



CZ2002 Object Oriented Programming & Design

Topic: STARS Planner

Members:

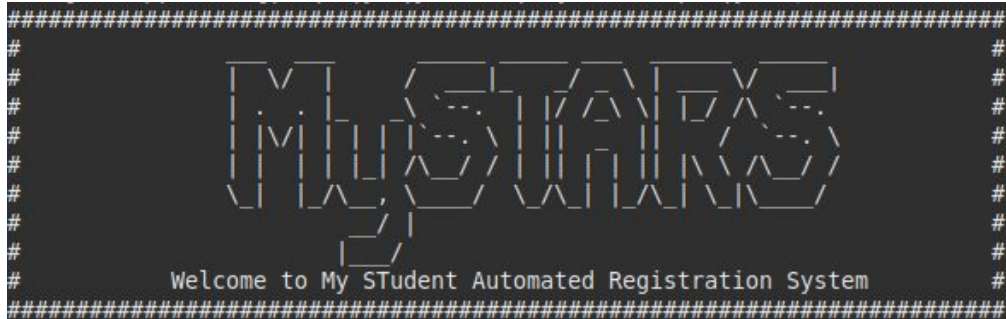
1. Chandrasekhar Aditya (U1923951A)
2. Jeffery Cao Siming (U1720566J)
3. Devansh Sunil Koppar (U1823660K)
4. Ong Jing Hong, Elliott (U1922981C)

Project Demo: <https://youtu.be/2QGsgbQl3t0>

Table of Contents

1. Introduction	3
2. Design Considerations	3
2.1 Approach Taken	3
2.2 Principles Used	4
3. OO Concepts Used	5
4. Extensibility	6
5. Assumptions Made	7
6. UML Class Diagram	8
7. UML Sequence Diagram	9
8. Test Cases	10
9. Declaration	15

1. Introduction



The aim of this project is to implement a STudent Automated Registration System Planner (STARS Planner) exclusively used by both Admins and Students. This application allows the Admin to manipulate various Configuration Settings and Managing Courses and Students. The Students are able to make use of these features and plan their timetable for their upcoming semester. In this application, we focus on the creative use of Design Principles, apply System Design Practices and Object-Oriented-P concepts to model.

This report will cover the concepts learnt in this course, and key design considerations used to implement the application. Appropriate class and sequence diagrams have been shown to understand the code better. Moreover test cases have been included which proves that the application works well and passes all the test cases mentioned in the assessment requirement.

2. Design Considerations

2.1 Approach Taken

We Designed our STARS Planner such that it is well-defined, conceptually simple and independent units interact with each other through well defined interfaces. This enabled us to achieve loose coupling and high cohesion. We also designed our system based on the Model-View-Controller architecture. With this architecture, we were able to incorporate the Observer and Strategy Pattern into our system. Entity classes such as 'Student' and 'Course' models the core classes that handle the state of the program. Model classes such as 'StudentUI' act as a view (Boundary) which users directly interact with. Controller Classes such as CourseManager contain the logic in which users can make changes to Model classes.

2.2 Principles Used

2.2.1 Single Responsibility Principle (SRP)

The principle states that a class should be specialized in their own responsibilities. This is to avoid the creation of a “God” class where a single class assumes more than one responsibility and can perform all functions. We have adhered to this principle by having separate Controller Classes for each Entity as follows:

- 1) CourseManager.java - It only deals with logic (create, update, delete) related to Courses
- 2) IndexManager.java - It only deals with logic (create, update, delete) Indexes
- 3) StudentManager.java - It only deals with logic (create, update, delete) Students
- 4) StudentCourseManager.java - It only deals with logic (create, update, delete) Courses
- 5) FileManager.java - It only deals with logic involved with reading, writing, and manipulating the data stored in files/databases.

As such, this allowed the classes above to only handle one aspect of the program, thus achieving low coupling.

2.2.2 Open-Closed Principle (OCP)

The principle states that classes or methods should be open for extension but closed for modification. In our program, we utilized interfaces such as User.java to achieve this. If the staff wishes to implement a new type of User, they can easily add another class implementing ‘User’ without changing the source code of other classes. For example, ‘User’ can be used to implement ‘Professor’ who will be added to the system as a new type of user without modifying the content in other classes .

2.2.3 Liskov Substitution Principle (LSP)

The principle states that functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. An example would be the UserUI.java and the classes that implement it are - StudentUI.java and AdminUI.java. The login() method in both the subclasses and superclasses returns the same result to the user. The method of the superclass

(UserUI) will return the same value even if the login() method gets overridden by the login() in the subclass.

2.2.4 Interface Segregation Principle (ISP)

This principle states that no client should be forced to depend on methods that it doesn't use. In our program, we ensured our interfaces to have the minimum required methods and attributes. For example, our STARS Planner has an interface, User.java and it is realized by Student.java and Admin.java to implement their own version of the functions. This helped us avoid 'FAT' interfaces, thus adhering to the Interface Segregation Principle.

2.2.5 Dependency Injection Principle (DIP)

This principle states that higher-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features. In our STARS Planner, the FileManager.java is a high-level module which is used to perform data storage and retrieval on Students, Admins and Courses. However, this high-level module does not depend on these classes, ie: Student.java, Admin.java, Course.java

3. OO Concepts

3.1 Composition (SRP)

We use Composition to show a whole-part relationship between 2 classes. For example, the Composition in our application is between the class Index and class Session. This is because an Index needs to have a tutorial or a lab (type of Session) to exist. This is established with a 'has a' relationship between Index and Session. Another example is , Course and Index.

3.2 Data Structures

For File I/O we used Serialization and deSerialization to store and retrieve Class Objects. We stored these files with a ".dat" file extension (binary). It's easier to perform read and write functions in a ".dat" file than a ".txt".

For Students in WaitList, we implemented a linked-list which is popped from the end and new students are added to the beginning.

3.3 Abstraction

Abstraction is a principle used to only show relevant information to external classes. In our program, we used this principle in many forms. For instance, in the Index.java class, we implemented a waitlist using a linked list that is hidden from external classes by setting its access visibility to private. External classes are only aware of its effects when the index is full and students can be added and removed from the waitlist.

3.4 Encapsulation

Encapsulation is used to hide data from external classes by allowing access only through 'get' and 'set' methods. In our program, we set the access visibility of all class variables to be private and implement 'get' and 'set' methods for each class variable to achieve encapsulation. For instance, in the Index.java class, we kept the numStudentsEnrolled' variable as private and implemented the 'getNumStudentsEnrolled()' and 'setNumStudentsEnrolled()' methods.

3.5 Inheritance

In Inheritance, when a class derives from another class, the child class inherits all public, protected methods and properties. In our application, the Student and Admin class inherits from the 'User' parent class thus promoting code reuse.

3.6 Polymorphism

Polymorphism is the ability of a function to take many forms. It is the ability of displaying the same functionality in multiple forms. In our code, we used polymorphism when representing lectures, tutorials and laboratory sessions. They all fall under the same parent class Sessions. However, each of them represents the functionality of Sessions in different ways.

4. Extensibility

In order to cater to the additional requirement of sending notifications other than email, classes can implement from the Notification interface, thus making its methods to be used both by Student Control and Admin Control classes to send a mode of notification. Additional methods

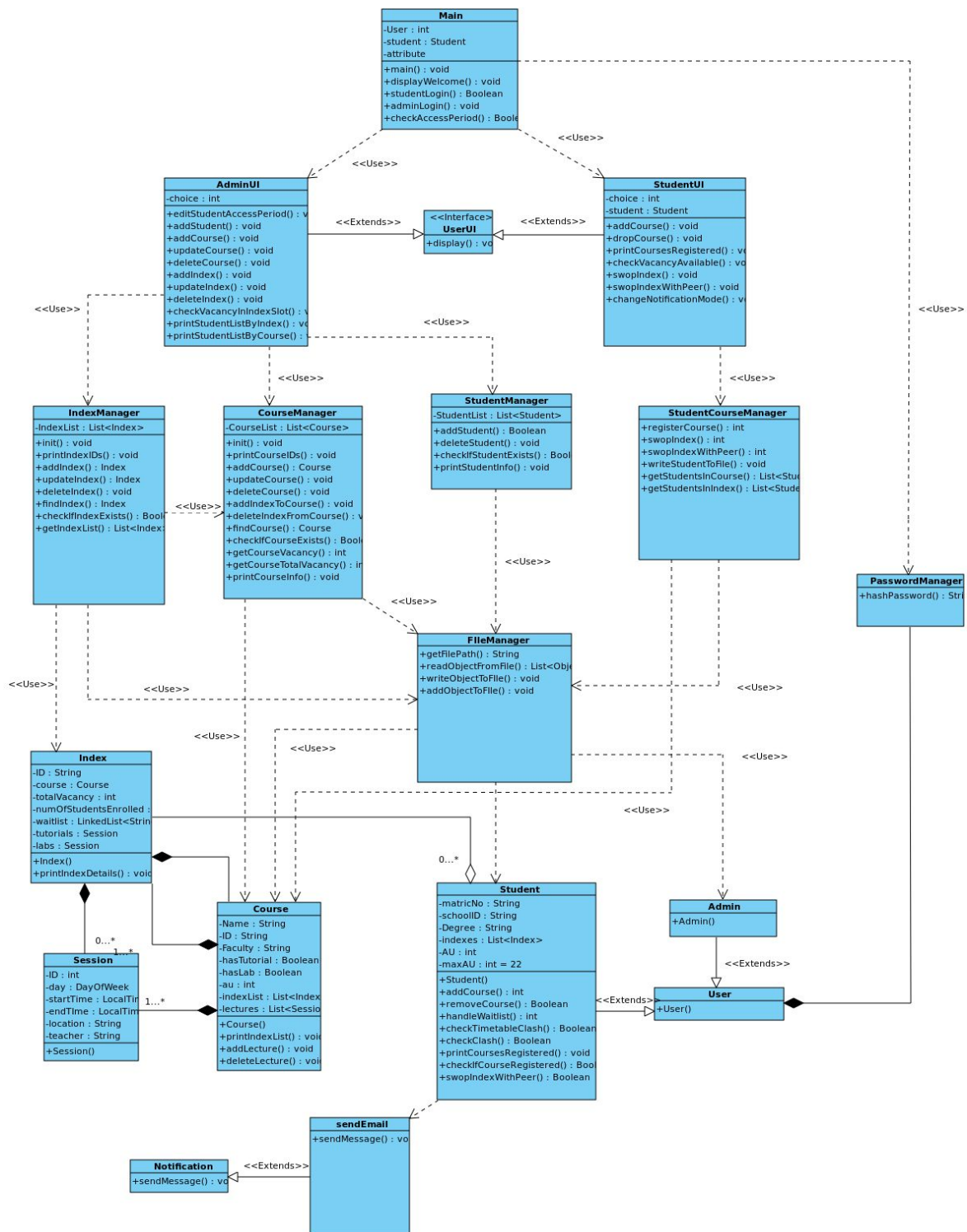
of sending notifications include notifying via SMS and WhatsApp: For these two examples, instead of just having a class named “SendMail”, we can have a class named “SMS” and “WhatsApp” which implements the Notification interface and allows our system to send Notification through other platforms.

5. Assumptions Made

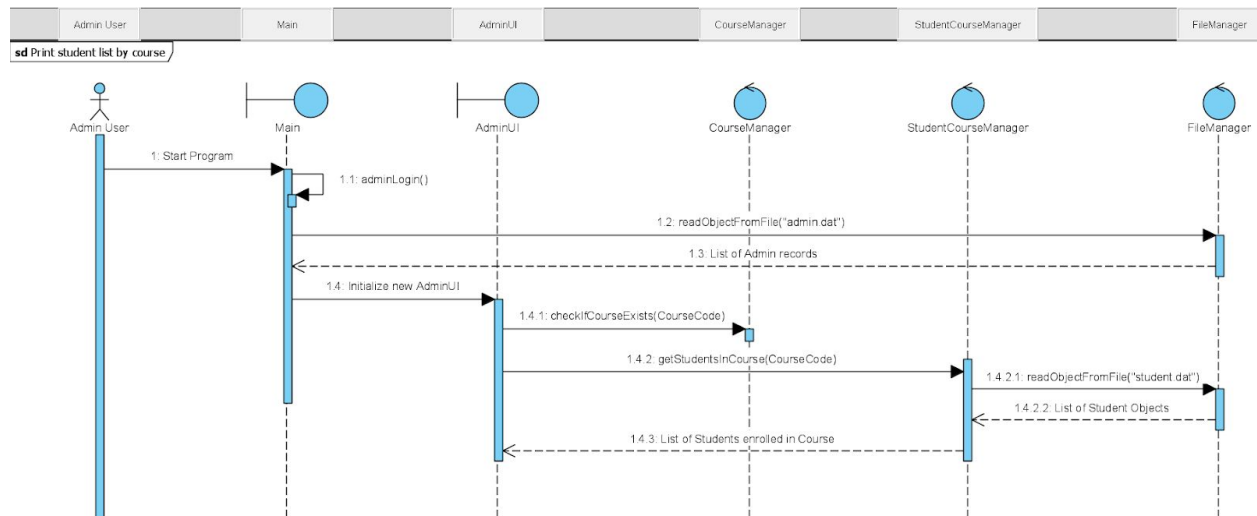
The team has decided to make the following assumptions for our Application, STARS Planner.

- 1) The STARS Planner is Open 24*7
- 2) The Admin will set the Student Access Period beforehand
- 3) Assume that all Students will be able to enroll in all Courses
- 4) To cater to future improvements, we design our application to consider multiple users and single users. Therefore, whenever a user enters the Main menu, we will assume it's a new user.

6. UML Class Diagram



7. UML Sequence Diagram



9. Test Cases

1) Student Login

	Test Case	Result
	Login not during Allowed Access Period	<pre> 1. Student 2. Admin 3. Exit 1 Cannot Login now. STARS Planner is not Active </pre>
	Login during Allowed Access Period	<pre> 1. Student 2. Admin 3. Exit 1 Enter your Matric Number: U0 Enter your password: asd ### Login Successful! ### </pre>

	Invalid Password	<pre> 1. Student 2. Admin 3. Exit 1 Enter your Matric Number: U0 Enter your password: asjn Incorrect Login Credentials </pre>
--	------------------	---

2) Add a Student

	Test Case	Result
	Add a new Student	<pre> Full name of student: Aditya Student account password: asd Student account email: Aditya@gmail.com Student's date of birth (DD-MM-YYYY format): 28-12-2001 Enter the student's school ID: SCSE Enter the student's degree code: CSE Student's Matric Number: U01 Successfully Added Student Record Successfully added! Student added successfully! Name: test Matric Number: U1 Degree:CSE Name: Aditya Matric Number: U0 Degree:CSE Name: Aditya Matric Number: U01 Degree:CSE </pre>
	Add an existing Student	<pre> 2 Enter the following details to add a new student to the system: Full name of student: Aditya Student account password: asd Student account email: aditya@gmail.com Student's date of birth (DD-MM-YYYY format): 28-12-2001 Enter the student's school ID: SCSE Enter the student's degree code: CSE Student's Matric Number: U0 Matriculation Number already exists Cannot add students with duplicate matric numbers. Do you want to try again? (y/n) </pre>

	Invalid data entries for adding new Student	<pre> Enter the following details to add a new student to the system: Full name of student: Aditya Student account password: asd Student account email: adi Invalid email address. Please enter the email ID again: </pre>
--	---	--

3) Add a Course

	Test Case	Result
	Add a new Course	<pre> Adding a new course... What is the course id? e.g. CZ2002 CZ2002 What is the course name? e.g. Object Oriented Programming and Design OODP What is the course faculty? e.g. SCSE SCSE How many Academic Units (AU) does the course have? eg. 3 3 Does the course have tutorials? e.g. True/False true Does the course have lab sessions? e.g. True/False false How many lecture sessions does the course have per week? eg. 1 1 Adding lecture 0... What day of the week is the session held? e.g. 4 = Thursday 1 Which hour is the session held? e.g. 13 = 1pm 13 What minute is the session held? e.g. 30 = 1:30pm 30 Duration of session in minutes? e.g 60 60 </pre>
	Add an existing Course	<pre> Adding a new course... What is the course id? e.g. CZ2002 CZ2001 Course Already Exists! Do you want to try again? (y/n) </pre>
	Invalid data entry for adding new Course	<pre> Adding a new course... What is the course id? e.g. CZ2002 CZ20334 What is the course name? e.g. Object Oriented Programming and Design newcourse What is the course faculty? e.g. SCSE SCSE How many Academic Units (AU) does the course have? eg. 3 three Invalid input! ##### </pre>

4) Register Student for a Course

	Test Case	Result
	Add a student to a course index with available vacancies.	<pre> Enter the Course Code: CZ2001 Enter the Course Index: I00 Succesfully Added CZ2001 ##### Course Code: CZ2001 Course Name: Algorithms AUs: 3 Index: I00 </pre>
	Register a student to the same course again	<pre> Enter the Course Code: CZ2001 Enter the Course Index: I01 You're already registered to this course! Try Registering to Another Course. ##### </pre>
	Add a student to a course index with 0 vacancies in Tut / Lab	<pre> Enter the Course Code: CZ2001 Enter the Course Index: I00 There are no vacancies. You've been added to the waitlist </pre>

5) Check available slot in a class (vacancy in a class)

	Test Case	Result
	Check for vacancy in course index	<pre> Enter the Index: I00 I00 has 0 vacancies ##### </pre>


	Invalid data entries for checking vacancy in course index	<pre> Enter the Index: ansd Index does not exist Do you want to try again? (y/n) </pre>
--	---	--

6) Day/Time clash with other course

	Test Case	Result
	Add a student to a course index with available vacancies.	<pre> Enter the Course Code: CZ2002 Enter the Course Index: I11 There is a TimeTable Clash. Cannot Add this Course ##### Course Code: CZ2001 Course Name: Algorithms AUs: 3 Index: I00 ##### </pre>

7) Waitlist notification

	Test Case	Result
	Add a student to a course index with available vacancies.	<pre> Enter the Course Code: CZ2001 Enter the Course Index: I00 There are no vacancies. You've been added to the waitlist </pre>
	Drop studentB from the same course index	<pre> Enter the Course Code: CZ2001 Succesfully Dropped the Course CZ2001 No Courses Registered ##### </pre>

	Display studentA notification	<p>STARS Planner Inbox x</p> <p> Aditya Chandrasekhar <donotreplyblackboard5@gmail.com> to me ▾</p> <p>Registered Courses: CourseCZ2001 Algorithms Index: 100</p>
--	-------------------------------	--

8) Print student list by index number, course



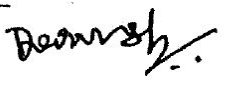
	Test Case	Result
	Print list by (i) Course (ii) Index	<pre> Student Name: test ,Matric Number: U1 Student Name: Aditya ,Matric Number: U0 ##### Enter the Index: 03 Student Name: Aditya ,Matric Number: U0 ##### </pre>
	Invalid data entries to Print list by (i) Course (ii) Index	<pre> Enter the Course: jsad Course does not exist Do you want to try again? (y/n) Enter the Index: amsd Index does not exist Do you want to try again? (y/n) </pre>

10. 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 the 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	Lab Group	Signature/Date
Chandrasekhar Aditya	CZ2002	BCG2	
Jeffery Cao Siming	CZ2002	BCG2	
Devansh Sunil Koppar	CZ2002	BCG2	
Ong Jing Hong, Elliot	CZ2002	BCG2	