CIS 2430 A2 README/Checklist

|  |  |
| --- | --- |
| Name: | Peter Hudel |
| UoG Email Address: | [phudel@uopguelph.ca](mailto:phudel@uopguelph.ca) |
| Partner’s name (if applicable) | Garnet Mclaren |
| Percentage of your submission that is taken from the starter code provided (approximately) | 0% - My code was the basis of what was used for A2 |
| How to run your application | To run this program, included in the .zip file there is a folder labelled “src”. This folder contains all of the source files for the assignment. Go into that directory and connect to the database, compile the program and run the program Planner. This will launch the gui. javac -cp .;mysql-connector-java-8.0.13.jar \*.java Then run the program. javac -cp .:mysql-connector-java-8.0.13.jar Planner |
| Notes for TAs (anything special we should know when grading your assignment) | In general, I did all the backend calculations. This includes completing the methods and functions, as well as setting up the database connections.  Garnet completed the frontend work, primarily the GUI. He used my backend code and turned it into a visual representation, demonstrating the user stories visually. **Anything in bold is what he did.**  We did not use any sample code that Judi provided other than the database code. This can be seen by comparing many of my functions from my A1 compared to my A2. Although refactoring occurred, many of the functions will be carried over.  To use the GUI:   * You can load/ save a student by using the File option * To remove a student, click on the course you want to remove and click the button * To change a grade in the transcript, you should click on the course you want to change, add the grade in the text box under the button and click the button * There are currently 2 students in the database: * name: “beep” number: 1234567 * name: “dumdum” number: 9876543 |

|  |  |
| --- | --- |
| **Learning Outcomes** | **3 examples from your code. File name, line number** |
| refactor and restructure class design for improved encapsulation, modularity, cohesion and coupling | * Remove plan of study, turn it into 2 lists: one of attempts and a hashmap of planned courses (Student.java, line 260) * Remove attempt info from course and create Attempt class (Attempt.java, all of it) * Move course and degree info into a package called univ (Course.java, line 1). Set setters to protected (Course.java, line 117) |
| demonstrate use of inheritance through super/sub classes as well as through the use of interfaces | * The degree hierarchy has the use of inheritance. The Degree.java class is a parent class that implements basic setters and getters. This helps to reduce unnecessary code. The children of Degree can use these methods, alongside overridden methods of their own through the use of super() (Degree.java, 78, example of a method the children can use) * The Degree classes, such as HonoursDegree and GeneralDegree and examples of child classes that extend the Degree parent. These classes are primarily where the requirements are determined. (GeneralDegree.java, line 13 shows it extending degree, line 29 shows super() * BCG, CS, Seng and BTA are children of the type of degree, which makes up what the major is. (CS.java, line 10 shows that it extends HonoursDegree) * In general, it goes Degree, Degree type, Major |
| demonstrate clear understanding event driven programming through well designed listeners and gui components | * **Please refer to the Planner class. Each button had an anonymous method/ lambda which was the Listener for each action performed.** **Garnet did the GUI** |
| demonstrate service-based error handling through a rich set of exception classes that communicate specific errors to client classes | * Didn’t really have time to finish this tbh |
| create a repeatable testing suite and justify the choice of test cases | * The Test.java class is where I did most of my testing for the backend. I would have the user input numbers for what test to perform, similar to what Planner.java did in A1, but this was only for testing. * I could add/ remove test cases by creating new options for the user to input. I could also run the program multiple times with different test cases, making it repeatable * Some test cases include: loading/ saving a student. Loading a student, adding courses to the transcript, saving the student, exiting, and reloading the student to determine if it did save, view a list of prereqs for a specified course, calculate the GPA with the 10 most recent courses. * All these test cases were implementations of the user stories required in the assignment specifications. |
| design and create a graphical user interface that is learnable and usable | * **Please refer to the Planner class** |
| use inner classes, anonymous classes, and/or lambdas effectively | * Anonymous class – Having multiple Comparators to sort based off of what is needed (Attempt.java, line 79) * Utilize these anonymous classes in order to properly sort the user’s plan according to semester taken * ActionListeners are inner classes that determines when an action must be done in the GUI. Garnet did this. Examples are in (GUI.java, line 779,785) **Garnet did the GUI** |

|  |  |
| --- | --- |
| **Required elements** | **Examples from your code (File name, line number)** |
| Exceptions and try/catch loops | * oops |
| Error prevention/handling (might also be try/catch or might be input checking) | * We made sure that a student was loaded beforehand, so none of the options had any functionality before a student was loaded |
| Two different layout managers | * **Grid (GUI.java, line 179) and border layouts (GUI.java, line 127) were used.** * **Garnet did the GUI** |
| Separate window/panel for administration | * **A JPanel was used for the admin view (GUI.java, line 358)** * **Using setVisible to switch between two different views. This was done in numerous locations. (GUI.java, line 793)** * **Garnet did the GUI** |
| Listeners | * **Lambdas were used to perform action listeners. (GUI.java, line 779, 785, 791)** * **Garnet did the GUI** |
| Course class refactored and immutable | * No course attempt info (stored in Attempt.java, all of it) * Setters set to protected (Course.java, line 117, 137) |
| Attempt class created | * Attempt class is created (Attempt.java) |
| Classes in package | * University classes are in a univ package (Course.java, CourseCatalog.java, Degree hierarchy) |
| Refactor Plan of Study (eliminating it) | * Plan of study is gone, now is an Array List of Attempts (Student.java, line 27) * Plan of Study is gone, planned courses is a HashMap of a course in reference to a semester to be taken (Student.java, line 28) |
| Database usage | * I tested out how we would connect to the database by testing it in the Planner.java class (line 48, line 179) * **We then connect to the database during the GUI to load a student or save it, depending on the user’s choice (saving a student is done in GUI.java, line 813)** |
| Javadoc comments (the most complete examples) | * All Javadoc comments are completed for each class. * Most complete examples include: (Degree.java, line 139-145) (Student.java, line 357) |
| Evidence of testing | * The entire planner class included was used to test backend stuff. * Each user story has a comment following the code used to test if it works. Values would be changed and results printed to the console. * (Planner.java, all lines) Example test includes calculating the most recent 10 courses for the GPA (Planner.java, line 141) |

**INSTRUCTIONS FOR PARTNERS**

Partners must each complete this sheet and both partners must submit. If you do not submit you will not get a grade for A2. Complete this sheet in such a way as to illustrated YOUR contribution to this assignment.