RMIT UNIVERSITY

Assignment 1
Technical report

Course: Programming 1

Lecturer: Thanh Nguyen

Team members:

Minh Hoang - s3871126

Dat Nguyen - s3870837

Trung Tien - s3836390

Taehyeon Jeong - s3799019

Khang Duc - s3824317

1. System overview

The system written in Java serves the purpose of a Customer Relationship Management (CRM) with the ability to manage leads and interactions, both are crucial data forms for businesses. Based on the requirements, we can describe the system's functionality as follow:

1. Syncing with the data files when opening the program for the first time, the system will update the current data within the system to the data in the file.
2. Provide a console-based interface for the user to navigate and control the system, commands provided are:

    -Viewing the data stored in the system

    -Add a new data to the system

    -Delete a data from the system

    -Update a data stored in the system

    -Create and view reports based on each data type and requirements (by Months, by Age, etc)

3. Update the data file based on the data stored in the system and the command executed.

This will create a loop for the user to continuously execute commands and consequently update the data file until the user decides to exit the program with a special command.

2. Technologies used in the program

- The program is created based purely on Java (version 15.0.1) as the programming language
- IDE for development: Intellij IDEA
- Tool for class diagram creation: starUML

- References and guides are the lectures from the course and information from the internet

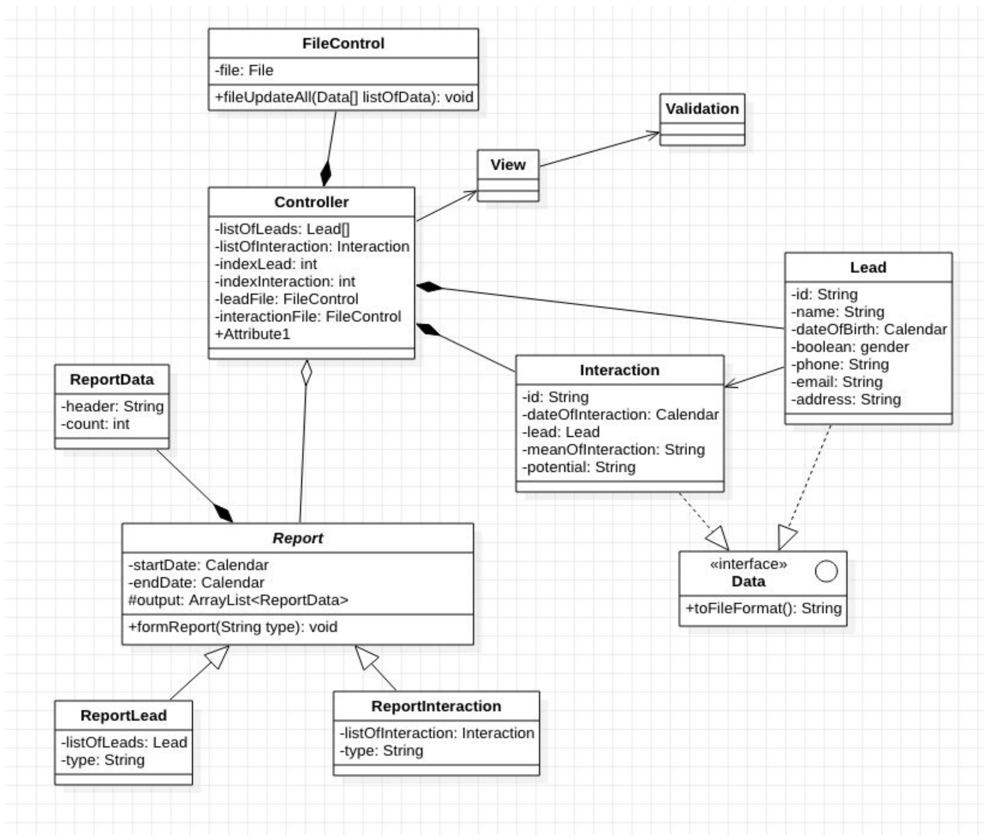
1. Progress

a) <u>Sketching the class design for the application</u>

The most important thing we had to do before starting to code is to design a class diagram based on the description provided. We decided to use the MVC model as our guideline for class structure.

The details of the MVC model will be omitted as more information can be found online, but the basic idea is to have a ***C***ontroller class represent the system which holds all the data process during the application execution, a ***V***iew class to display the necessary information including proper requests for input and formatted data to the user (in this case is through the console) and ***M***odel classes representing all the data types in the application.

The diagram below will serve as a structural view on the system, thus excluding unimportant/ambiguous details. The description of each class can be found on the later sections.



The system's process can be summarized as follow:

- Class "Controller": Controls the system, contains the Models (Lead, Interaction, Report) and uses helping classes (Validation, FileControl, View) for managing the system. Which command is available for the user and what should the system do is based on the current location:

    i. Use view (Class View) to show initial words explaining the situation
    ii. Use validate (Class Validation) to get the processed input from the user (looping if necessary)
    iii. Turn the processed input to proper data -> executes proper methods (including saving the input to the data stored in the Controller and update the file using FileControl)
    iv. Move to the next location

- Models: Class "Lead", class "Interaction" and "Report" are the data and will be updated during the program's execution

- Class "View": Show the output to the console and return required input (validated by "Validation" class), which will be moved to the Controller for processing
- Class "Validation": Check the String input from the console and return the processed data from the input
- Class "FileControl": Update the file based on the data stored in the Controller, initialize the data in the Controller at the start of the program
- Interface "Data": to indicate polymorphism with Lead and Interaction class for the method "toFileFormat": write the Data to the file

b) Developing internal functions

- Class “Controller”
  + **Controller()**: create constructors for the class.
  + **start()**: to start the Controller and use changeLocation() to move to different classes.
  + **changeLocation()**: determine and navigate the user to different “menus” with the user’s input from View class.

- Class “View”
  - **firstWords()**: show the message to announce that the program is started and notes for usage (important)
  - **mainMenu()**: present the main menu of the program and get the input from the user. Then, transfer the user input to the validation class to make sure that it is the correct format.
  - **subMenuLead()**: if the main menu input is validated, print the sub menu of the list of lead and if after checking the input through the validation.
  - **subMenuInteraction()**: same process with subMenuLead, if the user input is correct, execute by showing the options.
  - **subMenuReport()**: if the sub menu report input is validated, print the sub menu report “options” and get the next option from the Validation class’s corresponding method.

- **viewLeadList()**: if the users choose the view lead list option in the subMenuLead(), this will show the list of leads for them.
- **addLeadMenu()**: if the users choose the add lead option in the subMenuLead(), they will be asked to enter all the input for a new lead.
- **deleteLeadMenu()**: if the users choose the delete lead option in the subMenuLead(), they will be asked to delete a lead based on an id input.
- **updateLeadMenu()**: if the users choose the update lead option in the subMenuLead(), they will be asked to update an existing lead with their input.
- **getNewLead()**: users are asked to enter data for a new lead and it will be added after getting validated in validation class.
- **viewInteractionList()**: if users choose the view interactions list option in the subMenuLead(), this will show the list of interactions for them.
- **getNewInteraction()**: users are asked to enter data for a new interaction and it will be added after getting validated in validation class.
- **addInteractionMenu()**:if the users choose the add interaction option in the subMenuLead(), they will be asked to enter input for a new interaction.
- **deleteInteractionMenu()**: if the users choose the delete interaction option in the subMenuLead(), they will be asked to delete an interaction based on input.
- **updateInteractionMenu()**: if the users choose the update interaction option in the subMenuLead(), they will be asked to update an existing interaction with their input.
- **viewReportListByAge()**: a list of reports will be shown based on Age.
- **viewReportInteractionByPotential()**: a list of interactions will be shown based on potential.
- **viewReportInteractionByMonth()**: a list of interactions will be shown based on the month.
- **printReport()**: show the report with required format.

- Class “Validation”

- **Validation()**: create constructor for this class.
- **trimInput(String input)**: remove the space and reverse uppercase to lowercase from a string input.
- **isInteger()**: check if an String input can be converted to integer.
- **getMainMenuInput()**: ask and validate the correct input format for Main Menu options, prompting the user to enter again if incorrect.
- **getLeadMenuInput()**: ask and validate the correct input format for Lead Menu options, prompting the user to enter again if incorrect.
- **getInteractionMenuInput()**: ask and validate the correct input format for Interaction Menu options, prompting the user to enter again if incorrect.
- **getReportMenuInput()**: ask and validate the correct input format for Report Menu options, prompting the user to enter again if incorrect.
- **getNewLeadInput()**: ask and validate the correct input format for New Lead option, prompting the user to enter again if incorrect.
- **getNewInteractionInput()**: ask and validate the correct input format for New Interaction option, prompting the user to enter again if incorrect.
- **getCurrentLeadIdInput()**: ask and validate the correct input format for Current Lead Id option, prompting the user to enter again if incorrect.
- **getLeadNameInput()**: ask and validate the correct input format for Lead Name option, prompting the user to enter again if incorrect.
- **getLeadDateOfBirthInput()**: ask and validate the correct input format for Lead Date Of Birth option, prompting the user to enter again if incorrect.
- **getLeadGenderInput()**: ask and validate the correct input format for Lead Gender option, prompting the user to enter again if incorrect.
- **getLeadPhoneInput()**: ask and validate the correct input format for Lead Phone option, prompting the user to enter again if incorrect.
- **getLeadEmailInput()**: ask and validate the correct input format for Lead Email option, prompting the user to enter again if incorrect.
- **getLeadAddressInput()**: ask and validate the correct input format for Lead Address option, prompting the user to enter again if incorrect.

    - **getCurrentInteractionIdInput()**: ask and validate the correct input format for Current Interaction Id option, prompting the user to enter again if incorrect.
    - **getInteractionDateInput()**: ask and validate the correct input format for Interaction Date option, prompting the user to enter again if incorrect.
    - **getReportLead()**: ask and validate the correct input format for Report Lead option, prompting the user to enter again if incorrect.
    - **getReportInteraction()**: ask and validate the correct input format for Report Interaction option, prompting the user to enter again if incorrect.
    - **getMeanOfInteractionInput()**: ask and validate the correct input format for Mean Of Interaction option, prompting the user to enter again if incorrect.
    - **getInteractionPotentialInput()**: ask and validate the correct input format for Interaction Potential option, prompting the user to enter again if incorrect.

- Class “FileControl”
    - **fileControl()**: constructor to create new file, with the file name from user input
    - **fileUpdateAll()**: Whenever the data in either List of Leads or List of Interaction is modified(updating or deleting), this function can update all of the final data to the existing file.
    - **transferData(Lead[] listOfLeads)**: Transfer the data from the existing file (list of Leads) to the system and return the index to keep track of the number of elements of lead in the system.
    - **transferData(Interaction[] listOfInteractions, Lead[] listOfLeads)**: Transfer the data from the existing file (List of Interaction). We have to include the constructor List of Leads to link with the leadID. Then add to the system and return the index to keep track of the number of elements of interaction in the system.

- Class “Lead”

    + **Lead()**: has two constructor functions, one has the int attribute id, one has the String attribute id(converted by using processId() function), Each has its own purpose.
    + **processId()**: used by one of the Lead constructor to convert the int input to the String input with the right format.
    + **toFileFormat()**: Implemented from Data interface and override to format data into readable text to write in a file.
    + **Getter functions**: getId(), getName(), getDateOfBirth(), getPhone(), getEmail(), getAddress().
    + **Setter functions**: setId(), setName(), setDateOfBirth(), setPhone(), setEmail(), setAddress().

- Class “Interaction”
    + **Interaction()**: create constructors for the class.
    + **processId()**: used by the Interaction constructor to convert the int input to the String input with the right format.
    + **Getter functions**: getId(), getDateOfInteraction(), getLead, get MeanOfInteraction(), getPotential().
    + **Setter functions**: setId(), setDateOfInteraction(), setLead, set MeanOfInteraction(), setPotential().

- Interface “Data”
    + **toFileFormat()**: empty function for its implementing class to override (Lead class and Interaction class). Purpose is to turn the Data object to a String suitable for printing into the file.

- Class “Report”
    + **ReportData[] output:** the array of processed data, used for viewing (View class) and further processing
    + **Report()**: create constructor for three classes, one with startDate and endDate, one only has startDate and one with none.
    + **getPeriod()**: return String to let the user know the period (start day and end day) of the report shown.

    + **inReport()**: to check that the date is in the report date range (start day < x < end day) or not.
    + **getOutput()**: return the output array.
    + **formReport(String type)**: the abstract method for other classes extended from the Report class (Report Lead and Report Interaction).
- Class “ReportData”
    + String header, int count: header and the number of occurrence for each type of date created from the Report
    + **ReportData**: create constructor for the class.
    + **Getter functions**: getCount(), getHeader().
    + **Setter functions**: setHeader(), setCount().
    + **increase()**: used to increase count by 1.
- Class “ReportLead”
    + **ReportLead()**: create constructor for three classes, one has starting date and end date, one only has starting date and one only has end date.
    + **getAge()**: return the age of the specific lead
    + **formReport()**: Override the abstract method extended from the Report class to get the input from the user and call the formAgeReport
    + **formAgeReport()**: create the age report.
- Class “ReportInteraction”
    + **ReportInteraction()**: create constructor for three classes, one has starting date and end date, one only has starting date and one only has end date.
    + **Getter functions**: getPotential(), getInteractionDate()
    + **monthTransfer()**: change the month format of the Calendar Object from integer to a string.
    + **formReport()**: Override the abstract method extended from the Report class to get the input from the user and call the **formPotentialReport()** or **formMonthReport()** according to the user input.
    + **formPotentialReport()**: create the interaction potential report.

        + **compareX(Calendar date1, Calendar date2)**: compare 2 dates according to a special requirement (in this case is the Year and Month of the Calendar only)
        + **formMonthReport()**: create the interaction monthly report. The premise would be get all the dates in range (from start date to end date entered by the user) as a list. Sort the list using Collection.sort and then count all the similar dates together and form a ReportData, loop the process and we get a list of ReportData (output)

- Class “CalendarSortingComparator”: use for create a custom comparator to use **Collection.sort** for the **formMonthReport()**
    + **compare()**: we will compare the year first then to the month. Return 1 if date1 is bigger than date2, return 0 if date1 equals date2, return -1 otherwise. Similar to compareX

3. Task Division

- **Hoang Minh (Leader)**: Design the working flow between classes and the architecture of the system. Responsible for the task contribution for the other developers. Review the implemented code blocks and offer the guide to the other team members who deal with difficulties.
- **Dat Nguyen (Developer, Analyst)**: Analyze the requirements of the projects through profound comprehension, and give suggestions and clarify the potential problems that may occur during the development phase. Study the system in detail and implement the assigned tasks.
- **Trung Tien (Developer, Consultant)**: Fully understand how the system works in order to consult other developers to deal with their assigned tasks. Supervise coding conventions for better performance and successful outcome of the project.

- **Khang Duc (Developer, Tester)**: be aware of the correct requirement how the system should work in order to spot any possible flaw during testing. Testing the program to ensure the system meets the requirement with high quality.
- **Taehyeon Jeong (Developer, Technical writer)**: Have a grasp of the architecture of the system and code blocks to document everything in the program to the report. In charge of the documentation of the project process from the designing phase to the technical report. Managing the documentation requirements.

4. Result and what are not done

Overall, we successfully implemented all the features that we intended to do at the beginning. Even though the code is not so clean and the algorithms for solving problems are not ultimately optimized, we are happy with the final result. We believe that we can improve our code to be cleaner and more optimized. Also, in our system, interaction with the users is only done through the console. We could have implemented a more interactive interface to improve the convenience if we started earlier and had a more detailed and structured working plan.

The most notable problem needs to be solved: As we decided to choose Array as the data storage model and the “correct id index” as the search technique for getting the data out of the storage, modification to the file with random id (not sequential) will result in undefined output. This is an inherent problem due to initial design choices, changes would drastically change the source code.

5. Challenges and reflection

Building a whole system is never an easy thing since most of us are not having any experience with this system. At the beginning of the week when the assignment was released, we have come up with various ideas about how we should build our system. Initially, we decided to keep it simple and apply what we have learned in the course to make a simple system like one interface, two classes, some simple functions. However, Minh - our team's leader, told us about the Model-View-Controller (MVC) software design pattern, which is widely known in the real working environment. After discussion, we decided to apply that pattern to develop our system. Undoubtedly, MVC is complicated and sophisticated to understand since we were not familiar with the pattern. As a result, we continuously have encountered obstacles and problems. Luckily, all members of our team were eager to find solutions when the problems came up. Therefore, even if the progress might be slow and hard, we are finally able to pull it off.

The first challenge that we confronted was trying to understand the MVC architecture. Most of the team members had not much experience to deal with the controller system pattern. Thus, the general comprehension about MVC architecture, classes, methods took longer than we expected. Thankfully, one of the team members Minh, who has experience with that pattern, is really supported and more than willing to guide our team. We learnt it from each other, asking whenever we couldn't understand and stuck in it. If no one has an answer, we looked up for it on the internet or asked our lecturer. At the end, we could gradually understand the code, how it functions and how to write it.

Second obstacle, besides the understanding about the MVC architecture and structure of the project itself, was the difficulties with the working process along with GitHub. To facilitate the collaboration of implementing a computer program based on Java language, we took advantage of GitHub which enables the track of all iterations and modifications that team members make. However, despite the fact that we all have somewhat experienced with GitHub, the working process was not as smooth as we all expected. GitHub made it easy to contribute and document to our project but it required a certain amount of understanding. At the initial phase, Minh, the host of the github page, allowed the other members to push our code directly to the github page. However, some members accidentally deleted the original codes without noticing and pushed it which resulted in minor and severe progress loss. As a result, Minh changed the github page to have pull requests only so that we don't make mistakes again.

Thirdly, we had difficulty with time management. Even though we started early with the planning, we spent the first two weeks getting to know the code. All of us studied three or four courses this semester and had other assignments and projects to work on. This severely hinders our progress. When we were confident enough to start working on the system practically, we only had two weeks left. We decided to have a short team meeting and divide our work fittingly. At this point, we all realized the importance of detailed and concrete plans for better performance and the final outcome. Fortunately, everyone reasonably did their work before their deadline, this greatly helped us stable the project.

As stated above, during this assignment, we had to deal with the complicated codes of MVC, the time pressure, even get our progress deleted at some point due to some error in Git. All of the challenges above has put us to have been under a lot of stress. However, we didn't crack under pressure. Our teamwork was very good. We understood that some aren't as good as others and we didn't blame anyone as long as that person tries his/her best. If one had too much work that he/she couldn't handle beyond his/her ability, other team members available helped that member. We believe that a good leader or member is the one who not only does his part, but also helps his/her team to overcome any obstacle.

We wanted to have an informative challenge that can give us valuable experience and knowledge along the way that could help us in the future. We went through lots of hardship and problems during every phase from the beginning to the end, but none of us gave up. In conclusion, we can say with confidence that we are satisfied with the achievement we constitute, quality of the project and the skill that we acquired during this assignment.