# CIS*2430: Assignment 2

Due November 26, 8:30 a.m. (server time)

You will create a planning application that assists a student to plan their path through the B.Comp program. The application will keep track of student progress through the program, assist with planning the courseload for each semester, and help the student determine if they have fulfilled the requirements to graduate.

This second assignment will focus on user interaction, error handling, data management, and business logic.

## Learning Outcomes

As a learner I will:

- refactor and restructure class design for improved encapsulation, modularity, cohesion and coupling
- demonstrate use of inheritance through super/sub classes as well as through the use of interfaces
- demonstrate clear understanding event driven programming through well designed listeners and gui components
- demonstrate service-based error handling through a rich set of exception classes that communicate specific errors to client classes
- create a repeatable testing suite and justify the choice of test cases
- design and create a graphical user interface that is learnable and usable
- use inner classes, anonymous classes, and/or lambdas effectively

## Modules

### Exceptions and Error Handling

Add error handling to Course, Degree and its subclasses, and Student. Write your own exception classes where appropriate and use them. You may additionally use the built in exception classes. Modify your code to use try/catch loops. When appropriate your code should handle exceptions by giving the user another chance to provide the correct input.

### GUI

Create a Graphical User Interface for your Planner using Swing that provides functionality for all of the user stories given with this assignment description. Your GUI should include a separate window or popup for administration that provides functionality for the administrative tasks. Your GUI must be easy to use and visually appealing. As a minimum, your GUI must include menus, buttons, text input, dialog boxes, panels, frames, at least two different layout managers, and at least two custom listener classes or methods invoked via lambasthat you write. You may use additional components as well. Persons with prior experience using Swing who wish to learn something new may choose to do their GUI in javafx.

### Refactoring/Restructuring

A common activity in software development is refactoring. Refactoring consists of redesigning and then changing the code to reflect the new design.

The A1 design has two classes that are poorly encapsulated. Course contains information about a Course, but also contains information about a Student's attempt at a course. That class needs to be split into two classes, one called Course and one called Attempt. Attempt should contain all the information about a Student's attempt such as semester and final grade. Attempt should also have a reference to the actual course. Students can have more than one attempt for the same course. Course should have a field added called "semesterOffered" that has a value of F, W or B (both) that indicates when that course is normally offered.

The Course class should be immutable to client classes as students should not be able to change the information about a course. Change the course class so that the getters and setters are available only to the classes in the same package (using protected). Then refactor Course so that it is in a package called univ. You will also need to put Degree and the subclasses in that package. Add any necessary methods.

The second poorly designed class is the Plan of Study. The plan of study mixes the idea of a student's plan with their transcript of courses that they have already taken. Remove the plan of study class and refactor the Student class so that a student has two collections, one that holds a set of Attempts, and one that can hold a reference to a course that is planned and the semester it is planned to be taken. Revise and add methods as necessary to provide existing functionality.

**Demonstrate the efficacy of your new design by creating an implementation plan(including partial code if appropriate) for adding the Bachelors of Arts to the program. Include Theatre Studies and Geography majors in your implementation plan. Include any new classes you would need to write, the details of fields and methods for those classes as well as descriptions of what those methods would do. You do not have to implement this part, you just need to design it. A UML class diagram is a suitable document for describing this design as is a well structured pdf document.**

### Distributed Data

Use a provided Course Calendar library. Make queries on it to instantiate Courses and use them in your program. Use the Student Record library to save/load student records. Provide functionality in your GUI for one student to save and close their plan and another student to open one, without terminating the program and starting over.

## User Stories

Your final submission must contain a class called Planner that contains the main method to launch the gui. It does not (and likely should not) be the same Planner class from A1

As a user I can:

- select my degree (required choices: BCH (BComp Honours) , BCG (BComp General), optional: one other degree)
- select a major from my degree if it offers a major
- maintain courses in my plan of study (add/remove)
- maintain courses in my transcript, or list of courses I have taken (add/change grade/remove)

- save my program (the plan and my transcript)
- view a list of *required* courses for my degree and major that are *not* represented in my plan of study nor in my transcript
- view a list of *required* courses that are *not* represented in my transcript
- view the number of credits I must add to my plan of study in order to have a plan that allows me to complete my degree.
- view a list of the prerequisite courses for any required course for my degree and major.
- view a list of prerequisite courses that I must take in order to take the courses currently in my plan of study.
- view the number of credits I have completed in my program
- view the number of credits I have remaining to complete my program
- determine if I have met the completion requirements of my chosen degree
- view my course plan, organized by the semester I have, or plan to, take the courses
- view my overall GPA
- view my GPA for CIS courses
- view my GPA for my most recent 10 credits

As a program administrator I can

- connect to the database containing the list of courses and degrees
- maintain the list of degrees in the database (add/remove/change)
- maintain the list of courses in the database(add/remove/change)

## Required Model

The required model is slightly changed from that of A1. **Read the method headers carefully to identify situations where you will need to refactor.** It is an incomplete model and you will need to add methods and classes in order to complete this assignment.

All classes must also have a default (zero parameter) constructor, an overridden toString() method and an overridden equals() method.

### Concrete Classes in the Degree Hierarchy

The concrete classes (CS, SEng, and BCG) must provide the following public methods. In some cases these methods should be overridden methods of the abstract parent classes.

- String  getDegreeTitle()
- void setDegreeTitle(String title)
- void setRequiredCourses(ArrayList<Course> listOfRequiredCourseCodes)
- ArrayList<Course> getRequiredCourses()
- boolean meetsRequirements(ArrayList<Course> allTheCoursesPlannedAndTaken) (meetsRequirements ignores grades
- double numberOfCreditsRemaining(ArrayList<Course> allTheCoursesPlannedAndTaken)
- ArrayList<Course> remainingRequiredCourses(ArrayList<Course> allTheCoursesPlannedAndTak

You may use static final class variables to model the different requirements for each degree. Those class variables should also be placed at the appropriate level of abstraction in the Degree hierarchy.

### Course

The Course class should be immutable for non-package clients. This means that it should be placed in a package and the setters for the class should be changed from public, to protected access. Getters should still be public.

- String getCourseCode()
- void setCourseCode(String courseCode)
- String getCourseTitle()
- void setCourseTitle(String courseTitle)
- double getCourseCredit()
- void setCourseCredit(double credit)
- ArrayList<Course> getPrerequisites()
- void setPrerequisites (ArrayList<Course> preReqList)
- String getSemesterOffered() //one of F, W, or B we will ignore summer classes

## Attempt

- void setAttemptGrade(String grade) // a number, P, F, INC or MNR
- String getAttemptGrade()
- void setSemesterTaken(String semester)
- String getSemesterTaken()
- void setCourseAttempted(Course theCourse)
- Course getCourseAttempted()

### Course Catalog

The CourseCatalog class should be immutable for non-package clients. This means that it should be placed in a package and the setters for the class should be changed from public, to protected access. Use at least one package called univ to contain the Course and CourseCatalog classes. You may choose to put the Degree classes in that package as well. Do not put the Student or Plan of Study in that package.

The course catalog will use the database code provided to connect to the database and obtain course descriptions. It will no longer need to hold an entire list of all the available courses.

The CourseCatalog class must provide the following methods:

- void addCourse(Course toAdd)
- void removeCourse(Course toRemove)
- Course findCourse(String courseCode)

### Student

The Student class will connect to the database to save and retrieve the record for a given student. The student number can be used to look up a student.

The Student class must provide the following public methods:

- String getFullName()
- void setFirstName(String first)

- void setLastName(String last)
- String getFirstName()
- String getLastName()
- void setStudentNumber(Integer studentNum)
- Integer getStudentNumber()

Student will additionally need methods for getting and setting the Transcript and the Plan of Study for a student. Those methods can be defined by you.

## Grading Notes

- Late submissions are penalized at 2% per hour. Submissions more than 36 hours late will not be graded and grade of zero will be assigned. You may spend participation points if you wish to mitigate the grade penalty.
- Part of your grade for this assignment will come from automated testing.
- The marks breakdown will be released closer to the due date of the assignment. There will be bonus marks for persons who exclusively use their own solution from A1 as the basis for A2. There will also be bonus marks for early submission.
- You will be expected to have complete javadoc comments in all source code and you will also be expected to submit written evidence of testing in the form of testing scripts, unit testing programs, or junit classes.

## Submission Requirements

- You will submit a zip file (no other compression formats will be accepted). The file will contain a single folder named with your UofG login name. All of your java source files should be placed in that folder. Classes must be named exactly as specified.
- In addition to your java source files, you must submit a README file. The structure of that README file is given as a separate document. We will not grade your assignment if your README file is missing.
- This assignment may be done individually or with a partner. **You are permitted to work with a partner for ONE of the two assignments for the course**. You must previously have registered as partners in order to work with a partner for this assignment as registration is currently closed. Both partners must make a submission (of the same code).
  - There are additional submission requirements for partnered work described at the end of this document.

## Extras for Partnered Submissions

If you choose to work with a partner for this assignment, you must submit the following, in addition to the individual submission requirements.

1. The javadoc comments must be complete for all classes and must indicate the contribution of each partner to each method in the class.
2. Implement the classes required to add the Bachelor of Arts degree: Theatre studies major to your software.

3. Implement the following additional user story: Estimate my last semester of study, based on the courses I already have credit for, the current semester, my desired number of courses per semester, and the normal semester of offering for my planned courses.