Team name: KPMGComeGetUs

Name: Sng Hong Yao **Student ID:** 17205050
Name: Ankish Raj Prajapati **Student ID:** 17202456
Name: Ronan Mascarenhas **Student ID:** 17379773

Setting up Environment

1. Open a terminal and navigate to the file where run.sh is.
2. Execute `./run.sh` (Uses maven to build the JAR, then Java to run the JAR)

Meeting Demands

Data Containers

- Data containers were already written during Sprint 0, during this sprint, we need only reorganize it to be more clear structured, include their interfaces, and enforce encapsulation.

Test Cases (written via JUnit)

- These tests will run during `./run.sh` or `mvn test`
- CSVReader
  - Tested project/student csv against expected csv format.
  - Tested project/student csv against illegal separators. (i.e. ';' instead of ',')
  - Tested project/student csv against missing fields.
  - Tested project/student csv against wrong data types. (i.e. expected Integer, given String)
  - Tested project/student csv against wrong csv layout (i.e.).
- CSVWriter
  - Tested if written project/student csv is able to be read back into the system.
- Project / StaffMember / Student
  - Tested if initialization and other methods work as expected.

Reading Input Files to Store Data

- To do this we have the **CSVFileReader** function which has two constructors. We invoke the default constructor for now and have reserved the overloaded constructor for more functionality in the next sprint.
- We have used **OpenCSV** to read our input files, which is a robust open source file parser for Java.
- We use the **CSVReader** object to read our input files line by line. Each line/row is split up into strings based on the delimiter, thus each field becomes a String in the String array in the same order it was in the file.
- To read the projects, the **readProjects(filename)** function takes the name of the file to be read and returns an Array List of Project type. Each row is read and the fields are parsed into appropriate types (e.g. "preferred probability" parsed to Double).
- To read the students, the **readStudents(filename, projects)** requires the filename as well as Array List (Project type) of all the projects the students have to pick from. This is because the input file for the students only has the research activity and to create Student objects, we need an Array List of their preferred projects. So we go through each preference listed by the student and match it against the researchActivity of each project

in projects (which is given as an argument). Any project whose researchActivity matches is added to a temporary Array List of projects which contains all the current students project preferences. We use this along with the other details of the student which are stored in the first 4 fields of the row, parse appropriately, and create Student objects.

- To read staff members, the **readStaffMembers(filename)** takes a filename to read and returns an Array List of StaffMember objects. We read the name and stream as they are in the line and create Strings for researchActivities and researchAreas, which, to create the StaffMember objects, may contain multiple comma separated values for either. The constructor of StaffMembers handles the formatting for us when we create the objects.

Writing CSV files

- We use the **CSVFileWriter** class to write the files using Java native I/O library **FileReader** to read the file and **File** class to create a new file.
- All methods in this class follow a similar format. We build a string (line) to write to the file which contains all the fields required in that row, delimited by commas and ending in a new line character. We loop through each object, gather the required information, build the String, write it and so on. Exceptions are handled and the naming convention from the previous sprint is followed.
- Output files are printed with semicolons used as in-field delimiters. Functions follow a very similar structure to **ExcelWriter** class.