CuTAES Design Doc- Assignment 2
Zack Dawson 100851846

What must be saved between sessions:

Student information and application information must be saved between sessions. This includes a student's general info, an application's status and data (rel. courses, ta'd courses, work exp.), and TA evaluations (saved in application file). Additionally, reports & summaries will be saved to file.

File organization

CuTAES has a main 'data' folder, with two nested folders 'student' and 'application'. Saved reports & summaries will be saved in 'data'.

Students will be saved in 'student', and will be named <stuNum>.txt. It's safe and intuitive to name the file with the student number, since students have to be unique; this design reflects that.

Similarly, TA Applications will be saved in 'application', and will be named app_<course>_<timestamp>.txt. This will ensure applications are never overwritten.

Storing each unique student and application in their own file makes the program very scalable. Trying to fit them all in one file would be very messy, and adding, editing, and removing data would be difficult and error-prone. Splitting them up keeps it clean and maintainable. Plus, they can be easily read & modified by the user in a text editor, if they want to do that for some reason (Pretty useful for generating a lot of data for debugging).

Since there is no good cross-platform way to read every file in a directory, two master files will also be saved; one for students and one for applications. They will contain a list of filenames, which will then be read. These files will be fairly easy to maintain; new filenames will simply be appended to it.

Since you can't change a student's student number (would be equivalent to making a new student) and you can't change what course an application is for, these filenames will never change. So, the list of files can grow easily, without having to change any of the filenames. The only downside will be having to find and remove filenames from the list when data is deleted, but this won't be too bad in terms of implementation & performance.

Folder hierarchy example:

```
-data
    -application
        -app_COMP1405_1360889734.txt
        -app_COMP1405_1360890012.txt
        -app_<course>_<timestamp>.txt
    -student
        -100851846.txt
        -<stuNum>.txt
```

-applications.txt
-students.txt
-Summary_<course>.txt

It's unfortunate that there needs to be a master list of filenames, but I don't want to write platform dependent code and can't use libraries to wrap it for me. This solution will work, but does make removing students and applications a little more difficult, since they will need to be removed from the list. If I can figure out a way to eliminate the list entirely, I will do that.

Student data format:
<stuNum>.txt:
<firstName>
<lastName>
<stuNum>
<email>
<major>
<CGPA>
<major GPA>

Application data format:
app_<course>_<time>.txt
<course>
<stuNum>
<status>
related
<course>
<year>
<term>
<grade>
(repeat above 4 lines for each course user entered)
ta
<course>
<year>
<term>
<supervisor>
(repeat above 4 lines for each course user entered)
work
<job>
<desc.>
<start>
<end>
(repeat above 4 lines for each row user entered)

Master file list format
(for students, applications will look similar since it's just a list of filenames)

100851846.txt
100111111.txt
100888888.txt
<stuNum>.txt

In each file, I'm saving 1 piece of data per line to keep it user-readable and to make it easy to parse. There will never be a ton of information for a student or application, so file size isn't an issue.

Indentation in the application format is mostly for aesthetic purposes, but will probably be used to detect the end of a list of data (eg. parsing related courses, since there can be any number of courses entered).

Internal data storage and saving:

I created a Database class. This is a singleton which stores all of the students and applications, and loads them on startup. It provides methods to get students by id, get applications by course, save students and applications, etc.

The database class hides all of the backend work of maintaining the file structure, saving, reading, etc. This keeps the rest of the code clean, and allows for easy modifications of the file saving design in the future. (For example, switching it to use a relational database or changing the file format would be simple- only the Database class would need to change.)