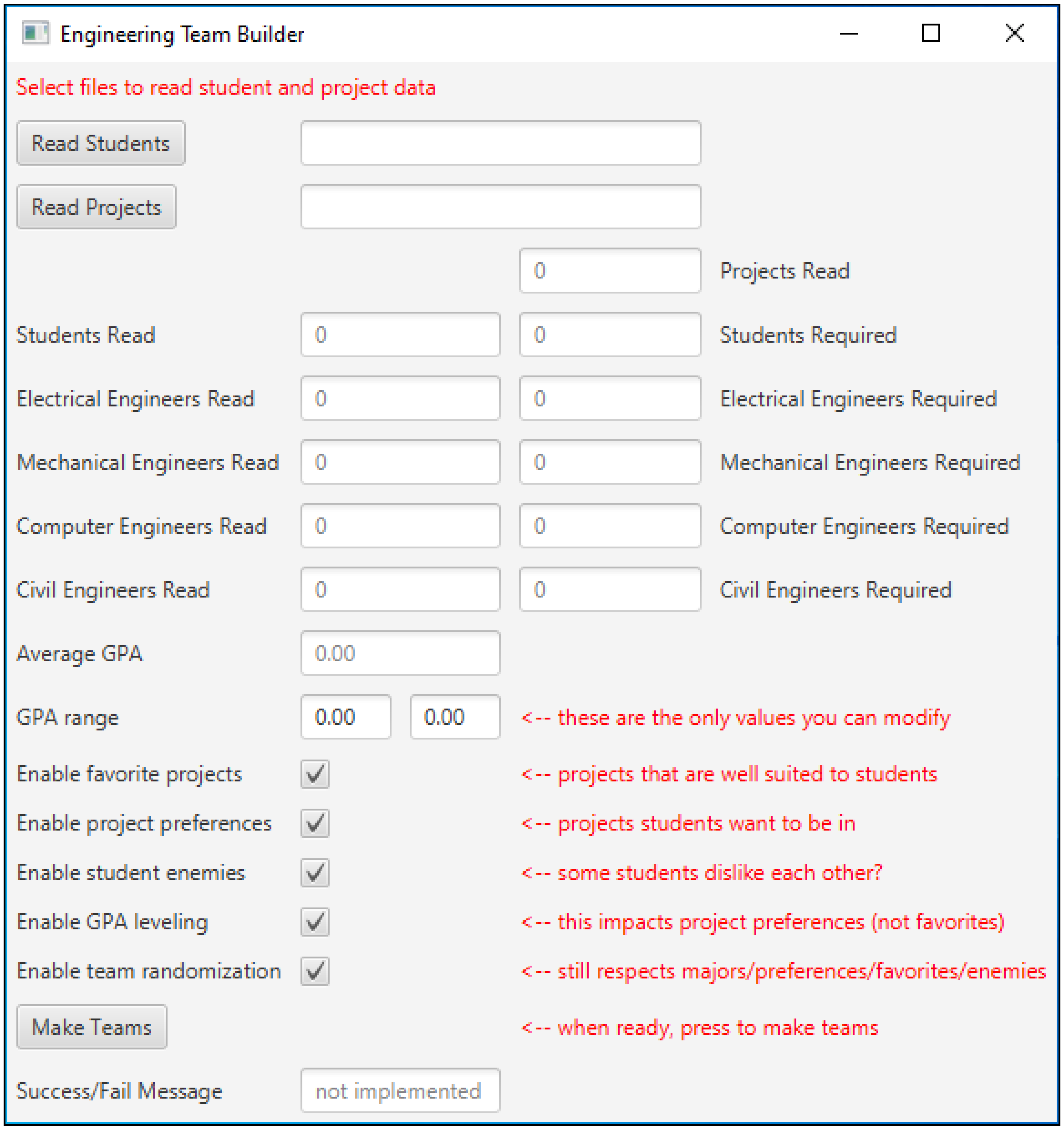
SENIOR DESIGN PROJECT TEAM BUILDER

https://github.com/chrisamarkey/tp_2_snrengrdesignteam.git

SHA-1: 260a83505b254fc99be4429292e39715a59fa4da

Andy Ludewig, Craig Gabel, Chris Markey

1. How to use the program!
    a. Execute code from Eclipse or JAR file (JAR file found in the “program” directory) Hopefully you see the GUI (the red text is there to help you, but I will go into a little more detail here):



Engineering Team Builder

Select files to read student and project data

		0	Projects Read
Students Read	0	0	Students Required
Electrical Engineers Read	0	0	Electrical Engineers Required
Mechanical Engineers Read	0	0	Mechanical Engineers Required
Computer Engineers Read	0	0	Computer Engineers Required
Civil Engineers Read	0	0	Civil Engineers Required
Average GPA	0.00		
GPA range	0.00	0.00	<-- these are the only values you can modify
Enable favorite projects	✓		<-- projects that are well suited to students
Enable project preferences	✓		<-- projects students want to be in
Enable student enemies	✓		<-- some students dislike each other?
Enable GPA leveling	✓		<-- this impacts project preferences (not favorites)
Enable team randomization	✓		<-- still respects majors/preferences/favorites/enemies
Make Teams			<-- when ready, press to make teams
Success/Fail Message	not implemented		

    b. Press the “Read Students” button to read a VALID students file. The provided “Student Input File.xls” should work and is the expected format.
    c. Press the “Read Projects” button to read a VALID projects file. The provided “Project Input File.xls” should work and is the expected format.
    d. The next 7 rows of information **cannot** be directly modified by the GUI. These values are drawn from the input files you chose. Ideal scenario is that each number of “Read” is equal to it’s “Required” pair. Any differences **should not** prevent the teams from

being built. But some students might be left out (if there are too many students) or some projects might not be completely filled (if there are too few students)

   e. GPA range can be adjusted within the GUI. Its values are defaulted to +/- 0.2 from the student body's average GPA. There is no guarantee that teams will fall within this range. Please read code comments about the GPA leveling mechanic and its deficiencies.
   f. Checkboxes to enable/disable various preferences.
      i. Favorite projects: these are projects students are well suited towards, there is an associated weight in the students file. This has the highest priority in assigning students to projects and nothing can override it.
      ii. Project preferences: when students list projects they want to do. This is the 2nd highest priority sorting decision.
      iii. Student enemies: if a student has an enemy, or the student is the enemy of another student, those students will never be placed on the same team. This has the least testing of all sorting mechanisms (but has worked so far). It appears we only support 1 enemy per student (that is all that has been tested so far). ExcelIO.readNextRow() would need updating to support multiple enemies. The input file might also need some change. The rest of the code should already support multiple enemies per student.
      iv. GPA leveling: this attempts to bring up the teams with low GPAs by trading their lowest GPA student with a high team's highest GPA student. The algorithm is described in detail inside TeamBuilder.java (line 167ish). This will probably break some student project preferences (not major, enemies, or favorites) by trading a student off a project they wanted to be on. A fancier algorithm could certainly prevent that. This currently does not care about high GPA teams, it only tries to fix low GPA teams by stealing high GPA students from high GPA teams. The algorithm will give up eventually, so there is no guarantee that all teams will fall within the GPA window (its like 50% of the time right now, but very few teams squeak by with a low GPA…1 or 2 tops).
      v. Team randomization: it respects all other rules. It just randomizes the rest of the students. Turn this off, and the output will always be the same.
   g. Press "Make Teams" when you are ready to go. Save as a .xlsx please (.xls works, but you will be warned about the file type when you try to open it).
   h. The Success/Fail message was never implemented and will always show that
   i. There are many ways that this code can fail to produce good results. We have only written code and tested to ideal circumstances. The files we've provided should produce good output.
2. How to get this working in Eclipse
   a. Install JavaFX
      i. Help->Install New Software
      ii. Add
         1. Name = Oxygen (or whatever you want to call it)
         2. Location = http://download.eclipse.org/releases/oxygen
      iii. Select that repository from the "Work with" dropdown menu

    iv. General Purpose Tools
    v. Select and install: e(fx)lipse
        1. I'm working from memory here, it might be called something else. But its name is pretty close to this

b. FIX YOUR BUILD PATH
    i. The build path is currently set to my machine, and the paths are explicit. Unless you happen to have the same directory structure as me, you must change your build path.
    ii. Right click on the project->Build Path->Configure Build Path
    iii. delete everything with an explicit path (be sure not to remove JRE system library and Junit 5)
    iv. Add External JARs
    v. Select everything in the "All needed libraries" directory
    vi. Apply & close

c. Run/debug main.java

d. Run any test/*Test.java you'd like (unfortunately, we could not get a test suite to work). Note there are no test files for Main.java, or the GUI MainWindow.java

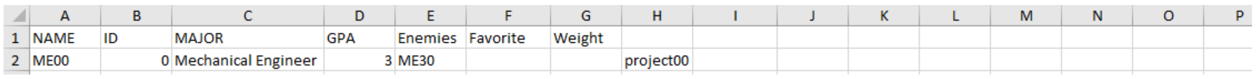
e. Let us know if there are problems, we will work with you to solve them

3. Excel File Instructions.
Input file instructions:

The program can interpret excel files with the .xls or .xlsx extensions.

The program currently requires that the input files be formatted as expected. For precise details, see the instructions below or the example input files included in the documentation.

Student Input File



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	NAME	ID	MAJOR	GPA	Enemies	Favorite	Weight									
2	ME00	0	Mechanical Engineer	3	ME30			project00								

Row 1 should contain the headers for each column. Rows 2 onward should each contain the data for an individual student.

Column A: This column must not be blank. Contains the name, first and last, of the student.

Column B: This column must not be blank. Contains the ID Number of the student.

Column C: This column must not be blank. Contains the Major of the student.

Column D: This column must not be blank. Contains the GPA of the student.

Column E: Contains the name, first and last, of the student's incompatible partner. This column may be left blank if the student does not have an incompatible partner.

NOTE: The program can currently only handle one incompatible partner per student.

Column F: Contains the name of the student's one favored project. This column may be left blank if the student does not have a favorite project.

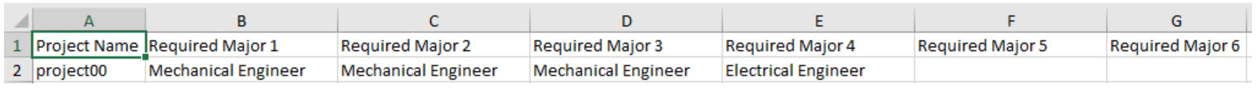
NOTE: The name must exactly match the name of the project as written in the project input file.

Column G: Contains the weight value, on a scale of 1-5, of the weight assigned by the user to a student's project preference, with higher numbers referring to a more valid reasoning for their preferred project. This column may be left blank if the student does not have a favorite project.

Column H onward: These columns contain the student's project preferences as displayed in the file resulting from the qualtrics survey. These values can be directly copied from that file into the input file. This file can handle as many projects as are needed, and allows an individual student to have as few or as many preferred projects as they like.

NOTE: The name in each cell must exactly match the name of the project as written in the project input file.

Project Input File



	A	B	C	D	E	F	G
1	Project Name	Required Major 1	Required Major 2	Required Major 3	Required Major 4	Required Major 5	Required Major 6
2	project00	Mechanical Engineer	Mechanical Engineer	Mechanical Engineer	Electrical Engineer		

Row 1 should contain the headers for each column. Rows 2 onwards should each contain the data for an individual project.

Column A: This column must not be blank. Contains the name of the project.

NOTE: This name must be matched at several places throughout the input files.

Column B onward: These columns contain the types of engineer required for the project. The program can handle as many or as few required students as the sponsor wants.

Output File:

The program will attempt to ouptut a modern Excel file, with the .xlsx extension. If the user specifically requests a .xls file, it will be readable, but Excel will throw an error when the results file is first opened.

UST Senior Project TeamBuilder Java & JavaFX Project Use Case

Use Case: Operator builds project teams
Scope: Team Builder operation
Level: user goal
Primary Actor: Operator
Stakeholders:

- Operator: Runs the team builder program
- Students: Placed on teams
- Projects: Receive team members

Preconditions: Student and Project lists compiled and formatted in Excel file
Success Criteria (Postconditions): Project and Team member lists' output to Excel file
Main Success Scenario:

1. Operator loads student and project data
2. Operator adjusts team building settings
3. Operator initiates team building process
4. System creates teams according to described settings
5. System outputs final teams

Extensions (or Alternative Flows):

3a. Team building process stops with system error.
  1. Operator finds an error with the student input file format.
    1a. Operator corrects input file error.
      1. Return to step 1
  2. Operator has a memory issue with his/her computer.
    2a. Operator shuts down unneeded applications.

1. Return to step 1

4a. Settings are too restrictive to build teams

1. Operator chooses less restrictive settings

1a. Return to step 3

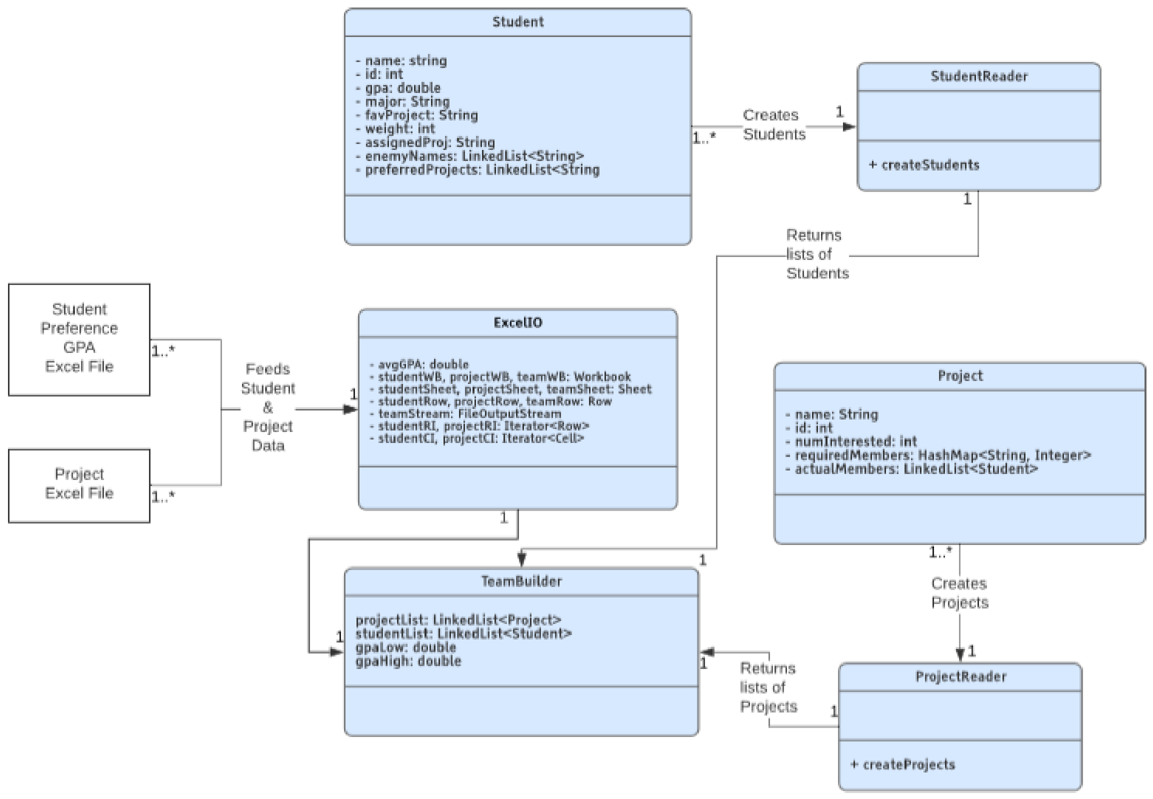
5a. Operator does not like the project teams' configurations
  1. Return to step 3
  2. If Operator does not want two students assigned to the same team, he/she adds student2 to student1's enemy list.
    2a. Return to step 1

5b. Operator needs to remove one project and add another, he/she makes the change to the project input file.
  1. Return to step 1

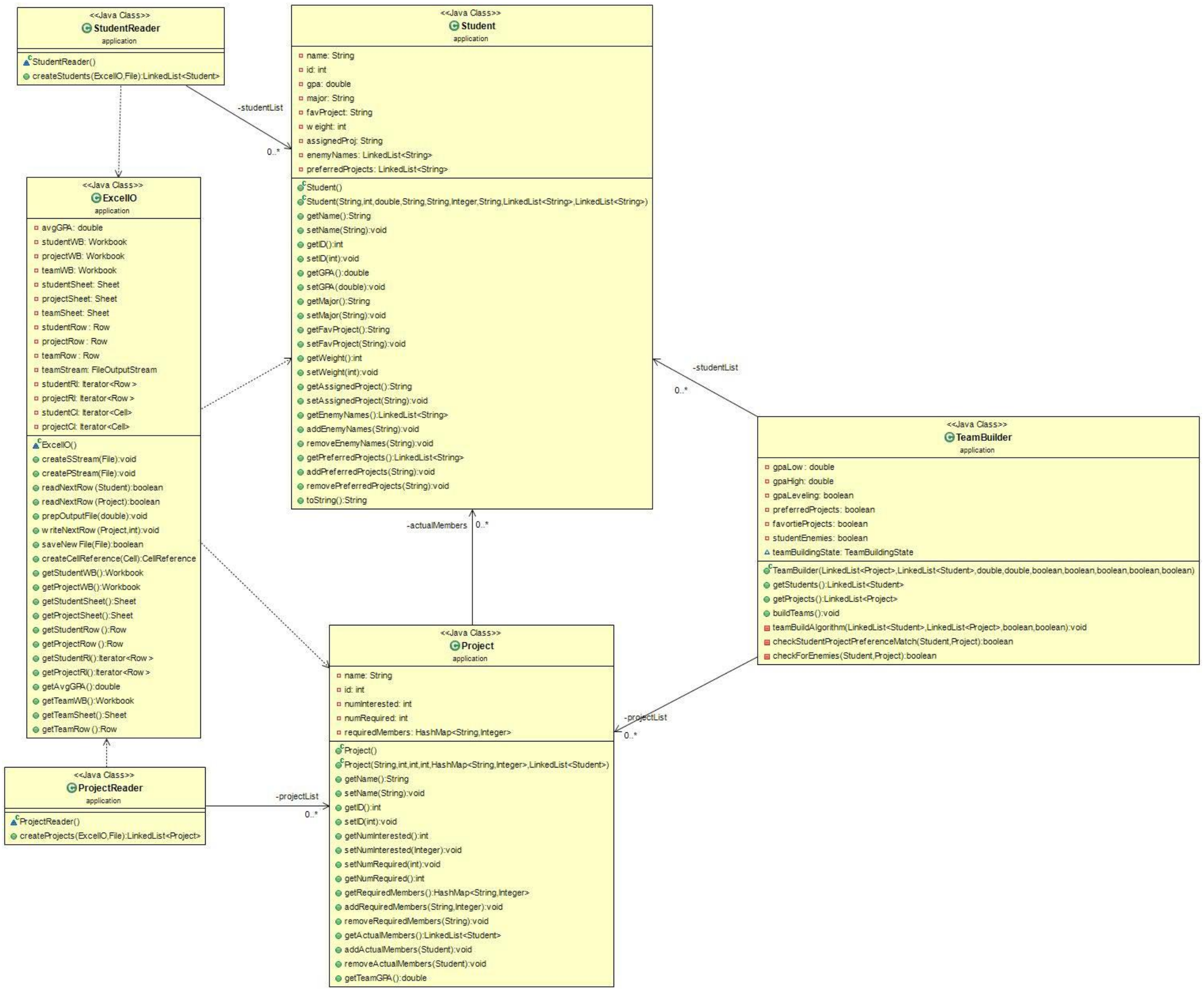
5c. Operator needs to add a new student, he/she makes the change to the student data Input file
  1. Return to step 1

UST Senior Project Domain Class Diagram

Craig Gabel, Andy Ludewig, Chris A Markey | December 13, 2019


Student
- name: string
- id: int
- gpa: double
- major: String
- favProject: String
- weight: int
- assignedProj: String
- enemyNames: LinkedList<String>
- preferredProjects: LinkedList<String
Creates Students
1..*
1
StudentReader
+ createStudents
1
Returns lists of Students
Student Preference GPA Excel File
Project Excel File
1..*
1..*
Feeds Student & Project Data
1
ExcelIO
- avgGPA: double
- studentWB, projectWB, teamWB: Workbook
- studentSheet, projectSheet, teamSheet: Sheet
- studentRow, projectRow, teamRow: Row
- teamStream: FileOutputStream
- studentRI, projectRI: Iterator<Row>
- studentCI, projectCI: Iterator<Cell>
1
1
Project
- name: String
- id: int
- numInterested: int
- requiredMembers: HashMap<String, Integer>
- actualMembers: LinkedList<Student>
1..*
Creates Projects
1
TeamBuilder
projectList: LinkedList<Project>
studentList: LinkedList<Student>
gpaLow: double
gpaHigh: double
1
1
Returns lists of Projects
1
ProjectReader
+ createProjects

Reverse-Engineered Class Diagram


<<Java Class>>
StudentReader
application
StudentReader()
createStudents(ExcelIO,File):LinkedList<Student>
-studentList
0..*
<<Java Class>>
Student
application
name: String
id: int
gpa: double
major: String
favProject: String
w eight: int
assignedProj: String
enemyNames: LinkedList<String>
preferredProjects: LinkedList<String>
Student()
Student(String,int,double,String,String,Integer,String,LinkedList<String>,LinkedList<String>)
getName():String
setName(String):void
getID():int
setID(int):void
getGPA():double
setGPA(double):void
getMajor():String
setMajor(String):void
getFavProject():String
setFavProject(String):void
getWeight():int
setWeight(int):void
getAssignedProject():String
setAssignedProject(String):void
getEnemyNames():LinkedList<String>
addEnemyNames(String):void
removeEnemyNames(String):void
getPreferredProjects():LinkedList<String>
addPreferredProjects(String):void
removePreferredProjects(String):void
toString():String
<<Java Class>>
ExcelIO
application
avgGPA: double
studentWB: Workbook
projectWB: Workbook
teamWB: Workbook
studentSheet: Sheet
projectSheet: Sheet
teamSheet: Sheet
studentRow : Row
projectRow : Row
teamRow : Row
teamStream: FileOutputStream
studentRI: Iterator<Row>
projectRI: Iterator<Row>
studentCI: Iterator<Cell>
projectCI: Iterator<Cell>
ExcelIO()
createSStream(File):void
createPStream(File):void
readNextRow(Student):boolean
readNextRow(Project):boolean
prepOutputFile(double):void
w riteNextRow(Project,int):void
saveNewFile(File):boolean
createCellReference(Cell):CellReference
getStudentWB():Workbook
getProjectWB():Workbook
getStudentSheet():Sheet
getProjectSheet():Sheet
getStudentRow():Row
getProjectRow():Row
getStudentRI():Iterator<Row>
getProjectRI():Iterator<Row>
getAvgGPA():double
getTeamWB():Workbook
getTeamSheet():Sheet
getTeamRow():Row
-studentList
0..*
<<Java Class>>
TeamBuilder
application
gpaLow: double
gpaHigh: double
gpaLeveling: boolean
preferredProjects: boolean
favortieProjects: boolean
studentEnemies: boolean
teamBuildingState: TeamBuildingState
TeamBuilder(LinkedList<Project>,LinkedList<Student>,double,double,boolean,boolean,boolean,boolean,boolean)
getStudents():LinkedList<Student>
getProjects():LinkedList<Project>
buildTeams():void
teamBuildAlgorithm(LinkedList<Student>,LinkedList<Project>,boolean,boolean):void
checkStudentProjectPreferenceMatch(Student,Project):boolean
checkForEnemies(Student,Project):boolean
-actualMembers
0..*
-projectList
0..*
<<Java Class>>
Project
application
name: String
id: int
numInterested: int
numRequired: int
requiredMembers: HashMap<String,Integer>
Project()
Project(String,int,int,int,HashMap<String,Integer>,LinkedList<Student>)
getName():String
setName(String):void
getID():int
setID(int):void
getNumInterested():int
setNumInterested(Integer):void
setNumRequired(int):void
getNumRequired():int
getRequiredMembers():HashMap<String,Integer>
addRequiredMembers(String,Integer):void
removeRequiredMembers(String):void
getActualMembers():LinkedList<Student>
addActualMembers(Student):void
removeActualMembers(Student):void
getTeamGPA():double
<<Java Class>>
ProjectReader
application
ProjectReader()
createProjects(ExcelIO,File):LinkedList<Project>
-projectList
0..*