Full Stack Application Development
Software Workshop 2

Assignment 1:
GUI

Set by
Jacqui Chetty

Early Testing Deadline:
TBA

Final Submission Deadline:
Monday 14th March 2022 4pm GMT

Use Canvas for the definitive deadlines

This assignment is worth
25 % (**twenty-five** percent)
of the overall course mark

Overview

Table of contents

1 Introduction

The aim of this assignment is to complete a GUI application based on the idea of a student capturing system. A student can be added to the system when a studentId, year of study and the allocation of 3 module choices is entered. The system also ensures that some editing can take place; as well as the deletion of a student from the system. The application has been simplified for practical purposes.

Descriptions of the overall requirements are given in Coding Tasks, but specifics of these requirements are generally left to 'TODO' comments in the skeleton code. You also need to analyse differences between your version's output against an expected student-data.txt. You will be given a student-data.txt. The file holds an initial set of student data. When you get to Testing your code you will be asked to complete a set of testing operations, such as add, edit and delete. The Testing your code section will provide an expected set of data for viewing before as well as after testing (add, edit and delete) has been completed.

2 Marking scheme

Question	Description	Marks
1	`Controller`	40
2	`EditStudentController`	20
3	`2 x .fxml files each 10 marks`	20
4	`Student`	10
5	`StudentData`	10
	Total	100

3 Coding tasks

We are making use of the MVC pattern so all of the following classes pertaining to the project form part of the following package(s):

```
Assignment1
├── .java/fsd/assignment/assignment1/Controller.data
├── .java/fsd/assignment/assignment1/EditStudentController.data
├── .java/fsd/assignment/assignment1/datamodel/Student.java
├── .java/fsd/assignment/assignment1/datamodel/StudentData.java
├── .resources/fsd/assignment/assignment1/main-view.fxml
├── .resources/fsd/assignment/assignment1/edit-students.fxml
└── student-data.txt
```

The following sections describe each class and Figure 1 below shows an example of the application when it is executed. There is also a video on Canvas under Assignment 1 that shows the execution of the application.

Note 1: When you create the Labels as well as the fx:id's take note that these correspond to the controller class variables, **these variables in the controller classes have been specified in the template**. Do NOT change any variable names in the controller classes, otherwise your application will not execute when auto-testing takes place.

Note 2: The window / scene(s) that you develop do not have to be identical to the image provided, and the vertical / horizontal gaps are optional); **the words window and scene are used interchangeably**.

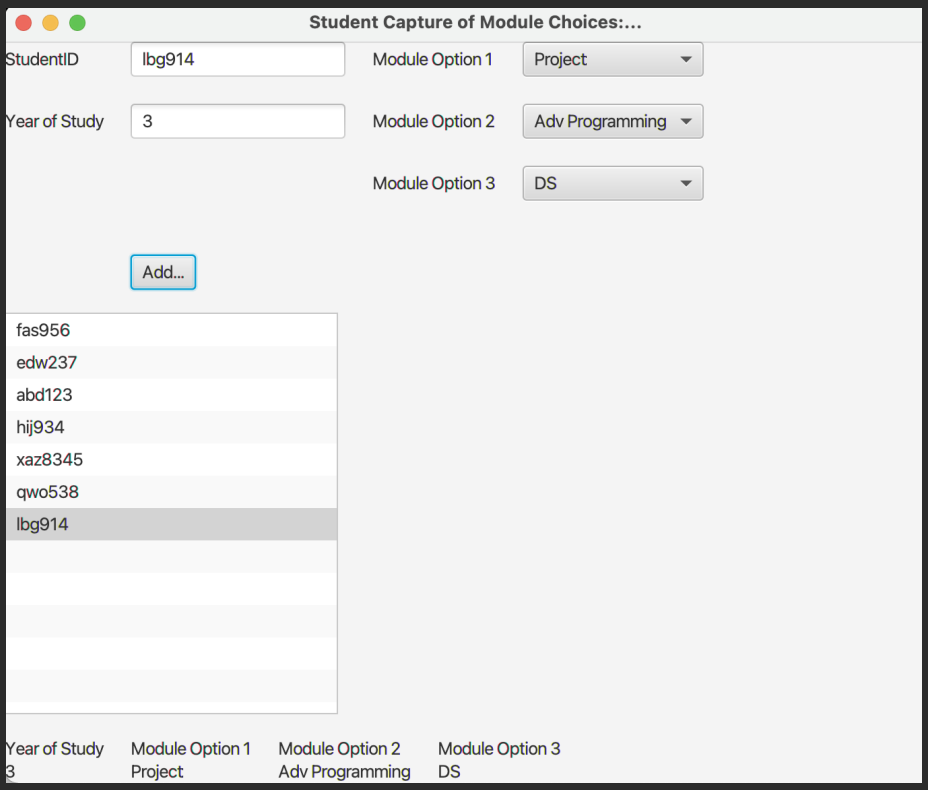


Figure 1 Expected window when application loads, **note that data may differ**.

3.1 Class 1 - Student.java

The `Student` class creates a data structure on which the functionality of the application is built.

Functionality:

1. include a constructor that accepts all of the instance variables as parameters;
2. set each parameter to the appropriate instance variable;
3. include getters() for all instance variables;
4. include a toString() that returns studId.

3.2 Class 2 - StudentData.java

The `StudentData` class is a Singleton class that ensures the integrity of the data by including an Observable ArrayList. The ArrayList stores the student data that must be written to / read from a .txt file called student-data.txt (provided to you).

Functionality:

1. create an instance of StudentData according to the rules of a Singleton class;

2. complete the getters() as instructed as part of the TODO;
3. complete the loadStudentData() according to the TODO's;
4. complete the storeStudentData() according to the TODO's;
5. include an addStudentData() so that a student can be added to the Observable ArrayList;
6. include an deleteStudent() so that a student can be removed from the Observable ArrayList.

3.3 Class 3 - main-view.fxml

The `main-view.fxml` class creates a structure for the main window / scene, as seen in Figure 1 above.

Functionality:

1. create a BorderPane<> and ensure that you connect the fxml to the correct controller;
2. include a <top> section;
3. include a GridPane<> with a vertical and horizontal gap of 20;
4. make use of the variables declared in the Controller class to create the <top> section within the GridPane<> (use the image above to assist you;
5. there is a Button<> element towards the end of the section, ensure that you include onAction="addStudentData", no fx:id is required for the button;
6. the last element is the Label<> tag that you may not notice, this label is used for a message when a studId or year of study is not entered for adding a student (discussed later);
7. the <left> section of the border has been included for you, no further code is required here;
8. the <bottom> section of the border must be included by creating a VBox<>;
9. inside the VBox<> create a GridPane<> with a horizontal gap of 20;
10. complete the Lables with the applicable fx:id's - note that the variable names for the fx:id's can be found in the controller class.

3.4 Class 4 - Controller.java

The `Controller` class is responsible for carrying out the actions when a user interacts with the main-view.fxml scene / window.

Functionality:

1. all the variables have been declared as well as explained, especially in terms of linking / corresponding to the main-view.fxml document;
2. **initialize()**-> housekeeping and ongoing tasks required with action events:

2.1. first complete the changed() method for the ChangeListener;
2.2. the setOnAction for the ChoiceBox has been completed for you;
2.3. the modChoices array must be added to each ChoiceBox (remember that there are 2 ChoiceBoxes;
2.4. create a new listContextMenu object for the delete functionality;
2.5. create a MenuItem for the deleteStudent;
2.6. complete the handle() of the delete by getting the student to be deleted and call the deleteStudent();
2.7. these actions need to be repeated for the edit;
2.8. compete the updateItem() as part of the cell factory so that studId's are either in the ListView or not;
2.9. create the code to ensure that the studId's are sorted in the ListView by year of study in ascending order, so year 1 students are found at the top of the list, and so on;
2.10. follow step 1 to 3 regarding the items appearing on the window.

3. **getChoice()**-> this method determines which choice of modules has been made:
   3.1. pass the ActionEvent into the method and assign values to choice1, choice2 and choice3.
4. **addStudentData()**-> this method is used to add a new student to the file:
   4.1. get the values from the textfields entered by a user;
   4.2. ensure that both the studId and yearStudy fields are occupied, if not a message should be displayed in the validateStudent label to indicate that the student cannot be added;
   4.3. if the student can be added the add the new student to the studentToAdd object,passing the values as parameters;
   4.4. use getInstance() to addStudentData in the StudentData class;
   4.5. ensure that the student recently added is selected when the list of studId's appears after the add.
5. **deleteStudent()**-> this method is responsible for ensuring that the delete action is initiated when a student needs to be deleted:
   5.1. create an alert so that a "are you sure you want to delete" window pops up;
   5.2. if the user wants to delete the deleteStudent() must be called.
6. **editStudent()**-> this method sets up the editing functionality, only module choices can be edited, the method makes use of a DialogPane<> with a

seperate edit-students.fxml, see Class 5 below for the details of this as well as Class 6 - the controller attached to the fxml:

6.1. ensure that the mainWindow has been specified;

6.2. set the title as well as the header around which student is to be edited;

6.3. create a FXMLLoader object and ensure that the edit-student.fxml is specified as the one to be loaded;

6.4. use a try...catch block to load the dialog window;

6.5. create an object for the EditStudentController so that the edit can take place via the processEdit();

6.6. call the setToEdit() so that the student to be edited is displayed in the window;

6.7. ensure that the buttons OK and Cancel form part of the dialog window;

6.8. if the intension of the user is to edit call the processEdit() and ensure that once the edit has taken place the edited student is highlighted.

3.5 Class 5 - edit-students.fxml

The `edit-students.fxml` class creates a structure so that a student's details can be edited (only module choices are editable). Figure 2 shows the edit window (note that this is an example only and does not link to the student-data.txt file that is provided to you).

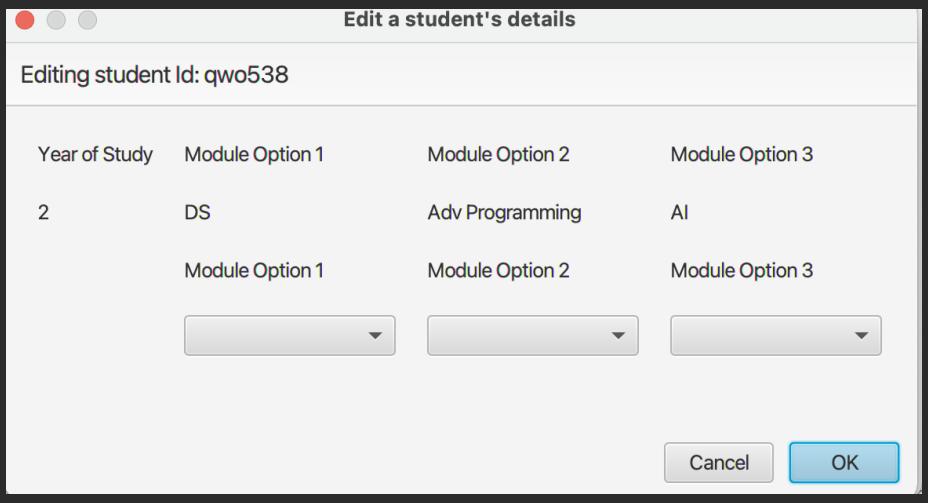


Figure 2 The edit window.

Functionality:

1. complete the .fxml according to the instuctions within the edit-students.fxml;

2. you will find all the variables required for the .fxml in the EditStudentController class.

3.6 Class 6 - EditStudentController.java

The `EditStudentController` class is responsible for carrying out the actions when a user interacts with the edit-students.fxml scene / window.

Functionality:

1. all variables required for the class are provided;
2. **initialize()**-> housekeeping and ongoing tasks required with action events:
    2.1. add all modChoices to each ChoiceBox so that the user can see the choices when the dropdown box appears.
3. **setToEdit()**-> this method ensures that the student to be edited displays the details before the edit:
    3.1. ensure that the current details if the student to be edited are displayed;
    3.2. get the new module choices.
4. **processEdit()**-> this method completes the editing process:
    4.1. create an observableList based on the data structure Student;
    4.2. get the studId and year of study;
    4.3. remove the student to be edited from the arrayList;
    4.4. add the student back on with the edited module choices.
5. **getChoiceEdit()**-> this method determines which choice of modules has been made to edit:
    5.1. pass the ActionEvent into the method and assign values to mod1S, mod2S and mod3S.

4 Other important sections

4.1 Submission requirements

The submission part of your assignment is every bit as important as the coding part. Your submission `zip` file **must**:

1. be a `zip` file of your own IntelliJ project;
2. be created by IntelliJ;
3. be renamed according to the requirements in §4.7 How to submit.

Your submission `zip` file **must not**:

1. be bigger than 1 MB (one megabyte) in size when compressed;

2. have contents bigger than 1 MB (one megabyte) in size when uncompressed;
3. contain any one file bigger than 500 kb (five hundred kilobytes);
4. contain more than 100 files in total.

Failure to comply with the submission requirements can result in penalties.

4.2 About the project

Example:

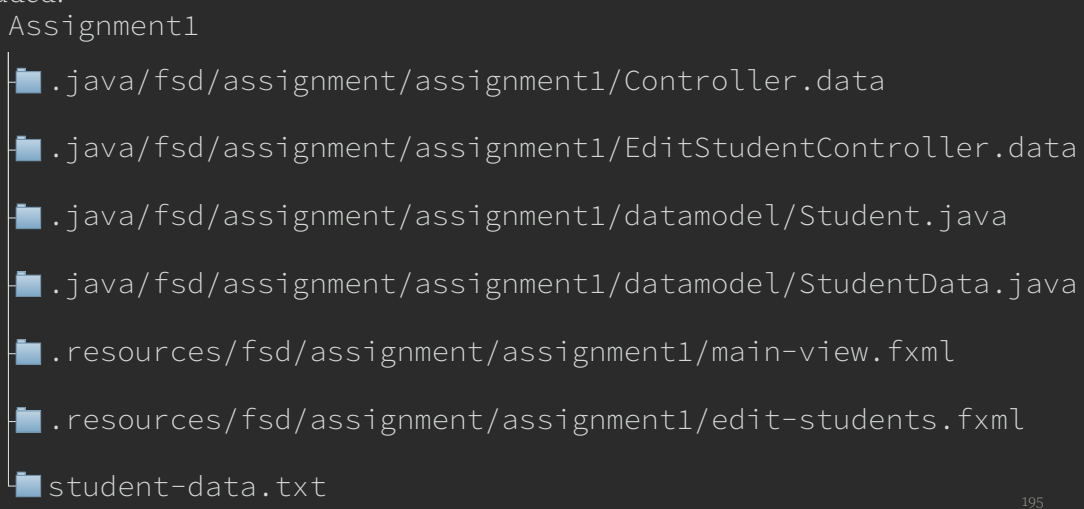
You are given an IntelliJ project containing skeleton code: this a framework that lacks functionality. You must use IntelliJ to expand this skeleton to complete the assignment according to the instructions in §3 Coding tasks.

4.3 Obtaining the assignment project

Go to Canvas, then go to the FSAD module, then to Assignment 1.[1] There is a link to a `.zip` file. Download and unpack this file to its own folder and move the unpacked directory (folder) somewhere sensible.

4.4 Project structure

The structure of the project is as follows (take note that the use of packages is included:



The IntelliJ project has a `src` folder and a sub-folder `main` which contains the `java` sub-folder. Within `java` there is a `fsd.assignment.assignment1` package that holds Controller, EditStudentController and Main. Within `fsd.assignment.assignment1` there is another package `datamodel` that contains Student and StudentData. See the structure above of see Figure 3 below to understand the full package

[1] Direct link: https://canvas.bham.ac.uk/courses/56081/assignments/343492

structure. Take note of the student-data.txt file from where data is loaded and stored to / from when data is added, edited or deleted.

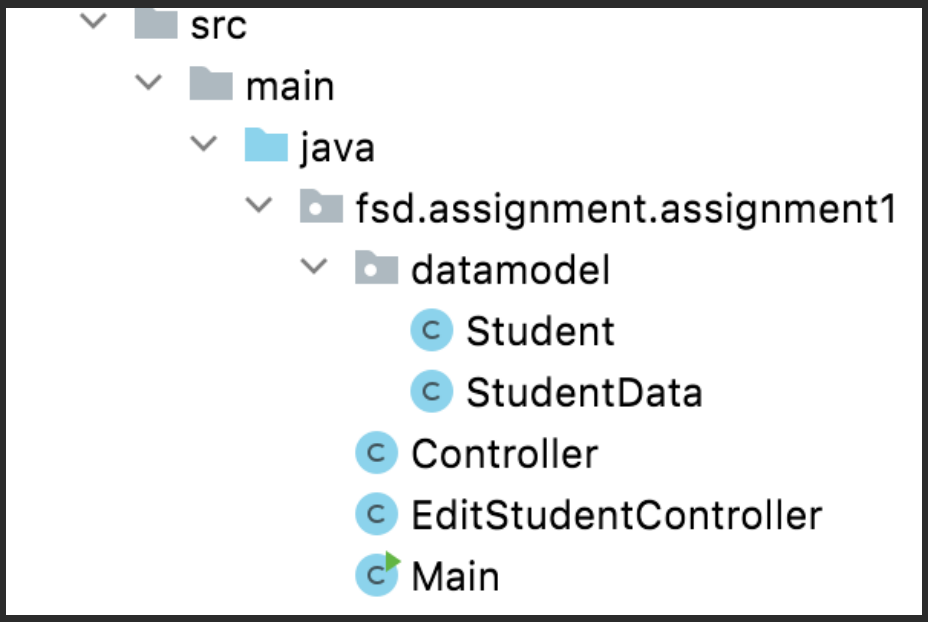


Figure 3 The folder structure.

4.5 Loading the assignment project

Remember when you load this project into IntelliJ to open the directory itself rather than one of the files inside it. You are required to work in IntelliJ because you must submit your completed IntelliJ project.

4.6 Testing your code

The student-data.txt file that you receive should look as follows:

```
fas956 1 OOP Maths AI
xaz8345 2 DS Adv Programming AI
hij934 2 DS Adv Programming AI
lbg914 3 Project Adv Programming DS
qwo538 2 Maths Project AI
edw237 2 Data Algo Adv Programming Project
```

Take note (see Figure 4 below) that the file list and the ListView are ordered differently, due to the ListView being a sorted list by year of study.

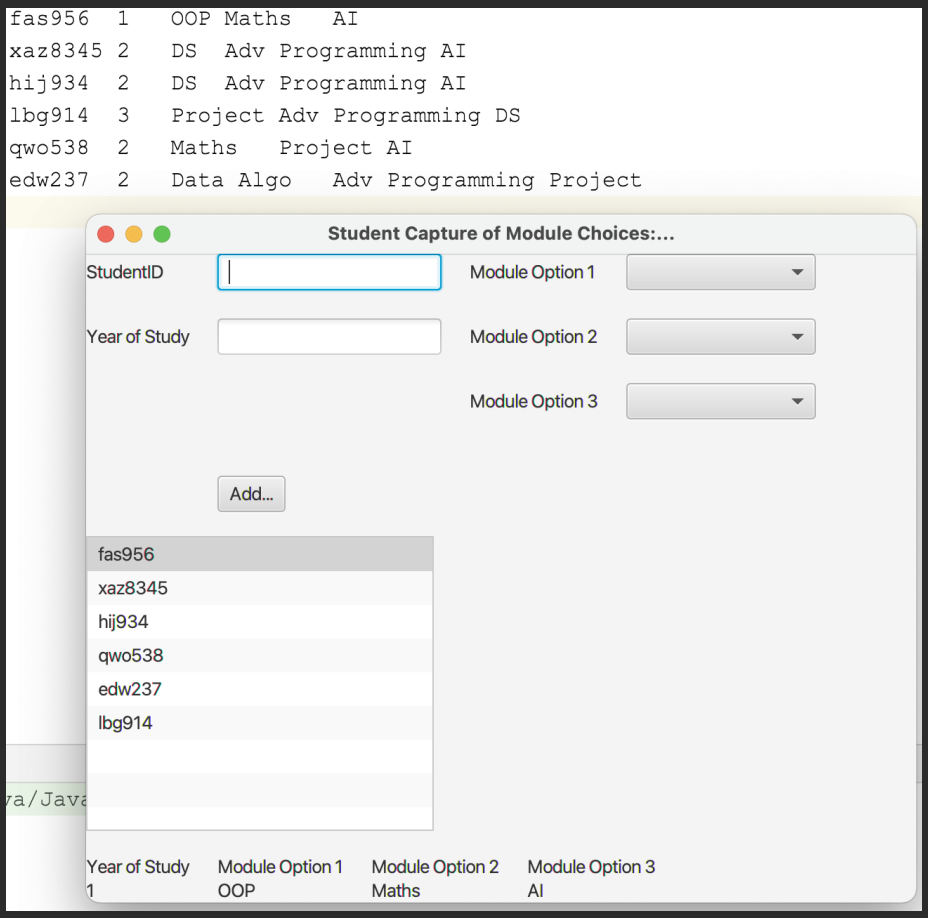


Figure 4 Contrasting the .txt with ListView from main scene.

Let's get testing ===»>

Add a student:

1. enter a studId -> **pew1234**
2. enter a year of study -> **2**
3. click module choice 1 choice box and choose -> **DS**
4. click module choice 2 choice box and choose -> **Advanced Programming**
5. click module choice 3 choice box and choose -> **Project**

Edit a student:

1. right-click on student **hij934** and choose edit
2. click module choice 1 choice box and choose -> **Maths**
3. click module choice 3 choice box and choose -> **Project**
4. click on **OK** to accept the changes

Delete a student:

1. right-click on student **qwo538** and choose delete
2. click **OK** to confirm deleting the student

Cause an error when adding a student:

1. enter a studId -> **abc123**
2. do not enter the year of study

3. click module choice 1 choice box and choose -> **AI**
4. click module choice 2 choice box and choose -> **Maths**
5. click module choice 3 choice box and choose -> **Project** - this should result in an error message as seen in Figure 5
6. now enter a year of study so that the student can be added, taking note that the Error Message should disappear
7. delete studId **abc123**

After the testing the student-data.txt should look as follows:

```
fas956 1 OOP Maths AI
xaz8345 2 DS Adv Programming AI
lbg914 3 Project Adv Programming DS
edw237 2 Data Algo Adv Programming Project
pew1234 2 DS Adv Programming Project
hij934 2 Maths Project AI
```

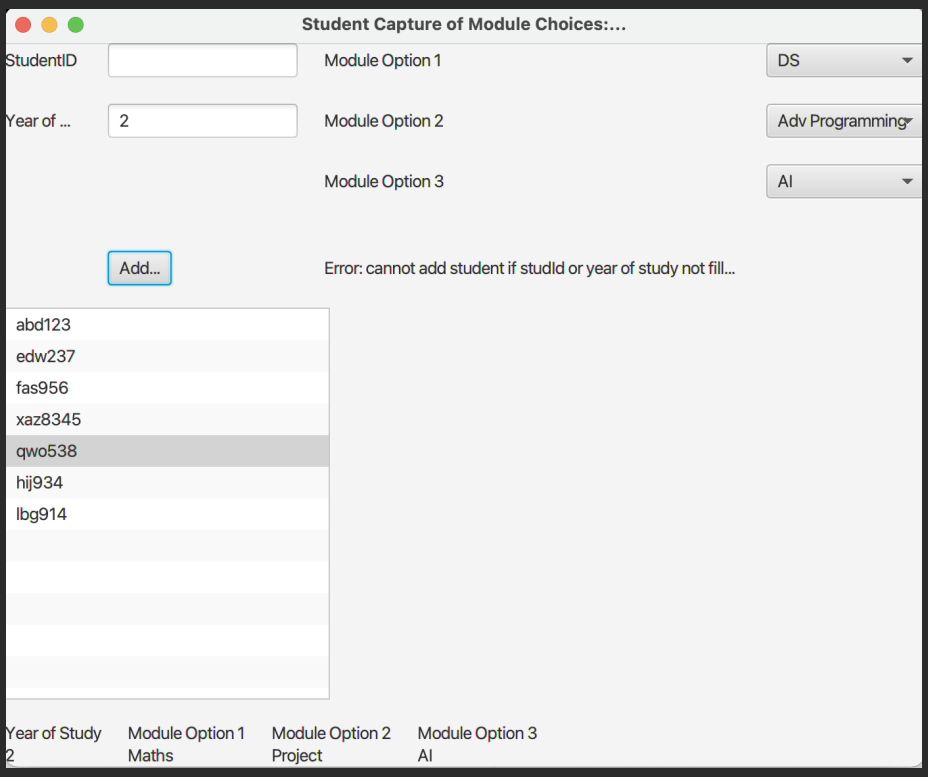


Figure 5 Expected output if **StudId** or **yearStudy** is not entered when adding a student.

4.7 How to submit

1. Build › Rebuild Project and fix any compilation errors.
2. File › Export › Project to Zip file… and rename the resulting `zip` file to `FSADAssign1_Givenname_Familyname_studentIDnumber.zip`
   If you officially have only one name then use `X` for the 'missing' name.
3. Ensure you do not include a previous exported `zip` file in the latest project export (a `zip` inside a `zip`) because this means you cannot guarantee the autotester will run the latest version of your code.
4. Upload the renamed `zip` file to Canvas for the Java course Assignment 1.
5. Canvas will probably add some extra information to the end of the file-name you have uploaded: do not worry about this.