**Project Manager: Ian Chiu**

**Group Members: Ian Chiu, Darren Chou, and Maxwell Wang**

**Mr. Peck**

**First Period**

**Case Study and Lab: Lynder Group Maker**

In this project, we consider a more useful classroom tool for teachers and students. Inspired by the popular dating app *Tinder,* **Lynder** is a program that will collect the respective ratings (from 1 to 5) that each students gives each other in terms of preference (1 being the lowest, and 5 being the highest), and analyze each rating to produce a group that will generate the ideal grouping for everyone in the group.

In this project, we will be creating a profile for each student that logs into the Lynder app. The profile will keep track of the student name, student GPA, as well as contain a map with every other student’s name and GPA.This map, called the studentMap, will be updated continuously as every student is logged in.

After logging in, the system will wait until all entries are entered, and will then bring up a secondary GUI for the user to use. This second GUI will display each student’s name in the map, display their respective GPAs, and provide a spot for the user to rank them from 1-5. Once the user has finished ranking every student in their class, they will click “finish.” Once the “finish” button has been clicked, the GUI will close, and come up again with the next student’s name, allowing the next student to undergo the same process the first student had done in ranking every student.

Once everyone has ranked everyone, an algorithm will determine the perfect grouping for every student based on how they have ranked each other. **This algorithm will work with two specific steps:**

First, for each student, the algorithm will calculate the total amount of points that their classmates have ranked them, and store it in a priority queue. In the priority queue, students with more points will be paired/grouped before students with lesser points. The use of this priority queue is because the higher the amount of points a student has, the more popular they are among their classmates, meaning that it will be easier and faster to pair them up. As the algorithm moves towards the students with lesser point totals, which means that more people rated those students more negatively than positively, it will be become more difficult for the algorithm to find the ideal pairing for those students.

Next, starting from the students with higher point totals, the algorithm will systematically go through each of that student’s rankings of other students, and collect how those other students, using the *.getRankingOf (String studentName)* method, ranked that particular student. It will calculate the average between that student’s ranking of the other student and the other student’s ranking of the student. After going through all of the student’s ranking, it will select the pairing with the highest average, and take those two names off of the map, reserving them as a pair. This algorithm will store this new pairing in an array.

Once all the pairings are complete, the algorithm will pass on the pairings to a third GUI, which will print out all of the pairings in a list for everyone to see. The third GUI will also provide the opportunity for the user to click “START OVER,” in case for any unappealing groupings, as there will be several cases where there are multiple high rankings of each other, in which case the algorithm will randomly select one to pair up.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Our design process for SafeTrade consists of four parts. The first part is structural design, which determines which data structures will be used in the program. The second part is object-oriented design, which determines the types of objects to be defined and the classes and interfaces to be written. The third part is detailed design, which determines the fields, constructors, and methods in all the classes. The fourth part is developing a test plan. We are going to discuss the structural design first, then the classes involved, and after that the detailed design and testing.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Our structural design decisions are summarized in Table 1:

|  |  |
| --- | --- |
| **Data** | **Interface => Class** |
| Student GPA, Student Name, Short Description of each Student  Methods for Data “Crunching” : Calculate averages of rankings, return ideal pairing through use of priority queue and treeMap | Object => Student Class contains a map of all other students using name as the key |
| Registered Students | Map ==> TreeMap<Score, Student> *score is the total amount of points per student* |
| Determine order in which students will be paired | Queue ==> Priority Queue *sorted in terms of total amount of points from TreeMap* |
| Collect information to store into the Student Class | GUI: UserWindow\_logIn |
| Give student opportunity to rank other students | GUI: UserWindow\_ranking |
| Return pairings in an easy-to-use user interface | GUI: UserWindow\_result |

We have chosen to create a Student object class that will utilize several data structures to create a ranking system and several data holding structures that will be able to contain the data that comes out of each step of the algorithm. We have chosen a TreeMap to hold the registered students because it will be much easier to place the students into the priority queue when they have already been sorted by

total points. Also, we can’t risk to waste any space or time, because the algorithm will be an O(n^2) algorithm, which will already take much of the time in the creating a group.

The priority queue will be comparing the total amount of scores, and placing it into a priority queue. There will be no need for a new comparator class in the implementation of this priority queue, because we will be comparing integers, which already has a comparator class.

Everything will be stored in the *Classroom* class. The Classroom class will hold an array of all the students, as well as contain the sorting algorithms.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. Object-oriented design:

Figure 1 shows a class diagram for Lynder. The project involves eight classes. 

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Testing*:

Proper testing for an application of this size is in many ways more challenging than writing the code. Entering and executing a couple of orders won’t do. The QA specialist has to make a list of possible scenarios and to develop a strategy for testing them methodically. In addition, the tester must make a list of features to be tested.

Several of the most important aspects that must be tested are the following:

1. Ensure that the data exchange between the GUI and the Classroom class happens as expected -- the GUI should transfer Student Name, Student GPA to the Student class, while the second GUI should get the Map from the Student Class, and the third GUI should get the result Array from the Classroom class.
2. Ensure that the ranking algorithm works as expected: it should be finding the average of each rating, and the highest average should be saved into an array
3. The GUI works as expected -- each ActionListener should be working

There will be other aspects that will need to be tested that will be further specified as exact details of implementation are confirmed as we continue work on creating each class.