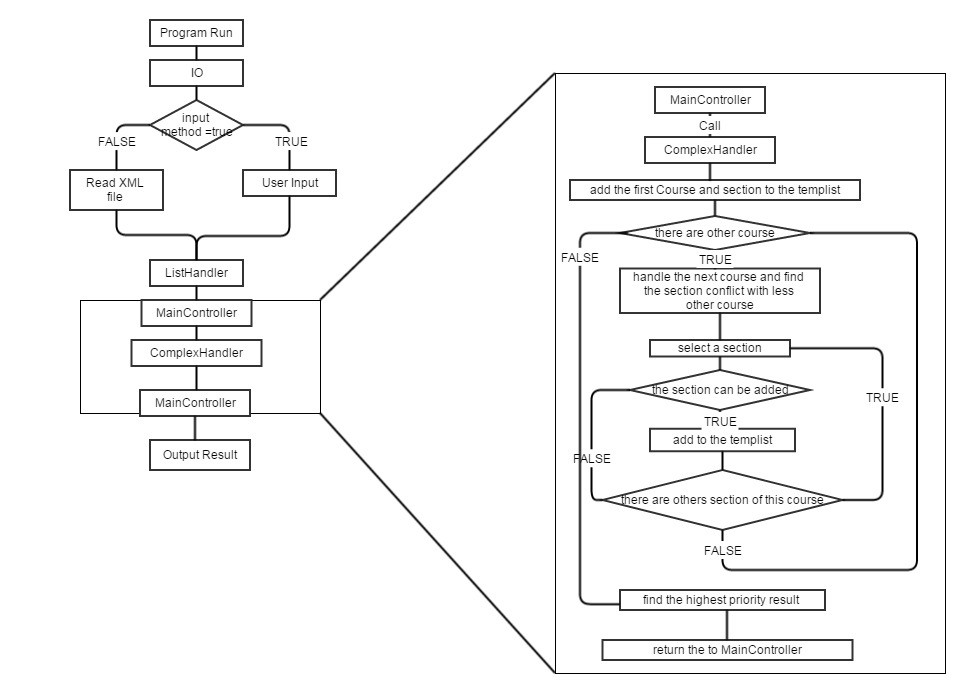
Class Scheduler is providing the students the suggestion to register their preferred course more efficiently. In order to let the others know about the Design Principle and the Design of Algorithm, the following report will introduce all our design principle and Design Pattern we use.

Design

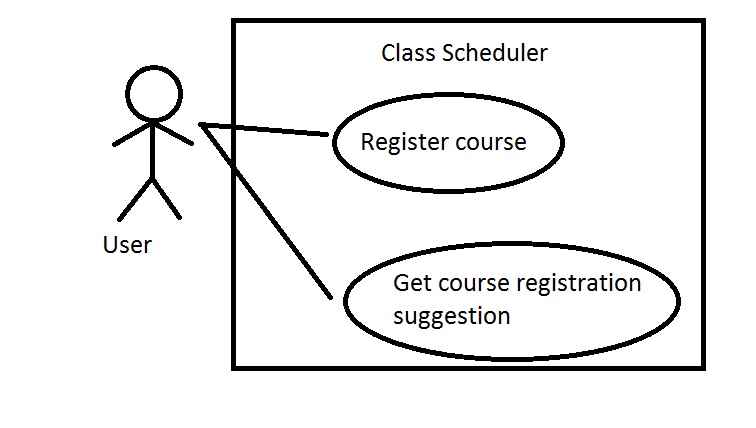
Class Description

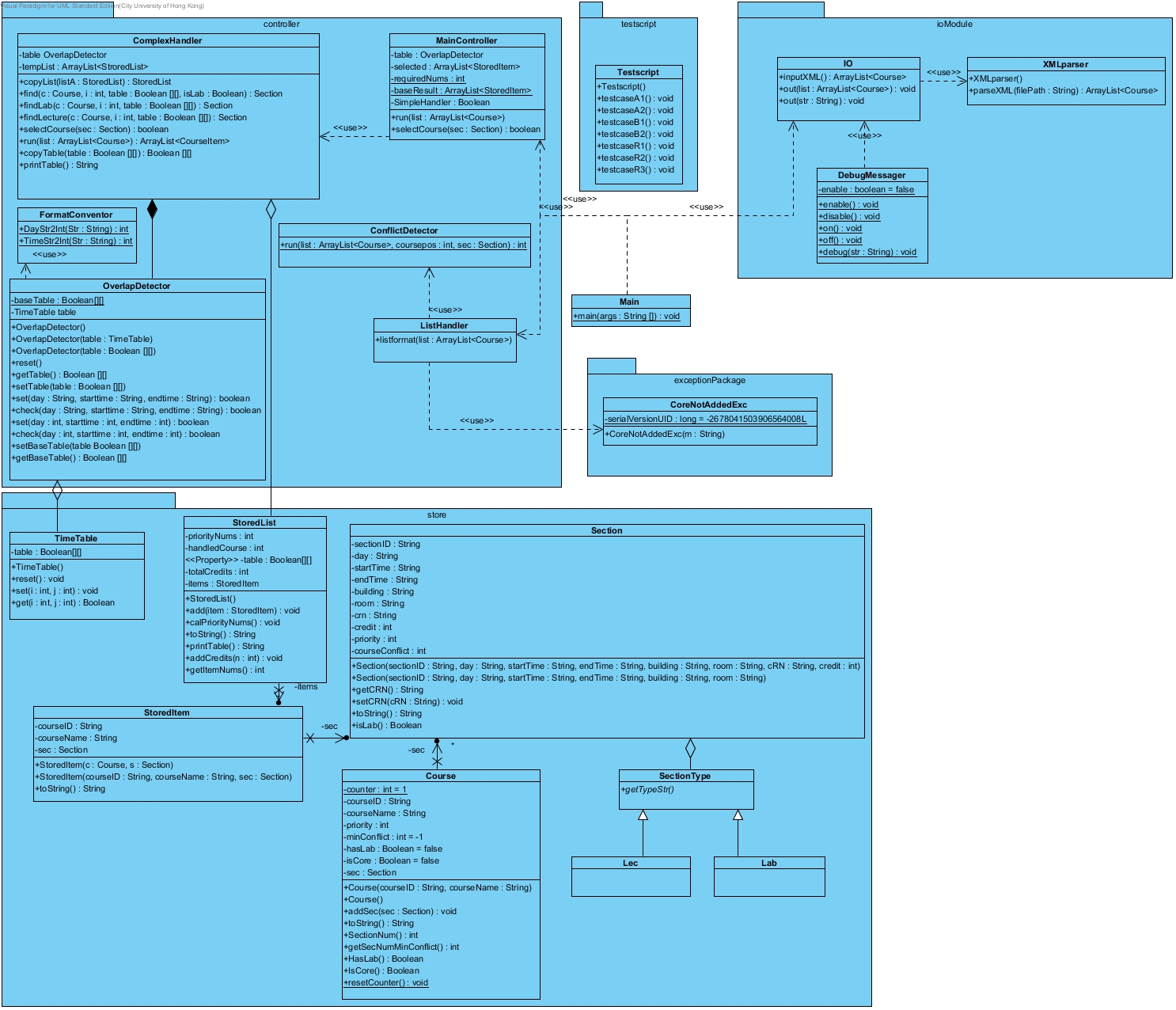
In the following, we will introduce our project design. The following list all the classes of our project that we have developed.

|  |  |
| --- | --- |
| Class | Brief introduction |
| **Package controller** | |
| ComplexHandler | The controller used to actually find the result of the input |
| ConflictDetector | A controller helps to find out all the conflict will happen |
| FormatConventor | A controller helps to convert some String(such as day, hour) to a integer value for other controller to use |
| ListHandler | A controller that helps to add more data that does not in the input to the course and section which can help the ComplexHandler find the result |
| MainController | The first controller will be run, it will call ComplexHandler and get the result for further action |
| OverlapDetector | A important controller helps to check can a section be selected for some situation |
| **Package exceptionPackage** | |
| CoreNotAddedExc | Throwed when a core course is not possible to be add. |
| **Package ioMudule** |  |
| DebugMessager | Used to output the debug message, can turn on or off |
| IO | Used to read the simple input and write the simple output |
| XMLparser | Used to read the XML file as a input |
| **Package store** | |
| Course | Provided all the accessor and mutator methods for the course object, and have method to find the smaller conflict number and find the number of section that conflict number equal to the minimum number of the course. |
| Section | Provided all the accessor and mutator methods for the section object |
| StoredItem | Provided method to get the courseID, courseName and courseSection for a particular course, and genetate these details in the output. |
| StoredList | Used arraylist to put all the course details |
| TimeTable | Used arraylist to allocate a course into a specific time slot |
| Lab | Return the Instance Object of Lab |
| Lec | Return the Instance Object of Lecture |
| SectionType | Return the type of section |

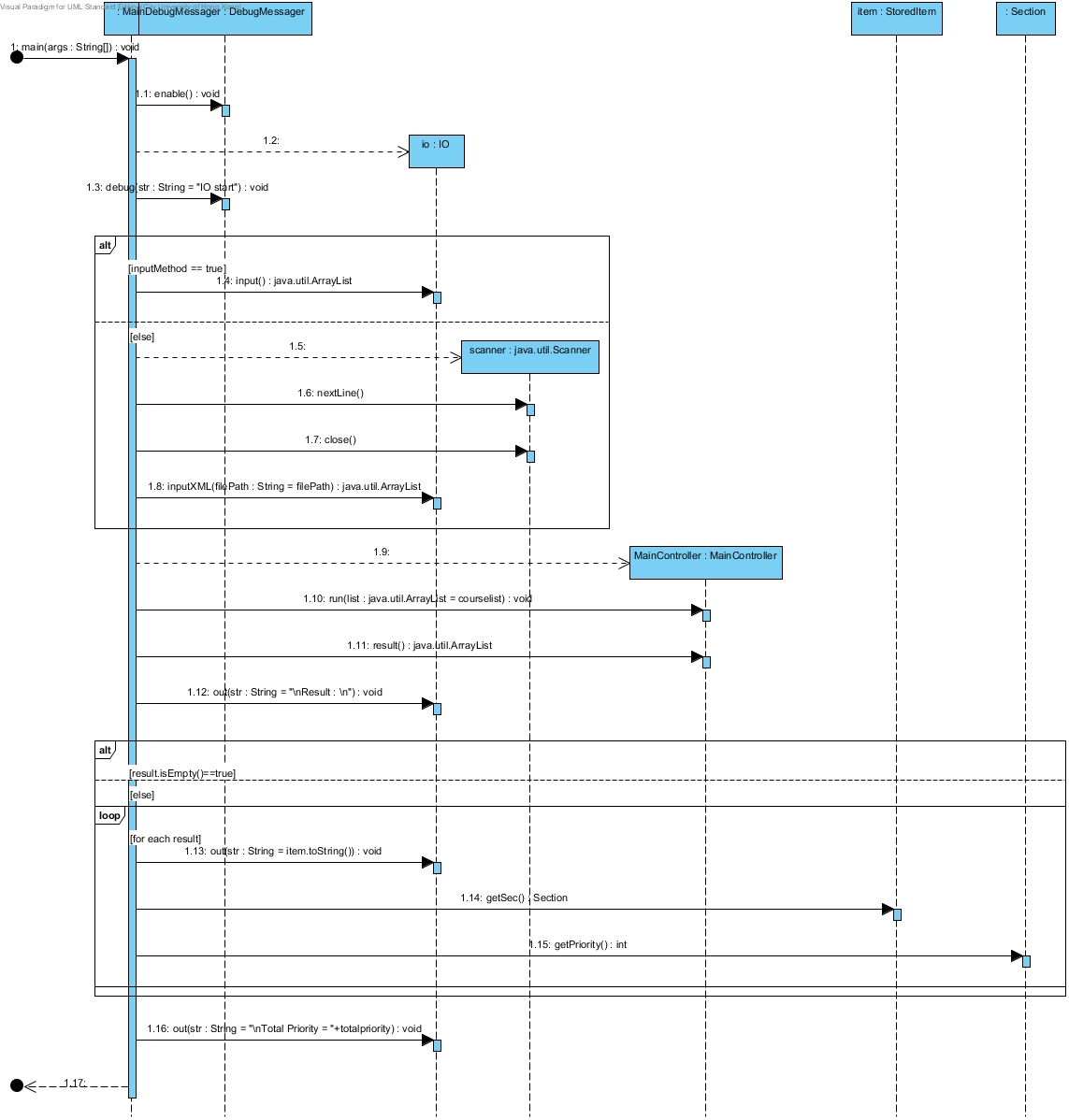
Work Flow

Use Case Diagram

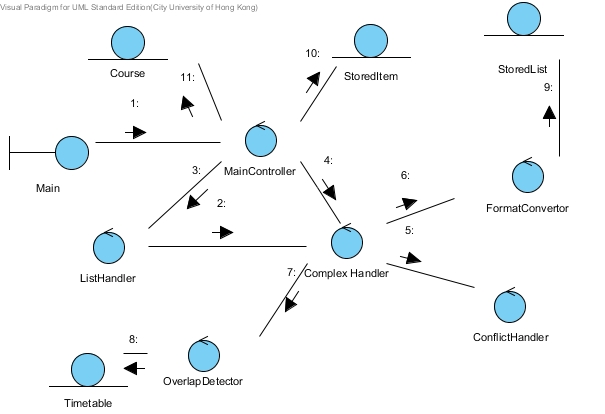


Class Diagram

Sequence Diagram



Analysis Class Diagram



Design Pattern and Design Principle Used:

1. Open Close Principle (OCP)

In this program, all the class in the Stored Package are use the accessor and mutator methods to get the require object for the operations. This can prevent the direct access of these class objects for direct access the methods directly. Besides, it can easily add the methods easily if we add more require object.

2. Single Responsibility Principle (SRP)

Our controller class designed are for different responsibilities, we do not want one class to handle more than one task, in order to make the program maintainable . For example, ConflictDetector’s responsibility is to find out the conflict number of a particular section in a course to other course, and the OverlapDetector’s responsibility is to check whether a course is able to add to a time table.

3. Façade Pattern:

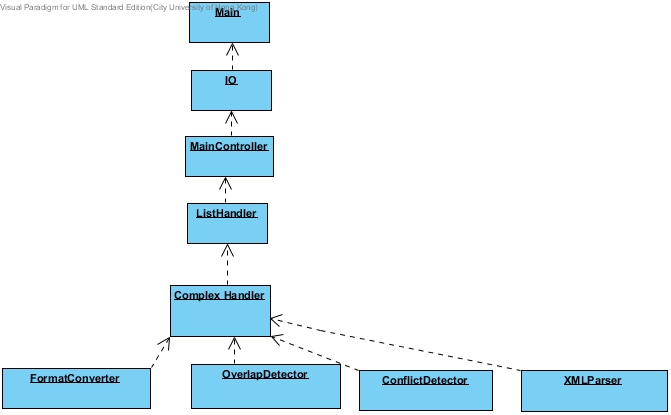
We used different types of controller as interface to handle all of the courses. The controller object will handle those complicate operations in these interfaces, such as the computation of the conflict number and conflict situations of the course.

4. Singleton Pattern

In the Lab and Lec class model we apply singleton pattern since it is just represent the type of section, we do not want a multiple instance appear at the same time.

Test Structure

In this project, we use bottom-up Testing, and the following is our test structure. The reason why we use this testing is we do not need to use test stub. Besides, the ability to test the program and the ability to plan and control are easy.



Terms

Conflict Number

Conflict Number is a number used to representing the effect of selecting a section of a course. This value will be calculated when the ListHandler run.

This value is the number of course that could not be selected LATER. if there are at least one session conflict with the current session, this value will +1, no matter how many session are conflicted, this value will only +1, as this can mark the calculation more easier.

**Example**

CS0001 TEST1 5  
C01 Fri 1200 1400 AC1 LT6 10001 3  
C02 Mon 1100 1300 AC1 LT8 10002 3   
C03 Wed 2000 2200 AC1 LT7 10003 3   
C04 Mon 0900 1100 AC1 LT3 10004 3   
C05 Fri 1500 1700 AC1 LT2 10005 3  
  
CS0002 TEST2 5  
C01 Fri 1200 1300 AC1 LT6 20001 3  
C02 Mon 1200 1300 AC1 LT6 20002 3  
C03 Tue 1200 1300 AC1 LT6 20003 3  
C04 Wed 1200 1300 AC1 LT6 20004 3  
C05 Thu 1200 1300 AC1 LT6 20005 3  
  
CS0003 TEST3 5  
CA1 Fri 1200 1400 AC1 LT6 30001 3  
CA2 Mon 1100 1300 AC1 LT8 30002 3   
CA3 Wed 1800 2000 AC1 LT7 30003 3   
CA4 Mon 0900 1100 AC1 LT3 30004 3   
CA5 Fri 1500 1700 AC1 LT2 30005 3  
  
CS0004 TEST4 1  
C01 Tue 0900 1200 AC2 2550 40001 3  
  
CS0005 TEST5 3  
C01 Wed 0900 1300 AC2 2550 50001 3  
C01 Mon 1700 1900 AC2 2550 50001 3  
C01 Mon 1500 1700 AC2 2550 50001 3  
  
CS0006 TEST6 1  
C01 Fri 1200 1400 AC2 2550 60001 3

* For CS0001 - C01, it conflict Number = 3 (conflicted with CS0002 - C01, CS0003 - C01 & CS0006-C01) because **if current section conflict with any session of a single course, it Confilct number will +1**
* For CS0002 - C01, it conflict Number = 2 (conflicted with CS0003 - C01 & CS0006-C01) because **the conflict number ONLY consider the course that having lower priority than the current course.**

Priority

Priority is used to determine the satisfy result. For each Sessions, their priority will be calculate as follow:

1. Get the ordered course list and session list from the user input
2. For the Courses, starting at the bottom number it from 1 to n.
3. Find the largest number of the sessions of all the Courses, calculate the digit number needed to represent the priority sessions.
4. For the Sections, starting at the top and number it from 1 to n.
5. Combine the number of course and Session.

**Example**

1. Get the course list and Numbering the Course list, Find largest Session Number of all the Course

CS0001 TEST1 120 (Priority Number = 2)  
C01 Fri 1200 1400 AC1 LT6 10001 3  
C02 Mon 1100 1300 AC1 LT8 10002 3   
...  
C120 Sat 1100 1300 AC1 LT8 10102 3   
  
CS0002 TEST1 2 (Priority Number = 1)  
C01 Fri 1500 1700 AC1 LT6 20001 3  
C02 Mon 1600 1800 AC1 LT8 20002 3   
  
Max(Session Number) = 120  
Digit needed = 3

1. Numbering the Session

CS0001 TEST1 120 (Priority Number = 2)  
C01 Fri 1200 1400 AC1 LT6 10001 3 (Priority Number = 001)  
C02 Mon 1100 1300 AC1 LT8 10002 3 (Priority Number = 002)  
...  
C120 Sat 1100 1300 AC1 LT8 10102 3 (Priority Number = 120)  
  
CS0002 TEST1 2 (Priority Number = 1)  
C01 Fri 1500 1700 AC1 LT6 20001 3 (Priority Number = 001)  
C02 Mon 1600 1800 AC1 LT8 20002 3 (Priority Number = 002)  
  
Max(Session Number) = 120  
Digit needed = 3

1. Combine the number

CS0001 TEST1 120 (Priority Number = 2)  
C01 Fri 1200 1400 AC1 LT6 10001 3 (Priority Number = 001, Final Priority Number = 2001)  
C02 Mon 1100 1300 AC1 LT8 10002 3 (Priority Number = 002, Final Priority Number = 2002)  
...  
C120 Sat 1100 1300 AC1 LT8 10102 3 (Priority Number = 120, Final Priority Number = 2120)  
  
CS0002 TEST1 2 (Priority Number = 1)  
C01 Fri 1500 1700 AC1 LT6 20001 3 (Priority Number = 001, Final Priority Number = 1001)  
C02 Mon 1600 1800 AC1 LT8 20002 3 (Priority Number = 002, Final Priority Number = 1002)  
  
Max(Session Number) = 120  
Digit needed = 3

Our algorithm

All of the main logic are implement in the ComplexHandler.

The Step to find out the result is as follow.

Handle the Core course

Check that if there are any ***Core Course***. If yes, Copy that course and section to a ***BaseResult***. If the system found that some of the Core Course could not be added. The system will stop and ask the user to check again. As this feature is used for the courses that are pre-assigned by the academic unit. These course normally could not change the section, so that this feature used to ensure these course will always in the result.

After this step, we have a ***BaseResult*** and the corresponding time table which marked the availabilities for time slot.

Handle the first course

For the first course that user inputted and not a core, we will add all of the section to a list called ***possibleResult***. If the ***BaseResult*** is not empty, we will also check the sections of the first course, and add the section that will not make any conflict with the core course into the ***possibleResult***.

Adding course and section into possibleResult.

*StoredList[i]*

*C1S1+C2S2+…+CnS2*

*handledCourse = n*

Adding more section from Course[n+1]

*StoredList*

*C1S1+C2S2+…+CnS2+Cn+1S1*

*handledCourse = n+1*

*StoredList*

*C1S1+C2S2+…+CnS2+Cn+1Sn*

*handledCourse = n+1*

The above graph showing how the system handle the ***possibleResult***. For the element in the ***possibleResult*** we call it ***StoredList***. Which is a list of ***StoredItem***, a simplified object of course and section.

There is a loop keep looping all for the ***StoredList*** in ***possibleResult***, for each ***StoredList*** there is a counter ***handledCourse*** to counting how many course have been processed for this result. For example, a ***StoredList*** having ***handledCourse*** with a value 3, that means this result is generated after the system try to add the section of course 3. It also means for this result, we need to handle the 4thCourse if there is.

Let Call the current ***StoredList*** as “currList” and its ***handledCourse*** *= n*. It means we need to add the section from course[n+1] to produce the new result. Assume there are 3 sections could be added to currList, Then we have 3 new result generated, such as,

currList : ListOfSection = [*C1S1+C2S2+…+CnS2*], handledCourse = n;

newResult[0] : ListOfSection = [*C1S1+C2S2+…+CnS2+Cn+1S1*], handledCourse = n+1;

newResult[1] : ListOfSection = [*C1S1+C2S2+…+CnS2+Cn+1S2*], handledCourse = n+1;

newResult[2] : ListOfSection = [*C1S1+C2S2+…+CnS2+Cn+1S3*], handledCourse = n+1;

\*\* *C1S1* representing 1st-section of the 1st-course.

These new results will added to the ***possibleResult***, and the loop will take them as “currList” later, the picture is that:

***possibleResult***

***size = n***

***currentpt = n***

***nextpt = N/A***

n

Process the next course

3 new results generated

***possibleResult***

***size = n+3***

***currentpt = n***

***nextpt = n+1***

n

Therefore, the loop will keep on make new result until, no new result could be create, which mean all of the course have been processed.

And there are two global variable ***maxPriority*** and ***result***, ***maxPriority*** keep storing the largest sum of priority in the ***possibleResult***, and ***result*** storing the ***StoredList*** with the largest total priority. When there are a new ***StoredList*** add into the ***possibleResult***, the system will compare the total priority with ***maxPriority***, if total priority larger than ***maxPriority***, the ***maxPriority*** and ***result*** will be updated.

For other course

For the other course, the system will only try to add the section having the minimum conflict number to the existing result. This is used to reduce the runtime and the resources.

The loop of possibleResult is ended…

As we keep updating the ***maxPriority*** and ***result***, when the loop finished, the ***result*** is storing the possible result that have the largest total priority. That will be the output of the program. However, If the ***BaseResult*** (Core Courses) is not empty, the program will merge ***BaseResult*** and ***result*** together to produce the final result.