ASSIGNMENT COVERSHEET

PRAGUECITY UNIVERSITY

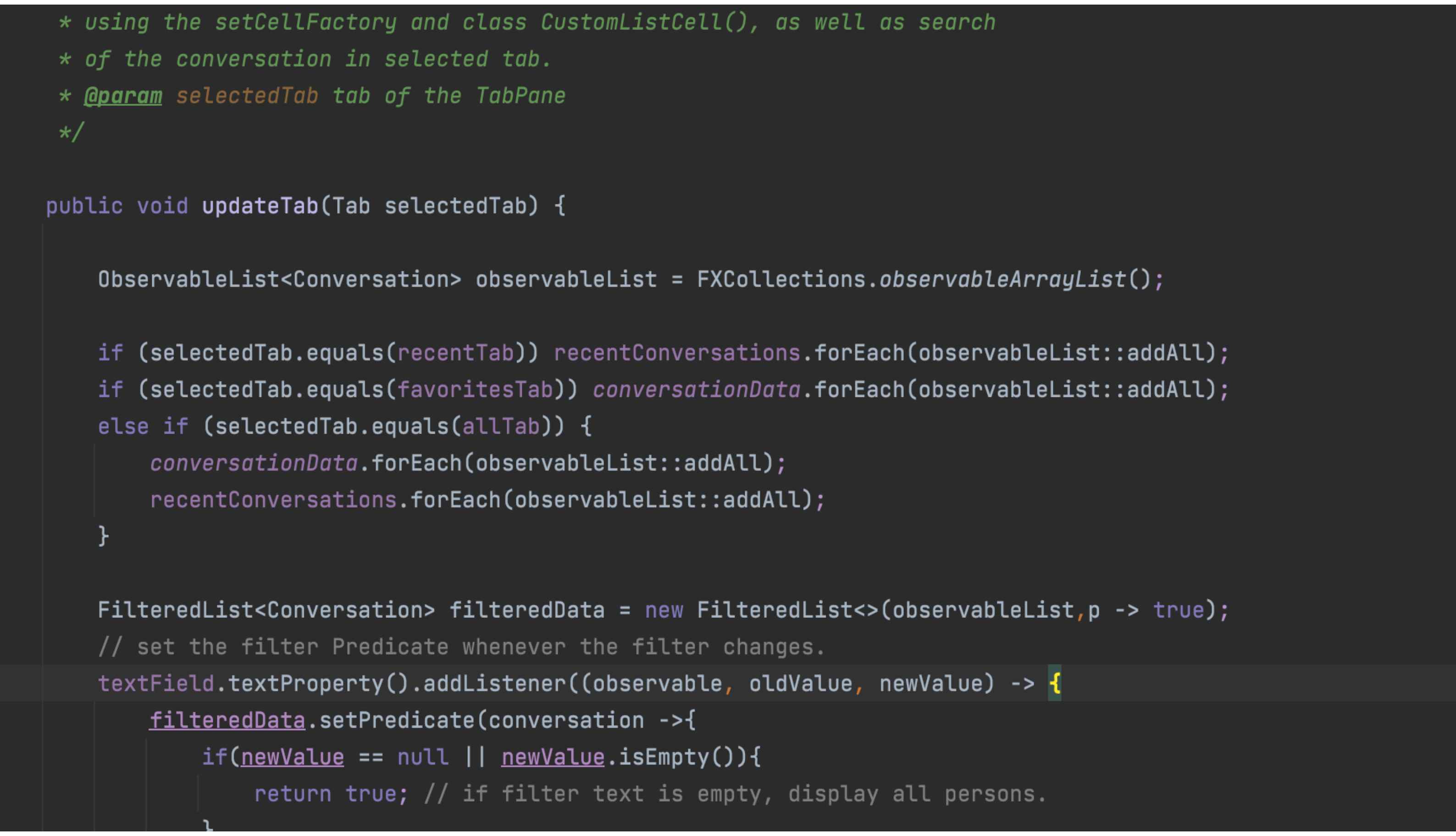
Student Name: Sebila Doubaeva	
Class: Java Programming (ESME)	
Assignment: Task 1	
Lecturer: Viktor Černý	Semester: 2104
Due Date: 24.01.2022	Actual Submission Date: 24.01.2022

Evidence Produced (List separate items)	**Location (Choose one)**	
	X	Uploaded to the Learning Center (Moodle)
		Submitted to reception
Note: Email submissions to the lecturer are not valid.		

Student Declaration:	
I declare that the work contained in this assignment was researched and prepared by me, except where acknowledgement of sources is made. I understand that the college can and will test any work submitted by me for plagiarism. **Note:** The attachment of this statement on any electronically submitted assignments will be deemed to have the same authority as a signed statement	
Date: January 24, 2022	Student Signature: Sebila Doubaeva

A separate feedback sheet will be returned to you after your work has been graded.
Refer to your Student Manual for the Appeals Procedure if you have concerns about the grading decision.

Student Comment (Optional)
Was the task clear? If not, how could it be improved?
Was there sufficient time to complete the task? If not, how much time should be allowed?
Did you need additional assistance with the assignment?
Was the lecturer able to help you?
Were there sufficient resources available?
How could the assignment be improved?



```
 * using the setCellFactory and class CustomListCell(), as well as search
 * of the conversation in selected tab.
 * @param selectedTab tab of the TabPane
 */

public void updateTab(Tab selectedTab) {

    ObservableList<Conversation> observableList = FXCollections.observableArrayList();

    if (selectedTab.equals(recentTab)) recentConversations.forEach(observableList::addAll);
    if (selectedTab.equals(favoritesTab)) conversationData.forEach(observableList::addAll);
    else if (selectedTab.equals(allTab)) {
        conversationData.forEach(observableList::addAll);
        recentConversations.forEach(observableList::addAll);
    }

    FilteredList<Conversation> filteredData = new FilteredList<>(observableList,p -> true);
    // set the filter Predicate whenever the filter changes.
    textField.textProperty().addListener((observable, oldValue, newValue) -> {
        filteredData.setPredicate(conversation ->{
            if(newValue == null || newValue.isEmpty()){
                return true; // if filter text is empty, display all persons.
```

(excerpt from the project)

Task 1

Sebila Doubaeva

January 24, 2022

1 Abstract

This report contains the requirements of the first task, specifically an analysis of this problem and the scope, the UML class diagram, as well as a textual description of the components and their relationship in the project.

Contents

2 Scope

The main goal of this project is to create a parser of .msg format files and present the result in a Graphical User Interface shell.

Each .msg file contains a conversation consisting of messages, which in turn consists of three lines :
- The first line containing the time in format hh:mm:ss following after the string "Time:".
- The second line consists of the sender's name following after the string "Name:".
- The third line consists of the message text itself, following the string "Message:".
- The third line consists of the message text itself, following the string "Message:".
- The line containing the message should be followed by an empty line.

The application requires the separation of these elements into JavaFX components. The main conditions are :
- The time must be enclosed in quotation marks : so we should see an explicit separation of time from the rest of the text.
- The name should also be separated with a color scheme from the rest of the message.
- The name should not be displayed if the previous message belonged to the same person.
- All char emoticons should be replaced with gif images related to them.
- Each message must follow the same order of separation of its components.
- Opening a file at the click of a button.

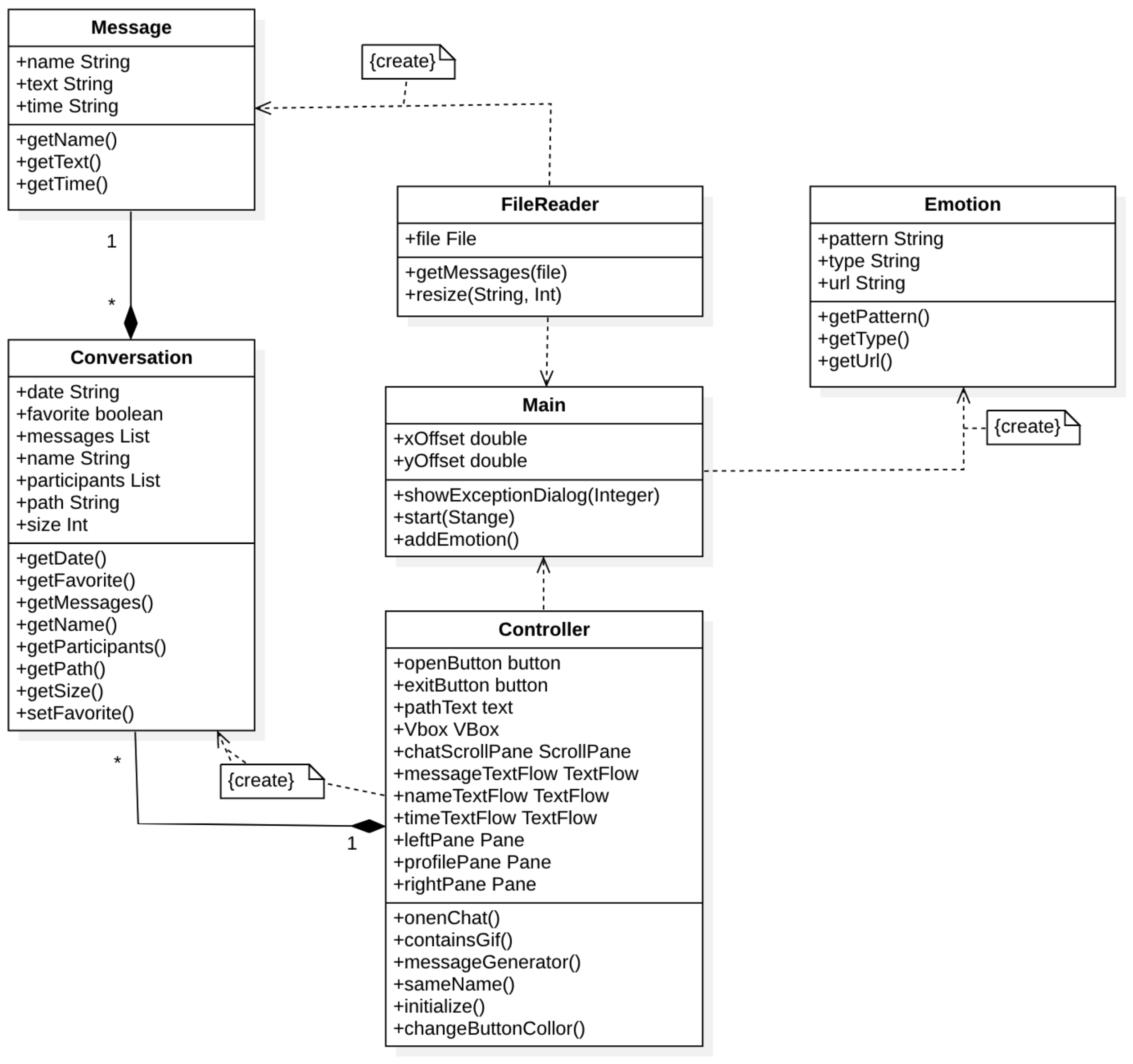
Therefore, it is necessary to make an application showing all the elements of the file in separated JavaFX components. The project should be built on a model that divides the dependency between graphics and the internal logic of the application. So, in case of changes in the file parsing method, this should not affect the work of the graphical part of the project.

The graphical interface should be reasonable, that is, understandable and user-friendly. Possible improvements are also welcome, whether internal, such as the ability to open files of a different type, or graphical, such as displaying recently opened files.

3 UML Class Diagram

After analyzing the scope, I identified the following class diagram model (look at Figure one). The logic of the project will consist in the complete separation of the graphical part and the parsing part. The file will be read in a class FileReader, independent of the Main or Controller class. In the Main class, the necessary methods will be initialized, and in the Controller class, methods related to user actions in the application itself will be executed.

Figure 1: Initial UML Class Diagram of the project



In the subsequent progress of the project, this diagram can be improved depending on the need for changes, however, the overall structure will be the same for most classes.

4 Description of components

The application will consist of a single window, which in turn will be divided into two sides: right and left, which will be presented in the form of boxes.

The right side is a vertical box, which will consist of two main components that will be generated later : a scroll pane that will display messages and an upper Pane panel that will contain the path to the file and button with option to add the current chat to one specific chat folder.

The left panel will be presented in the form of a horizontal box, with a Pane panel displaying recently opened files and a vertical box with a button for opening a file, button for creating folders with chats, a button for opening a specific folder, button for switching the application theme (light or dark), as well as an exit button.

On open button click, a dialog window will open with a file selection. When the file is selected, it will go through parsing and splitting into parts. So, an object of the Message class will be created, with the parameters of the text, name and time. Next to the name there will be a profile circle containing the first letter of the name, followed by the text of the message, and then the time. If this particular routine was chosen, it is only because it meets the requirements of the obvious separation of these three components and is the most natural for the user, since it complies with the current standards of chat display. Each message object will pass such a separation, and then it will be placed in a vertical box, which is located in the scroll pane.

The folder creation button will open a dialog window asking the user to write a name for the folder. This folder will be located in the internal list in the project.
Clicking on the open folder will open another dialog box with all folders, the folder selection will show the files present in the same window. Double-clicking on the file will open the selected conversation.

The last two buttons are obvious to understand,clicking on the theme change button will replace the dark colors of the components with light ones, and pressing the exit button will shut down the application.