SOFTENG754

SOFTWARE REQUIREMENTS ENGINEERING

ASSIGNMENT 4: Course Enrolment System for a University using TDD and CI practices

Submitted by: Group 9

Member Name	UPI	Github ID	Reponsible for Pages
Ahmed Suhail	asuh702	iahmedsuhail	1 to 12 23 to 25
Khalid Ali	kali435	kali435	18 to 22
Yue Xu	yxu314	yxu314	13 to 17

Date of Submission: 22 May 2020

Link to private GitHub repository: https://github.com/iahmedsuhail/softeng754-a4

Note: Some TravisCI builds that were supposed to fail, are passing in the repository. This was caused because of gradle errors, that were fixed midway through the project. TDD principles were still followed by checking the tests from IDE. A discussion on this can be found here.

Introduction

In Assignment 2, the team created a set of user stories documenting the requirements for a Course Enrolment System for a University. In this assignment, the team implements this product using TDD and CI techniques. Only stories with Must-Have priority from A2 were implemented. Functionality for the Back end is developed in this iteration.

Must Have User Stories

Mis prioritisation in Assignment 2

"Requirements labelled as Should have are important but not necessary for delivery in the current delivery timebox. While Should have requirements can be as important as Must have, they are often not as time-critical or there may be another way to satisfy the requirement so that it can be held back until a future delivery timebox" (International Institute of Business Analysis, 2008)

In the first iteration of development of the Backend for the course enrolment system, it was deemed that the following user stories deserved priorities of should-have, in that the system would not break if these were not implemented: -

ID	**User Story**	**Reason for mis-prioritising**	**Action**
01-14	Single-Sign on	Redundant user story. The ability to sign on using existing credentials is better suited as acceptance criteria for user story #03-03.	The user story was removed.
01-06	As a student, I want to view formulated timetables, so that I can see which course fits my schedule.	The ability to formulate schedules is a non-critical functionality for the system, in that another way to satisfy the requirements of the system is simply enrolling into a course searching through the course catalogue.	The priority of the story was changed to Should Have.
01-20	As a student, I want the mandatory courses for my program to be automatically selected, so that I don't waste time on selecting mandatory subjects.	The ability of the system to automatically select courses that are mandatory in a given course is another non-critical functionality for the system, in that there are usually other ways to enforce this requirement for the system. It is hence better off developed in a later sprint, and not in this iteration.	The priority of the story was changed to Should Have.

While some of the stories were over-estimated as Must- Haves, no story was found to be under-estimated. In other words, no story whose inclusion was critical was left out of the system in Assignment 2.

Revision of Acceptance Criteria

Acceptance criteria for some stories were not as specific, detailed as required for a suitable system. The acceptance criteria of the following stories were revised: -

User Story ID	User Story	Old Acceptance Criteria	New Acceptance Criteria
02-04	As an admin, I want to be able to set course restrictions, so that there are guidelines to get into a course	- Admins should be able to set criteria such as maximum number of students, prerequisites when setting guidelines on a course, availability. - Admins should be able to confirm that there are no impacts of the guidelines on the teaching team.	- Admins should be able to limit the maximum number of students in a course. - Admins should be able to check for prerequisites of a course for a student wanting to enrol into a course. - Admins should be able to set the availability for a course - The system shall enforce these restrictions.
01-02	As a student, I want to be able to apply for concessions whenever the requirements enforced by the system are not met, so the admin can make decisions on my enrollment manually.	- Students should be able to see information regarding the course, such as Availability, Class, Description, Instructor - Students should be able to submit enrolment concession requests, with a description of why they want the concession - Students should be able to see some help regarding submitting enrolment concessions	- Students should be able to see the instructor for a course, along with the information. - Students should be able to see help regarding enrolment through the system. - Students should be able to apply for enrolment concessions for any course run by the university - Students should be able to submit enrolment concession requests, with a description of why they want the concession
03-03	The system should integrate with the current university systems, so that students have a streamlined experience of trying to access other university applications.	Missing Acceptance Criteria	- Persons should be able to enter University SSO login details for logging into the system - Persons should be able to login into the system - If incorrect details are entered, persons should not be able to login into the system.
01-13	As a student, I should be able to see the list of courses I have submitted for concessions, so	Students should be able to see course details of the courses they	Students should be able to see the course name and the course code of the courses they have applied concessions for.

	that I can see which courses I am enrolled in.	have applied concessions for.	
01-01	As a student, I want all concession requests to be handled before the enrollment starts, so that the enrollment process can be first come first serve to ensure transparency.	The process of concessions, should be handled before the process of enrolment begins.	- The process of concessions, should be handled before the process of enrolment begins. - Enrolment concession should be first come first serve to ensure fairness and transparency.
01-10	As a student, I want to be able to search for a class, so that I can add courses to my schedule in multiple ways.	Students should be able to search for a class using multiple options.	Students should be able to search for a class using any of the following: Course Name Course Code Instructor

Change of User Story

Following user story was deemed limiting the solution scope for developers and was hence revised.

User Story ID	**Old User Story**	**New User Story**
01-01	As a student, I want all concession requests to be handled before the enrollment starts, so that the enrollment process can be first come first serve to ensure transparency.	As a student, I want all concession requests to be handled before the enrolment starts, so that the enrolment process is fair.

Final User Stories for A4

Following is a table of all the Must-Have Stories for the Course Enrolment System, with their acceptance criteria: -

User Story ID	User Story	Acceptance Criteria
01-07	As a student, I want to be able to see my current enrollment status, so that I can see the courses I am enrolled in.	- Students should be able to see course name,course code, enrollment status.
01-11	As a student, I want to be able to enrol into courses, so I can work my way towards getting a degree.	- Students should be able to enrol into courses. - Students should be able to see related information such as fee liabilities while confirming enrolment.
03-03	The system should integrate with the current university systems, so that students have a streamlined experience of trying to access other university applications.	- Persons should be able to enter University SSO login details for logging into the system - Persons should be able to login into the system - If incorrect details are entered, persons should not be able to login into the system.
02-04	As an admin, I want to be able to set course restrictions, so that there are guidelines to get into a course.	- Admins should be able to limit the maximum number of students in a course. - Admins should be able to check for prerequisites of a course for a student wanting to enrol into a course. - Admins should be able to set the availability for a course - The system shall enforce these restrictions.
03-05	As a student, I want to be able to see the list of courses, so that I can look at the options I have for taking courses.	- Students should be able to see course code, course name, points, availability, lecture times, lecturer name. - Students should be able to sort the courses by departments and specialisations.
01-01	As a student, I want all concession requests to be handled before the enrolment starts, so that the enrolment process is fair.	- The process of concessions, should be handled before the process of enrolment

		begins. - Enrolment concession should be first come first serve to ensure fairness and transparency.
01-02	As a student, I want to be able to apply for concessions whenever the requirements enforced by the system are not met, so the admin can make decisions on my enrollment manually.	- Students should be able to see course information as maintained by the course catalog - Students should be able to see the instructor for a course. - Students should be able to see help regarding enrolment through the system. - Students should be able to apply for enrolment concessions for any course run by the university - Students should be able to submit enrolment concession requests, with a description of why they want the concession
01-13	As a student, I should be able to see the list of courses I have submitted for concessions, so that I can see which courses I am enrolled in.	- Students should be able to see the course name and the course code of the courses they have applied concessions for.
02-02	As an admin, I want to approve concessions, so that students can get into courses.	- Admins should be able to see Student info, such as ID, name while making the decision - Admins should be able to see Course Info like Program Department, Title, Course code and requirements enforced by the university. - Admins should be able to approve/reject requests. - The system shall send a notification to the student regarding the concession decision made by admin.

01-10	As a student, I want to be able to search for a class, so that I can add courses to my schedule in multiple ways.	Students should be able to search for a class using any of the following: - Course Name - Course Code - Instructor
01-12	As a student, I should be able to drop out of courses, so that I can get out of the courses I don't want to take anymore.	- Students should be able to see their current enrolments. - Students should be able to drop out of courses.

Implementation

User Story ID: 01-07

User Story Title: Current Courses

User Story: As a student, I want to be able to see my current enrolment status, so that I can see the courses I am enrolled in.

Acceptance Criteria

- Students should be able to see course name, course code, enrollment status.

```
    @org.junit.Test
    public void Given_StudentIsEnrolledInACourse_Expect_StudentCanSeeTheEnrolledCourses() {
        Student student = new Student();

//          Create 3 new courses
        Course course = new Course( name: "Software Requirements Engineering", code: "SOFTENG754", points: 15, availability: true);
        Course course1 = new Course( name: "High Performance Computing", code: "SOFTENG751", points: 15, availability: true);
        Course course2 = new Course( name: "Software Tools and Techniques", code: "SOFTENG750", points: 15, availability: true);

//          Enroll the student into the courses
        student.enrol(course);
        student.enrol(course1);
        student.enrol(course2);

//          Result object for comparison
        ArrayList<Course> resultEnrolledCourses = new ArrayList<>();
        resultEnrolledCourses.add(course);
        resultEnrolledCourses.add(course1);
        resultEnrolledCourses.add(course2);

        assertEquals(resultEnrolledCourses, student.getEnrolledCourses());
    }
```

- Note that mocking was not used here because the ability to enrol into the course was developed prior to this test. One could argue that mocks should've been used rather than the original classes for the tests to only test the said functionality (which is a core principle for Unit tests), but it could also be argued that the said functionality was tested prior as well.

*Further in the discussions section

Integration Test

```
@Category(IntegrationTests.class)
@Test
public void test_Student_CanSubmit_EnrollmentConcession_ForA_Course_WithA_Teacher(){
    Student student = new Student();
    Teacher teacher = new Teacher( name: "Kelly Blincoe");
    Course course = new Course( name: "Software Requirements Engineering",
            code: "SOFTENG754", points: 15, availability: true, teacher, fee: 500);

    student.enrol(course);

    ArrayList<Course> resultEnrolledCourses = new ArrayList<>();
    resultEnrolledCourses.add(course);

    assertEquals(resultEnrolledCourses, student.getEnrolledCourses());
}
```

User Story ID: 01-11

User Story Title: Ability to enrol

User Story: As a student, I want to be able to enrol into courses, so I can work my way towards getting a degree.

Acceptance Criteria

- Students should be able to enrol into courses.

```
@org.junit.Test
public void testCanEnrolIntoCourses() {
    Student student = new Student();
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);

    List<Course> resultEnrolledCourses = new ArrayList<>();
    resultEnrolledCourses.add(course);

    student.enrol(new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true));

      Won't work probably because the they aren't the same objects in memory
      assertEquals(resultEnrolledCourses, student.getEnrolledCourses());

      This test gets details (as a string) of the enrolledCourses and the course we just added
    assert(student.getEnrolledCourses().get(student.enrolledCourses.size()-1).showInfo().equals(resultEnrolledCourses.get(0).showInfo()));
}
```

- Note: When this test case was committed to the repository and then code for it was written, it was deemed that the assertEquals was not the right way to compare objects in memory, and that two ArrayLists<> shouldn't have been compared. Instead, the test was changed to correctly compare the results.
- Students should be able to see related information such as fee liabilities while confirming enrolment.

```
@org.junit.Test
public void Given_StudentEnrolsIntoACourse_Expect_TheFeeOwedByTheStudentIncreases(){
    Student student = new Student();
    Teacher teacher = new Teacher( name: "Kelly Blincoe");
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true, teacher,  fee: 500);

    student.enrol(course);

    assertEquals( expected: 500, student.feeLiable);
}
```

- Note: Mocking for testing only the code under test here was not suitable, because both the student and course classes had aspects that were supposed to be tested.

User Story ID: 03-03

User Story Title: Integrate with University Systems

User Story: The system should integrate with the current university systems, so that students have a streamlined experience of trying to access other university applications.

Acceptance Criteria

- Persons should be able to enter University SSO login details for logging into the system:
  This acceptance criteria could not be verified because it depends on an external system.
- Persons should be able to login into the system

```
@org.junit.Test
public void testPersonCanLogin() {
    Person person = new Person();
    Controller.login(person, username: "username", password: "password");
    assertTrue(person.isLoggedIn());
}
```

- If incorrect details are entered, persons should not be able to login into the system.

```
@Test
public void Given_PersonEntersIncorrectDetails_Expect_PersonShouldNotBeAbleToLogin() {
    Person person = new Person();
    Controller.login(person, username: "IncorrectUsername", password: "IncorrectPassword");
    assertFalse(person.isLoggedIn());
}
```

User Story ID: 03-05

User Story Title: System Catalog

User Story: As a student, I want to be able to see the list of courses, so that I can look at the options I have for taking courses.

Acceptance Criteria

- Students should be able to see course code, course name, points, availability, lecture times, lecturer name.

```
@org.junit.Test
public void testCourseDetailsCanBeSeen() {
    Course course = new Course( name: "Software Requirements Engineering", code: "SOFTENG754", points: 15, availability: true);
    assertEquals( expected: "Software Requirements Engineering", course.getName());
    assertEquals( expected: "SOFTENG754", course.getCode());
    assertEquals( expected: 15, course.getPoints());
    assertEquals( expected: true, course.getAvailability());
}
```

- Students should be able to sort the courses by departments and specialisations.
  This functionality belongs to the front end of the application, and hence couldn't be fulfilled.

User Story ID: 01-02

User Story Title: Apply for concession

User Story: As a student, I want to be able to apply for concessions whenever the requirements enforced by the system are not met, so the admin can make decisions on my enrollment manually.

Acceptance Criteria

- Students should be able to see the instructor for a course, along with the information.

```
@org.junit.Test
public void testCourseDetailsCanBeSeen() {
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);
    assertEquals( expected: "Software Requirements Engineering", course.getName());
    assertEquals( expected: "SOFTENG754", course.getCode());
    assertEquals( expected: 15, course.getPoints());
    assertEquals( expected: true, course.getAvailability());
}
```

```
@org.junit.Test
public void testCanSeeInstructor() {
     Create a new course with an instructor.
    Teacher teacher = new Teacher( name: "Kelly Blincoe");
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true, teacher);

    assertEquals( expected: "Kelly Blincoe", course.getInstructor());
}
```

- Students should be able to see help regarding enrolment through the system.
  This acceptance criteria is purely for the Front-End of the application.
- Students should be able to apply for enrolment concessions for any course run by the university

```
    @org.junit.Test
    public void testCanSubmitEnrollmentConcession() {
//        Mocked student because the getName function doesn't exist
        Student mockedStudent = Mockito.mock(Student.class);
        Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);

//        Create a new Enrollment Concession
        EnrollmentConcession ec = new EnrollmentConcession(mockedStudent, course);

        Mockito.when(mockedStudent.getName()).thenReturn("Ahmed");
        assertEquals( expected: "Ahmed wants to enrol in SOFTENG754", ec.getDetails());
    }
```

- Students should be able to submit enrolment concession requests, with a description of why they want the concession

```
@org.junit.Test
public void Given_StudentSubmitsEnrollmentConcession_Expect_StudentCanSubmitDescription(){
    Student student = new Student();
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);

    EnrollmentConcession ec = new EnrollmentConcession(student, course,  description: "Prereqs were done in another institution");

    assertEquals( expected: "Prereqs were done in another institution", ec.getDescription());
}
```

User Story ID: 02-04

User Story Title: Course Guidelines

User Story: As an admin, I want to be able to set course restrictions, so that there are guidelines to get into a course.

Acceptance Criteria

- Admins should be able to limit the maximum number of students in a course.
- Admins should be able to set prerequisite restrictions for a course.
- Admins should be able to set the availability for a course

```
@org.junit.Test
public void testAdminCanSetGuidelines() {
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);
      setting maximum number of students that can get into a course
    course.setMaxNumberOfStudents(50);
    assertEquals( expected: 50, course.getMaxNumberOfStudents());

      set the prerequisites for a course
    course.addPrereqs( code: "SOFTENG251");
    ArrayList<String> resultPrereqs = new ArrayList<>();
    resultPrereqs.add("SOFTENG251");
    assertEquals(resultPrereqs, course.getPrereqs());

      set availability
    course.setAvailability(false);
    assertEquals( expected: false, course.getAvailability());
}
```

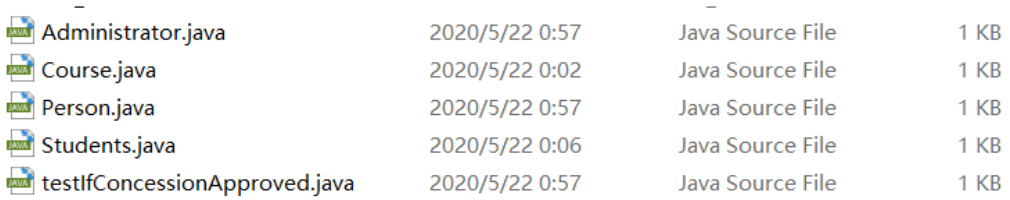
- The system shall enforce these restrictions.

```
@org.junit.Test
public void Given_CourseIsNotAvailable_Expect_StudentsCantBeEnrolledIntoCourse(){
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);
    course.setAvailability(false);

    Student student = new Student();
    ArrayList<Course> emptyListOfCourses = new ArrayList<>();

    student.enrol(course);

    assert(emptyListOfCourses.equals(student.getEnrolledCourses()));
}
```

User story: concession approval

Use case: As an administrator, want to approve the concession of students.

There are five classes I have created which consist of administrator.java; course.java; Person.java; Students.java; TestifConcessionApproved.java.



Administrator.java	2020/5/22 0:57	Java Source File	1 KB
Course.java	2020/5/22 0:02	Java Source File	1 KB
Person.java	2020/5/22 0:57	Java Source File	1 KB
Students.java	2020/5/22 0:06	Java Source File	1 KB
testIfConcessionApproved.java	2020/5/22 0:57	Java Source File	1 KB

Administrator.java | Course.java | Person.java | Students.java | testIfConcessionApproved....

```
public class Students extends Person{

    public Students(int ID, String name, int age) {
        super(ID, name, age);
        // TODO Auto-generated constructor stub
    }

    public String getStuInfo() {
        return "Student information: student ID is: " + ID + ", student name is: " + name +
                ", Student age is :" + age;
    }



}
```

```
public class testIfConcessionApproved {


    public static void main(String args[]) {


    //check student information
    Students stu1 = new Students(131992,"Hanhan",24);
    String info = stu1.getStuInfo();
    System.out.println(info);
    System.out.println("-----------------------------------");
```

First of all, get the student information and Determine whether a student is eligible to apply for the concession.

```
///check course info
Course course1 = new Course("SXAU","General Economic","COMP722",
        "pass with grades above 80%","Amanda");
String info2 = course1.getCourseInfo();
System.out.println(info2);
System.out.println("-----------------------------------");
```

Then, get the course information.

```
//set or reject the course
System.out.println(course1.getStatus());
```

Check if the concession is approved.

```
public class Course {

    String programDepartment;
    String title;
    String courseCode;
    String requirements;
    String stuNameOfThisCourse;
    Boolean status;

    public Course(String programDepartment, String title,String courseCode,String requirements,
            String stuNameOfThisCourse) {
        this.programDepartment = programDepartment;
        this.title = title;
        this.courseCode = courseCode;
        this.requirements = requirements;
        this.stuNameOfThisCourse = stuNameOfThisCourse;
        this.status = false;
```

The default course status is false.

```
System.out.println(course1.getStatus());
course1.setStatus(true);
System.out.println(course1.getStatus());
System.out.println("-----------------------------------");
```

Change the status of the concession to “true”.

```
public String getStuName() {
    return stuNameOfThisCourse;
}

public void setStatus(boolean decision) {
    status = decision;
}
public boolean getStatus() {
    return status;
}

public String  getCourseInfo() {
    return "Course Information: ProgramDepartmant is:" + programDepartment +
            ", Title is: "+ title + ", Course code: " + courseCode +  ", Requirements"+requir
            +", student name of taking this course" + stuNameOfThisCourse;

}
```

Update the current status of the course.

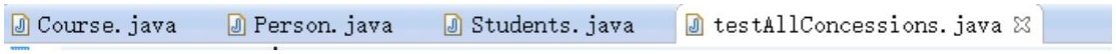
```
//set or reject the course
System.out.println(course1.getStatus());
course1.setStatus(true);
System.out.println(course1.getStatus());
System.out.println("-----------------------------------");
assertThat(true,course1.getStatus());
```

TDD: Use one of the JUnit methods assertThat to test if the result is true.

User story: concession list

In this part, I will intend to test a student whose name is Hong checks the list of his courses submitted for concession.

At the beginning, I created four classes which consist of Corse, Person, Students, and testAllConcessions.



In the Course class, it contains all the basic information of course such as programDepartment, course code, and etc.

```
public class Course {

    String programDepartment;
    String title;
    String courseCode;
    String requirements;
    String stuNameOfThisCourse;
    Boolean status;

    public Course(String programDepartment, String title,String courseCode,String requirements,
            String stuNameOfThisCourse) {
        this.programDepartment = programDepartment;
        this.title = title;
        this.courseCode = courseCode;
        this.requirements = requirements;
        this.stuNameOfThisCourse = stuNameOfThisCourse;
        this.status = false;

    }
```

And then let them have ability to return outcome.

```
    public String getStuName() {
        return stuNameOfThisCourse;
    }

    public void setStatus(boolean decision) {
        status = decision;
    }
    public boolean getStatus() {
        return status;
    }

    public String  getCourseInfo() {
        return "Course Information: ProgramDepartmant is:" + programDepartment +
                ", Title is: "+ title + ", Course code: " + courseCode +  ", Requirements"+requirements
                +", student name of taking this course" + stuNameOfThisCourse;

    }
}
```

And the class Students is a extension of class Person. Class Students consist of the basic information of the student, such as student ID, student Name, and student age. I also

imported a package called java.util.LinkedList to let the student have ability to add all course that needed to be applied concession into this list and linked the list to the student.

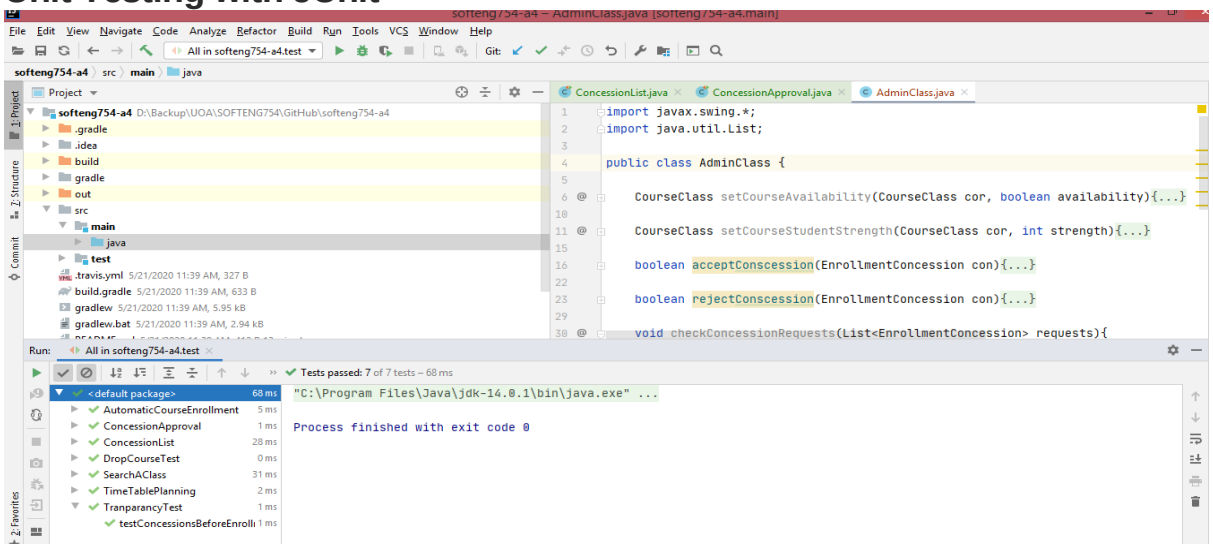
```
import java.util.LinkedList;

public class Students extends Person{
    public Students(int ID, String name, int age) {
        super(ID, name, age);
        // TODO Auto-generated constructor stub
    }



    LinkedList<Course> allCourses;

    public Students() {
        this.allCourses = new LinkedList<>();
    }

    public LinkedList<Course> getAllCourses(){
        return allCourses;
    }

    public void addCourse(Course course) {
        allCourses.add(course);
    }

    public LinkedList<Course> showAllCourses() {
        return allCourses;
    }


    public String getName() {
        return name;
    }
```

In the last test class, I imported two package called java.util.LinkedList, and java.util.List. In order to test it , I created a student called Hong, I also created two course called Data Structure and Algorithm which are the student wants to apply concession for. Then I add these two courses into the AllCourse List which has linked with the student before. Finally. TDD:I called assert function to test if these two courses have existed in the concession list. If these course have existed, then it will return True, otherwise return False.

```java
import java.util.LinkedList;
import java.util.List;

public class testAllConcessions {

    public static void main(String args[]) {
        Students student1 = new Students(338337,"hong",24);
        Course course1 = new Course("BCIS","Data Structure","TMSK881",
                "Three assignments with above 80% grades","hong");


        Course course2 = new Course("BCIS","Algorithm","COMP007","pass with excellent","hong");


        student1.addCourse(course1);
        student1.addCourse(course2);

        student1.showAllCourses();


        assert(student1.getAllCourses().get(0).getCourseInfo().equals(course1.getCourseInfo()));

    }


}
```

Kali435

Unit Testing with JUnit



ID No	Title	User Story	Priority	Acceptance Criteria	Estimatio	Test Cases	Who is working currently	Status
01-01	Transparency, Fairness	As a student, I want all concession	Must Have	The process of concessions, should be handled before	L	testConcessionsBeforeEnrollment()	Khalid Ali	
01-06	Timetable Planning	As a student, I want to view formulated	Must Have	Students should be able to see a timetable for the	XL	testGetTimeTable()	Khalid Ali	Removed from must haves
01-10	Search for a class	As a student, I want to be able to search	Must Have	Students should be able to search for a class using	M	testSearchClass()	Khalid Ali	
01-12	Ability to drop	As a student, I should be able to drop out	Must Have	Students should be able to see their current	M	testDropCourses()	Khalid Ali	
01-13	Concession List	As a student, I should be able to see the	Must Have	Students should be able to see course details of the	S		Khalid Ali	
01-20	Automatic mandatory course	As a student, I want the mandatory	Must Have	The system shall enrol students in mandatory courses	M	testAutoEnrol()	Khalid Ali	Removed from must haves

Production Code

Name	Date modified	Type	Size
AdminClass.java	5/21/2020 10:05 PM	JAVA File	2 KB
Classes.java	5/21/2020 10:05 PM	JAVA File	2 KB
CourseClass.java	5/21/2020 10:05 PM	JAVA File	2 KB
EnrollmentConcession.java	5/21/2020 10:05 PM	JAVA File	1 KB
EnrollmentManager.java	5/21/2020 10:05 PM	JAVA File	3 KB
PersonRole.java	5/21/2020 10:05 PM	JAVA File	1 KB
Student.java	5/22/2020 11:29 AM	JAVA File	1 KB
StudentClass.java	5/21/2020 10:05 PM	JAVA File	3 KB
TimeTable.java	5/21/2020 10:05 PM	JAVA File	1 KB
TimeTableEntry.java	5/21/2020 10:05 PM	JAVA File	1 KB

Testing Code

Name	Date modified	Type	Size
AutomaticCourseEnrollment.java	5/22/2020 12:10 AM	JAVA File	1 KB
ConcessionApproval.java	5/21/2020 11:52 PM	JAVA File	1 KB
ConcessionList.java	5/21/2020 10:51 PM	JAVA File	1 KB
DropCourseTest.java	5/22/2020 12:02 AM	JAVA File	1 KB
SearchAClass.java	5/21/2020 10:05 PM	JAVA File	1 KB
StudentTest.java	5/22/2020 11:29 AM	JAVA File	2 KB
TimeTablePlanning.java	5/21/2020 11:58 PM	JAVA File	1 KB
TranparancyTest.java	5/21/2020 11:04 PM	JAVA File	1 KB

Transparency, Fairness

TranparancyTest.java

```
import org.junit.Before;
```

```
import org.junit.Test;

public class TranparancyTest {
    StudentClass studentClass;
    CourseClass courseClass;
    @Before
    public void init(){
        studentClass = new StudentClass();
        courseClass = new CourseClass("Software Requirements
Engineering", "SOFTENG754", 15, true);

    }

    @Test(expected = IllegalStateException.class)
    public void testConcessionsBeforeEnrollment(){
        studentClass.applyForConcession(courseClass);
        studentClass.enrol(new CourseClass("test", "code",01,true));
    }
}
```

Timetable Planning

TimeTablePlanning.java

```
import org.junit.Test;

import java.util.Date;

import static org.junit.Assert.assertNotNull;

public class TimeTablePlanning {

    @Test
    public void testGetTimeTable(){
        EnrollmentManager system = new EnrollmentManager();
        TimeTable tb = new TimeTable();
        CourseClass courseClass = new CourseClass("Software
Requirements Engineering", "SOFTENG754", 15, true);
        TimeTableEntry entry = new TimeTableEntry((new Date()),new
Classes("102", courseClass), courseClass);
        tb.addToTimeTable(entry);
      assertNotNull(tb.getTimeTableEntryList());
            }
```

Search for a class

SearchAClass.java

```
import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.assertEquals;
```

```
public class SearchAClass {

    @Test
    public void testSearchClass(){
        EnrollmentManager sys = new EnrollmentManager();
        CourseClass courseClass = new CourseClass("LMN", "789", 0,
true);
        Classes c1 = new Classes("101", courseClass);
c1.setDepartment("CS");
        Classes c2 = new Classes("102", courseClass);
c2.setDepartment("CS");
        sys.addToClassesList(c1);
        sys.addToClassesList(c2);
        ArrayList<Classes> aray = new ArrayList<>();
        aray.add(c1);
        aray.add(c2);
        assertEquals(c1, sys.searchClassByID("101"));

assertEquals(aray.toString(),sys.searchClassByDepartment("CS").toStr
ing());
    }

}
```

Ability to drop

DropCourseTest.java

```
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class DropCourseTest {
    StudentClass studentClass;
    CourseClass courseClass;
    @Before
    public void init(){
        studentClass = new StudentClass();
        courseClass = new CourseClass("Software Requirements
Engineering", "SOFTENG754", 15, true);
    }

    @Test
    public void testDropCourses(){
        CourseClass courseClass1 = new CourseClass("Software
Engineering", "SOFTENG751", 12, true);
        studentClass.enrol(courseClass);
        studentClass.enrol(courseClass1);
        assertTrue(studentClass.dropOutCourse(courseClass));
        assertEquals(1, studentClass.getEnrolledCours().size());
    }
}
```

Concession List

ConcessionList.java

```
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class ConcessionList {
    StudentClass studentClass;
    CourseClass courseClass;
    @Before
    public void init(){
        studentClass = new StudentClass();
        courseClass = new CourseClass("Software Requirements
Engineering", "SOFTENG754", 15, true);

    }

    @Test
    public void testGetEnrolledCoursesDetails(){
        courseClass.setProgramDept("CS");
        studentClass.enrol(courseClass);
        assertEquals("CourseClass title: Software Requirements
Engineering\nCourseClass Code: SOFTENG754\nCourseClass Dept: CS",
studentClass.showEnrolledCoursesDetails());
    }


}
```

Automatic mandatory course selection

AutomaticCourseEnrollment.java

```
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class AutomaticCourseEnrollment {
    @Test
    public void testAutoEnrol() {
        List<CourseClass> courseClassList = new ArrayList<>();
        courseClassList.add(new CourseClass("ABC", "123", 0, true));
        courseClassList.add(new CourseClass("XYZ", "456", 0, true));
```

```
        courseClassList.add(new CourseClass("LMN", "789", 0, true));
        CourseClass cor2 = courseClassList.get(1);
        cor2.makeMandatory();
        EnrollmentManager sys = new
EnrollmentManager(courseClassList);
        StudentClass studentClass = new StudentClass();
        sys.AutoEnroll(studentClass);
        assertEquals(cor2, studentClass.getEnrolledCours().get(0));
        assertTrue(studentClass.getEnrolledCours().contains(cor2));
    }

}
```

Discussion about Test Driven Development and CI

- ### Requirements are ever evolving

  Even though the developers were quite familiarized with the domain here, some requirements were overlooked, and were realised in development. Because the developers were themselves students, for the purposes of this assignment, requirements were refined on the go. But, for real world projects, user stories and their acceptance criteria will need to continuously evolve to reflect the reality.

- ### TDD gives us control over feedback

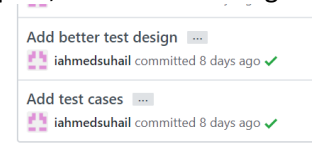
  When I started on the project, I found it hard to visualise the end of the project (amateur developer problems!). However, I realised that if one focuses on just one module at a time, and gets that working, one is made aware of the feedback on small, incremental steps and that feedback can be very helpful in getting to the destination faster. This feedback can be of great value in larger projects where it is hard to see the end of the project.

- ### Writing good test cases is not easy but can help better requirements

  Effective and good quality test cases can be hard to write. However, good quality test cases can reveal boundary conditions as well. I realised this while writing the test case for a user being able to login into a system, when I realised it is also important to test that a user is not able to login when they enter the incorrect details. This led to me revising the acceptance criteria for the user story, and hence, improved requirement. In complex situations, incremental progress like this can be very beneficial for reduced bugs in the codebase.

- ### Invalid Test Cases

  Often, some Junit Test cases were deemed inappropriate. A Junit test would not focus on the right aspects, as in the following example where I was comparing two objects hoping that the test would pass, but I had to change the test.




```
@org.junit.Test
public void testCanEnrolIntoCourses() {
    Student student = new Student();
    Course course = new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true);

    List<Course> resultEnrolledCourses = new ArrayList<>();
    resultEnrolledCourses.add(course);

    student.enrol(new Course( name: "Software Requirements Engineering",  code: "SOFTENG754",  points: 15,  availability: true));

      Won't work probably because the they aren't the same objects in memory
      assertEquals(resultEnrolledCourses, student.getEnrolledCourses());

      This test gets details (as a string) of the enrolledCourses and the course we just added
    assert(student.getEnrolledCourses().get(student.enrolledCourses.size()-1).showInfo().equals(resultEnrolledCourses.get(0).showInfo()));
}
```

- ### Unit Tests should use mocking more often

  While working through the assignment, I became more aware of the difference between Unit Tests and Integration tests. An important use case of Mocking that I had missed, was that mocking helps avoid the dependence of a test on other modules, and the test for a particular unit should not be invalidated because of bugs in other modules (versus only using mocking when a certain class/function isn't implemented yet). Now this didn't play a role in my particular project, as I had a user story map sorted according to the business process workflow, i.e. things a user would do first were developed rather than things that a user would do later. And the individual objects were tested so mocking was not needed. But I still appreciated in the report cases where if mocking were used, it would have been better. If more people had worked on the other user stories, mocking would definitely have been done more often.

- ### Note on Travis CI builds failing.

  Kent Beck, in (2002), advocates for a development environment to provide rapid response to small

changes. I followed a similar approach in writing test cases, wherein I would write test cases, ensure that they failed to compile, run them from my IDE, ensuring that they failed, and then write the code to make those test cases pass. This is, in my opinion, perfectly in line with TDD principles.
However, when I checked the build history from TravisCI, all my tests that were supposed to fail, were also passing the build. This (I realised later) was because of incomplete compile dependencies in Gradle and the project not conforming to default directory structure for Gradle. After fixing these issues midway through the project, I sought guidance from Kelly, on how I should proceed. She seemed to be of the opinion that if I had followed TDD principles, I shouldn't worry about the build failing to compile in the Travis environment, because I had run the tests in my IDE (I had marked the directories as Source and Test not realising the directory structure was not the same).
I wasn't very familiar with Travis when I started working on this project and should have spotted the build errors earlier. But I was still following TDD principles, writing tests before writing the code for a functionality and refactoring as needed. Here is a table of all the commits in which tests were failing and then compiling in the IDE but still passing in Travis builds.

Tests that fail from the IDE but passed the Travis build
https://github.com/iahmedsuhail/softeng754-a4/commit/dec71fc4ec0764a04121d3404c216af3847cf060
https://github.com/iahmedsuhail/softeng754-a4/commit/f22a653ec621ee435fbc4d359410f6310a2444c0
https://github.com/iahmedsuhail/softeng754-a4/commit/5b27e91da18a89204764151a6e53eb9c5ff2f252
https://github.com/iahmedsuhail/softeng754-a4/commit/6d758a847e4bc69e719bfc17401db206cb7c7ac4

Figure 1: Tests were not compiling because of Gradle Problems.

TRACEABILITY MATRIX

ID No	Title	User Story	Priority	Acceptance Criteria	Estimation Points	Test Cases	Who is working currently	Status
01-01	Transparency, Fairness	As a student, I want all concession requests to be handled before the enrollment starts, so that the enrollment process can be first come first serve to ensure transparency.	Must Have	The process of concessions, should be handled before the process of enrolment begins. Enrolment concession should be first come first serve to ensure fairness and transparency	L	testConcessionsBeforeEnrollment()	Khalid Ali	Pending Build Pass
02-02	Concession Approval	As an admin, I want to approve concessions, so that students can get into courses.	Must Have	Admins should be able to see Student info, such as ID, name while making the decision Admins should be able to see Course Info like Program Department, Title, Course code and requirements enforced by the university. Admins should be able to approve/reject requests. The system shall send a notification to the student regarding the concession decision made by admin.	XS	testIfConcessionApproved()	Yue Xu	Pending Build Pass
02-04	Course Guidelines	As an admin, I want to be able to set course restrictions, so that there are guidelines to get into a course.	Must Have	Admins should be able to limit the maximum number of students in a course. Admins should be able to check for prerequisites of a course for a student wanting to enrol into a course. Admins should be able to set the availability for a course The system shall enforce these restrictions.	M	testAdminCanSetGuidelines() Given_CourseIsNotAvailable_Expect_StudentsCantBeEnrolledIntoCourse()	Ahmed Suhail	Merged into master
01-02	Apply for concession	As a student, I want to be able to apply for concessions whenever the requirements enforced by the system are not met, so the admin can make decisions on my enrollment manually.	Must Have	Students should be able to see course information as maintained by the course catalog Students should be able to see the instructor for a course. Students should be able to see help regarding enrolment through the system. Students should be able to apply for enrolment concessions for any course run by the university Students should be able to submit enrolment concession requests, with a description of why they want the concession	S	testCanSubmitEnrollmentConcession() testCanSeeInstructor() Given_StudentSubmitsEnrollmentConcession_Expect_StudentCanSubmitDescription()	Ahmed Suhail	Merged into master
03-03	Integrate with University Systems	The system should integrate with the current university systems, so that students have a streamlined experience of trying to access other university applications.	Must Have	Persons should be able to enter University SSO login details for logging into the system Persons should be able to login into the system If incorrect details are entered, persons should not be able to login into the system	L	testPersonCanLogin() Given_PersonEntersIncorrectDetails_Expect_PersonShouldNotBeAbleToLogin()	Ahmed Suhail	Merged into master
03-05	System catalog	As a student, I want to be able to see the list of courses, so that I can look at the options I have for taking courses.	Must Have	Students should be able to see course code, course name, points, availability, lecture times, lecturer name. Students should be able to sort the courses	L	testCourseDetailsCanBeSeen()	Ahmed Suhail	Merged into master
01-06	Timetable Planning	As a student, I want to view formulated timetables, so that I can see which course fits my schedule.	Must Have	Students should be able to see a timetable for the courses they have selected. Students should be presented with multiple timetables whenever there are multiple classes for a course. The system should sort and display only suitable timetables for a student.	XL	testGetTimeTable()	Khalid Ali	Pending Build Pass
01-07	Current Courses	As a student, I want to be able to see my current enrollment status, so that I can see the courses I am enrolled in.	Must Have	Students should be able to see course name,course code, enrollment status.	S	Given_StudentIsEnrolledInACourse_Expect_StudentCanSeeTheEnrolledCourses()	Ahmed Suhail	Merged into master
01-10	Search for a class	As a student, I want to be able to search for a class, so that I can add courses to my schedule in multiple ways.	Must Have	Students should be able to search for a class using any of the following: - Course Name - Course Code - Instructor	M	testSearchClass()	Khalid Ali	Pending Build Pass
01-11	Ability to enrol	As a student, I want to be able to enrol into courses, so I can work my way towards getting a degree.	Must Have	Students should be able to enrol into courses. Students should be able to see related information such as fee liabilities while confirming enrolment.	XL	testCanEnrolIntoCourses()	Ahmed Suhail	Merged into master
01-12	Ability to drop	As a student, I should be able to drop out of courses, so that I can get out of the courses I don't want to take anymore.	Must Have	Students should be able to see their current enrolments. Students should be able to drop out of courses	M	testDropCourses()	Khalid Ali	Pending Build Pass
01-13	Concession List	As a student, I should be able to see the list of courses I have submitted for concessions, so that I can see which courses I am enrolled in.	Must Have	Students should be able to see the course name and the course code of the courses they have applied concessions for.	S	testALLconcessions()	Khalid Ali and Yue Xu	Pending Build Pass