COSC1295 Advanced Programming Assignment Report

S3815738 Wenkai Li

1. Justification of Algorithms, Data structures and Designs

Model class TeamFormation works as a mediator. It modifies and retrieves data from classes include Company, ProjectOwner, Project and Student. To ensure class TeamFormation has only one instance, I used singleton pattern. For GUI, I used MVP pattern. The mainController class works as a presenter to communicate between model and view. To reduce coupling, there is no direct link between model and view. A UML graph is shown in Appendix I for a detailed structure presentation.

In model class TeamFormation, to manipulate objects, I put objects and their id in hash maps. These hash maps are generic classes. Parameters for generic are String and Objects accordingly. Some maps need to keep the input order with LinkedHashMap while some maps need to order by value with TreeMap. I put student objects in a LinkedHashMap< String, Student > to keep the input order of students. I put Company, ProjectOwner and Project objects in a TreeMap< String, Object > to order map entries by id. I used LinkedHashSet < String > to store id of students who have been added to projects to avoid repeating project assignment and keep the input order. I used TreeSet< String > to store id of students who are available so they can be ordered by value. Two generic class stacks are used to store array lists of students and projects in a swap. Generic parameter for these stacks is ArrayList<String>. Generic parameter for these array lists is String which is id of students and projects.

In model class Project, I used HashMap<String, Integer> to store skill levels needed by the project. Each skill is mapped to an integer value. I used ArrayList<String> to store id of students who have been assigned to this project. I used HashSet<String> to store personality types of students in this project. It is easier to count distinct personality types with hash set because set can guaranty there is no repeated value. If the size of the hash set is less than three when there are 4 students in the project, a PersonalityImbalance exception will be thrown. I used an int variable to store the number of leaders. If it is 0 when there are 4 students in the project, a NoLeader exception will be thrown.

In model class Student, I used ArrayList<String> to store id of students who have a conflict with this student. Id of the student itself is also in this array list to avoid repeating. I used HashMap<String, Integer> to store skill levels. I used LinkedHashMap<String, Integer> to store project preferences and keep the input order. Values are assigned according to the order of project id input. For example, the first input project will be mapped to 4 and the second input project will be mapped to 3.

In mainController, I used ArrayList<CheckBox> to store checkboxes. To separate and manipulate checkboxes of each project, I stored array lists of checkboxes in ArrayList<ArrayList<CheckBox>> teams.

The heuristics algorithm to auto swap students is designed with greedy strategy. I used the sum of three standard deviations as objective function. The algorithm will do ten rounds of optimizations. In each round, it will find the largest standard deviation first. Then regarding this metric, it finds projects with max and min scores. In these two projects, it then finds students who caused max and min scores in these projects. Then it will try to swap these two students. If it throws exception or sum of metrics is not improving, it will swap them back and try second max or min student in these two projects, and so on. If students in these two projects are all tried but no improving, it will go to the next round. If the sum of metrics improved, it will keep the change and go to the next round of optimization. The whole process of auto swap will be printed in the console.

2. Scalability

A preview of the GUI is shown in Appendix II. The GUI is designed for if there are only 5 projects. There is no place in the stage for more teams. To make it more scalable, I can use expandable list view, accordion or other room-saving containers instead of checkboxes in panes. For bar charts, if there are more teams, they might need larger spaces to show all bars clearly. It can be improved by using separate scenes. For example, press a button to pop up a separate window to show a single bar chart. The scroll panes that show detailed information on projects and students scala well as users can scroll down to read all information. To make it more user friendly, I can also use a separate scene to show the information in a separate window.

The scalability of heuristics algorithm is good. If the number of projects is n, the time complexity is O(n) as it needs to scan all projects and find max and min projects according to the metric. When trying to swap students in max and min projects, the worst-case time complexity is $O(m^2)$ if m is the number of students in each project. However, m would not be very large because there would not be lots of students in each project. So, in each round of optimization, the time complexity is $O(n) + O(m^2)$, but it dominates by the number of projects. It also depends on how many rounds it needs to be done. Although this algorithm does not always achieve global optimization, it helps project managers to balance projects in a scalable way. Another issue is when no students swapping can improve the largest standard deviation, it will try to improve this metric again in next rounds. To avoid this, I can add a method to check whether the largest metric is the same as last round and terminate the loop if so.

3. Lessons learnt

I did not use composition pattern and interface to implement undo. Instead, I added two stacks to store array lists of swapped students and projects. When undo, it will pop the array lists out and swap students back. It is an easy way but might not as good as using composition pattern in terms of usability. Besides, undo is only for swap students but not for add students. For adding students, I can put array lists of the added student and project in the stack when adding students. Then add an if-else clause in undo method. If the size of the array list is 1, it means it is an undo for adding student. Then pop the array lists out and remove the student from the project. If the size of the array list is 2, it means it is an undo for swapping students. Then pop the array lists out and swap the students back.

I figured out how to use lambda expressions in the assignment. When calculating standard deviations, there were redundant codes at first. Then I used a functional interface and lambda expressions to replace those methods. Then pass the interfaces as parameters and combine methods for different standard deviations in one method.

I had an issue to undo the text swap in checkbox because text swap is in mainController and students swap is in model class TeamFormation at first. Then I had a solution. I made undo method to return the array list of the swapped students id. Then transfer this array list as a parameter to mainController to swap these two students id back in checkbox texts.

I serialized student and project objects to a file. Then I modified the objects. When I try to load the serialization file, it throws an InvalidClassException. I searched online and found that if the objects are modified, the serialization is not usable. The reason is if there is no declaration of serialVersionUID, JVM will generate one automatically at run-time. However, if the object is modified, the auto-generated serialVersionUID will change as well. To solve this, I can declare a serialVersionUID.

I used hash set to store members of projects. Then as the development of the program, I need to get members of projects by index. I found hash set has no method to get elements by index. So, I changed the data structure to array list. I learnt that the usage of data structure needs to be justified at the beginning. Then I found that I can simply convert hash set to array list without modification. I learnt that before modifying, try to find out whether there is an easier way first.

I had an issue in using SQLite with IntelliJ community. I searched online and found that I need a third-party plugin. When setting up JDBC classpath in IDE, I didn't know I have to right-click the java file to build path. I searched online and figured it out.

References:

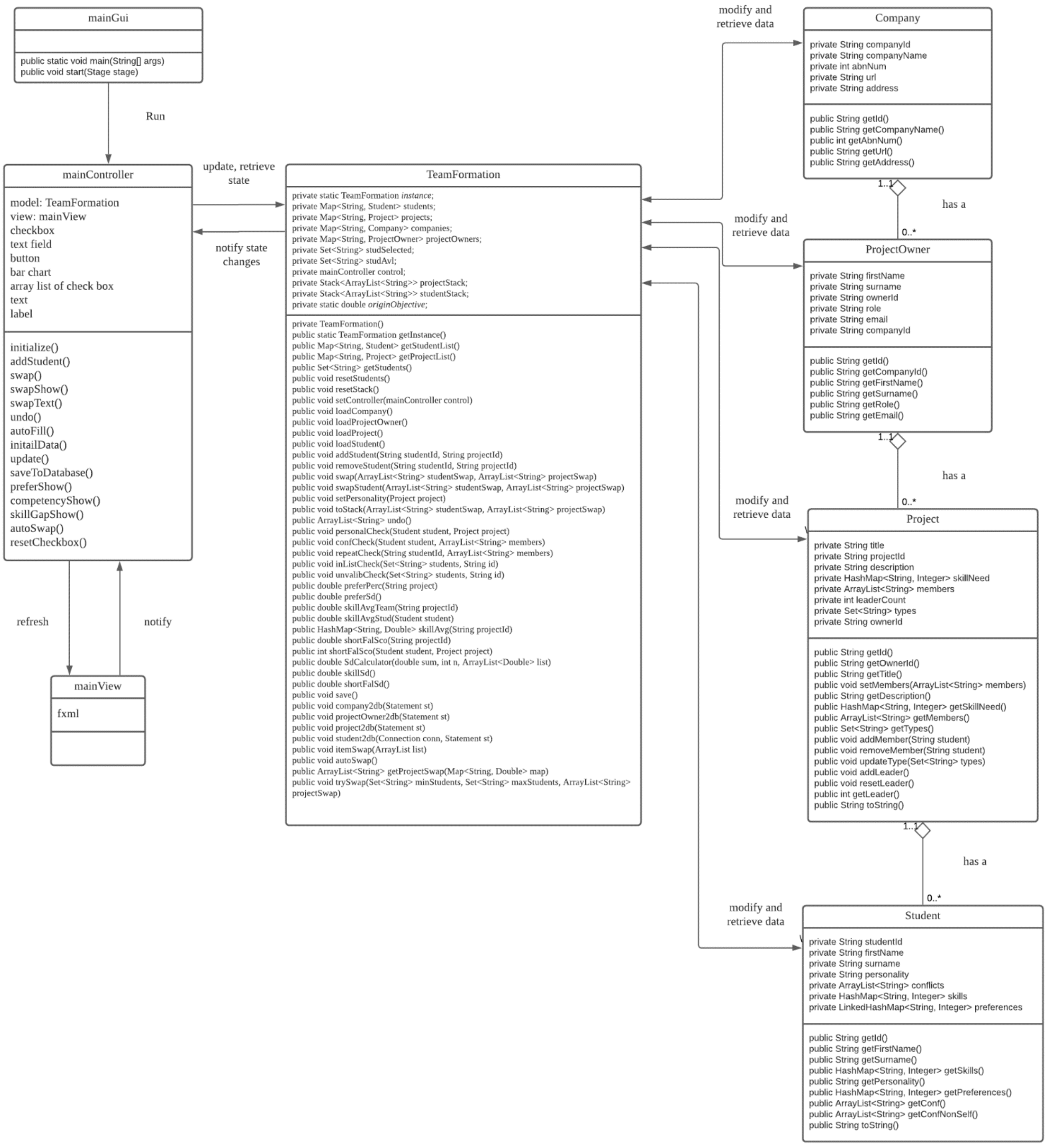
[1] "Introduction to Java Serialization | Baeldung", *Baeldung*, 2020. [Online]. Available: https://www.baeldung.com/java-serialization. [Accessed: 09- Oct- 2020].

[2] M. Pitt and M. Myers, "Why doesn't java.util.Set have get(int index)?", *Stack Overflow*, 2020. [Online]. Available: https://stackoverflow.com/questions/769731/why-doesnt-java-util-set-have-getint-index/769732. [Accessed: 09- Oct- 2020].

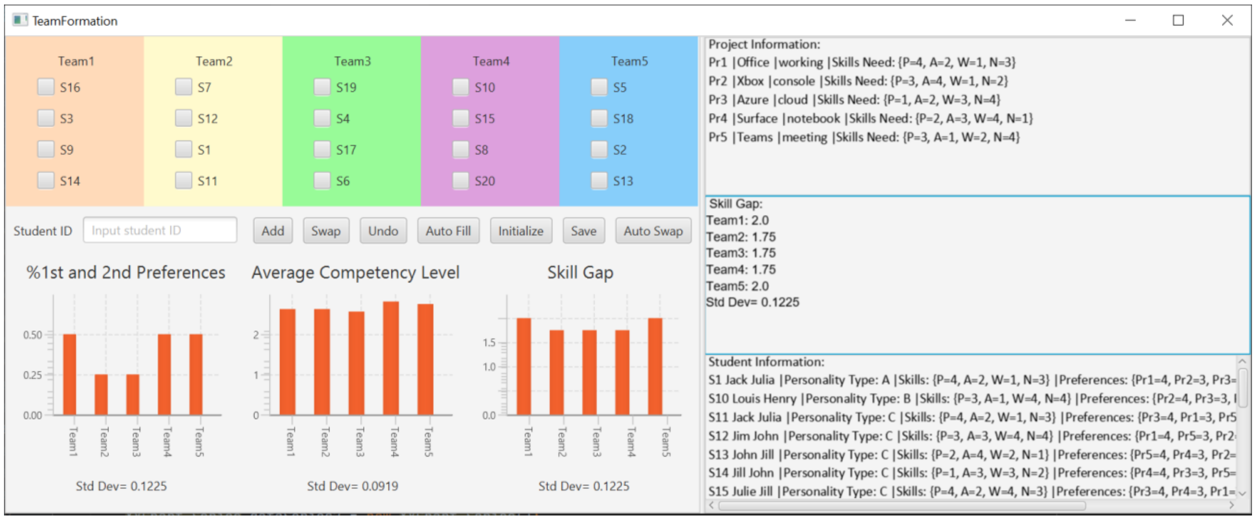
[3] "How to connect a Database server in Intellij Community Edition?", *Logicbig.com*, 2020. [Online]. Available: https://www.logicbig.com/how-to/intellij/intellij-community-edition-connecting-database.html. [Accessed: 09- Oct- 2020].

[4]"JDBC Tutorial: Connecting to Your Database Using JDBC", *Progress Blogs*, 2020. [Online]. Available: https://www.progress.com/blogs/jdbc-tutorial-connecting-to-your-database-using-jdbc. [Accessed: 09- Oct- 2020].

Appendix I


mainGui
public static void main(String[] args)
public void start(Stage stage)
Run
mainController
model: TeamFormation
view: mainView
checkbox
text field
button
bar chart
array list of check box
text
label
initialize()
addStudent()
swap()
swapShow()
swapText()
undo()
autoFill()
initailData()
update()
saveToDatabase()
preferShow()
competencyShow()
skillGapShow()
autoSwap()
resetCheckbox()
refresh
notify
mainView
fxml
update, retrieve state
notify state changes
TeamFormation
private static TeamFormation *instance*;
private Map<String, Student> students;
private Map<String, Project> projects;
private Map<String, Company> companies;
private Map<String, ProjectOwner> projectOwners;
private Set<String> studSelected;
private Set<String> studAvl;
private mainController control;
private Stack<ArrayList<String>> projectStack;
private Stack<ArrayList<String>> studentStack;
private static double *originObjective*;
private TeamFormation()
public static TeamFormation getInstance()
public Map<String, Student> getStudentList()
public Map<String, Project> getProjectList()
public Set<String> getStudents()
public void resetStudents()
public void resetStack()
public void setController(mainController control)
public void loadCompany()
public void loadProjectOwner()
public void loadProject()
public void loadStudent()
public void addStudent(String studentId, String projectId)
public void removeStudent(String studentId, String projectId)
public void swap(ArrayList<String> studentSwap, ArrayList<String> projectSwap)
public void swapStudent(ArrayList<String> studentSwap, ArrayList<String> projectSwap)
public void setPersonality(Project project)
public void toStack(ArrayList<String> studentSwap, ArrayList<String> projectSwap)
public ArrayList<String> undo()
public void personalCheck(Student student, Project project)
public void confCheck(Student student, ArrayList<String> members)
public void repeatCheck(String studentId, ArrayList<String> members)
public void inListCheck(Set<String> students, String id)
public void unvalibCheck(Set<String> students, String id)
public double preferPerc(String project)
public double preferSd()
public double skillAvgTeam(String projectId)
public double skillAvgStud(Student student)
public HashMap<String, Double> skillAvg(String projectId)
public double shortFalSco(String projectId)
public int shortFalSco(Student student, Project project)
public double SdCalculator(double sum, int n, ArrayList<Double> list)
public double skillSd()
public double shortFalSd()
public void save()
public void company2db(Statement st)
public void projectOwner2db(Statement st)
public void project2db(Statement st)
public void student2db(Connection conn, Statement st)
public void itemSwap(ArrayList list)
public void autoSwap()
public ArrayList<String> getProjectSwap(Map<String, Double> map)
public void trySwap(Set<String> minStudents, Set<String> maxStudents, ArrayList<String> projectSwap)
modify and retrieve data
Company
private String companyId
private String companyName
private int abnNum
private String url
private String address
public String getId()
public String getCompanyName()
public int getAbnNum()
public String getUrl()
public String getAddress()
1..1
has a
0..*
modify and retrieve data
ProjectOwner
private String firstName
private String surname
private String ownerId
private String role
private String email
private String companyId
public String getId()
public String getCompanyId()
public String getFirstName()
public String getSurname()
public String getRole()
public String getEmail()
1..1
has a
0..*
modify and retrieve data
Project
private String title
private String projectId
private String description
private HashMap<String, Integer> skillNeed
private ArrayList<String> members
private int leaderCount
private Set<String> types
private String ownerId
public String getId()
public String getOwnerId()
public String getTitle()
public void setMembers(ArrayList<String> members)
public String getDescription()
public HashMap<String, Integer> getSkillNeed()
public ArrayList<String> getMembers()
public Set<String> getTypes()
public void addMember(String student)
public void removeMember(String student)
public void updateType(Set<String> types)
public void addLeader()
public void resetLeader()
public int getLeader()
public String toString()
1..1
has a
0..*
modify and retrieve data
Student
private String studentId
private String firstName
private String surname
private String personality
private ArrayList<String> conflicts
private HashMap<String, Integer> skills
private LinkedHashMap<String, Integer> preferences
public String getId()
public String getFirstName()
public String getSurname()
public HashMap<String, Integer> getSkills()
public String getPersonality()
public HashMap<String, Integer> getPreferences()
public ArrayList<String> getConf()
public ArrayList<String> getConfNonSelf()
public String toString()

Appendix II



Team1	Team2	Team3	Team4	Team5
S16	S7	S19	S10	S5
S3	S12	S4	S15	S18
S9	S1	S17	S8	S2
S14	S11	S6	S20	S13

Std Dev= 0.1225

Std Dev= 0.0919

Std Dev= 0.1225

```
Project Information:
Pr1 |Office |working |Skills Need: {P=4, A=2, W=1, N=3}
Pr2 |Xbox |console |Skills Need: {P=3, A=4, W=1, N=2}
Pr3 |Azure |cloud |Skills Need: {P=1, A=2, W=3, N=4}
Pr4 |Surface |notebook |Skills Need: {P=2, A=3, W=4, N=1}
Pr5 |Teams |meeting |Skills Need: {P=3, A=1, W=2, N=4}
```

```
Skill Gap:
Team1: 2.0
Team2: 1.75
Team3: 1.75
Team4: 1.75
Team5: 2.0
Std Dev= 0.1225
```

```
Student Information:
S1 Jack Julia |Personality Type: A |Skills: {P=4, A=2, W=1, N=3} |Preferences: {Pr1=4, Pr2=3, Pr3=
S10 Louis Henry |Personality Type: B |Skills: {P=3, A=1, W=4, N=4} |Preferences: {Pr2=4, Pr3=3, I
S11 Jack Julia |Personality Type: C |Skills: {P=4, A=2, W=1, N=3} |Preferences: {Pr3=4, Pr1=3, Pr5
S12 Jim John |Personality Type: C |Skills: {P=3, A=3, W=4, N=4} |Preferences: {Pr1=4, Pr5=3, Pr2
S13 John Jill |Personality Type: C |Skills: {P=2, A=4, W=2, N=1} |Preferences: {Pr5=4, Pr4=3, Pr2=
S14 Jill John |Personality Type: C |Skills: {P=1, A=3, W=3, N=2} |Preferences: {Pr4=4, Pr3=3, Pr5=
S15 Julie Jill |Personality Type: C |Skills: {P=4, A=2, W=4, N=3} |Preferences: {Pr3=4, Pr4=3, Pr1=
```