Criterion C: Development

# Structure

The entirety of the project’s data is stored in ArrayLists and Vectors. I used Vectors for data that would directly interact with lists, as they accept vectors as valid input for their model. This made it easy for any data stored in a Vector collection to be displayed to the user interface. For all other data, I used ArrayLists. I chose this data collection type because they had flexible sizes, allowing the user to add as many users and assignments as needed. In addition, the modernity of ArrayLists makes them less likely to have problems with serialization, which was my file i/o method of choice.

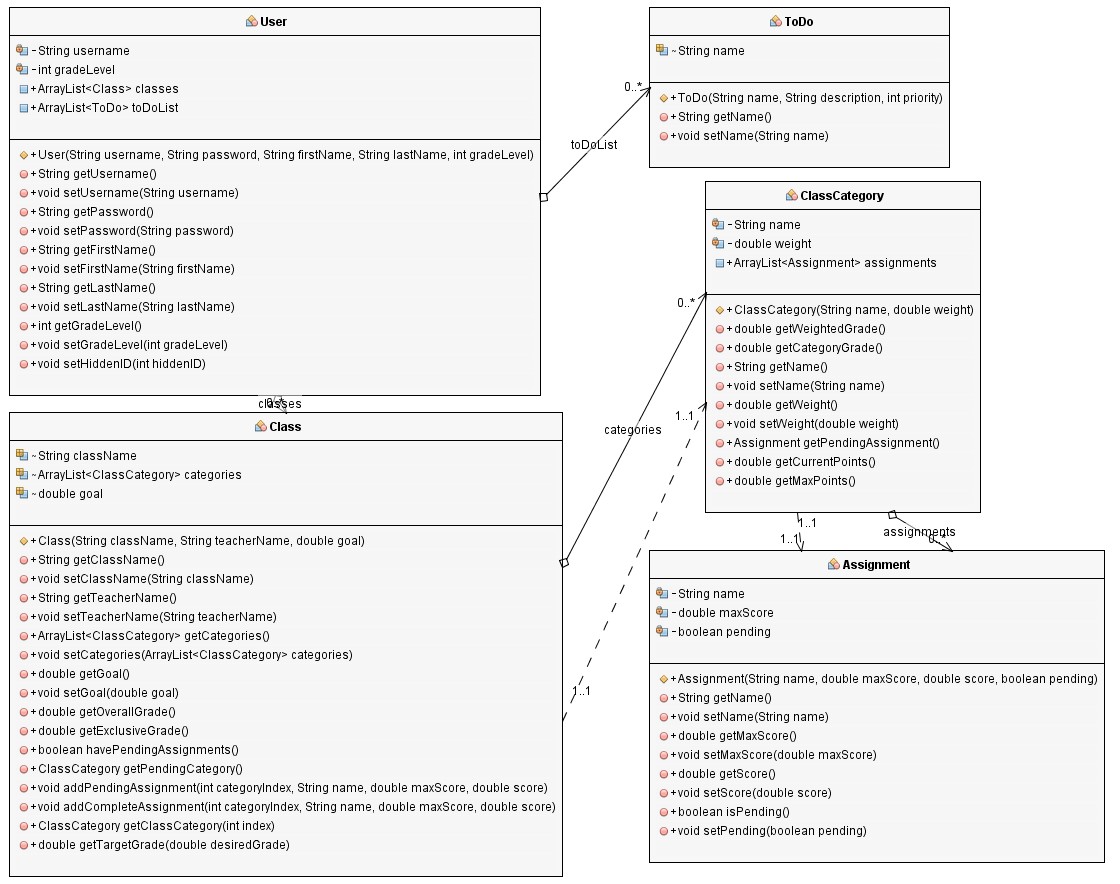
# Techniques Used

In my project, I used the following techniques. The list is not comprehensive of every programming technique I utilized, but rather the ones of most importance.

* Class decomposition
* File i/o using serialization
* Method decomposition
* Linear search
* Complex selection
* Nested loops

# Class Decomposition

Seeing that there was the possibility of multiple users of this product, classes to track for each user, and assignments to track for each class, class decomposition was greatly beneficial in simplifying the problem.



Above is a UML diagram showing the basic classes that will be needed in this solution. The user class will hold all of the user information, including a list of their classes and to-do list. The to-do list is made up of to-do objects. The user class stores their classes in an ArrayList called classes, which keeps track of the class name and goal, and has methods pertaining to various features such as calculating the overall grade of a class. Each class has a list of class categories, which have a name and a weight value, which determines how much weight each assignment in the category receives when calculating the overall grade. Finally, each category is made up of individual assignments, which are what make up a user’s overall score in a class.

# File i/o using serialization

To help simplify the file i/o, I decided to use serialization. This was due to the fact that nearly all of my data was stored within classes and would be clunky to store and retrieve this information from file. With serialization, I was able to cleanly save and load user objects, which held all the internal classes of each user much more easily.



# Method Decomposition

To help further break down the solution’s process, I used method decomposition. Within each class, there are many methods to help complete various processes. The best example of this is in the Class class, where multiple methods were used in order to calculate the target grade that the user would need on an assignment in order to achieve their goal. This is shown below, where the method getTargetGrade() utilized many other methods within the class in order to break down the process and make it more easily understood.

# Linear Search

The login system utilized a linear search to help obtain the correct account attempting to log in. It used a loop that went through all of the users from an array which was extracted through file i/o in order to save user objects in between sessions.

# Complex Selection

Complex selection was used to determine the look of the dashboard based on an instance variable called dashboardState. This would change the appearance of the dashboard when the user selected different tabs. There are four different tabs: Home, Grades, Calendar/To-Do, and Settings.



# Nested Loops

Besides searches for the sake of the user, the program also runs searches in order to function itself. An example of this is when the Class class needs to find the category or assignment that is currently pending. To accomplish this, I used nested loops to quickly go through every assignment that the user object had under it.

