Object Oriented Programming - A2 README/Checklist

NOTE: this is essentially A1 refactored and overhauled (also has a GUI now)

Name:	Mitchell Van Braeckel
UoG Email Address:	mvanbrae@uoguelph.ca
Partner's name (if applicable)	N/A -> worked alone
Percentage of your submission that is taken from the starter code provided (approximately)	0% -> I didn't bother looking at Judi's posted A1 code because I was satisfied with my first A1 grade (mine worked almost perfectly)
How to run your application	Open the project called "Planner", that is in the zipped submission, in NetBeans IDE 8.1 Then, once it's open in net beans, run "Planner.java". This will execute the program I've created for A2. (I cannot guarantee it works perfectly if your run this another way, I used this to create it. Note that we are allowed to use a GUI builder, like NetBeans or Eclipse, so I assume we should be running the program using that as well. Also note that I'm unfamiliar with running java programs using terminal. The A2 description never stated how we had to execute/run our program, so I assume that's what this section is for.) EDIT: It was brought to my attention that this method is not acceptable (even though there was not anything in the A2 instructions saying that it must be instructions for command line). Thus, I am sending this with instructions of how to run it on command line. Note that because NetBeans works differently, doing the below instructions will work but it won't be able to find my files. So, you first must move all the *.csv and *.bin files into the "src" folder from where it currently is in "Planner". 1) cd to wherever my submission folder is (eg. Desktop) 2) cd mvanbrae/Planner/src 3) javac planner/*.java univ/*.java 4) java planner/Planner
Notes for TAs (anything special we should know when grading your assignment)	Most of my program handles erroneous/bad input (even for file reading/writing) and it may confuse them while testing. Just look at the tooltips and/or the documentation in the code for what good input is. I don't use exceptions for everything, but it should be handles with other methods (i.e. using if statements, or some other conditional check) Generally: Numbers are handled by checking >= 0 Strings are checked that they aren't empty or "" (sometimes if null, but usually not because they never should be) There's also a mini readme / "how-to-use" that pops up when u start the program. Just reading that, and the tooltips where applicable, should be enough to run my program and test it successfully.

	NOTE: sometimes, due to changes made to some .java files, the reading of saves (.bin files that had objects written to it) won't work. Unfortunately, that means you need to 'Start Over'. This should be avoidable by just not making changes to .java files. You will need to spend time loading courses into the transcript and plannedList. This does error checking regarding prerequisites etc. See error prompts and code comments and tooltips for more details (NOTE: I check grade and status where possible as well). Also, unfortunately I have not provided a fully completed courseList.csv file to load the program (the degreeList.csv program is good though), so you will most likely need to update it for full thorough testing. Using the one I've provided should be good enough, but some prerequisites for courses are missing from that section or a course might be missing from the list. My program only works properly if every course listed as a prerequisite has a course in the CSV file. Also, the subject identifier for "HK" needs to be changed to "HKIN" because my program only takes subject of 3 or 4 character length. NOTE: in admin mode, you can add degrees and it saves. Then, it updates the combo box for selecting it. However, it changes from what it was originally (that's ok, both are accounted). The main issue is that selecting degrees that have been added won't work because that means I would need a Degree concrete class specifically for that. Also note, that in admin mode, all added degrees are stored as an ALLCAPS version, but removing checks input without ignoring case. NOTE: I think I changed it all, but sometimes I may say PlanOfStudy instead of Student (because I refactored it) NOTE: Student contains a PlanOfStudy which has transcript and plannedList, which are both ArrayList of object type Attempt NOTE: Plan of study refers to the Student's entire plan for attending university: courses taken and planned Transcript are all the taken and in progress courses Planned are all the courses student plans to take but not taken yet NOTE: updating uses as many values as applicable from input to update more than just the grade. You can only update things in the transcript. So to simulate updating plannedList, you have to remove from plannedList, then add to transcript NOTE: POS = Plan of study

Learning Outcomes	**3 examples from your code. File name, line number**
refactor and restructure class design for improved encapsulation, modularity, cohesion and coupling	Student now has a POS instead of other way around Some things are now immutable (in univ package). Also, Course now has protected mutator methods so user can't change it (lines 100, 121, 142, 159, and 192)

	Degree now has altered numberOfCreditsRemaining, remainingRequiredCourses, and meetsRequirements methods. (eg. BCG.java line 81, Degree.java line 115, BCG.java line 132)
demonstrate use of inheritance through super/sub classes as well as through the use of interfaces	My *.java files implement Serializable so they can write an object to a .bin file Planner.java line 1704 SaveButtonListener inner class implements ActionListener BCG is a subclass of GeneralDegree (line 15 "extends" shows inheritance) GeneralDegree is a subclass of Degree (line 14 "extends" shows inheritance)
demonstrate clear understanding event driven programming through well designed listeners and gui components	GUI builder was used. I have lots of nice little extra stuff like hotkeys see code to find them (mainly numbers 1 through 7, Q, W, E, R, S, D, F, G, X, I, O, K, L). These have been mapped in a similar grid to the button layout. Some have a logical letter associated with it. Use Alt+KEY for buttons and Ctrl+KEY for menu item options. Ctrl+Alt+X for exiting.
demonstrate service-based error handling through a rich set of exception classes that communicate specific errors to client classes	All button listeners first check input if applicable Eg. Planner.java lines 691, 785, and 943 (may need to follow to appropriate method call as well)
create a repeatable testing suite and justify the choice of test cases	Did not do this, see images for testing evidence. I tested everything as possible as seen by my extensive error-checking (do not have evidence for everything). I also used the moodle VPL junit tests.
design and create a graphical user interface that is learnable and usable	Planner.java and Admin.java (no line numbers or other things applicable) make up the GUI
use inner classes, anonymous classes, and/or lambdas effectively	Planner.java on lines 54 to 69 and AdminMode.java 40 to 47 The above have custom listeners using lambdas and/or inner/anon classes

Required elements	**Examples from your code (File name, line number)**
Exceptions and try/catch loops	See bottom of Planner.java for try-catch example (for file stuff) See lines 1747 to 1770 for custom methods that parse a number and try-catch the exception thrown if it didn't work (not a number) Student.java mutator methods on lines 74 and 95 throw NullPointerExceptions that are caught if necessary. (eg. See Planner.java lines 727 to 734 for catching this)
Error prevention/handling (might also be try/catch or might be input checking)	Literally look anywhere in my code for error-checking and you'll probs find something.

	See bottom of Planner.java for a try-catch example (file stuff) See line 691 method for Login action that checks first and last name to be good input, and checks ID input to be a non-negative number
Two different layout managers	I used a GUI builder, so could not access the code directly to change layout managers, thus almost everything uses its default GroupLayout (eg. Planner.java line 203) I added custom stuff in to have an additional GradLayout. See lines 49 to 69 in Planner.java for setting a panel's layout to GridLayout and adding buttons to it (and adding lambda-style listeners to those buttons)
Separate window/panel for administration	AdminMode.java It pops up as a new frame and hides the main frame while running. Later, it sets the original main frame visible again while disposing itself on exit
Listeners	Planner has listeners everywhere for all the buttons and menu items, etc My custom lambda listeners are in Planner.java on lines 54 to 69 and AdminMode.java 40 to 47
Course class refactored and immutable	Course.java (now located in univ package) It now has protected setters so it can't be changed by the user, thus immutable The now protected mutator methods are on lines: 100, 121, 142, 159, and 192
Attempt class created	Attempt.java (located in planner package)
Classes in package	Package planner: AdminMode.java (the secondary frame as separate window for popups) Attempt.java PlanOfStudy.java Planner.java (the main frame) Student.java Package univ: BCG.java CS.java Course.java CourseCatalog.java Degree.java GeneralDegree.java HonoursDegree.java SEng.java

Refactor Plan of Study (eliminating it)	I have not removed PlanOfStudy. Judi said it was not necessary to do. Instead I have refactored the class , reversing the original connection, so that Student has a PlanOfStudy, which contains the required lists of Attempts for transcript and plannedList, instead of the other way around (makes more sense that a Student has a plan, compared to plan having a student) See: Line 22: attributes are different from A1 File read and write methods are not in PlanOfStudy anymore. It Now has ArrayList<Attempt> for transcript and plannedList and other corresponding behaviour methods Other smaller changes like toString() (Line 496) have also been made
Database usage	Since we were not really taught how to use the database (SQL stuff etc.) I had trouble and did not have the time to worry about figuring it out. Thus, my program runs using files: courseList.csv degreeList.csv LastnameFirstnameID.bin (<- for saving student plans of study) Bottom of Planner.java has file read and writes (Line 1778 to 1900) CourseCatalog.java line 95: initializeCatalog to read courseList.csv Degree.java line 146: readRequiredCourses to read degreeList.java
Javadoc comments (the most complete examples)	Attempt.java && Student.java (most complete / best two)
Evidence of testing	I can provide screen shots and paper brainstorming notes if desired. For now, I'll just include a few images.

The only change I would make, would be to add error checking that ensures courses the user is trying to add are offered during the specified semester. For example, if a course (eg. TEST*1000) is only offered in the Fall semester ("F") and the user tries to add it to their "W18" semester (either Transcript or PlannedCourseList), it won't work because it isn't offered during winter semesters.