Persistent Saving

Infinite Tears

Our method of persistent saving was chosen to be just a single file with each application taking up one line. Each different object was parsed by having a '%' placed between them, and within the objects themselves there is a second paring item '$' to separate the different parts of the object.

We chose this method for now because it was the easiest and fastest for our purposes. We had also considered saving each course individually with all of the applications for that specific course but decided the organization would take too long to work through. Another option was saving each student with some identifier numbers, and having related courses, ta courses, and work experience all in different files, each individual element having a unique identifier that would then correlate to a student's needs. This also seemed too extensive and extremely unorganized as everyone's related courses would be stored in the same file to be pulled out later and assigned to a student.

In the future we plan on implementing an improved method of either saving where we would save each application individually with the student number, course, and either the work experience, related course, and ta courses just loaded into the text, or use the idea of identifiers to call specific work experience, related courses, and ta courses from the student objects. Students would be saved and categorized by their student number, containing a list of all their application numbers, and in the case of the students holding their work experience, related courses, and ta courses queues they would have each individual element of the queue have a unique identifier for the loader to recognize and use.

Sadly due to time constraints and other issues popping up we were only able to implement a very basic and inefficient saving method, but with our ideas, we should be able to fix that in a future release, opting for a much more organized and efficient method of saving, keeping in mind our need for loading.

Our future method will be faster and easier to read outside of the program, and also would lead to not having to load everything all at once, as we would be able to just load the applications to display their status when an admin wants to access the system, or just load one student and all their applications when a student wants to log in, so that more time is saved not loading pointless applications or unneeded students. We believe that this will lead to using less memory while running, a faster start-up time, and overall a better experience.

In conclusion, we realize our method is not complete as of now, but we hope that in the coming weeks we will be able to implement improved designs.

Colin Kealty 100855810

Yugo Brunet 100853306