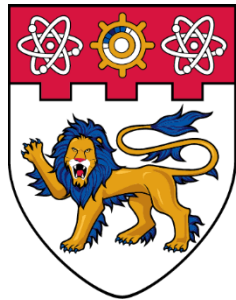


Nanyang Technological University
School of Computer Science and Engineering
CE2002: Object Oriented Design and Programming
Student Automated Registration System (STARS) implementation using Java



**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

Members –

Kesarimangalam Srinivasan Abhinaya	U1923302J
Musthiri Sajna	U1922559D
Singh Aishwarya	U1923952C
Unnikrishnan Malavika	U1923322E
Zhang Yilin	U1740676A

Lab Group – SE2

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.




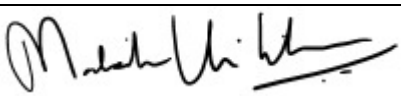

Name	Course	Lab Group	Signature	Date
Kesarimangalam Srinivasan Abhinaya	CE2002	SE2		17/11/20
Musthiri Sajna	CE2002	SE2		17/11/20
Singh Aishwarya	CE2002	SE2		17/11/20
Unnikrishnan Malavika	CE2002	SE2		17/11/20
Zhang Yilin	CE2002	SE2		17/11/20

Table of Contents

1. Introduction

1.1 Java Libraries Used

1.2 Tools Used

2. Design Considerations

2.1 Approach

2.2 Principles

2.2.1 Single Responsibility Principle (SRP)

2.2.2 Open Closed Principle (OCP)

2.2.3 Liskov Substitution Principle (LSP)

2.2.4 Interface Segregation Principle (ISP)

2.2.5 Dependency Injection Principle (DIP)

2.3 Object Oriented (OO) Concepts

2.3.1 Abstraction

2.3.2 Encapsulation/Information Hiding

2.3.3 Inheritance

2.3.4 Polymorphism

2.4 Assumptions

3. Unified Modelling Language (UML) Class Diagrams

3.1 Description and Illustration

4. Unified Modelling Language (UML) Sequence Diagrams

4.1 Description and Illustration

5. Final Testing

5.1 Test Case 1: Student Login

5.2 Test Case 2: Add a Student

5.3 Test Case 3: Add a Course

5.4 Test Case 4: Register Student for a Course

5.5 Test Case 5: Check Available slots in a class

5.6 Test Case 6: Day/Time clash with another course

5.7 Test Case 7: Waitlist Notification

5.8 Test Case 8: Print student list by index number and course

6. Demo Video Link

7. References

1. Introduction

Student Automated Registration System (STARS) is a course registration IT system built with the intention of assisting students with registering and adding courses and efficiently planning their timetables. The purpose of this report is to provide analysis and demonstration of a mock STARS system designed by the team, incorporating the Object Oriented (OO) concepts learnt in CE2002 – Object Oriented Design and Programming. The report is structured to provide an extensive overview of the mock STARS system through the discussion of design considerations, analysis of scenarios through Unified Modelling Language (UML) Class and Sequence diagrams and final testing with the sample test cases.

1.1. Java Libraries used

The following Java Libraries were used in the project –

- java.util.ArrayList, java.util.Scanner, java.util.Properties
- java.io, java.io.IOException, java.time, java.awt
- javax.swing, javax.mail.Message, javax.mail.MessagingException, javax.mail.PasswordAuthentication, javax.mail.Session, javax.mail.Transport, javax.mail.internet.InternetAddress, javax.mail.internet.MimeMessage

1.2. Tools Used

The following tools were used in the entirety of this project –

- Eclipse IDE – Compilation of the entire project – code, debug and run
- Sublime Text + Command Prompt/Terminal – Initial coding of individual sections.
- iMovie – Video Editor
- Google Meet and Zoom meetings – group meetings and video recording

2. Design Considerations

Design Considerations are devised to apply flexibility and universal accessibility to model designs. They can be used to identify existing faults and potential barriers in model implementation. The following section describes the design considerations formulated by the team for the mock STARS system.

2.1. Approach

The team's approach was to keep the work style as interactive as possible, which meant designing and implementing the code simultaneously (side – by – side). The beginning stage involved making a brief draft plan to identify the main classes and packages that would be required for the project. A decision was made to use the following as the final packages – courses, student, faculty, main and emailClient. As the workload was divided and individual work started, the team realised the need to add control and boundary class, displayIO. Hence,

project design was further improved and was made more detailed, so that the system could accommodate more exceptional cases. The mock STARS system is divided into entity classes (course, student, faculty), control and boundary classes (all IO - related classes).

Another hurdle faced by the team was that as the project began to take its form, several classes started to contribute to the same functionality (due to divided workload and individual interpretation of the problem statement). An example of this was when all three classes course, index and session were programmed to construct the course object. However, due to the interactive working style and good communication, the team effortlessly cleared this hurdle by regrouping and discussing functions of individually proposed classes to not overlap functionalities across classes and packages. In the end, the team made the call to divide classes in packages based on functionality so that there would be no further confusion.

2.2. Principles

2.2.1. Single Responsibility Principle (SRP)

The proposed design considers SRP from the beginning. Therefore, each class only has only one main responsibility and the classes have attributes and methods that are only directly related to themselves. This also makes the system easy to manage when changes are to be made, which ensures overall cohesion is achieved. In our code, we have implemented this through the extensive use of IO classes to manage changes within the file without affecting the main entity classes

2.2.2. Open Closed Principle (OCP)

The Open Closed Principle works on the simple rule that all entities – classes, objects, methods – should be open for extension, however closed for modification. The general idea of this concept is that code should be written in a way that new functionality can be added without the modification of existing code. This simply means that a change in one class requires all dependent classes to adapt to the change without the need of modification. For example, the abstract class that has been used as the super class to derive the different IO Boundary Classes is open for extension but closed to modification from the sub classes. It contains the methods for reading and writing serialized objects to and from the binary file.

2.2.3. Liskov Substitution Principle (LSP)

This principle states that the objects of a parent class are replaceable with objects of its child class without breaking the application. This means that the objects of the child class must behave in the same way as the objects of the parent class. Therefore, an overridden method must accept the same input parameter values as the parent class. For example –in the CourseIO class, when you're reading from the file the object gets stored as a Course object but we are reading it as an instance of the class Object.

2.2.4. Interface Segregation Principle (ISP)

The principle has a simple concept that informs programmers to not force clients to depend upon interfaces that they do not use. Its goal is to reduce the impact and frequency of changes by splitting code into several,

independent parts. This principle has been followed extensively throughout the structure of the program, as illustrated in the class diagram.

2.2.5. Dependency Injection Principle (DIP)

The main concept of this principle is that modules that provide complex logic (high-level modules) should be reusable and unaffected by changes in utility features, provided by the low-level modules. This is achieved through abstraction, where both High and Low-level modules depend on abstraction instead of each other, and abstraction itself does not depend on details, details depend on abstraction. For example, the IO class which has the main read and write operations to the file is independent of changes made to its subclasses.

2.3. Object Oriented (OO) Concepts

2.3.1. Abstraction

Abstraction refers to the concept of an object containing only essential characteristics that distinguishes it from other objects. The main purpose of abstraction is to handle the complexity of the program for the understanding of the user. This concept is also realised through SRP. An example of this would be the Course entity class only containing essential attributes such as CourseCode and Index ArrayList to distinguish them from other course objects created.

2.3.2. Encapsulation/Information Hiding

Encapsulation is used to build a barrier to protect private data. Information hiding hides the detailed implementation of the class from outsiders and prevents users from accessing data that they do not require for the implementation of the program. In our entity classes, most of the attributes are defined as private. The attributes can only be accessed or modified by the public accessor and mutator methods. Information hiding helps provide data security and thus ensures data integrity and consistency. For example, to check timetable clashes, there are methods in Index and Session classes called checkIndexClash and checkSessionClash. So, when students are to add a course, the student only needs to call the function to validate timetable clashes, but do not need to know the details of how the Index or Session class works. This enables better Encapsulation because Student and Index classes are responsible for their own validation and persistence.

2.3.3. Inheritance

Inheritance is defined as the mechanism through which a child acquires the attributes and behaviours of a parent. The concept of inheritance works on the fact that new classes (child) can be created that are built upon existing classes (parent). When something is inherited from the parent class, its methods and properties can be reused. This can be seen in the project, where the attributes of Course class are called and used in Student and Admin classes in the mock STARS system.

2.3.4. Polymorphism

The concept of polymorphism refers to having many forms – this happens when multiple classes are related to one another through inheritance. Polymorphism enables the user to use the methods that were inherited via inheritance to perform different tasks. Therefore, a single action can be used in various ways to perform different tasks. For example - the read function in the main IO class is responsible for reading from different files depending on which subclass it is called in.

2.4. Assumptions

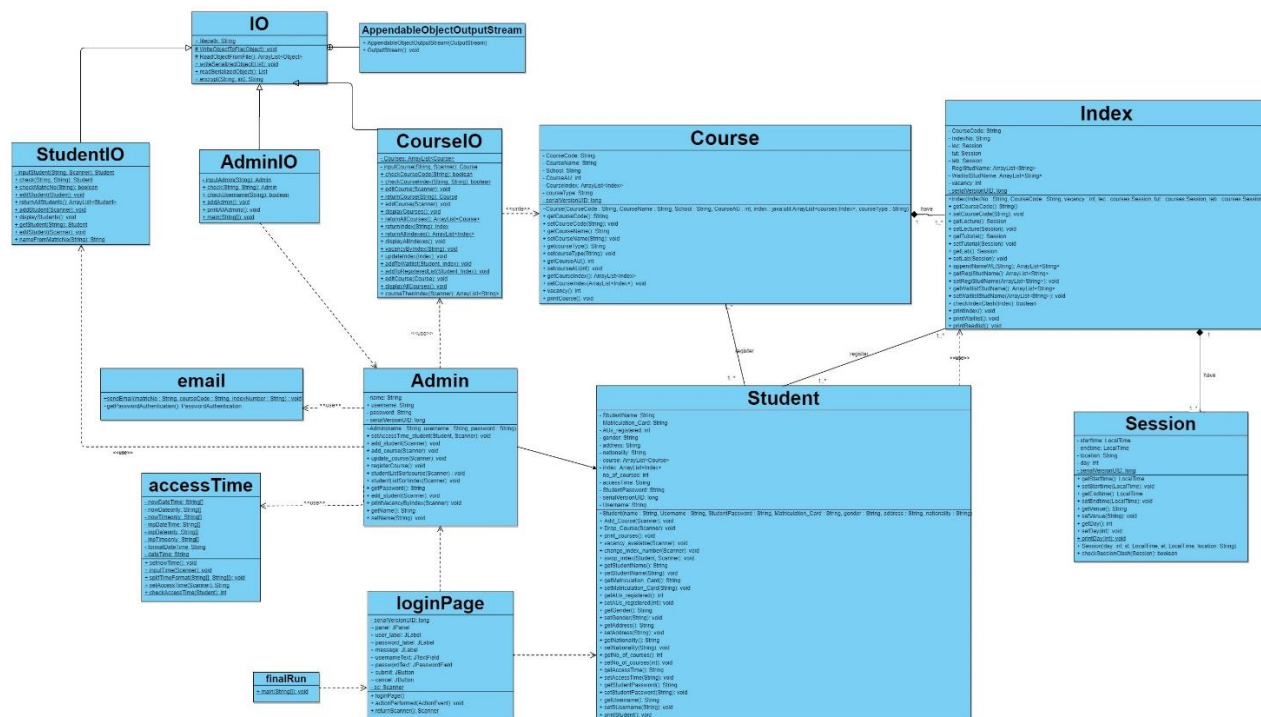
The following are some of the assumptions that were made during the design and programming phase of the entire project –

- a) The system does not take multi-user, concurrent login into consideration.
- b) Course Registration does not take course pre-requisites into consideration
- c) Courses only have Lectures, Tutorials and Laboratory sessions
- d) Login passwords are stored in a flat file in hashed format and not clear text
- e) Pre-existing records can be loaded from files
- f) Student records and courses stored in files – binary format – adding and updating modifies the files

3. Unified Modelling Language (UML) Class Diagram

3.1. Description and Illustrations

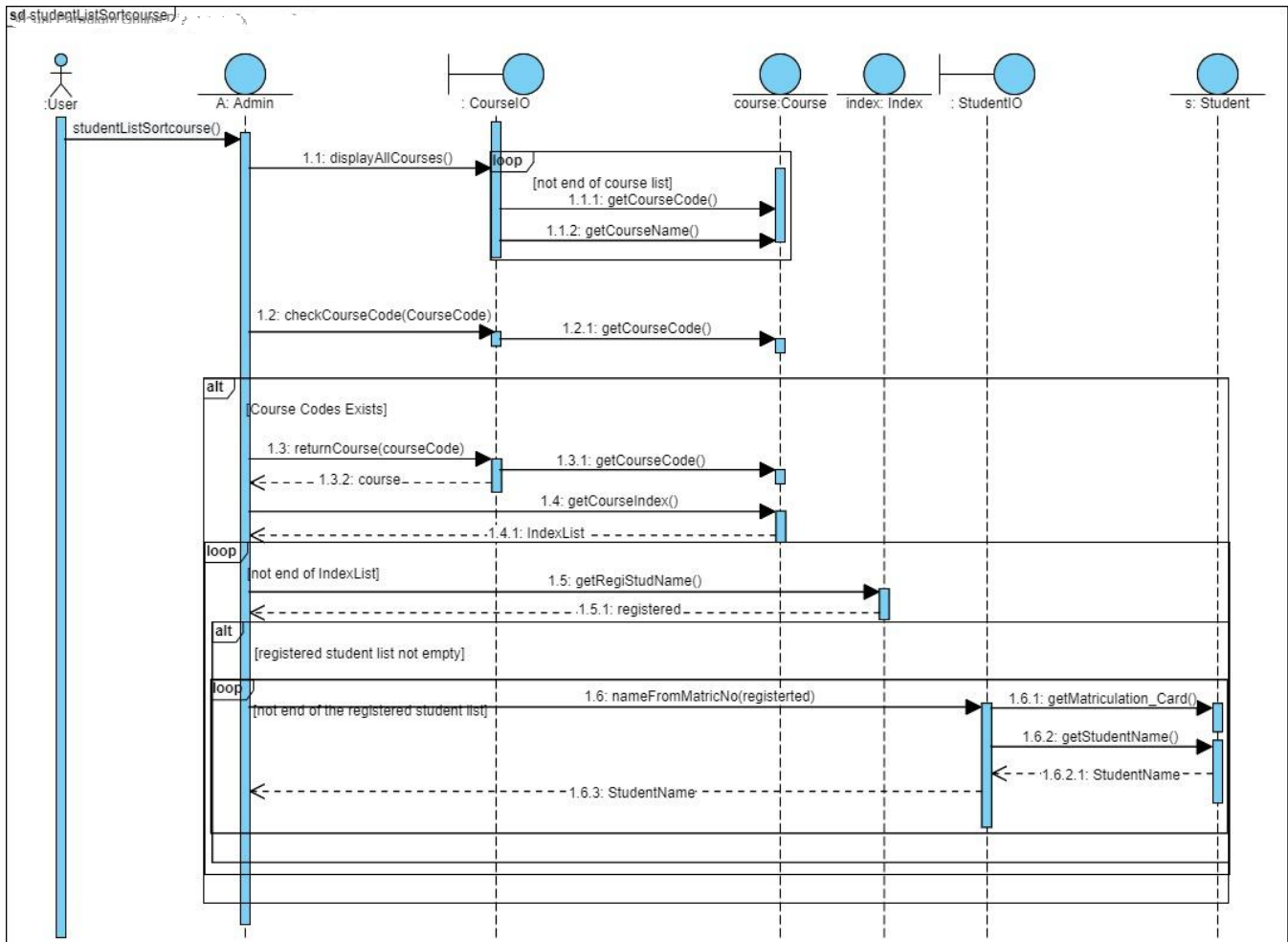
UML Class Diagrams are graphical illustrations used to visualise OO systems. It displays the system's classes, attributes, methods, and the relationship shared between different classes. The following diagram illustrates the mock STARS system's UML class diagram –



4. Unified Modelling Language Sequence Diagram

4.1. Description and Illustrations

UML Sequence Diagrams are graphical illustrations used to visualise interactions between objects. These diagrams are time dependent, and they illustrate the order of interactions by using the y-axis (vertical axis) to represent time.



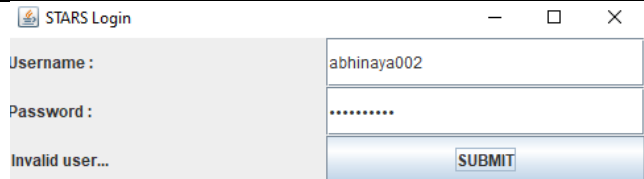
5. Final testing

5.1. Test Case 1: Student Login

This test case involves checking the following functionalities –

Test Case	Result
Login before access time (dates)	Time now: 23-11-2020 18:54:30 Access Restricted!
Login after access time (dates)	Time now: 23-11-2020 18:38:55 You have missed your date!

Incorrect Password



STARS Login

Username: abhinaya002

Password:

Invalid user...

SUBMIT

5.2. Test Case 2: Add a Student

This test case involves checking the following functionalities –

Test Case	Result
Add a new student	<pre> You have chosen: Add a student, as staff. Matriculation Number: U1923322E Student Name: Unnikrishnan Malavika Username: malavika002 Password: qwerty Gender: Female Address: Nanyang Avenue Nationality: Indian Student added. Number of Students read: 5 Student Name: K S Abhinaya Matriculation Number: U1923302J Student Name: Morgan Alex Matriculation Number: U1834654C Student Name: Heath Tobin Matriculation Number: U1723653G Student Name: Singh Aishwarya Matriculation Number: U1923463C Student Name: Unnikrishnan Malavika Matriculation Number: U1923322E Press enter to continue.. </pre>
Add an existing student	<pre> You have chosen: Add a student, as staff. Matriculation Number: U1923302J Student already exists. Please edit Student instead. </pre>
Invalid data entries	✓
Password Length Check	<pre> Password: time Weak Password, Try again. Password: </pre>

5.3. Test Case 3: Add a Course

This test case involves checking the following functionalities –

Test Case	Result
Add a new course	Course successfully added. The courses stored currently are:
Add an existing course	You have chosen: Add a course, as staff. Course Code: CE2002 Course already exists. Please edit Course first.
Invalid data entries	Course AUs: 4rw AUs can only be an integer. Please re-enter. Course AUs:
Index Check	Index Number: 10022 Index already exists. Please re-enter. Index Number:

5.4. Test Case 4: Register Student for a Course

This test case involves checking the following functionalities –

Test Case	Result
Add a student to a course with available vacancies	Welcome Musthiri Sajna! Registering students on the Waitlist... Please wait.... Pending Registrations Complete!
Add a student to a course with 0 vacancies	Student remains on the waitlist
Register the same course again	10022 Course is already registered! Press enter to continue..
Invalid Data Entries	✓

5.5. Test Case 5: Check available slots in a class

This test case involves checking the following functionalities –

Test Case	Result
Check for vacancy in course index	<pre>Please state your option here: 8 You have chosen: Check available vacancies for an index number, as staff. The available indexes are: 06087, 10305, 10306, 10022, 10021, 65463, 45397, Enter index: 10306 The vacancy in the given Index is: 9 Press enter to continue..</pre>
Invalid data entries	<pre>Please state your option here: 8 You have chosen: Check available vacancies for an index number, as staff. The available indexes are: 06087, 10305, 10306, 10022, 10021, 65463, 45397, Enter index: 52353 Index not found.</pre>

5.6. Test Case 6: Day/Time clash with another course

This test case involves checking the following functionalities –

Test Case	Result
Add a student to a course index with available vacancies.	<pre>Enter Course code: CE2005 The available index options are: 65463, Enter Course Index: 65463 Timetable clashes between session! CE2005 Lab clashes with CE2002 Lecture This Index clashes with your Registered Courses! Press enter to continue..</pre>

5.7. Test Case 7: Waitlist Notification

This test case involves checking the following functionalities –

Test Case	Result
Add Student A to a course index with 0 vacancies	<pre>Course Code: CS9680 Index Number: 87654 vacancy number: 0 lecture time: 17:30 to 19:30 lecture venue: CSLab 2 lecture day: 1 Status: Waitlist</pre>

Drop Student B from the same course index	<pre> You have chosen: Drop course, as a student. Enter Course code to drop: CS9680 Press enter to continue.. </pre>
Display Student A's timetable	<pre> Course Code: CS9680 Index Number: 87654 vacancy number: 0 lecture time: 17:30 to 19:30 lecture venue: CSLab 2 lecture day: 1 Status: Registered </pre>

5.8. Test Case 8: Print student list by index number and course

This test case involves checking the following functionalities –

Test Case	Result
Print list by Index number	<pre> Please state your option here: 9 You have chosen: Print Student list by index number, as staff. The available indexes are: 06087, 10305, 10306, 10022, 10021, 65463, 45397, Enter Index number: 10022 Singh Aishwarya Press enter to continue.. </pre>
Print list by Course	<pre> Please state your option here: 10 You have chosen: Print Student list by course (all students registered for the selected course), as staff. The available options are: Number of courses read: 5 CS8207 Fifty Discoveries, Fifty Inventions CE2101 Algorithms Design and Analysis CE2002 Object Oriented Design and Programming CE2005 Operating Systems EE8087 Cyber Security Enter Course code: CE2002 Singh Aishwarya K S Abhinaya Press enter to continue.. </pre>
Invalid data entries	√

6. Demo Video Link –

<https://www.youtube.com/watch?v=ZK30IYK4hFc>

7. References

Java Cryptography - Decrypting Data. (n.d.). Retrieved November 18, 2020, from https://www.tutorialspoint.com/java_cryptography/java_cryptography_decrypting_data.htm

Stack Exchange. (2014, February 8). Working with static constructor in Java. Software Engineering Stack Exchange. <https://softwareengineering.stackexchange.com/questions/228242/working-with-static-constructor-in-java/267311>

What is Sequence Diagram? (2020). Visual Paradigm. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

UML Class Diagram Tutorial. (n.d.). Visual Paradigm. Retrieved 2020, from <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

Janssen, T. (2020, April 28). SOLID Design Principles Explained: Dependency Inversion Principle with Code Examples. Stackify. <https://stackify.com/dependency-inversion-principle/>

Janssen, T. (2020b, April 28). SOLID Design Principles Explained: Interface Segregation with Code Examples. Stackify. <https://stackify.com/interface-segregation-principle/>

Janssen, T. (2020c, April 28). SOLID Design Principles Explained: The Liskov Substitution Principle with Code Examples. Stackify. <https://stackify.com/solid-design-liskov-substitution-principle/>

Janssen, T. (2020d, April 28). SOLID Design Principles Explained: The Open/Closed Principle with Code Examples. Stackify. <https://stackify.com/solid-design-open-closed-principle/>

Java Polymorphism. (n.d.). W3Schools. Retrieved November 18, 2020, from https://www.w3schools.com/java/java_polymorphism.asp

Loton, T. (2001, October 26). JavaMail quick start. InfoWorld. <https://www.infoworld.com/article/2075785/javamail-quick-start.html>