Assignment 4: Course Planner

- Submit deliverables to CourSys: https://courses.cs.sfu.ca/
- Late penalty: 10% per calendar day (each 0 to 24 hour period past due), max 2 days late.
- This assignment may be done **individually or in pairs.** Do not show other students your code, do not copy code found online, do not reuse solutions from previous semesters or courses or assignments, and do not post questions about the assignment online. Please direct all questions to the instructor or TA via the help list: cmpt-213-help@sfu.ca;
  - You may use *ideas* (short code snippets, tutorials, documentation...) you find online and from others, but your solution must be your own.
- See the marking guide for details on how each part will be marked.

1. Introduction

Create a tool which helps students plan their courses. Specifically, it will show students when a course has been offered in the past which may help predict when it will be offered in the future.

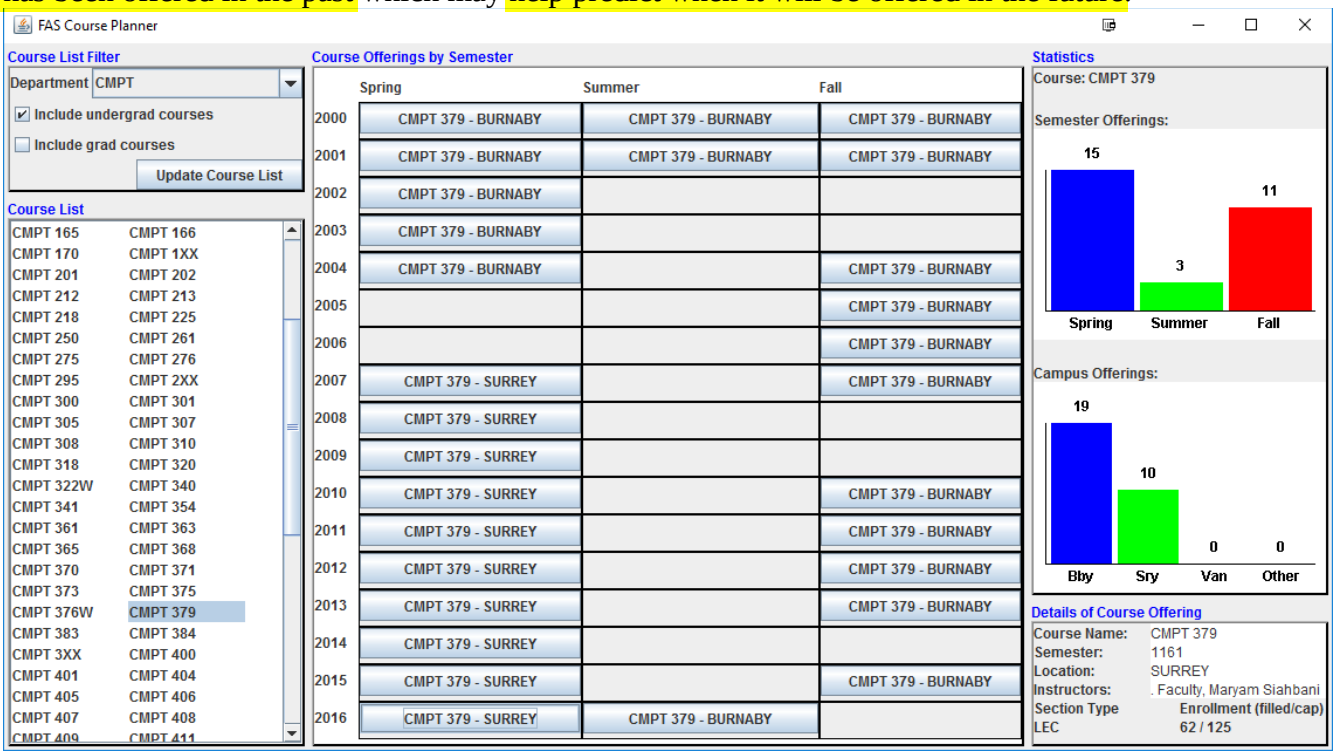


Figure 1: Screenshot showing CMPT 379 selected.

1.1 Use case walk thought

1. User selects a department and undergrad and/or grad courses and clicks "Update Course List"
2. User is shown all matching course in the scrolling "Course List" list box.
3. User clicks a course to have all its offerings (year, semester, campus) shown in the grid. Bar-graphs and statistics update on the right-hand side.
4. User clicks an offering to see the details of the course in the bottom right.
5. User may select a different department, different course, or different offering.

2. Requirements

Your submission must meet the following requirements.

2.1 Required Course Data Input

1. Create a folder in your project name `data/` (it will be at the same level as the `src/` folder (not inside it).
2. Copy the provided `course_data_2016.csv` file into this folder.
3. The file is a comma-separated text file with the following columns:
    1. **SEMESTER:**
       The semester number of the offering.
       - The first three digits refer to the year.
       - The last digit is the semester: 1=Spring, 4=Summer, 7=Fall. You need handle only these three semester (i.e., you will not be tested with a semester number 5).
    2. **SUBJECT:**
       The department. Will be a short string.
    3. **CATALOGNUMBER:**
       The course "number". May be a number like 213, or a string like 105W or 1XX.
    4. **LOCATION:**
       The campus:
       - Campuses of interest: `BURNABY`, `SURREY`, `HRBRCNTR`.
       - Other locations: `OFF`, `KIT`, `KAM`, `WML`, …. (anything else).
    5. **ENROLEMENTCAPACITY:**
       The enrollment cap for the section (i.e., maximum number of seats)
    6. **ENROLMENTTOTAL:**
       The enrollment total for the section (i.e., the number of people who took that section).
    7. **INSTRUCTORS:**
       The instructor(s) which taught that section of the course. Comma separated if more than one. "(null)" if unknown (information not stored in the database).
    8. **COMPONENTCODE:**
       The type of section as a short string such as `LEC`, `TUT`, `SEC`, `OPL`, ...
4. At startup, your program must be hard-coded (using a named constant) to load the "`data/course_data_2016.csv`" file.
    1. If the file is not found, or there was an error reading the file or its contents, display a message box and then quit. For example:

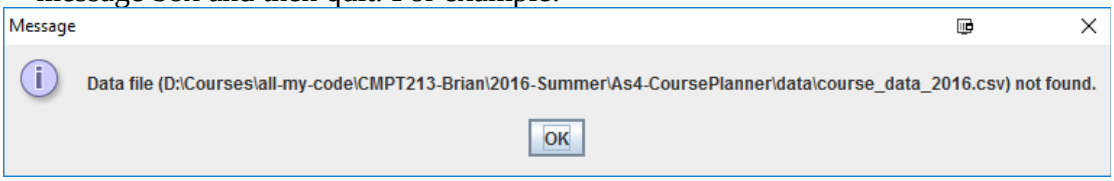


Figure 2: Data file not found error message before exiting.

    2. Your application will only be tested with well formed input files, such as the one provided. We may test with alternative contents, such as different subjects, courses, locations, years, and section types.
    3. You may *not* use a 3rd party CSV file reader.

2.2 Required UI Features

2.2.1 General

1. UI must be well laid out and usable.
2. Each panel described below must have a heading. (See hints)
3. Each panel's contents (below the heading) must have a solid border around it. (See hints)

2.2.2 Course list filter

The Course List Filter panel allows the user to select which courses should be shown in the Course List panel. It includes:

1. Drop-down combo-box to select a department (i.e., subject) for courses to be shown. These must be the subjects that are found in the input file.
2. Checkbox to select to include in the list all undergrad courses (number 499 and below). Must include courses with letters. For example, 100, 095, 105W, 1XX, XX1, 499.
3. Checkbox to select to include in the list all graduate courses (number 500 and above).
4. Button to update the list of courses displayed. Pressing the update button must not need to reprocess the data file: data must have already been loaded when the program started.

2.2.3 Course list

1. Display a list of courses which match the filter criteria the user has entered.
2. Show the department name and course number, such as “CMPT 106W”.
3. Sort the courses in alphabetical order.
4. Display the courses in at least 2 columns.
5. Allow the list to scroll if there is more data than will fit on screen.
6. The list may be empty (for example, selecting graduate MACM courses).
7. When the user clicks on a course, it selects the course for display in the “Course Offerings by Semester” panel, and “Statistics” panel.

2.2.4 Course offering by semester

See hints section for ideas on setting up the grid.

1. Display the offerings for the selected course (from the "Course List" panel) in a grid.
2. Across the top, display semester names (such as "Spring", "Summer", and "Fall") as column headers. You *may* list them by either calendar year (Spring through Fall), or academic year (Fall through Summer).
3. Down the left, list the year number.
   - If using calendar years, list as "2013"
   - If using academic year, list as "2012-2013"
   - Display the range of years necessary for the offerings of the selected course. For example, if the course was only offered in Fall 2001 and Spring 2005, display years between 2001 and 2005 (inclusive).
4. For each semester:
   - Add a button for each offering of the selected course for that semester.
   - Draw a border around each semester, forming a grid look.
5. Grid size:
   - The semester headings should be just tall enough for the words.
   - The year labels should be just wide enough for the year display.
   - All extra available space should be taken up by contents of the grid.
6. Updates
   - When the selected course changes, the grid must update to the new offerings.
   - When no course is selected, the grid must clear. You may either leave it blank, or display a message to the user prompting them to select a course.

2.2.5 Statistics

1. Display the name of the selected course, such as "CMPT 213".
2. Semesters Bar Graph
   - Display a bar graph showing how frequently the selected course was offered in each semester. Only handle the spring (1), summer (4) and fall (7) semesters. You may ignore any other semester.
   - Display bar labels of Spring, Summer and Fall (or Fall, Spring Summer of using academic years).
3. Campus Bar Graph
   - Display a bar graph showing how frequently the selected course is offered at each campus.
   - Display the bar labels of Bby, Sry, Van, and Other.
4. Update the statistics display and graphs when the selected course changes.
   - Correctly handle the case where no course is selected.

2.2.6 Details of course offering

1. Display detailed information about the selected course offering in the lower right panel.
2. Text display:
   - Course: department and the course number, such as "CMPT 213".
   - Location: such as `BURNABY`, `SURREY`, `OFF`, …
   - Semester code: such as 1131
   - Instructors: A comma separated list of instructors which taught the selected course offering.
     - Some offerings have multiple instructors, such as CMPT 165 Fall 2012 Burnaby.
     - Text too big to fit in one line should word-wrap to the next line (hint: use a `JTextField`). See Figure 3.
     - If the Instructor field is listed as "(null)", instead change it to be "" (the empty string).

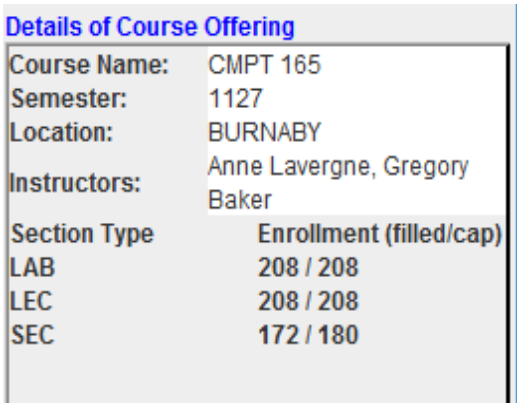


Figure 3: Course Offering details, including multiple instructors.

3. Display a table with the following columns:
   - Section type (such as LEC, LAB, …)
   - Enrollment for each section-type: number students enrolled and total capacity.
     - Sum together instances of a specific section-type within an offering. For example, add up the enrollment and capacity of all labs for a specific course offering.
     - For example, consider the following lines of the input file:

SEMESTER	SUBJECT	CATALOGNUMBER	LOCATION	ENROLMENTCAPACITY	ENROLMENTTOTAL	INSTRUCTORS	COMPONENTCODE
1147	CMPT	250	BURNABY	130	129	Anthony Dixon	LEC
1147	CMPT	250	BURNABY	26	26	Anthony Dixon	LAB
1147	CMPT	250	BURNABY	26	26	Anthony Dixon	LAB
1147	CMPT	250	BURNABY	26	25	Anthony Dixon	LAB
1147	CMPT	250	BURNABY	26	26	Anthony Dixon	LAB
1147	CMPT	250	BURNABY	26	26	Anthony Dixon	LAB
1147	CMPT	250	SURREY	82	81	Brian Fraser	LEC
1147	CMPT	250	SURREY	24	24	. Faculty	LAB
1147	CMPT	250	SURREY	24	24	Brian Fraser	LAB
1147	CMPT	250	SURREY	24	23	Brian Fraser	LAB
1147	CMPT	250	SURREY	10	10	Brian Fraser	LAB

Figure 4: Sample lines in the input data file.

- When processed, this generates the following two offerings (Burnaby and Surrey).

```
CMPT 250
      1147 in BURNABY by Anthony Dixon
            Type=LAB, Enrollment=129/130
            Type=LEC, Enrollment=129/130
      1147 in SURREY by Brian Fraser, . Faculty
            Type=LAB, Enrollment=81/82
            Type=LEC, Enrollment=81/82
```

Text 1: Course offering and section information for the lines of data shown in Figure 4.

- The data in the file may not seem to make sense. There could be more people in lecture than lab, there could be sections with 0 capacity, or 0 enrolled, or 9999. This is OK: just process the numbers as though they are valid numbers.
- Assume there is at most one offering of each course at each campus during a single semester. For example, group together all CMPT 120 information for Burnaby during the summer of 2000, even if there were actually two lectures in Burnaby that semester for CMPT 120.
- Figure 5 shows the details view for data in the example shown in Figure 4.

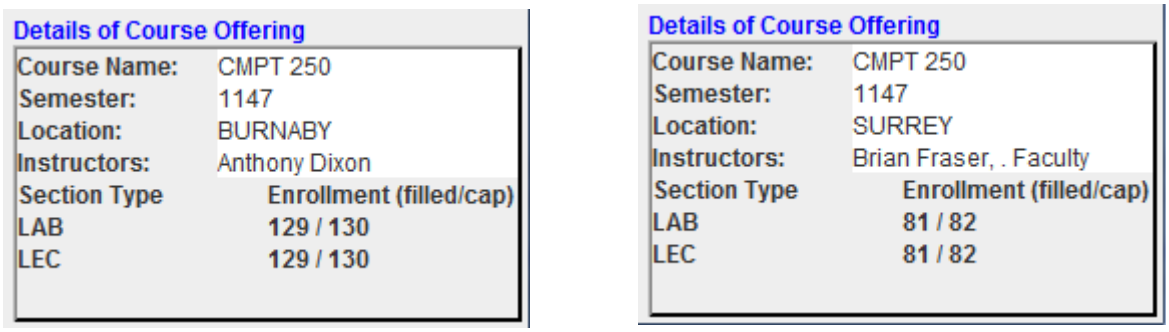


Figure 5: Details view for two different course offerings shown in previous table.

4. When selecting a different course, clear the information from the panel.
5. Update the display whenever the user selects a different course offering.

2.2.7 Resizing (see Figure 6 below)

1. UI must initially be displayed at a large enough size to show complete interface (Figure 1).
2. Resizing horizontally:
   - Left and right panels must not stretch horizontally.
   - Semester columns in the course offerings grid must expand or compress horizontally; the year titles (left of grid) must not resize horizontally.
   - Your application need not look good when the size is reduced beyond a minimum size. i.e., if some elements are not visible, you won't lose marks.
3. Resizing vertically:
   - Left: Course list grid must resize; the filter display must not.
   - Centre: Grid resizes, stretching all rows of the grid vertically; header at top must not resize.
   - Right: Extra space should appear below the “details of course offering” section (or in that section). Its elements should not stretch (such as the bar graphs or course details text.)

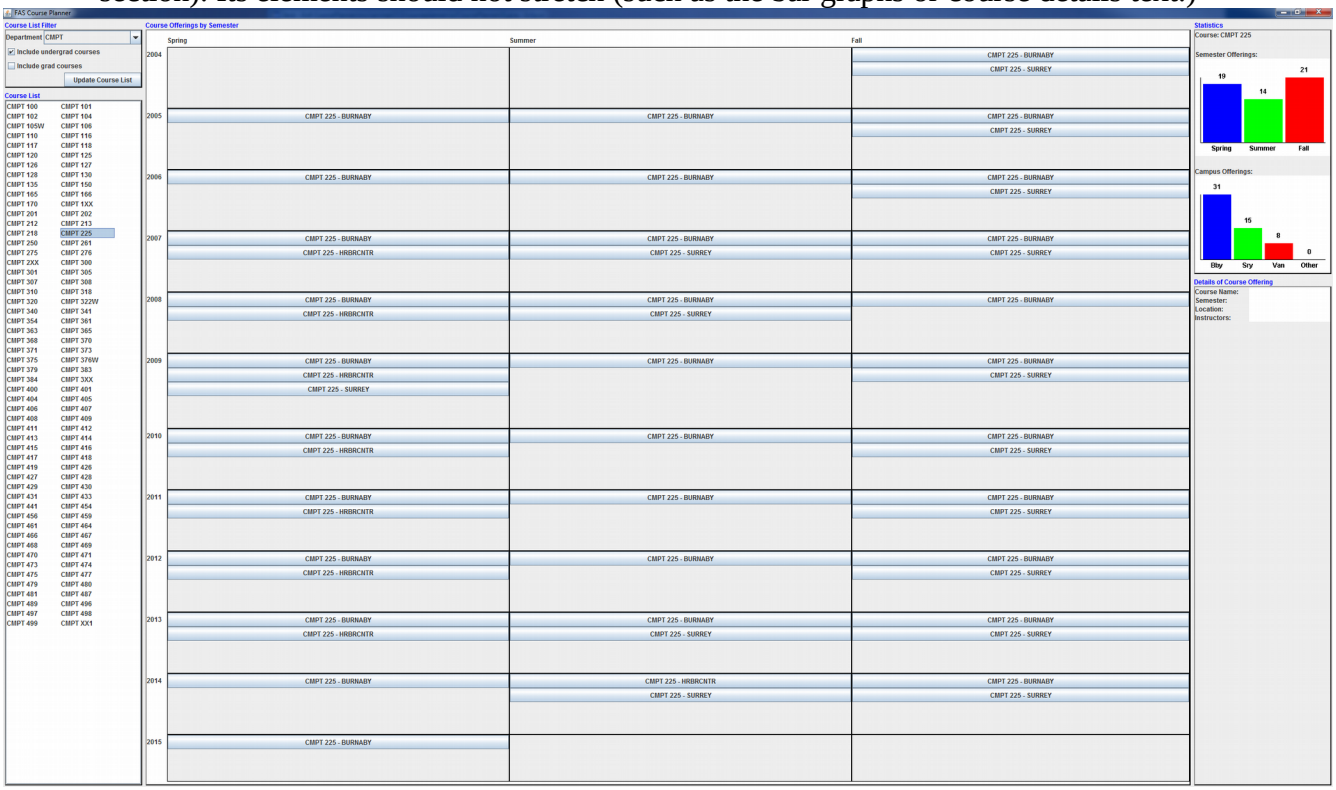


Figure 6: UI stretched vertically and horizontally.

2.3 Coding Requirements

Your program must:

1. Follow the course's code style guide.
2. Separate the UI and the model (data).
   - UI and model classes must be in two different package. For example: `ca.cmpt213.courseplanner.ui` and `ca.cmpt213.courseplanner.model`
   - Model must manage:
     1. Information loaded from the data file at program start (courses, offerings, sections, ...)
     2. Filtering options for the course list
     3. Currently selected course
     4. Currently selected offering
3. Modularity and class design
   - You must have well design classes.
   - Use numerous classes in the UI (derived from `JPanel`) to implement the UI. You must have at least 5 or more UI classes.
   - Use multiple classes in your model to store the system's information. You must have at least 5 or more model classes (or even 10+).
4. UI Class structure
   - You must create an abstract base class (ABC) to be used by all of your UI panels.
   - This ABC must support a title `JLabel`, and the border around the panel's main contents. The advantage of this is that all panels will have the same look without having duplicate code.
5. Observer use
   - Model must support notifying observers when its state changes using the Observer pattern.
   - Model *should* allow clients to register observers for a specific state change, such as when the selected course changes, or when the selected offering changes. This way clients are only notified when the desired change occurs.
   - UI panels must register observers with the model for notifications when a model state change has an impact on that specific panel.
6. Your program must not search through the entire data-set each time the user makes a selection on the UI. Processing all the data each click makes the UI too slow because with a huge data set, it is a *lot* of extra work.
   - Instead, process the data set into meaningful collections of objects, and then access these objects without having to search all records on each click. For example, when loading the data set, process it into courses, and then selecting a course can quickly load the required information.

3. Hints

The following hints may be useful as you create your application. These are not requirements, so you need not follow them. Please see the marking guide for exactly what is being marked.

3.1 Model Design

1. Start by designing the data storage classes.
   - Identify classes by analyzing input data-file format, and needs of the UI.
   - Do CRC cards for your classes.
2. Develop the API (i.e., method calls) the model needs to support in order for the UI to work.
   - Think about how each UI panel will get the information it needs from the model.
   - Think about what each UI changes in the model (select a course, …), and how the model will be told about this change, and store that state.
   - Consider using the facade pattern to expose a single model interface.
3. You must use the Observer pattern to update the UI (as described in the requirements section). Think about how the model will support registering an observer, and notifying that observer.
   - You may need multiple methods to add observers. For example, one to add an observer for a course-selection change, another for an offering-selection change.
   - You may need separate lists of observers for these different purposes.
4. Think about low level classes you could use. For example, would having a `Semester` class allow you to encapsulate some logic about converting a string (“1037”) into a `Semester`? What about a class which can encapsulate reading a CSV file without any knowledge of the meaning of the data?

3.1.1 General tips

1. Be *very* careful using ==. Most of the time you'll need `.equals()`!
2. Use asserts, and check that they are turned on! (Temporarily add an “assert false;” to test).
3. Considering overriding `equals()` for most of your data classes. This can be very useful when you are searching a list of items to see if it's already in the list.
4. When parsing a course number to check if it is grad or undergrad, you may find the following code helpful:

```
int getStringValue(String str) {
 return Integer.valueOf("0" + str.replaceAll("(\\d*).*", "$1"));
}
```

3.1.2 Model debug output

- Add a `dumpModel()` method which displays the entire data set to the console. Use this to check that your read-in and data-processing routines are working. Suggested output format shown to right.
- Note: This generates the output for iteration 1, so you must do it. However, the suggestion is to carefully ensure you have it correct!

```
CMPT 106
       1107 in SURREY by Harinder Khangura
              Type=LEC, Enrollment=29/32
              Type=TUT, Enrollment=29/32
       1117 in SURREY by Harinder Khangura
              Type=LEC, Enrollment=30/35
              Type=TUT, Enrollment=30/35
       1131 in SURREY by Harinder Khangura
              Type=LAB, Enrollment=57/61
              Type=LEC, Enrollment=57/61
       1137 in SURREY by
              Type=LAB, Enrollment=66/191
              Type=LEC, Enrollment=66/192
       1147 in SURREY by Harinder Khangura
              Type=LAB, Enrollment=43/50
              Type=LEC, Enrollment=43/50
       1157 in SURREY by Harinder Khangura
              Type=LAB, Enrollment=52/52
              Type=LEC, Enrollment=52/52
```

Text 2: Output generated by "dumping" the model.

3.2 UI Design

If you design each panel to rely on just the model (and not on each-other), the reduced coupling allows you to test components more independently and implement features and correcting bugs more easily.

3.2.1 Panel Abstract Base Class (ABC)

1. As described in the requirements section, you must have an abstract base class which inherits from `JPanel` be the base class for all your UI panels (see Figure 7).
2. Have ABC display the panel's title (such as "Course List Filter").
3. Have ABC display a border around the main area where panel's contents will go.
   - Create a border on a panel using:

```
myPanel.setBorder(BorderFactory.createBevelBorder(
    BevelBorder.LOWERED,
    Color.black, Color.gray));
```

4. Consider using `BorderLayout` for the ABC. Put the title in the `NORTH`, and the user's content in the `CENTER`.
5. All UI panels need a reference to the model to get the information.
   - Make each UI panel's constructor accept the model as a parameter and pass it to the ABC constructor.
   - Make ABC's constructor accept the model as a parameter and store it in a private field.
   - Have ABC provide access to the model for derived classes through a public or protected method (such as `getModel()`).

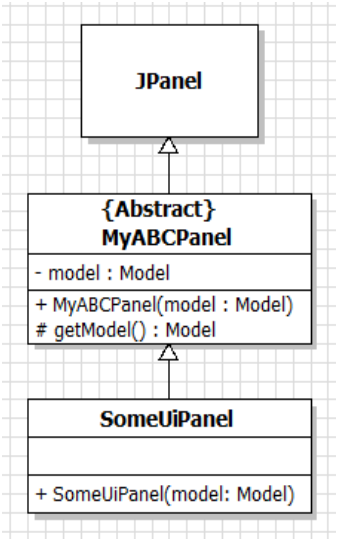


Figure 7: Suggested ABC design for UI panels.

6. Component Resizing:
   - The size that a component wants to be displayed at is accessible via `getPreferredSize()`
   - The maximum size it will allow itself to be stretched to is accessible via `getMaximumSize()`
   - Prevent a component (`daItem`) from resizing vertically but allow it horizontally:

```
Dimension prefSize = daItem.getPreferredSize();
Dimension newSize = new Dimension(
    Integer.MAX_VALUE,
    (int)prefSize.getHeight());
daItem.setMaximumSize(newSize);
```

   - Hint: Make the above code a function!
7. Think through how derived panels will place content in the main content area. You may need provide a method to access that content area (a `JPanel`, quite likely!).

3.2.2 Course List Filter

1. To create a combo box, use a `JComboBox`.
2. Provide the combo box a set of options to display in a `Vector<String>`:

```
Vector<String> options = new Vector<String>();
options.add("First");
options.add("Second");
JComboBox<String> myComboBox = new JComboBox<String>(options);
```

3. Consult the `JComboBox` documentation to find out how to get the selected item.

3.2.3 Course List

1. Consult Oracle's Java tutorial on `JListView` for help setting up the list view.
2. To space out items in the list, consider calling the `JListView`'s `setFixedCellWidth()` method.
3. To handle list selection events, consult Oracle's Java tutorial on list selection listener.

4. Give the list a preferred size (`setPreferredSize()`) to have it be a reasonable size at startup.
5. Register as a listener to the model for department selection change events.

3.2.4 Course Offering by Semester

1. Set the layout to be a `GridBagLayout`.
   - This allows you to create a grid of unevenly sized elements (unlike a `GridLayout`)
   - `GridBagLayout` documentation: Oracle tutorial, JavaDocs, page of useful information.
2. `GridBagLayout` hints:
   For the `GridBagConstraints` object, set the following fields:
   - Location in the grid to place the contents: `.gridx`, and `.gridy`
   - Add space around the contents of a cell with `.ipadx` and `.ipady`
   - Stretch contents to fill cell's area: `.fill = GridBagConstraints.BOTH;`
   - Make cells stretch evenly using `.weightx = 1;` and `.weighty = 1;`
3. To put multiple components inside a cell, you'll need to put the components into a `JPanel`.
   - To make your three columns the equal size, make the JPanels all start off the same width. If you have set the .weightx property (above) for all columns, Java will then scale your initially-equal sized JPanels at the same rate, keeping them the same size. You can set the preferred size such as:
     `myPanel.setPreferredSize(new Dimension(1, 1));`

3.2.5 Statistics – Bar Graphs

1. Use the `BarGraphModel` and `BarGraphIcon` classes provided on the course website to display the bar graphs.
2. Be sure to handle no selection by drawing an empty bar graph (give `changeData()` an array of 0's).
3. If your statistics panel registers as an observer with the Model, when the observer is called you'll want to update the data stored in the `BarGraphModels` for the two graphs. Plus, you can trigger the `BarGraphIcon` to redraw by calling the full panel's `updateUI()` function. This will force a redraw of all components in that part of the UI, and hence redraw the graphs with the new data.

3.2.6 Details of Course Offering

1. For the table of sections (such as lectures, tutorials, …) consider using a `GridLayout`.
   - Disable vertical stretching on the grid's panel (see Component Resizing above).
   - You can remove all elements from a panel using `myGridPanel.removeAll();`
     Useful for redrawing the grid from scratch after an update.
2. Register with the model as an observer for when the offering selection changes.
3. Use a JTextArea for the data display of the first 4 fields because it can be set to enable word wrapping (required for long instructor strings). To enable line-wrap on a JTextArea, use:

```
daArea.setLineWrap(true);
daArea.setWrapStyleWord(true);
```

4. Bonus

The following are bonus features you can implement for a few extra marks. Your application's implementation of the basic features must be largely complete for you to earn any of these bonus marks.

1. **Select multiple courses**
   - Allow the user select some “favourite” courses to view.
   - Allow the user to double-click on a course in the Course List and add it to the Favourites.
   - Display a new panel on the UI for showing the Favourites.
   - Allow the user to double-click a favourites course and remove it from the favourites.
   - Single clicking a course in the favourites list displays that in the rest of the UI.

2. **Restrict display of offerings**
   - Add some filtering options for the offerings grid to restrict what offerings are shown.
   - Able to filter based on campus (checkbox for showing Surrey, Burnaby, Vancouver, Other).
   - Able to filter based on showing only a limited range of years. Such as limit to past *N* number of years (where user can edit N).
   - For example, user only wants to see course offerings for last 3 years for the Surrey campus.
   - The bar graphs should show only those offerings displayed in the grid.

3. **Multi-channel bar graph**
   - Change the bar graph display of Semester Offerings to differentiate between campuses.
   - For example, the following bar graph gives the idea. You are able to adjust this as to how exactly you display the graph (you may not use any 3rd party libraries).

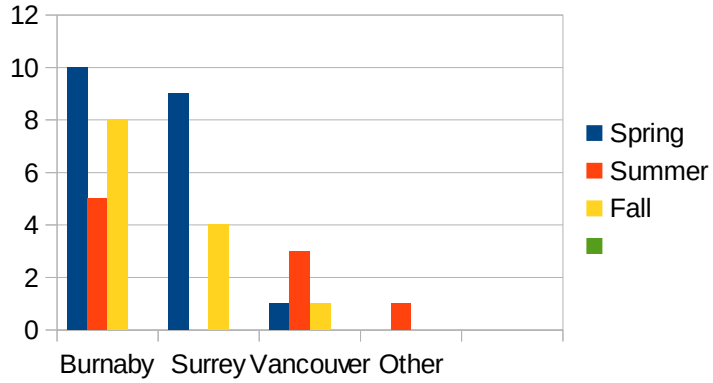


Figure 8: Bar graph display for bonus grouping.

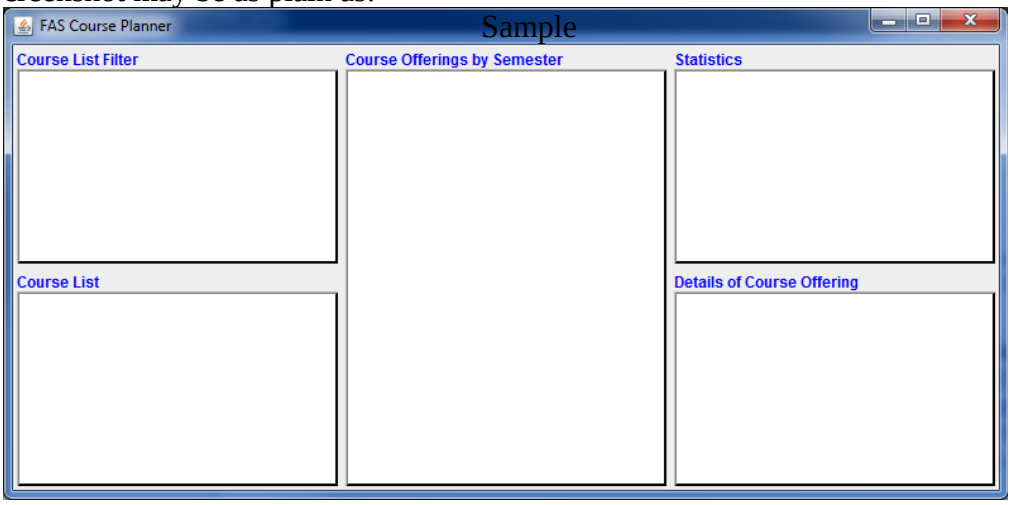
5. Deliverables

Submit each iteration to CourSys: https://courses.cs.sfu.ca/.
You must create a group in CourSys (even if you worked individually). Submit one solution per group.

5.1 Iteration 1: Dump Model & UI Skeleton

This is a proof of progress milestone. It shows you have some of the model implemented (parsing data, building a meaningful internal representation), and the UI skeleton. Submit the following two files:

1. `output_dump.txt`
   - A text file of the complete output generated by calling `dumpModel()` on your model after it processed the provided input file.
   - Format should be *similar* to Text 2, on page 10.
2. `ui_capture.png`
   - A screenshot of your (possibly) incomplete UI.
   - Must show all panels with titles and borders, but each panel may be empty. OK to have more completed.
   - Screenshot may be as plain as:



5.2 Final Submission

Full ZIP file of project submitted.

- *No* screen-shots or sample outputs are required.

Please remember that all submissions will automatically be compared for unexpected similarities.