# Description

The Application part 1 shows a CLI application for a Job Agency. It has option for listing all the Jobs and Job Seekers. There are two different classes for them.

JobVacancy Class is for Jobs and JobSeeker Class is for Job Seekers.

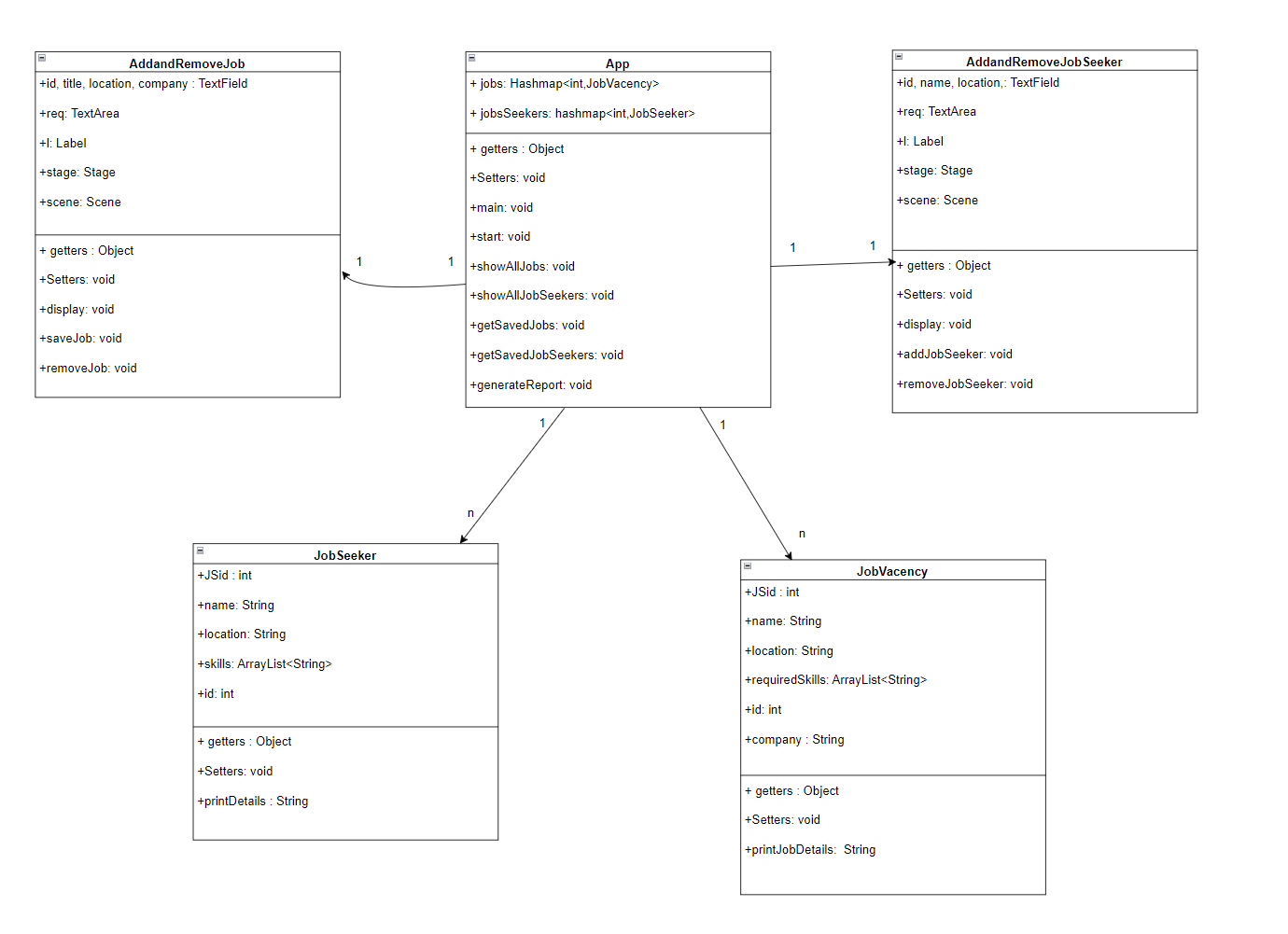
Then there is a main class that is the working layer of application and provides all features.

In Part 2, there are two additional classes named AddandRemoveJob and AddandRemoveJobSeeker. These classes provide GUIs for adding New Jobs and Job Seekers and Removing them as well.

To store the data permanently, there are text files that store the data. When Add or Remove Functions are called for any class, records in these files also change.

Part 2’s main class is App class that provides the startup interface with all options to the user.

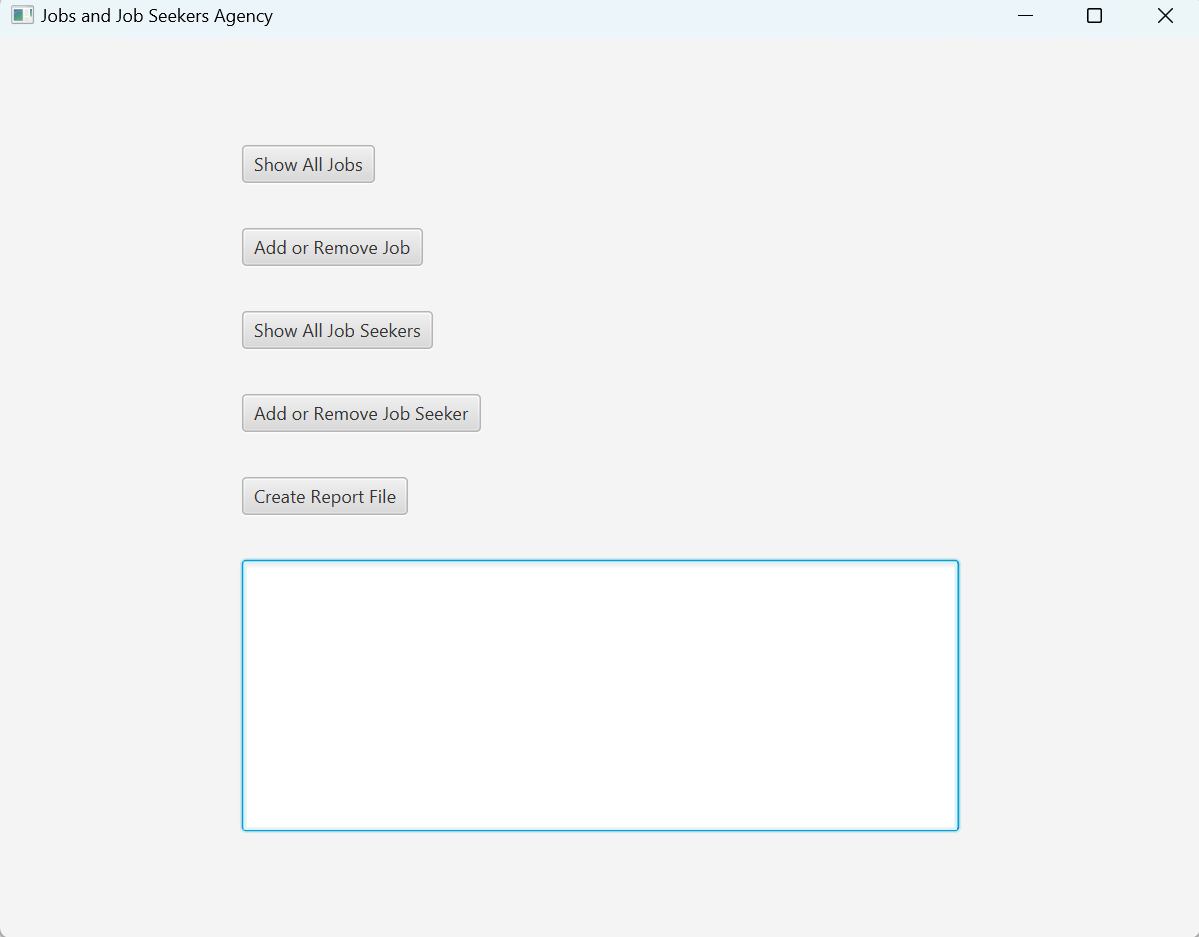
The Class Diagram for the GUI part is as following:

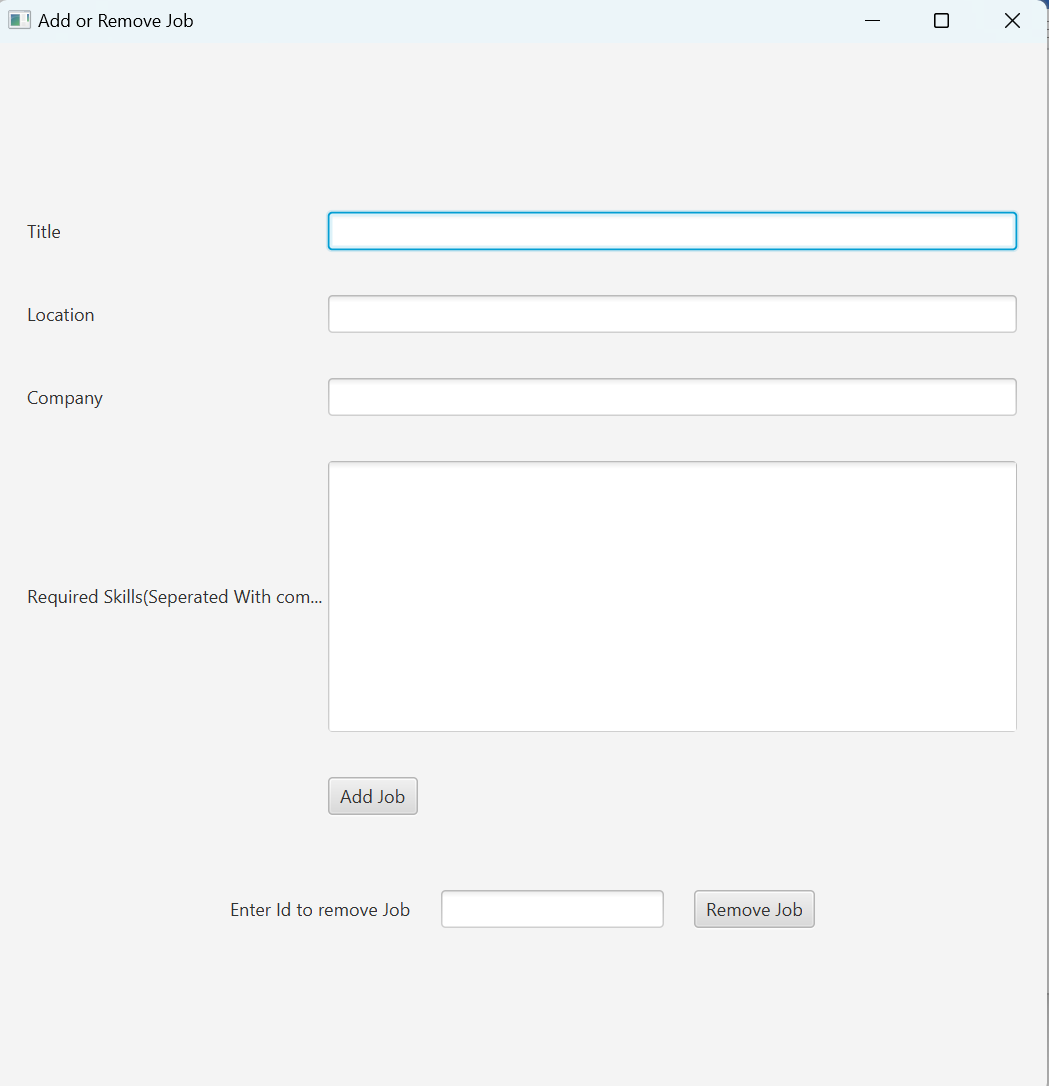


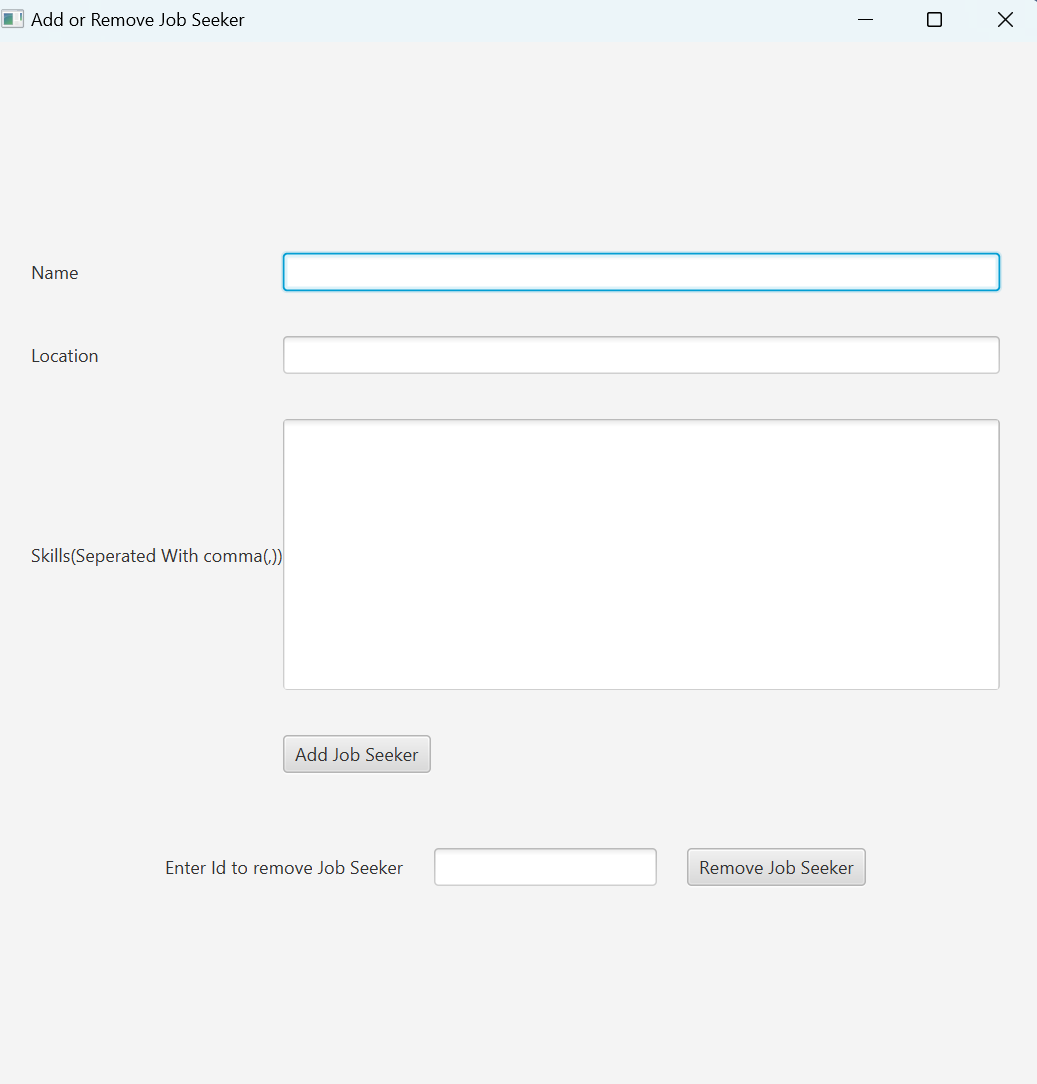
The Table for the Part 2 is as following:

|  |  |  |
| --- | --- | --- |
|  | **Example of Code** | **Comment** |
| **JavaFX** | @Override      public void start(Stage primaryStage) throws Exception {          root = new GridPane();          primaryStage.setTitle("Jobs and Job Seekers Agency");          Button button = new Button("Add or Remove Job");          Button button2 = new Button("Add or Remove Job Seeker");          Button showAllJobs = new Button("Show All Jobs");          Button showAllJobSeekers = new Button("Show All Job Seekers");          Button createReport = new Button("Create Report File");          area = new TextArea();          root.setAlignment(Pos.CENTER);          root.setVgap(30);          root.setPadding(new Insets(30,30, 30, 30));…………… | This is the start function from Application class. |
| **Lambda Expression** | button.setOnAction(e->AddandRemoveJob.display());          button2.setOnAction(e->AddandRemoveJobSeeker.display());          showAllJobs.setOnAction(e->showAllJobs());          showAllJobSeekers.setOnAction(e->showAllJobSeekers());          createReport.setOnAction(e->generateReport()); | These are some lambda expressions used in App class to add ActionEventHandler to buttons. |
| **Exceptions** | jobsSeekers = new HashMap<>();          jobs = new HashMap<>();          try {              Jobsreader = new BufferedReader(new FileReader("./Text Files/Jobs.txt"));              Jobswriter = new BufferedWriter(new FileWriter("./Text Files/Jobs.txt", true));              JobSeekersreader = new BufferedReader(new FileReader("./Text Files/JobSeekers.txt"));              JobSeekerswriter = new BufferedWriter(new FileWriter("./Text Files/JobSeekers.txt", true));          } catch (Exception e) {              e.printStackTrace();              area.setText("There must have been changes to text files");          } | Every Exception is handled and shown error details according on the GUI. |
| **Collection classes** | 1. //Hashmaps for storing jobs and jobSeekers       private static HashMap<Integer, JobVacency> jobs;      private static HashMap<Integer, JobSeeker> jobsSeekers;  public void setSkills(ArrayList<String> skills) {          this.skills = skills;      } | ArrayList, Hashmap and other collection classes are used to store data and perform operations quickly and in an efficient manner. |
| **File handling** | File toChange = new File("Text Files/Jobs.txt");              List<String> list;              try {                  list = Files.lines(toChange.toPath()).filter(line -> !line.startsWith(id + ""))                          .collect(Collectors.toList());                          Files.write(toChange.toPath(), list, StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE\_EXISTING);              } catch (IOException e) {                  l.setText("Something Went Wrong..");                  e.printStackTrace();              } | File Handling is used in nearly every function. All FileHandlers are closed after the use. |

Some GUI interfaces of the application looks like this:







A variety of java classes are used in the project like Hashmaps and ArrayLists to store data and manipulate the data and File Handling is done using BufferedReader classes and also java.nio.File.files class. Lambda expressions and Stream API is used to make the program work more in less code and in a sophisticated manner.

User will have all kind of information on the screen whenever he performs any operation.

# File Structure

Files are structured in a manner that Part 1 and Part 2 are in different directories.

That is why copy of some java files may exist in both folders and Also there is a Text Files folder in each Part that has the text files in it which are useful for storing data and the report file is also created in the same folder.

If any exception is thrown. It is handled in every part of application and the error stack trace gets printed on the console and a simple warning is shown to the user on the GUI interface too.

GUI creation in divided in three classes:

1. App class: This is the main class that runs the application and has the starting interface.
2. AddandRemoveJob class: This class opens a GUI interface for adding and Removing a job from files and hashmaps too.
3. AddandRemoveJobSeeker: This class opens a GUI interface for adding and Removing a job seeker.