Design Considerations

Approach Taken

During the initial phase of designing the SCRAME application, we decided that it would be the most optimal if the application was to be practical and realistic. As a result, our application aims to mimic closely a real life application, such as our very own school's NTU STARS system. However as STARS have a pre-existing database of courses and students to be built upon, we have to enable such features that allow for the creation of courses and students.

This feature comes with the capability to view and check course vacancies so that only limited students are able to register for the specific course. This extends to the real life aspect of how there are only limited slots for different courses. The courses components, to some extent, takes reference from how our school run its system as well. To elaborate, from a specific course code, there could be various lecture groups, tutorial groups and its respective lab groups (if need be). Within a specific course code, the exam weightage, course weightage and its subcomponents could be adjusted accordingly as well.

The SCRAME application also allows the viewing of student's transcript, where they can check how they fare for a certain module based on their overall grade percentage and can be further broken down into their components and respective sub-components.

Assumptions Made

The time slot for courses and courses offered based on semester period, for example exclusive availability only in first semester was not implemented.
The application, then, is only accessible by the administrator of the application.

Principles Used

As taught in Design Principles, the basis of good design aims to achieve loose coupling and high cohesion between the different classes, while keeping in mind a design with high reusability, extensibility and maintainability. Another important principle would be good encapsulation where each classes are solely responsible for its function and does not require the knowledge of how a separate class operates. In other words, if two classes are dependent on each other, the information presented by the passer should cater to the needs of the passee class.

While constructing the application, we aligned our design very closely to the SOLID Design Principles as these principles aid us in achieving the :

- **S**ingle Responsibility Principle (SRP)
- **O**pen-Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Injection Principle (DIP)

<u>Single Responsibility Principle:</u>

This principle advocates that a single class should exist solely for one responsibility. This would diminish the need to change more than one classes if one class requires certain changes, reducing the snowball effect. In essence, this would achieve the low coupling effect, approaching one of the good design principles. Before embarking on the coding phase, we ensure that our UML Class Diagram acknowledges this principle and adheres to it strictly. This can be seen from our class diagram in the proceeding components of the report.

<u>Open-Closed Principle:</u>

Fundamentally, this principle prescribes to the idea that **Abstraction is the key to OODP**, and that “a module should be open for extension but closed for modification”. This means that a certain class allows for more inheritance from it (if necessary), but the methods of the class are kept consistent so that existing classes that *realises* from it would not be disrupted.

<u>Liskov Substitution Principle:</u>

Barbara Liskov stated that “a user of a base class should continue to function properly if a derivative of that base class is passed to it”. This means that even if arguments were to pass to the subclass instead of the base class, the pre-conditions and postconditions of the subclass method are met. Essentially, it means that the derived class is substitutable for its base class.

Interface Segregation Principles:
This principle champions the idea that many specific interfaces are better than one general purpose interface. This means that while an interface is supposedly meant to be general where many classes implements its methods, the class should not need to implement those methods that they do not use. Therefore, interfaces should be separated based on their specifics function as well. Based on our implementation, we created two boundary interfaces for separate functions. While both classes might be designed for boundary classes, they serve two different functions and relevant classes can implement the interfaces that they need.

Dependency Injection Principle:
Exploring the dependency injection principle, the principle states that high level modules should not depend upon low level modules. Rather, both should depend upon abstractions. This would provide support for pluggable implementations in our application and allows for the building of loosely coupled components.

More often than not, some of these principles are interleaving and are designed concurrently. Accordingly, some of our code implementation applies the idea of a few design principles simultaneously.

Use of Object-Oriented Concepts

Abstraction:
Abstraction is the process of reducing various specific characteristics to a set of essential characteristics. In the process of abstraction, we ensure that an entity contains attributes that describe the state of the object as well as methods that describes the behavior associated with it. For example, the Student entity class only contains essential attributes such as matriculation number and student name which differs for the different student objects created.

Encapsulation/Information Hiding:

Encapsulation protects an object's private data while Information Hiding hides the details and implementation of the class from users. This means that a user only has to understand which methods are available to call and which input parameters are needed, without the need to know how each method is implemented. Other than protecting private data, it helps to facilitate programmers in adding more complex logic on top of a class. For example, the Entity class will contain various methods that describe its associated behavior as well as accessor and mutator methods. With information hiding, the Controller class can simply call these methods without understanding the implementation for those methods. The same goes for a Boundary class that calls methods from a Controller class.

Inheritance:

In OOP, inheritance is the mechanism of basing an object or class upon another object. It is an important feature that allows for the derivation of new classes by absorbing their attributes and behaviour. The derived class can be extended, adding on new capabilities to the classes. This enables code reuse and can greatly reduce the programming effort in implementing new classes. Initially, we planned to have various classes that implements the inheritance feature, where, for example, both 'Students' and 'Professors' is-a *'Human'*. However, while developing the application, we realised that this base entity class is not even necessary and instead of helping us ease the programming effort, it hugely hinders the flow of the program. As such, we compromise on not having the inheritance feature to improve the seamless flow between classes.

Polymorphism:

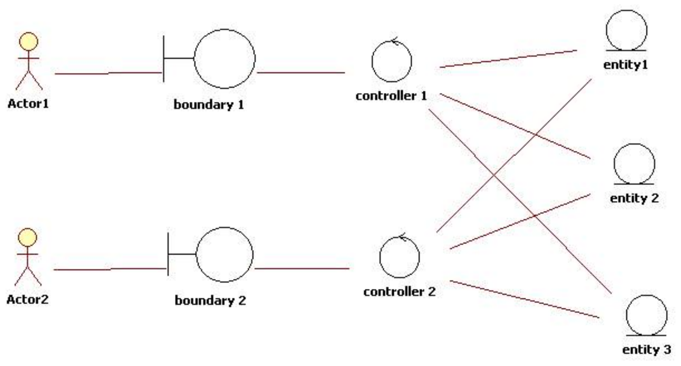
Polymorphism is the ability of an object to take on many forms. In OOP, it means the ability of an object reference being referred to different types; knowing which method to apply depends on where it is in the inheritance hierarchy. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object, where the child class

overrides the methods of the parent’s class by replacing or refining it depending on the needs of the child class.
Polymorphism forms the basis of inherited classes, and since we do not implement any inheritance classes, this feature is largely absent from our application.

Applying Entity-Control-Boundary (ECB) Pattern

Throughout the entire design of the application as well, we made certain that the relationship and the roles of the classes are sustained. Following the “is-a” and “has-a” relationships different classes have different relationships with one another. We ensured that the relationship between each classes (if it exists), to be logical and functional.

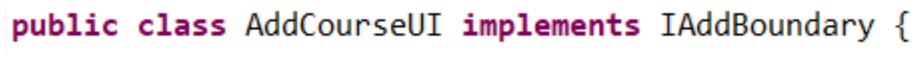
However, more importantly, different classes have their different stereotypes and responsibilities depending on their functions mainly: boundary, control and entity classes.



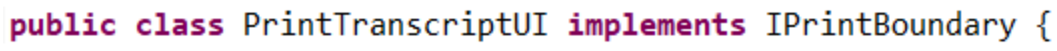
As seen from the diagram above, the separate types of class can only interact with the next level type of class. This means that the user only interacts with the boundary class, boundary class only interacts with controller class and controller class can then interact with entity classes.

Our SCRAME application strictly conforms to the integrity of this structure throughout the implementation.

Important code segment?

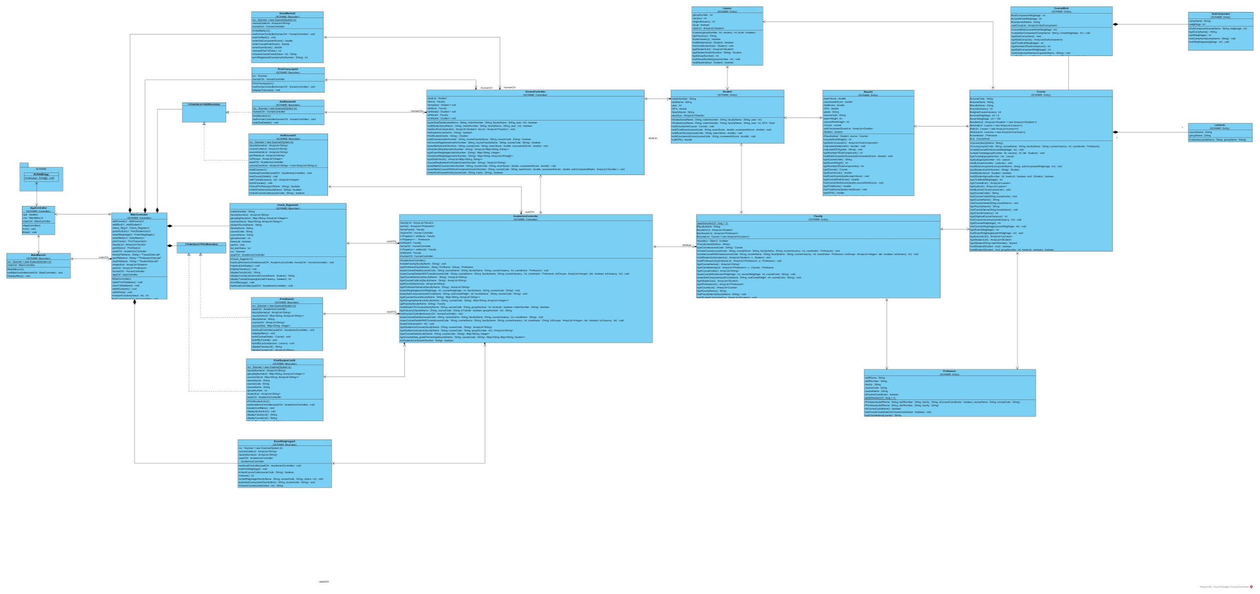


```java
public class AddCourseUI implements IAddBoundary {
```

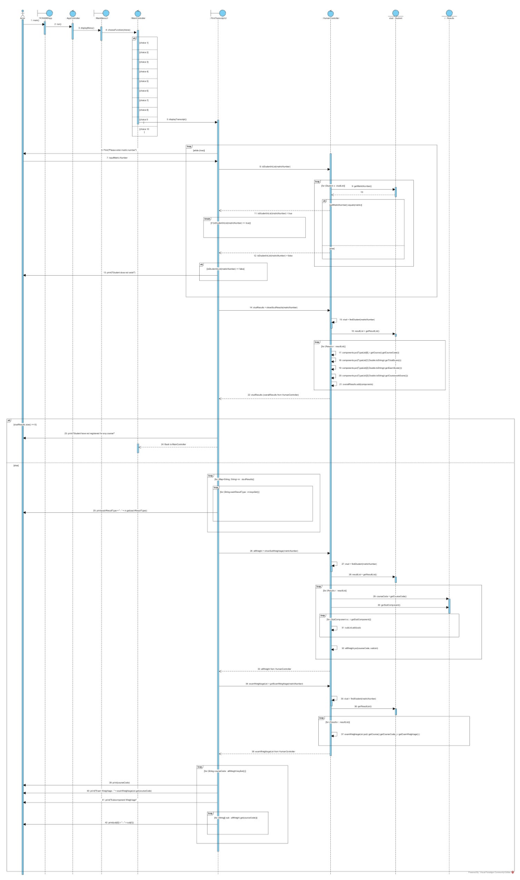


```java
public class PrintTranscriptUI implements IPrintBoundary {
```

Detailed UML Class Diagram



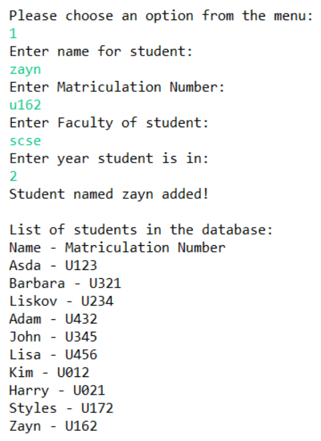
Detailed Sequence Diagram (Print Transcript) next 2 pages



Test Cases and Results

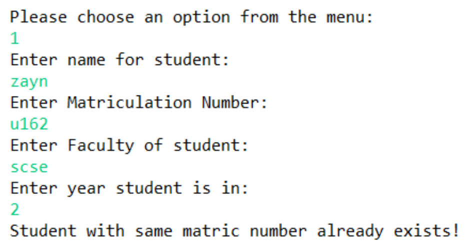
1. Add a student:

   a. Add a new student: (list of **all** students displayed after addition)



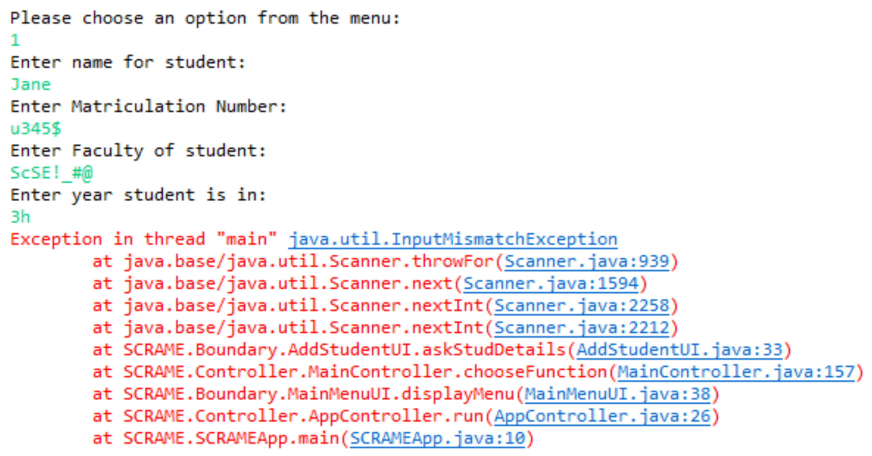
```
Please choose an option from the menu:
1
Enter name for student:
zayn
Enter Matriculation Number:
u162
Enter Faculty of student:
scse
Enter year student is in:
2
Student named zayn added!

List of students in the database:
Name - Matriculation Number
Asda - U123
Barbara - U321
Liskov - U234
Adam - U432
John - U345
Lisa - U456
Kim - U012
Harry - U021
Styles - U172
Zayn - U162
```

   b. Adding existing student: (based on matric number - error message prompt)



```
Please choose an option from the menu:
1
Enter name for student:
zayn
Enter Matriculation Number:
u162
Enter Faculty of student:
scse
Enter year student is in:
2
Student with same matric number already exists!
```

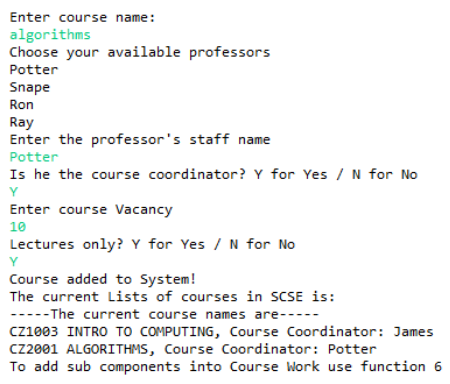
   c. Invalid data entries: (throw new exception)



```
Please choose an option from the menu:
1
Enter name for student:
Jane
Enter Matriculation Number:
u345$
Enter Faculty of student:
ScSE!_#@
Enter year student is in:
3h
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at SCRAME.Boundary.AddStudentUI.askStudDetails(AddStudentUI.java:33)
	at SCRAME.Controller.MainController.chooseFunction(MainController.java:157)
	at SCRAME.Boundary.MainMenuUI.displayMenu(MainMenuUI.java:38)
	at SCRAME.Controller.AppController.run(AppController.java:26)
	at SCRAME.SCRAMEApp.main(SCRAMEApp.java:10)
```

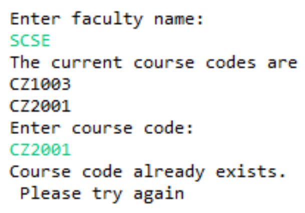
2. Add a course

a. Add a new course: (lists **all** courses with Course Coordinator)



```
Enter course name:
algorithms
Choose your available professors
Potter
Snape
Ron
Ray
Enter the professor's staff name
Potter
Is he the course coordinator? Y for Yes / N for No
Y
Enter course Vacancy
10
Lectures only? Y for Yes / N for No
Y
Course added to System!
The current Lists of courses in SCSE is:
-----The current course names are-----
CZ1003 INTRO TO COMPUTING, Course Coordinator: James
CZ2001 ALGORITHMS, Course Coordinator: Potter
To add sub components into Course Work use function 6
```

b. Adding an existing course: (based on course code - prompt to input again)



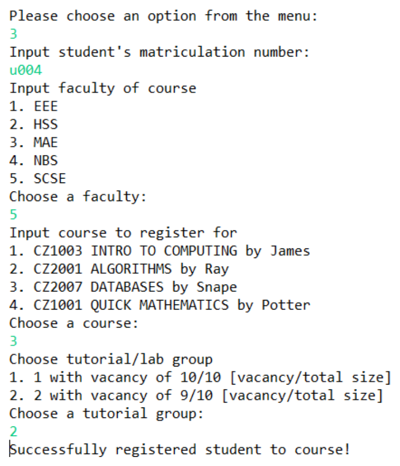
```
Enter faculty name:
SCSE
The current course codes are
CZ1003
CZ2001
Enter course code:
CZ2001
Course code already exists.
 Please try again
```

c. Invalid data entries:

(no way for invalid entry as most entries are string unless code for special chars; ! @#$)

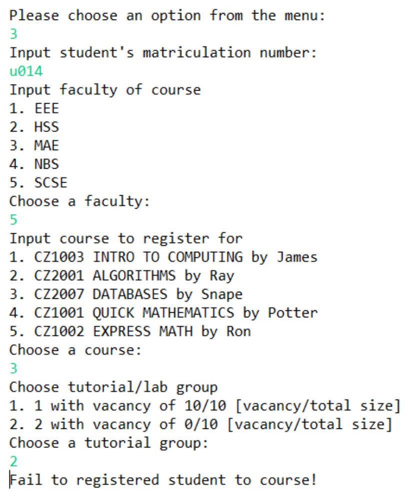
3. Register student for a course

a. Add student to course with available vacancies



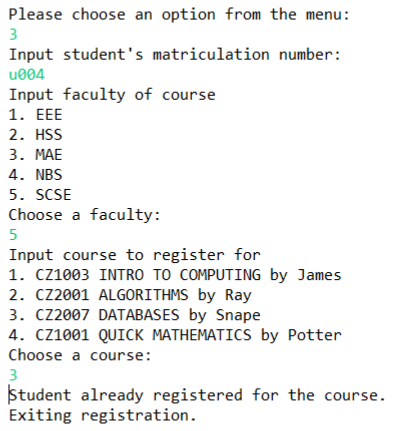
```
Please choose an option from the menu:
3
Input student's matriculation number:
u004
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
Choose a course:
3
Choose tutorial/lab group
1. 1 with vacancy of 10/10 [vacancy/total size]
2. 2 with vacancy of 9/10 [vacancy/total size]
Choose a tutorial group:
2
Successfully registered student to course!
```

b. Add student to course with 0 vacancies



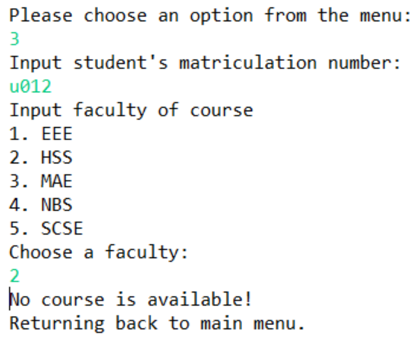
```
Please choose an option from the menu:
3
Input student's matriculation number:
u014
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
Choose tutorial/lab group
1. 1 with vacancy of 10/10 [vacancy/total size]
2. 2 with vacancy of 0/10 [vacancy/total size]
Choose a tutorial group:
2
Fail to registered student to course!
```

c. Register for the same course again



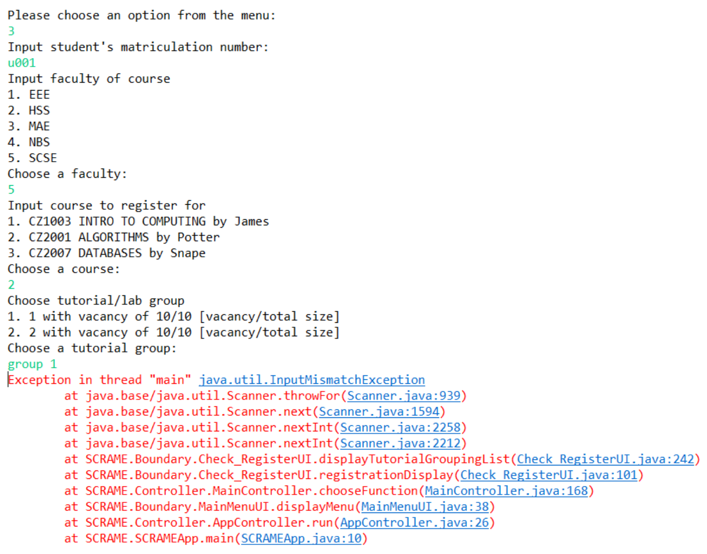
```
Please choose an option from the menu:
3
Input student's matriculation number:
u004
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
Choose a course:
3
Student already registered for the course.
Exiting registration.
```

d. Register for course in faculty (that has no courses added yet)



```
Please choose an option from the menu:
3
Input student's matriculation number:
u012
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
2
No course is available!
Returning back to main menu.
```

e. Invalid data entries

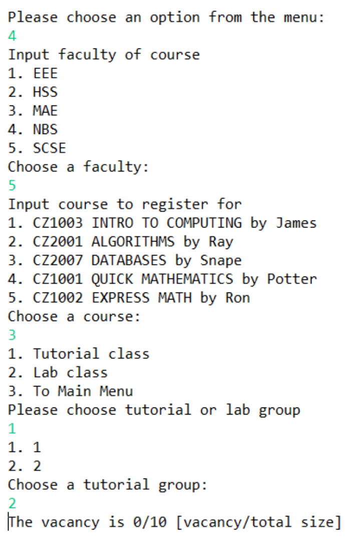


```
Please choose an option from the menu:
3
Input student's matriculation number:
u001
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Potter
3. CZ2007 DATABASES by Snape
Choose a course:
2
Choose tutorial/lab group
1. 1 with vacancy of 10/10 [vacancy/total size]
2. 2 with vacancy of 10/10 [vacancy/total size]
Choose a tutorial group:
group 1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at SCRAME.Boundary.Check_RegisterUI.displayTutorialGroupingList(Check_RegisterUI.java:242)
	at SCRAME.Boundary.Check_RegisterUI.registrationDisplay(Check_RegisterUI.java:101)
	at SCRAME.Controller.MainController.chooseFunction(MainController.java:168)
	at SCRAME.Boundary.MainMenuUI.displayMenu(MainMenuUI.java:38)
	at SCRAME.Controller.AppController.run(AppController.java:26)
	at SCRAME.SCRAMEApp.main(SCRAMEApp.java:10)
```

4. Check available slot in class

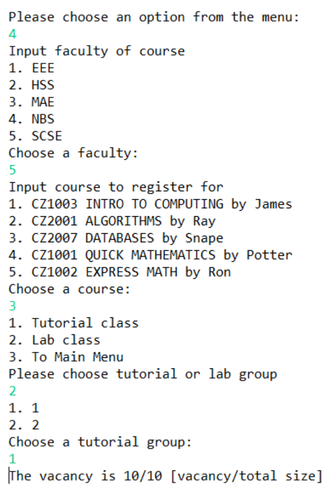
a. Check for vacancy:

i. Tutorial class



```
Please choose an option from the menu:
4
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
1. Tutorial class
2. Lab class
3. To Main Menu
Please choose tutorial or lab group
1
1. 1
2. 2
Choose a tutorial group:
2
The vacancy is 0/10 [vacancy/total size]
```

ii. Lab class



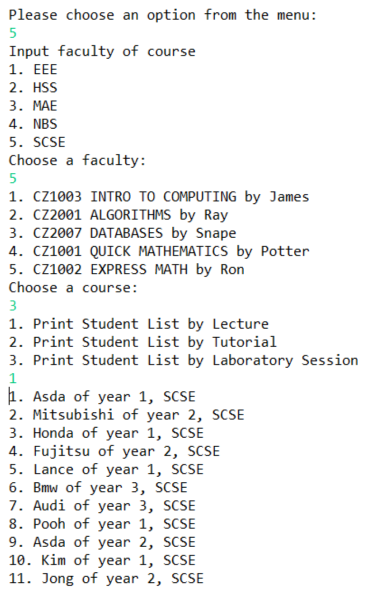
```
Please choose an option from the menu:
4
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
Input course to register for
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
1. Tutorial class
2. Lab class
3. To Main Menu
Please choose tutorial or lab group
2
1. 1
2. 2
Choose a tutorial group:
1
The vacancy is 10/10 [vacancy/total size]
```

b. Invalid data entries

5. Print student list by lecture, tutorial or lab for a course

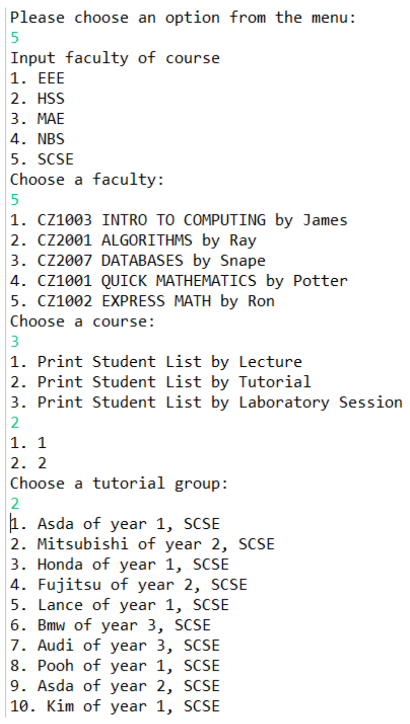
a. Print list by:

i. Lecture



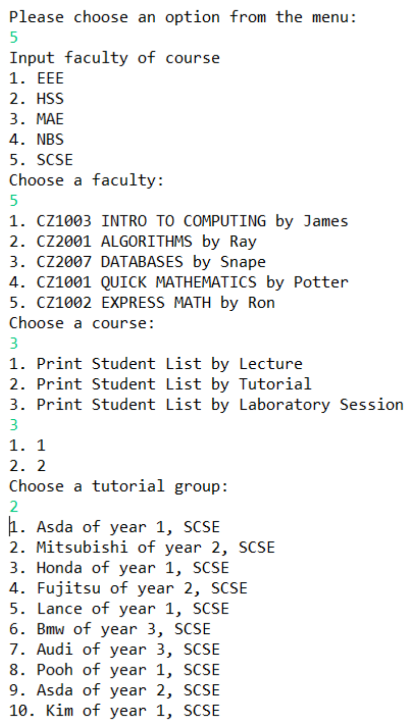
```
Please choose an option from the menu:
5
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
1. Print Student List by Lecture
2. Print Student List by Tutorial
3. Print Student List by Laboratory Session
1
1. Asda of year 1, SCSE
2. Mitsubishi of year 2, SCSE
3. Honda of year 1, SCSE
4. Fujitsu of year 2, SCSE
5. Lance of year 1, SCSE
6. Bmw of year 3, SCSE
7. Audi of year 3, SCSE
8. Pooh of year 1, SCSE
9. Asda of year 2, SCSE
10. Kim of year 1, SCSE
11. Jong of year 2, SCSE
```

ii. Tutorial group



```
Please choose an option from the menu:
5
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
1. Print Student List by Lecture
2. Print Student List by Tutorial
3. Print Student List by Laboratory Session
2
1. 1
2. 2
Choose a tutorial group:
2
1. Asda of year 1, SCSE
2. Mitsubishi of year 2, SCSE
3. Honda of year 1, SCSE
4. Fujitsu of year 2, SCSE
5. Lance of year 1, SCSE
6. Bmw of year 3, SCSE
7. Audi of year 3, SCSE
8. Pooh of year 1, SCSE
9. Asda of year 2, SCSE
10. Kim of year 1, SCSE
```

iii. Lab

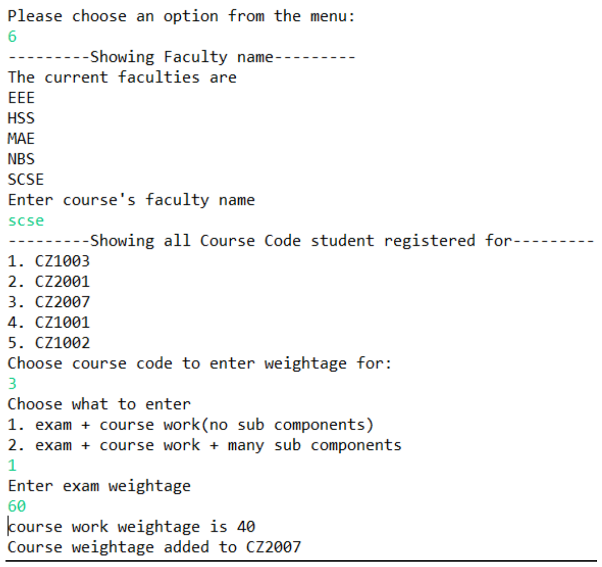


```
Please choose an option from the menu:
5
Input faculty of course
1. EEE
2. HSS
3. MAE
4. NBS
5. SCSE
Choose a faculty:
5
1. CZ1003 INTRO TO COMPUTING by James
2. CZ2001 ALGORITHMS by Ray
3. CZ2007 DATABASES by Snape
4. CZ1001 QUICK MATHEMATICS by Potter
5. CZ1002 EXPRESS MATH by Ron
Choose a course:
3
1. Print Student List by Lecture
2. Print Student List by Tutorial
3. Print Student List by Laboratory Session
3
1. 1
2. 2
Choose a tutorial group:
2
1. Asda of year 1, SCSE
2. Mitsubishi of year 2, SCSE
3. Honda of year 1, SCSE
4. Fujitsu of year 2, SCSE
5. Lance of year 1, SCSE
6. Bmw of year 3, SCSE
7. Audi of year 3, SCSE
8. Pooh of year 1, SCSE
9. Asda of year 2, SCSE
10. Kim of year 1, SCSE
```

b. Invalid data entries

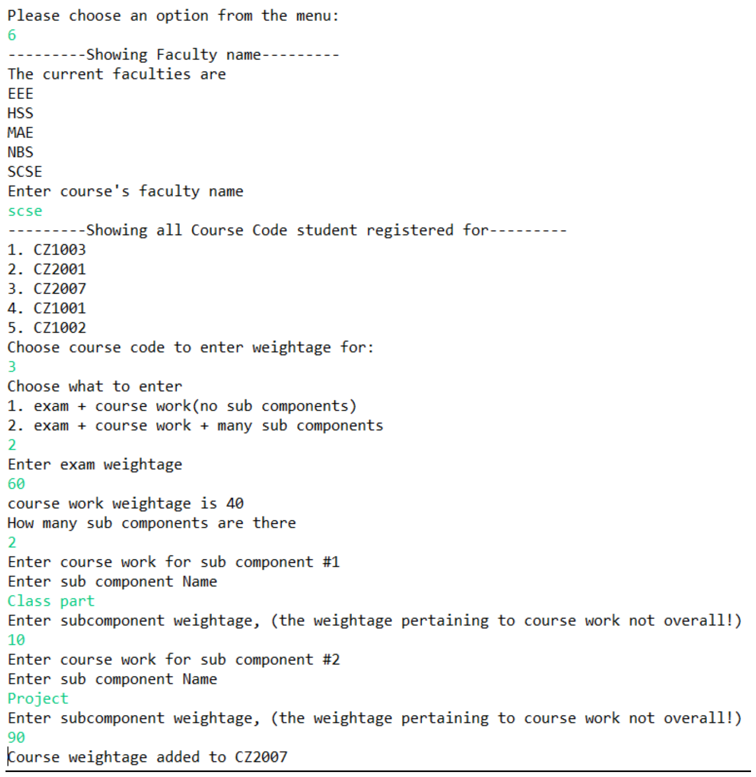
6. Enter course assessment component weightages

   a. Course assessment with only exam + 1 main coursework component without subcomponents



```
Please choose an option from the menu:
6
---------Showing Faculty name---------
The current faculties are
EEE
HSS
MAE
NBS
SCSE
Enter course's faculty name
scse
---------Showing all Course Code student registered for---------
1. CZ1003
2. CZ2001
3. CZ2007
4. CZ1001
5. CZ1002
Choose course code to enter weightage for:
3
Choose what to enter
1. exam + course work(no sub components)
2. exam + course work + many sub components
1
Enter exam weightage
60
course work weightage is 40
Course weightage added to CZ2007
```

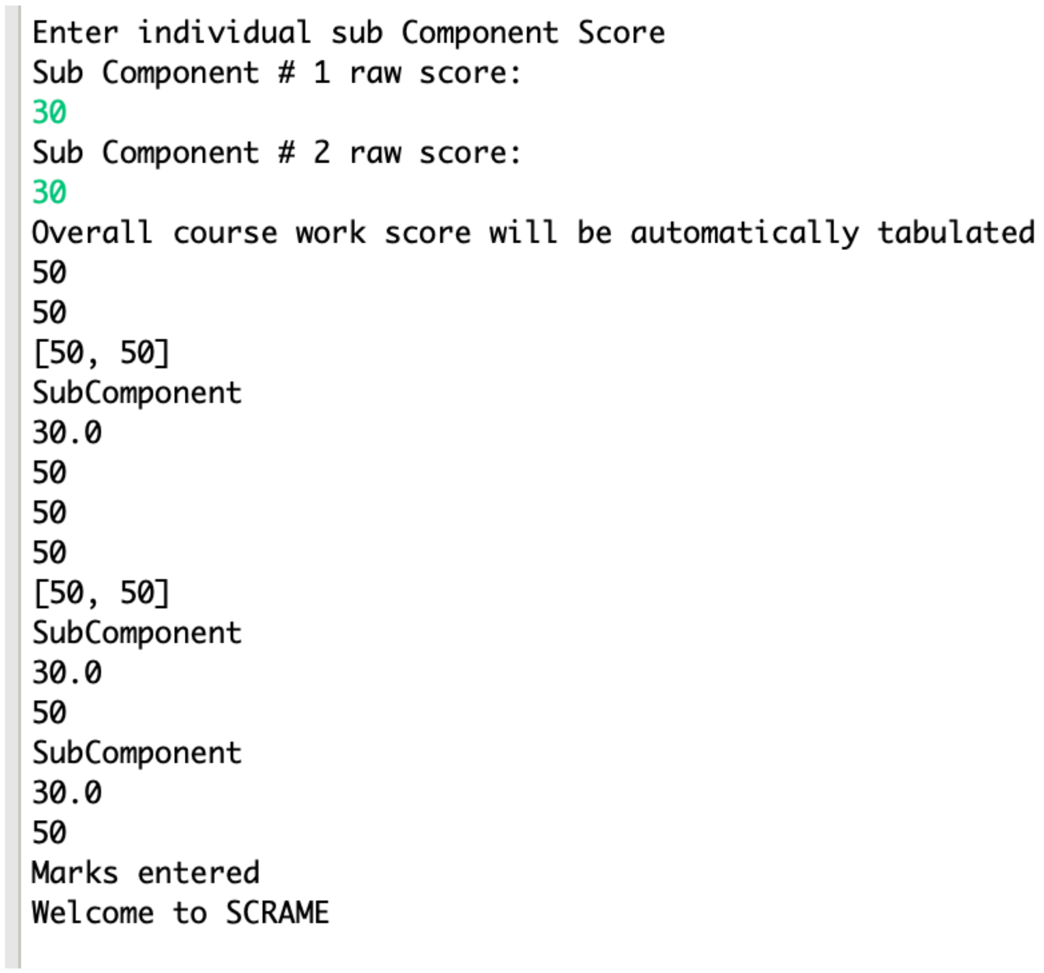
   b. Course assessment with exam + 1 coursework with 2 subcomponents



```
Please choose an option from the menu:
6
---------Showing Faculty name---------
The current faculties are
EEE
HSS
MAE
NBS
SCSE
Enter course's faculty name
scse
---------Showing all Course Code student registered for---------
1. CZ1003
2. CZ2001
3. CZ2007
4. CZ1001
5. CZ1002
Choose course code to enter weightage for:
3
Choose what to enter
1. exam + course work(no sub components)
2. exam + course work + many sub components
2
Enter exam weightage
60
course work weightage is 40
How many sub components are there
2
Enter course work for sub component #1
Enter sub component Name
Class part
Enter subcomponent weightage, (the weightage pertaining to course work not overall!)
10
Enter course work for sub component #2
Enter sub component Name
Project
Enter subcomponent weightage, (the weightage pertaining to course work not overall!)
90
Course weightage added to CZ2007
```

   c. Invalid data entries

7. Enter coursework mark - inclusive of its components



```
Enter individual sub Component Score
Sub Component # 1 raw score:
30
Sub Component # 2 raw score:
30
Overall course work score will be automatically tabulated
50
50
[50, 50]
SubComponent
30.0
50
50
50
[50, 50]
SubComponent
30.0
50
SubComponent
30.0
50
Marks entered
Welcome to SCRAME
```

   a. Enter valid coursework mark for course with only 1 main component

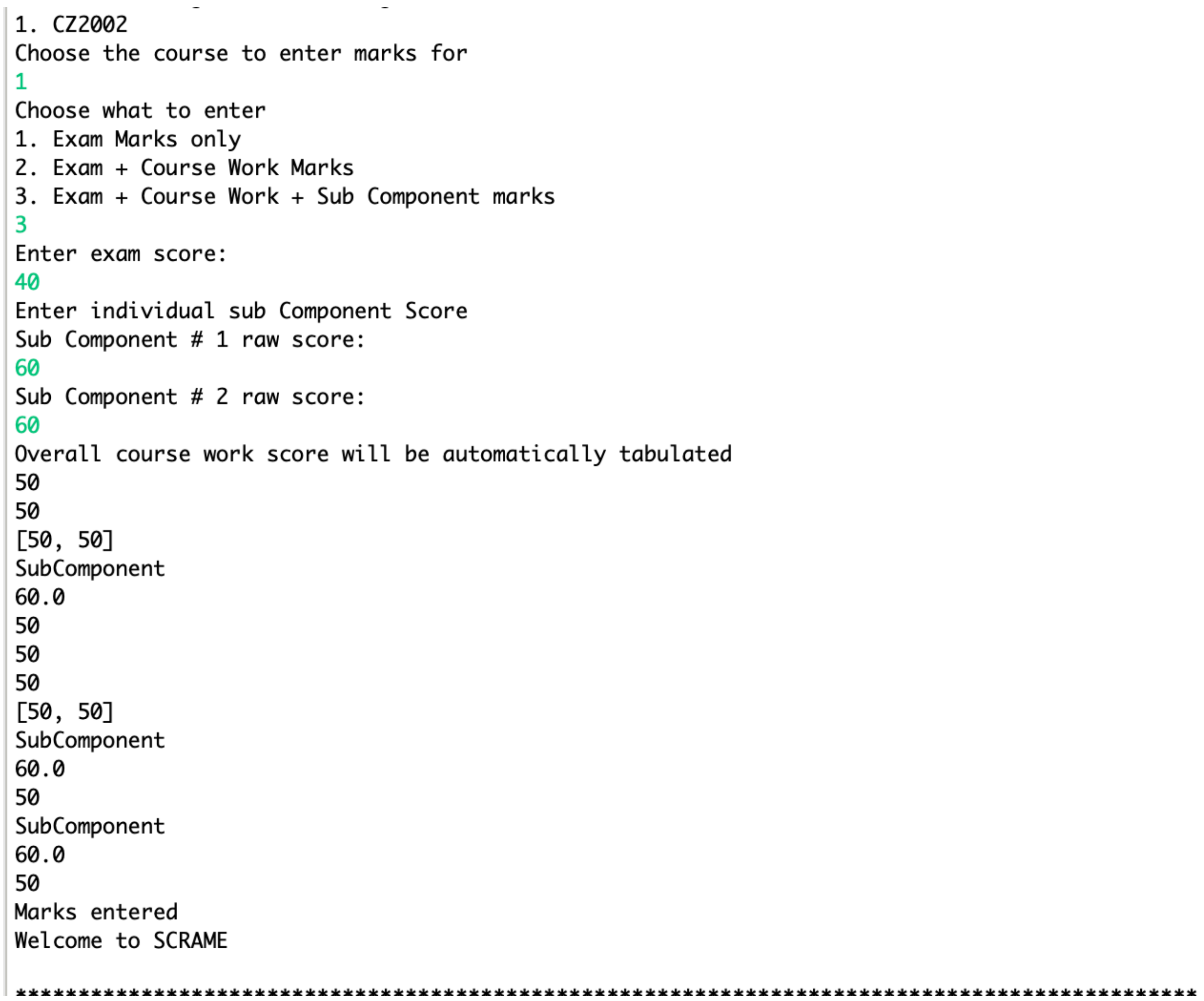
   b. Enter valid coursework mark for course with 2 sub-components

   c. Invalid data entries

8. Enter exam mark

   a. Enter valid exam mark for valid course

   b. Invalid data entries



```
1. CZ2002
Choose the course to enter marks for
1
Choose what to enter
1. Exam Marks only
2. Exam + Course Work Marks
3. Exam + Course Work + Sub Component marks
3
Enter exam score:
40
Enter individual sub Component Score
Sub Component # 1 raw score:
60
Sub Component # 2 raw score:
60
Overall course work score will be automatically tabulated
50
50
[50, 50]
SubComponent
60.0
50
50
50
[50, 50]
SubComponent
60.0
50
SubComponent
60.0
50
Marks entered
Welcome to SCRAME

**********************************************************************************************
```

References & Tools

- UML Diagram Tools - Visual Paradigm https://www.visual-paradigm.com/
- NTULearn CZ2002 main course site content