## Criterion E: Product development

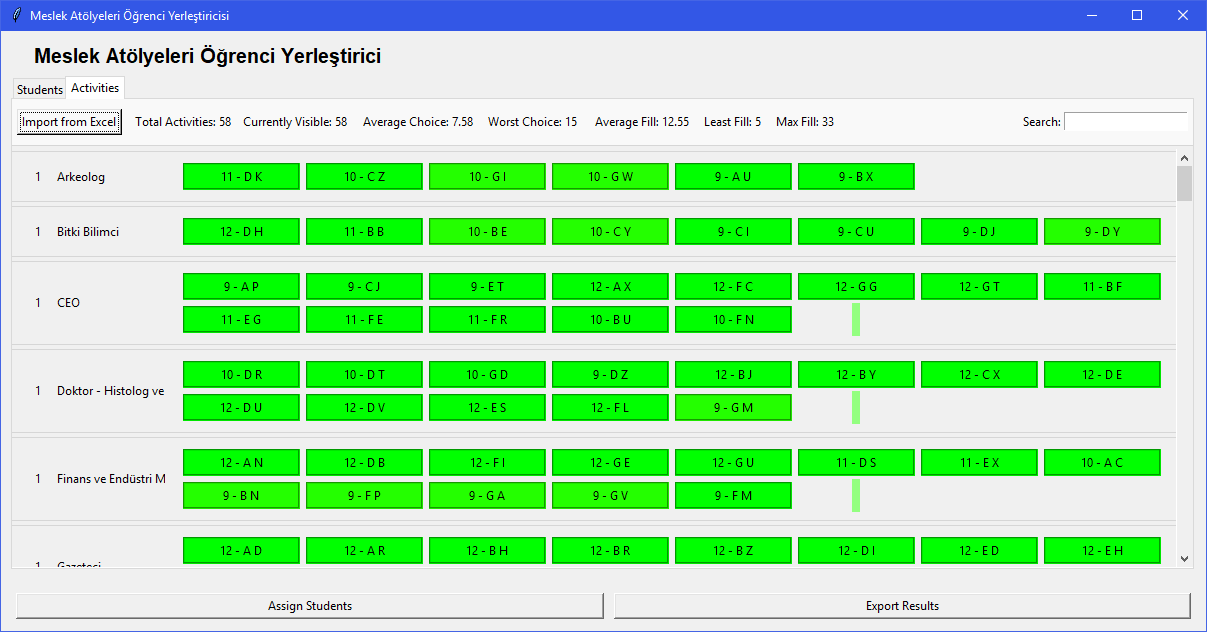
### Complex Techniques Used

* Multiple different classes
* Arrays
* File Handling
* Functions with parameters and returns
* for, foreach, while loops, with and without exit conditions
* If-else
* UI

### Final Interface

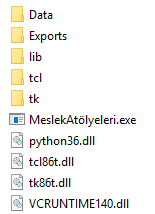
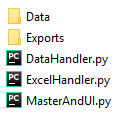


Students screen



Activities screen

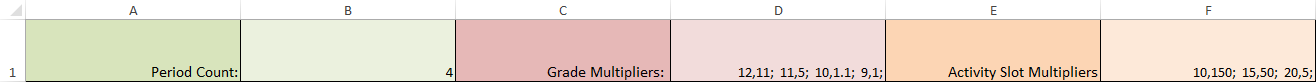
### Folder Structures



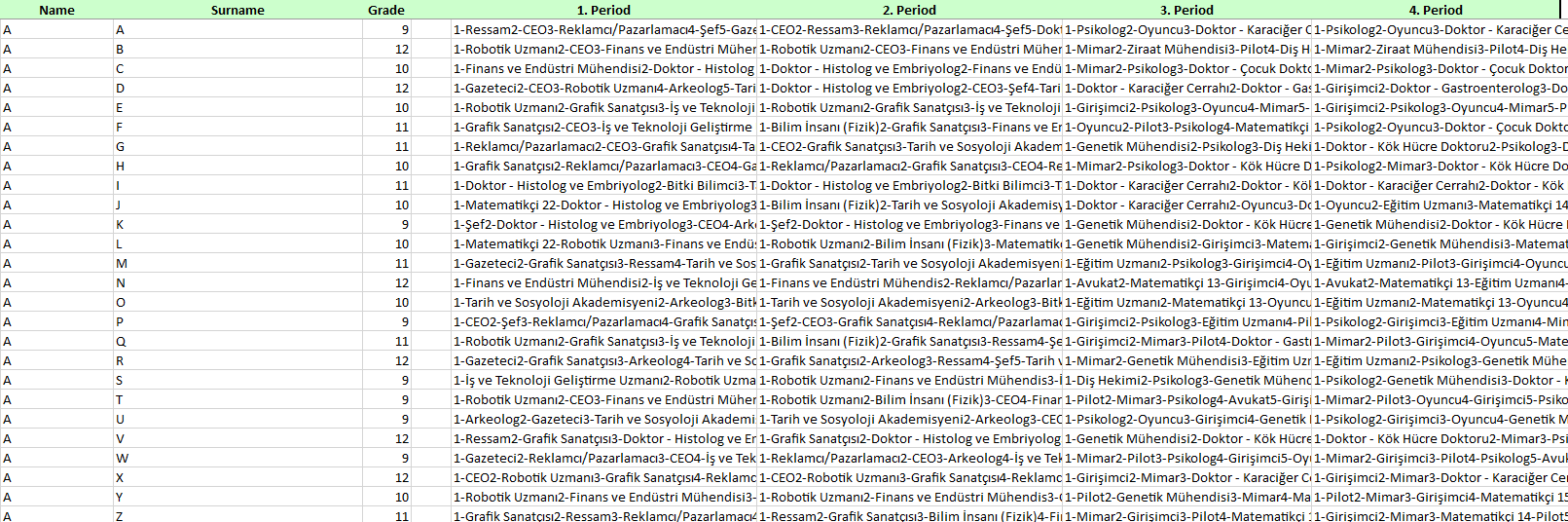
The folder structures for the source files (left), the final exe program (middle), the data folder (right).

The settings file is loaded at startup and needs to be edited by hand to change algorithm settings.

### Settings File



### Import Excel Structure (Generated by Jotfrom.com)

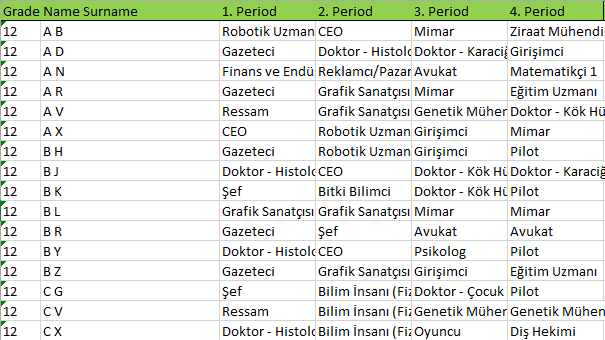


This is an example input file. There are student names and students’ activity choices for each period. The picture is the Example Students.xlsx file which includes the student choices collected by my client during the event. The student names are changed for privacy, and the seminar names are in their original Turkish form.

Period data is in the following format:

[choice index]-[seminar name][choice index]-[seminar name]…

### Export Excel Structure

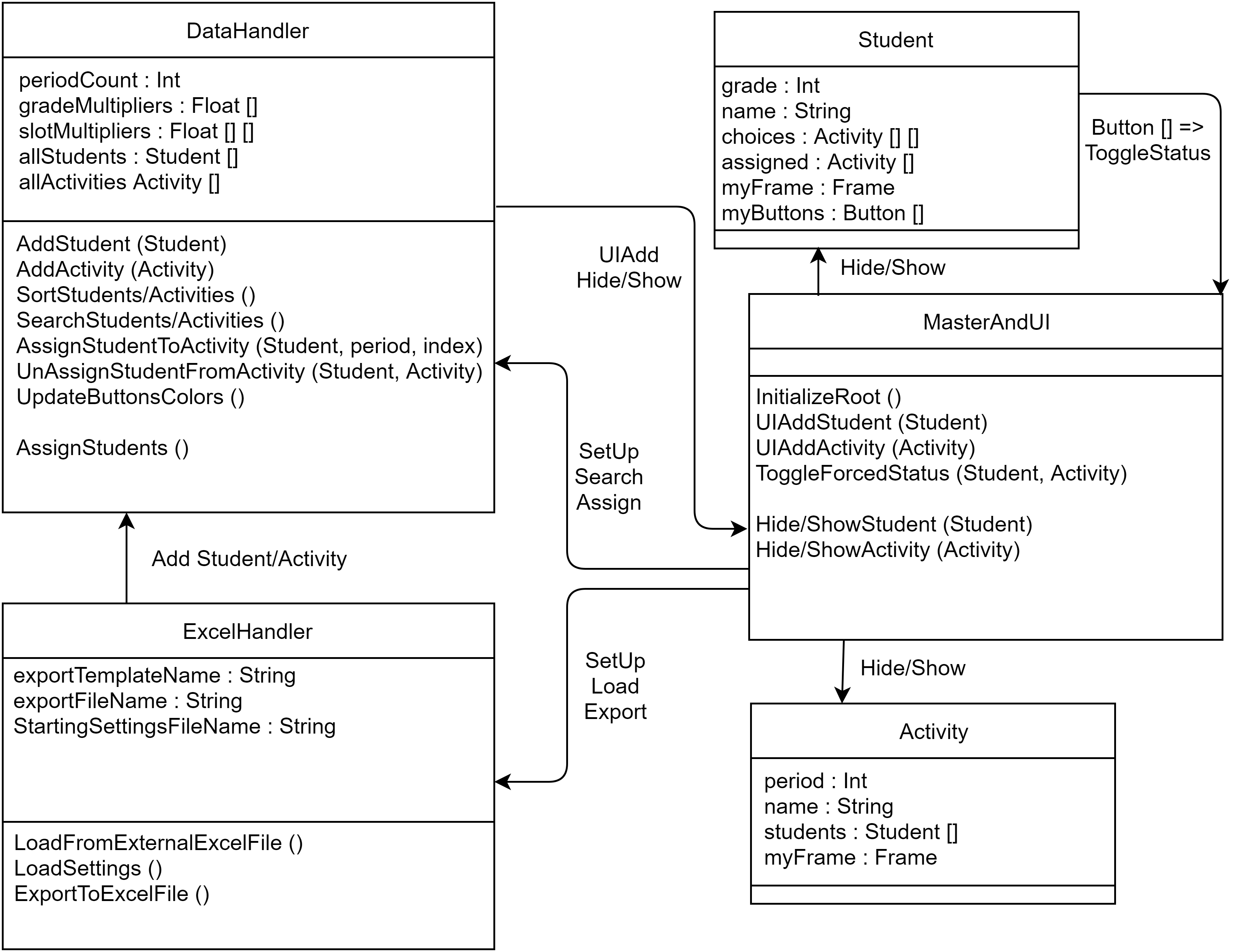


The file exported by the program.

Please note that originals of both of these files are in Turkish, and so the files exported by the program will have Turkish headers.

### UML Diagram – Overview of the classes

Multiple classes/Objects

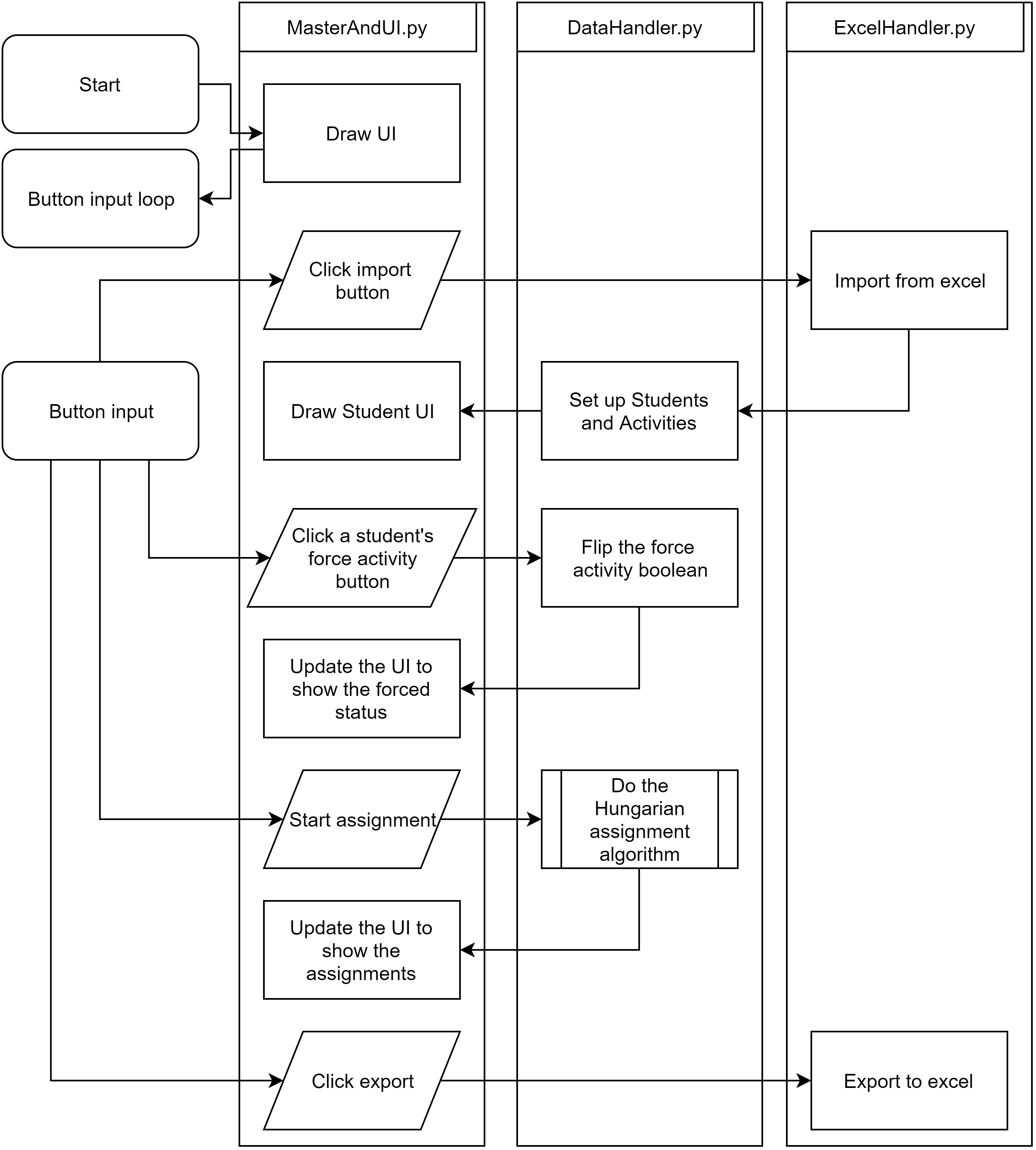


DataHandler.py: Stores the student and activity information and deals with operations for them. The main assignment algorithm is also in this file. The storage classes Student and Activity is also in this file.

MasterAndUI.py: Control and draws the UI. Initializes the other classes & passes the required references to those classes.

ExcelHandler.py: Deals with excel reading & writing

### Flowchart



### Explanation of the code

#### DrawUI:

A section from the MasterAndUI.py:

I used Tkinter to draw my UI. Tkinter “packs” elements into “frames” to organize the UI. I also use some of the premade UI elements like a “Notebook” for the tabs in my application. There are also some StringVars used here, which are updateable texts in the UI. I also used “VerticalScrolledFrame” which is a collection of other Tkinter elements made into a nice package that I took from <http://tkinter.unpythonic.net/wiki/VerticalScrolledFrame>.

Function with return

UI

#### Import Students

After drawing the UI the program waits for the user to import students.

This section from ExcelHandler.py imports students from the excel file. It creates Student objects in the DataHandler file and adds the student object to the list with the AddStudent method.

File I/O

Arrays

While loop

For loop

This bit of code in DataHandler.py adds the students, updates the UI and if the student have a never seen before activity, adds that to the activity list. AddActivity method does the same adding and updating the UI.

Foreach loop

Break condition

UIAddStudent in MasterAndUI.py just adds the specific UI elements and assigns the various variables.

Student and Activity classes in the DataHandler.py. These classes mostly hold data.

Multiple classes/Objects

Arrays

#### Force an Activity:



Clicking any of the activities selected by the student will force assign that activity to that student. Assigned activities are in bold and force assigned activities are in italic.

From the MasterAndUI.py:



Function with parameters

Assigning students to the activities is also simple:



If-else

#### Assign Students

Student assignment algorithm is the most complicated part of this application. I followed the matrix interpretation of the Hungarian algorithm from this Wikipedia article: <https://www.wikiwand.com/en/Hungarian_algorithm>. Unfortunately the article’s explanation had a few errors and I had to fix it but the algorithm worked in the end.

The Hungarian algorithm is an algorithm for a situation like this: if you have set of “jobs” and a set of “workers” that each will do each of the “jobs” for a different “cost”, the algorithm will match the jobs with the workers so that the cost is as small as possible. For my case, I need to match students with activities, trying to make sure everyone gets their best possible choice. So for that to work with the algorithm I give better choices better “value”. Also 12th graders choices get multiplied. To fill a minimum amount of slots while having flexibility to have more students in an activity I make the first slots in a activity more valuable, and the rest less valuable, so the minimum number of slots should be matched by the algorithm. All these “values” get flipped in the end because the algorithm works to find the least, not the maximum.

The algorithm generates a 2D array as the “cost matrix”



The cost function:

After this the more complicated Hungarian algorithm parts come in. The full explanation the Hungarian algorithm is not in the scope of this IA, but I will put some code snippets and will explain some of the code sections. Please check the source code while looking the Wikipedia article if you want a deeper understanding.

Some of the preparation steps:





The most complicated part:

This is a recursive method set that does one part of the 3 step process in the Wikipedia article. Each method calls the other one until there are no possible step points left.

After this process there are numerous more steps similar to these ones after. In the end all of the students are assigned to activities and the UI is updated:



I also wrote a debug console matrix drawer to fix the errors within the algorithm. You can only see this if you use the python version:

This bit of code draws the cost matrix into console. (it is disabled with a return statement at top as default to make the algorithm run faster)

#### Exporting to Excel

Export part is also simply writing the list of students & activities into an excel file (handled by the ExcelHandler.py):



A full source code is in the product folder.

**Word count**: 906