Faculty of Technology – Course work Specification 2014/15

Module name:	OO Software Design & Development		
Module code:	CTEC2602		
Title of the Assignment:	Assignment 2: Module Chooser GUI		
This coursework item is: (delete as appropriate)	Summative		
This summative coursework will be marked anonymously	Yes		
The learning outcomes that are assessed by this coursework are: 1. To use Java to implement standard object-oriented designs given in UML. 2. Address issues of quality, maintainability, correctness and robustness with respect to software design and development. 3. Make effective use of the Java SDK Application Programming Interfaces.			
This coursework is: (delete as appropriate)	Individual		
This coursework constitutes 25% **to the overall module mark.**			
Date Set:	Week 24 (Monday 9th March 00:01)		
Date & Time Due:	Week 29 (Wednesday 15th April 23:59)		
Your marked coursework and feedback will be available to you on: If for any reason this is not forthcoming by the due date your module leader will let you know why and when it can be expected. The Head of Studies (rgh@dmu.ac.uk) should be informed of any issues relating to the return of marked coursework and feedback.	Within 4 weeks of week 29. Your tutor will always aim to provide you with feedback as soon as is possible.		
When completed you are required to submit your coursework to: • Blackboard VLE through an assignment submission portal as a .zip file. (*see attached document*)			
Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations which state: *"the time period during which a student may submit a piece of work late without authorisation and have the work capped at 40% [50% at PG level] if passed is **14 calendar days**. Work submitted unauthorised more than 14 calendar days after the original submission date will receive a mark of 0%. These regulations apply to a student's first attempt at coursework. Work submitted late without authorisation which constitutes reassessment of a previously failed piece of coursework will always receive a mark of 0%."*			
Academic Offences and Bad Academic Practices: These include plagiarism, cheating, collusion, copying work and reuse of your own work, poor referencing or the passing off of somebody else's ideas as your own. If you are in any doubt about what constitutes an academic offence or bad academic practice you must check with your tutor. Further information and details of how DSU can support you, if needed, is available at: http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/academic-offences.aspx and http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/bad-academic-practice.aspx			
Tasks to be undertaken: See (following) attached document.			
Deliverables to be submitted for assessment: See (following) attached document.			
How the work will be marked: See (following) attached document.			
Module leader/tutor name:	**Luke Attwood**		
Contact details:	**lattwood@dmu.ac.uk (GH6.71)**		

Assignment 2: Module Chooser GUI

About this assessment

This individual summative coursework counts **25%** towards your module mark.

The deadline for submitting your work via Blackboard is **23:59 Wednesday 15th April 2015**.

Objectives

The objective of this assessment is for you to demonstrate your ability to design and implement an OO system consisting of a set of Java classes, using advanced libraries within the Java SDK. In particular:

1. To study and correctly make use of a prebuilt student profile data model.
2. To build a suitable user interface using the Java Swing and AWT libraries.
3. To implement event handling procedures that provide a basis for an interactive and user-friendly system.
4. To adhere to standard principles of the Model-View-Controller (MVC) design pattern and appropriately decompose classes through abstraction and encapsulation.

Submission

Submit your web application through the submission portal available on Blackboard.

You should create a new folder called "CTEC2602Assignment2" and copy your Eclipse project into this folder, ensuring all of the Java source files (and any other associated files) are in it. Then compress the folder into a .zip file so you end up with a file in the format CTEC2602Assignment2.zip. You should then submit this file through Blackboard. **Note**: *It is expected that ALL source code files WILL be attached - please check this before submission.*

Anonymity

The University has introduced the requirement to anonymously mark all assessment work. Anonymity will apply only to the marking process itself, and ends before feedback is given. This assessment will be marked anonymously. It should be noted however that where work is presented to or observed by the assessing tutor, for example to receive interim feedback or assistance, that this may compromise anonymity. However, all efforts will be made to mark the work anonymously.

Module Chooser GUI specification

A student profile captures the details of a second year undergraduate computing student and allows them to select their final year module options. There are compulsory modules that must be selected (depending on the course of study), and others that are only associated with certain courses.

Your task is to build an interactive graphical user interface that dynamically allows modules to be selected based on the chosen course of study, and then stores this information. Ideally the application should be user-friendly and contain appropriate validation to ensure only a legitimate selection of modules is made. There should be a feature allowing associated data to be saved to a file and then loaded back into memory at a later time.

For this prototype, you are only required to use the data of two courses, Computer Science and Software Engineering. However, the system should be designed such that it would be relatively simple to add further courses and modules in the future.

The table below shows all of the available modules, their credit amount, and whether they are an option (and/or compulsory) for Computer Science and Software Engineering students.

Code	Title	Credits	Computer Science option	Compulsory	Software Engineering option	Compulsory
CTEC3903	Software Development Methods	15	x	x	x	x
IMAT3451	Computing Project	30	x	x	x	x
CTEC3902	Rigorous Systems	15	x		x	x
CTEC3426	Telematics	15	x		x	
CTEC3604	Multi-service Networks	30	x		x	
CTEC3110	Secure Web Application Development	15	x		x	
IMAT3404	Mobile Robotics	15	x		x	
IMAT3406	Fuzzy Logic & Knowledge Based Systems	15	x		x	
IMAT3426	Information Systems Strategy & Services	30	x			
IMAT3429	Privacy and Data Protection	15	x		x	
IMAT3608	Mobile Games Development	30	x		x	

Computer Science students have 45 compulsory credits (via CTEC3903 and IMAT3451), whereas Software Engineering students have 60 compulsory credits (via CTEC3903, IMAT3451 and CTEC3902). Computer Science students can exclusively study IMAT3426. In total 120 credits must be selected through any legitimate combination of module options.

Guidance on building the application

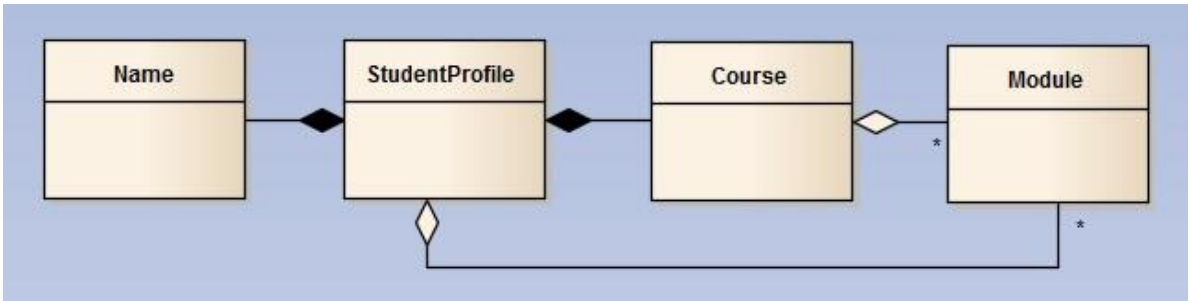
You are advised to thoroughly read this guidance and to continually refer to it as a means of helping you design and implement the Module Chooser GUI application.

Please place the model, controller and view classes into appropriate **packages** within your Eclipse project. This will help you logically separate them during development, and will help your tutor to mark these different components more easily.

The Model

You have been provided with the data model for this application - see the file **model.zip** on Blackboard. The model contains four classes – `StudentProfile`, `Course`, `Module`, and `Name`.

- A **Student Profile** encapsulates a student name, P number, a Course of study and a set of selected Modules.
- A **Course** has a name and a collection of all Modules available on that course.
- A **Module** has a code, name, credit amount and is either mandatory or not.
- A **Name** for a student is made up of a first name and a family name.



The model classes purposely have no javadoc comments, so you are required to study the source code in order to understand how to use them as the data model for your MVC application.

You should not need to add any further code to the model and are advised to make use of existing methods wherever possible. Should you find yourself wanting to add any further functionality to the data model you have been provided with, you should clarify this with your tutor who will advise you on whether it is appropriate or not.

The View

The GUI is made up of three forms, which you are advised to place on three different tabs, using a `JTabbedPane` component. They are as follows:

> Create Student Profile tab
> Should display a combo box, pre-populated with the two aforementioned second year computing courses, and three text fields for inputting a student first name, family name, and P number. There should be a *create profile* button.

Select Modules tab
Should display two list boxes, one of which contains the unselected module options (for the chosen course) and another which displays selected module options (including compulsory modules). There may optionally be a display of accumulated credits for the current selected modules. There should be a *reset, remove, add,* and *submit* button.

Overview Selection tab
Should display an overview of the student's details and selected modules based on their submitted profile and module selection from the previous two tabs. The information should ideally be clear and well presented. There should also be a *save overview* button.

Your GUI should also have a menu bar:

The Menu Bar
This Menu bar should have two menus: File and Help. The File menu is made up of menu items: "Load Student Data", "Save Student Data" and "Exit". The Help menu is made up of a single menu item: "About".

Note: You are strongly advised to separate logical containers in your view into their own top-level classes, e.g. extending `JPanel`. You will also need to provide suitable methods that allow relevant data and components to be accessed and modified within the view.

You can see the **Sample GUI Screenshots** section at the end of this document to assist in visualising how the GUI should generally look and behave.

The Controller

You are strongly advised to significantly decouple the model and view by deploying a MVC architecture with a separate controller, placing the event handling into its own top-level class.

Pre-loading course and module data: The aforementioned course and module data should be loaded into the application when it starts. The controller should handle this requirement and attach courses to the combo box on the create student profile tab. Whilst this can be hardcoded (*see further section*), a more advanced requirement of the application, which you may wish to attempt, would be to use a Scanner to dynamically load the course and module data in from a file. This would then allow for the input data to be easily evolved in the future.

See below details of what each event handler should broadly do. It is assumed you will consider appropriate validation where relevant:

- **Create Profile Handler** - captures the details of the student and stores them within the data model. In addition, this should also cause the select modules tab to be dynamically populated with the initial unselected and (compulsory) selected modules based on the chosen course.
- **Reset Handler** - resets the modules tab to populate the initial unselected and (compulsory) selected modules based on the chosen course.

- **Add Handler** - removes the current selected module from the unselected modules list box and adds it to the selected modules list box.
- **Remove Handler** - removes the current selected module from the selected modules list box and adds it back to the unselected modules list box.
- **Submit Handler** - captures the details of the selected modules and adds them into the data model. In addition, this should cause an overview of the student's details and selected modules to appear dynamically on the "Overview Selection" tab.
- **Save Overview Handler** - save an overview of the student's details and selected modules in text format using a PrintWriter (or suitable alternative), choosing a custom representation.
- **Save Profile Handler** - save the student profile (data model) to a file in binary format using an ObjectOutputStream.
- **Load Student Data Handler** - restore the student profile (data model) from a file using an ObjectInputStream. You may also wish to consider bringing the GUI (view) back to its previous state (at the point of saving).
- **About Handler** - trigger a popup window containing basic details of the application.
- **Exit Handler** - close the current window and terminate the application.

Model-View-Controller (MVC) design pattern

You should clearly showcase the MVC design pattern throughout your implementation and attempt to separate the concerns of the model, view and controller. You will be assessed on your ability to sensibly decouple these entities to make a maintainable and reusable solution.

Application Loader

Your `main` method test class can simply instantiate the model (i.e. `StudentProfile`), view (i.e. your class that extends `JFrame`) and (assuming you separate it,) the controller.

Useful tip: Populating the JComboBox with courses

When you load the GUI, the "Create Student Profile" tab should display a combo box, populated with the two aforementioned computing courses. There are various ways of achieving this. One approach you may take would be to have a `JComboBox<Course>` specified in an appropriate part of your view, e.g.

```
private JComboBox<Course> cboCourses;
```

Within that same class you could have a method that accepts an array of type `Course` and then populates the combo box with this, e.g.

```
public void populateComboBox(Course[] courses) {
        for (Course c : courses) {
                cboCourses.addItem(c);
        }
}
```

You could then specify a method within your controller class, which populates an array of type `Course` with the required hardcoded module data and then returns it, e.g.

```
        private Course[] setupAndGetCourses() {
                Module ctec3903 = new Module("CTEC3903", "Software Development Methods", 15, true);
                Module imat3451 = new Module("IMAT3451", "Computing Project", 30, true);
                Module ctec3902_SoftEng = new Module("CTEC3902", "Rigerous Systems", 15, true);
                Module ctec3902_CompSci = new Module("CTEC3902", "Rigerous Systems", 15, false);
                Module ctec3110 = new Module("CTEC3110", "Secure Web Application Development", 15,
false);
                Module ctec3426 = new Module("CTEC3426", "Telematics", 15, false);
                Module ctec3604 = new Module("CTEC3604", "Multi-service Networks", 30, false);
                Module imat3404 = new Module("IMAT3404", "Mobile Robotics", 15, false);
                Module imat3406 = new Module("IMAT3406", "Fuzzy Logic and Knowledge Based systems",
15, false);
                Module imat3429 = new Module("IMAT3429", "Privacy and Data Protection", 15, false);
                Module imat3608 = new Module("IMAT3608", "Mobile Games Development", 30, false);
                Module imat3426_CompSci = new Module("IMAT3426", "Information Systems Strategy and
Services", 30, false);

                Course compSci = new Course("Computer Science");
                compSci.addModule(ctec3903);
                compSci.addModule(imat3451);
                compSci.addModule(ctec3902_CompSci);
                compSci.addModule(ctec3110);
                compSci.addModule(ctec3426);
                compSci.addModule(ctec3604);
                compSci.addModule(imat3404);
                compSci.addModule(imat3406);
                compSci.addModule(imat3429);
                compSci.addModule(imat3608);
                compSci.addModule(imat3426_CompSci);

                Course softEng = new Course("Software Engineering");
                softEng.addModule(ctec3903);
                softEng.addModule(imat3451);
                softEng.addModule(ctec3902_SoftEng);
                softEng.addModule(ctec3110);
                softEng.addModule(ctec3426);
                softEng.addModule(ctec3604);
                softEng.addModule(imat3404);
                softEng.addModule(imat3406);
                softEng.addModule(imat3429);
                softEng.addModule(imat3608);

                Course[] courses = new Course[2];
                courses[0] = compSci;
                courses[1] = softEng;

                return courses;
        }
```

During the initial construction of the application, once the JFrame view container has been instantiated you could pass in this array to populate the combo box.

Please turn over…

Assessment Criteria

The following criteria show how you will be assessed:

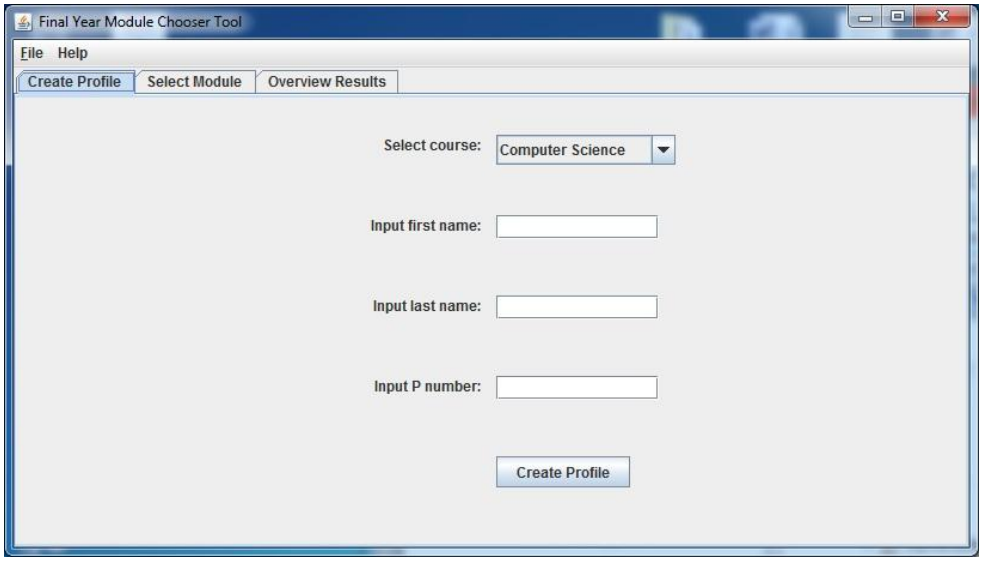
- **User Interface (30%):** The user interface appropriately displays and captures relevant data, with a suitable layout *.
- **Event Handling and Fitness for Purpose (30%):** The user interface is responsive to interaction and can be used to achieve any associated tasks. There is appropriate validation to ensure operations behave correctly.
- **MVC Design and View Decomposition (30%):** The Model-View-Controller (MVC) design pattern has been applied to separate concerns and reduce coupling. The view has had logical containers separated into their own top-level classes.
- **Saving & Loading (10%):** Courses and associated modules can be pre-loaded into the application. An overview of the student's details and selected modules can be written to a file in text form. The current student profile can be saved to a file in binary form and seamlessly restored at a later time.

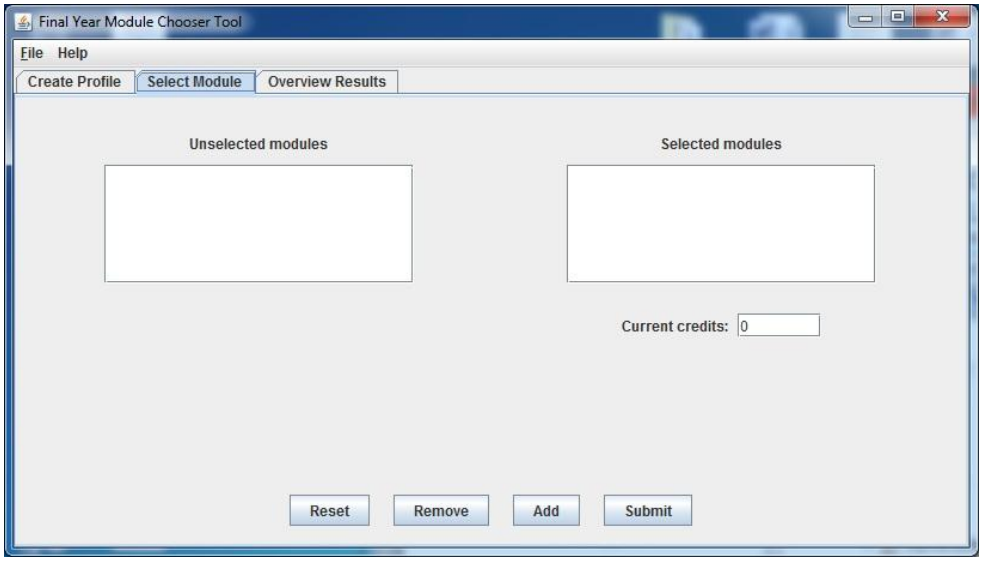
* You are being assessed on your understanding and correct use of common Java Swing layout managers (that you have been taught), such as BorderLayout, FlowLayout, GridLayout, BoxLayout and GridBagLayout. You will not receive credit for using a GUI builder or absolute positioning.

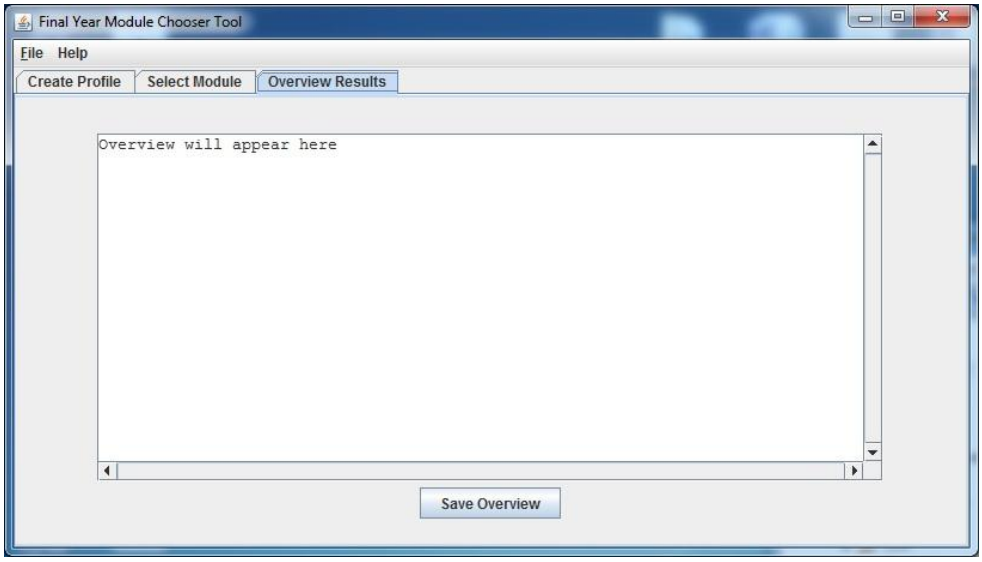
Sample GUI screenshots

These screenshots are provided to help you visualise the core details that the user interface should display and capture. This prototype design is purely provided as a guide and you do not need to try and replicate the exact layout shown. You are encouraged to be creative where applicable

Default tabs
When the application is first loaded, the three tabs will have default appearances, e.g.



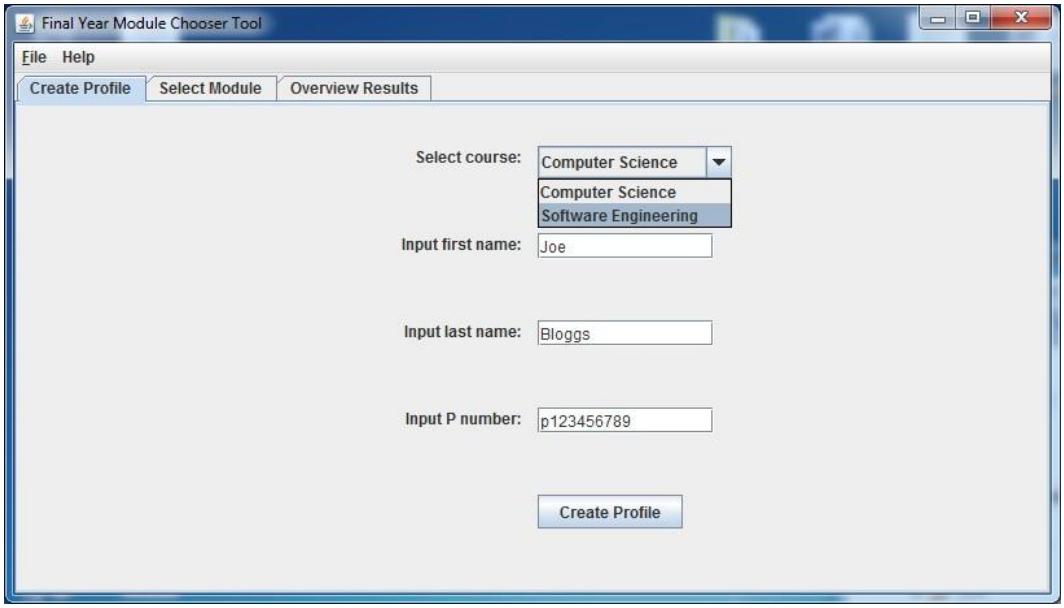




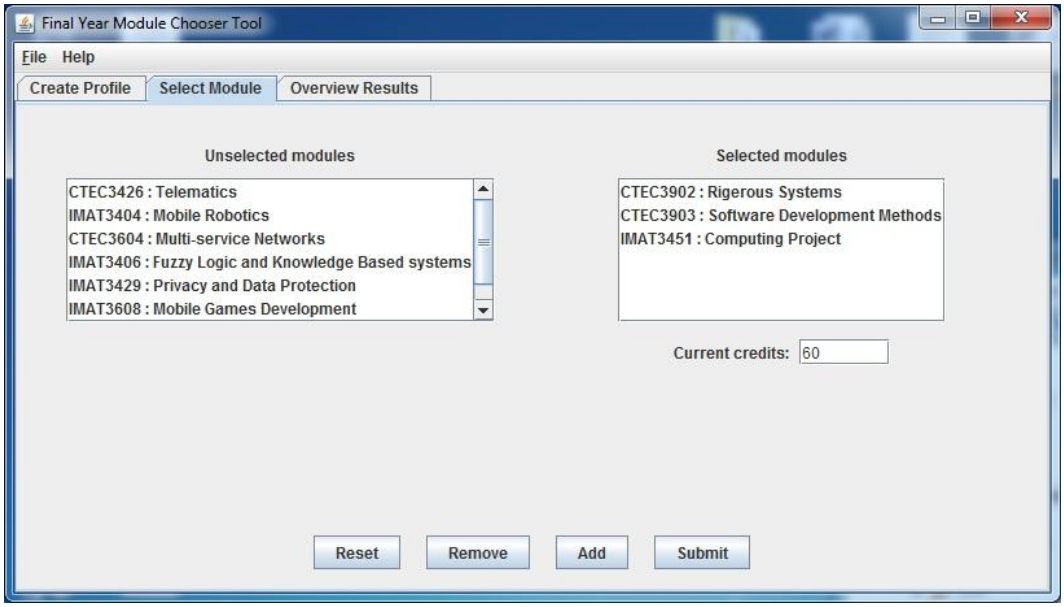
Please turn over…

Using the GUI

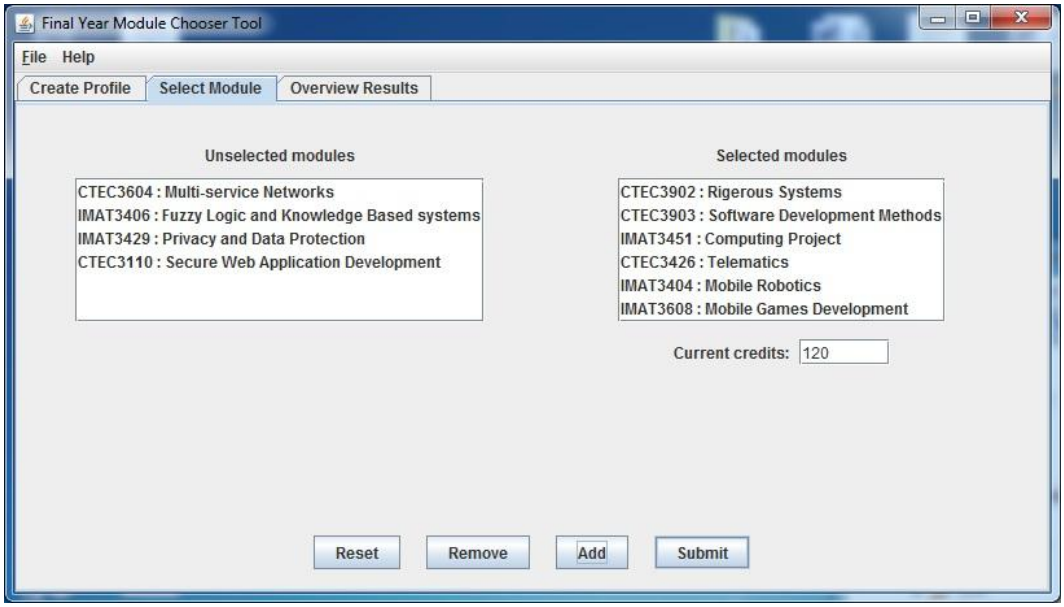
The following screenshots show how courses should be pre-populated into a combo box and that when a profile is created, the modules of the chosen course will be dynamically rendered, ideally automatically assigning mandatory modules to the "Selected modules areas". You can then add to the selected modules list, or remove modules back to the unselected modules list.



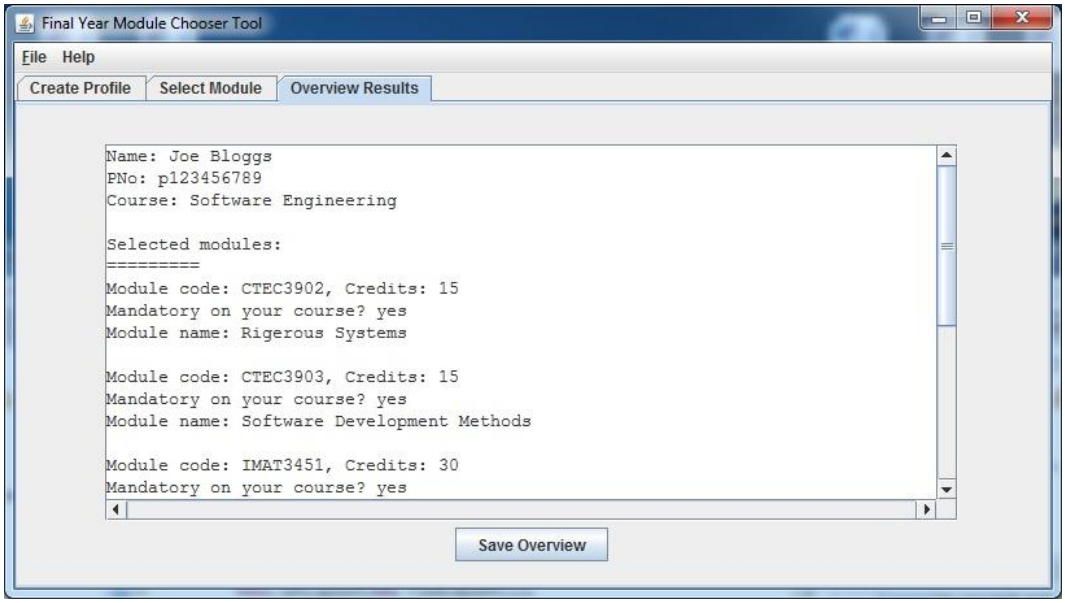
Courses in the combo box will either be prepopulated by 'hardcoding' them in your source code, or by dynamically reading them in from a text file, when the application initially loads.



This shows an initial hypothetical view, with prepopulated modules, and the current credit amount shown. **Note**: Whilst you can show additional details of a module, i.e. its credits and whether it's mandatory, inside of the list view, if you have a `JList<Module>`, which you can process behind the scenes, then you should not need to do this.



This shows how modules have been added up to the 120 credit amount, and if the user is happy with their selection, they can proceed to submit. You will want to carefully consider various validation clauses on this tab. Also, the reset button should bring this tab back to its initial state (i.e. that shown in the previous screenshot).



An overview of the user's selection and input details is displayed. The way in which you display this information is entirely up to you. There is also the ability to save this textual information to a file.

Please turn over…

The menu bar

A simple example of the proposed "File" menu. This will allow the data model to be saved to a file in binary form, and restored at a later time. Ideally, the interface should largely replicate its prior state.