ASSIGNMENT COVERSHEET

PRAGUECITY UNIVERSITY

Student Name: Daniel Shutov	
Class: Java Programming (FT/BL)	
Assignment: From Design to Implementation and Evaluation using Java technology	
Lecturer: Viktor Černý	Semester: 2201
Due Date: 9th June, 2023	Actual Submission Date: Submission date

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: May 31, 2023	Student Signature: Daniel Shutov

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?

Chat Viewer

Daniel Shutov

May 31, 2023

Contents

1 List of symbols and list of acronyms **3**

2 Introduction **4**

3 Analysis and design **5**

3.1 Problem analysis 5

3.2 Object-oriented solution 6

4 Implementation **8**

4.1 From design to implamentation 10

5 Testing **11**

6 User Documentation **12**

6.1 Introduction 12

6.2 System Requirements 12

6.3 Installation 12

6.4 Usage 12

6.5 Limitations 12

7 Project management **13**

7.1 Scope 13

7.2 Constraints 14

7.3 Project Management Tools 15

7.4 Adapting to Changes 15

8 Suggested improvements **16**

9 List of figures **17**

10 Self-reflection **18**

References **19**

List of Figures

1 UML proposed design 7
2 UML design 10
3 Reading file 13
4 Error 13
5 Main screen 14
6 Displaying file 14

1 List of symbols and list of acronyms

Class: As javatpoint.com (2021) defined: "In object-oriented programming, a class is a basic building block. It can be defined as a template that describes the data and behavior associated with the class instantiation. Instantiating is a class is to create an object (variable) of that class that can be used to access the member variables and methods of the class. A class can also be called a logical template to create the objects that share common properties and methods".
Method: As w3schools.com (2023) defined: "A method is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as functions".
JavaFX: As Pawlan (2013) defined: "JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms".
UI: As interaction-design.org (2021) defined: "User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable.".
Access modifier: As Team (2022) defined: "Access modifiers are keywords that can be used to control the visibility of fields, methods, and constructors in a class. The four access modifiers in Java are public, protected, default, and private."

2 Introduction

This technical report will examine the problem assigned for the ICA course "From Design to Implementation and Evaluation Using Java Technology." The report must contain a problem analysis, project scope, UML class design, implementation explanation, testing explanation, and possible enhancements.
The assignment was completed only by using the previous knowledge that was gained during the classes.

3 Analysis and design

3.1 Problem analysis

The objective of this task is to design and implement an independent graphical application titled "Chat Viewer." The application is responsible for displaying a stored messaging conversation in a particular file format. The application's user interface should display conversation messages in a formatted manner. The following entities are identified as crucial to the issue:

1. ChatMessage: Represents a single message in a chat conversation. It should contain information such as the sender, receiver, timestamp, and the message content.

2. ChatConversation: Represents the entire conversation between two or more individuals. It should contain a collection of ChatMessages.

3. ChatFileReader: Responsible for reading the chat conversation file and returning a ChatConversation object.

4. ChatViewerGUI: The main user interface of the application. It should contain a display area for the ChatConversation and controls for opening, closing and saving chat files.

5. ChatViewerController: Acts as a bridge between the ChatViewerGUI and the ChatConversation. It should handle user input and update the ChatViewerGUI with the ChatConversation data.

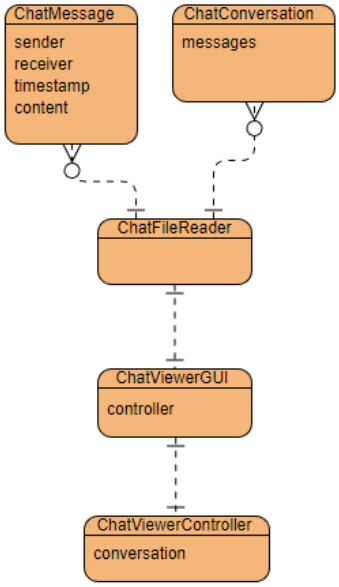
3.2 Object-oriented solution

The object-oriented Chat Viewer application solution proposed is based on the identified entities. The following UML class diagram describes the interactions and relationships between these entities:

- The ChatMessage class represents individual conversational communications. It includes sender, recipient, timestamp, and content as attributes. It is not directly related to any other classifications.
- The ChatConversation class represents a conversation's compilation of ChatMessages. It is composed of one or more ChatMessages, so it has a composition relationship with ChatMessage. The ChatConversation class has a relationship with the ChatFileReader class in order to read the conversation from a file.
- The ChatFileReader class reads the conversation file and returns an instance of the ChatConversation class. It is not directly related to any other classifications.
- ChatViewerGUI class represents the user interface of the application. As it contains an instance of the ChatViewerController class, it has a composition relationship with that class. The ChatViewerGUI class has a relationship with ChatConversation in order to display chat communications.
- The ChatViewerController class serves as a link between ChatViewerGUI and ChatConversation. It processes input from the user and updates the ChatViewer GUI with ChatConversation data. ChatViewerController is related to ChatConversation in order to access the chat communications.

The proposed solution is designed to be scalable and flexible, allowing for future enhancements and modifications to be easily incorporated.

Figure 1: UML proposed design



4 Implementation

First, implementation and analysis are distinct; it was much simpler to implement the system differently than was described in the "Analysis and design" section. Here are the three files that constitute the basis of this project:

1. MainScene.fxml - This file contains the architecture and design of the application's graphical user interface (GUI). It defines the structure of user-interactable UI elements such as buttons, navigation panes, and grids. It is written in FXML, a declarative language that enables the programmer to define the GUI using a hierarchical structure (FXML is an XML-based markup language for designing JavaFX application user interfaces).

2. MainSceneController.java - This controller class coordinates the interaction between the user interface defined in the "MainScene.fxml" file and the application's backend logic. The "MainScene.fxml" file contains the user interface elements displayed to the user, such as controls and scroll bars.

   Using the @FXML annotation, the "MainSceneController" class defines several methods that are associated with the numerous UI components in the "MainScene.fxml" file. When the user interacts with the corresponding UI component, these methods are automatically executed.

   Using the "onMouseClicked" attribute, the "GetText" method is associated with the "theopenbtn" button in the "MainScene.fxml" file. This method is invoked when the "theopenbtn" icon is clicked. This method retrieves user-entered text from the text field and displays it in the "gridpane_inner" GridPane.

   Here's how the GetText method works:

   (a) It retrieves the text entered by the user in the "input_text" "TextField" using the getText method.

   (b) It creates a new Label object with the retrieved text and adds it to the "gridpane_inner" GridPane using the add method.

   (c) It clears the "input_text" "TextField" using the clear method.

   There are additional methods within the "MainSceneController" class that manage various UI events, such as button presses, mouse events, etc. All of these methods update the UI

elements dependent on the actions of the user.

The "MainSceneController" class initialize method is a special method that is invoked when the UI components are imported. This method configures the initial state of the UI components, such as loading data from a database or establishing the initial text of a label.

The "MainSceneController" class functions as a bridge between the application's user interface and it is backend logic. It manages user events and modifies UI components in response to user actions, making the application interactive and responsive.

3. App.java - This file contains one's application's main method, which serves as the entrance point for the JavaFX application. It creates an instance of the "MainScene.fxml" file, configures a scene and stage (a JavaFX container at the upper level), and displays it to the user.

Now, let's delve deeply into the application's implementation:

1. The "MainScene.fxml" file is fetched when the App.java class's "start()" method is invoked. This file defines the application's layout, which consists of a "GridPane" layout containing a Button and a "ScrollPane". The Button is used to access a file, while the "ScrollPane" displays the file's contents.

2. The controller for the "MainScene.fxml" file is the MainSceneController.java file. When "MainScene.fxml" is imported, it is "initialize()" method is automatically invoked. This method initializes "ScrollPane's" underlying "GridPane" layout.

3. When a user clicks on a Button defined in the "MainScene.fxml" file, the "GetText()" method in the "MainSceneController.java" file is invoked. This method opens the file selector dialog, allowing the user to select a file to access. Once a file is selected, the file's contents are read and displayed in the "ScrollPane's" "GridPane" layout.

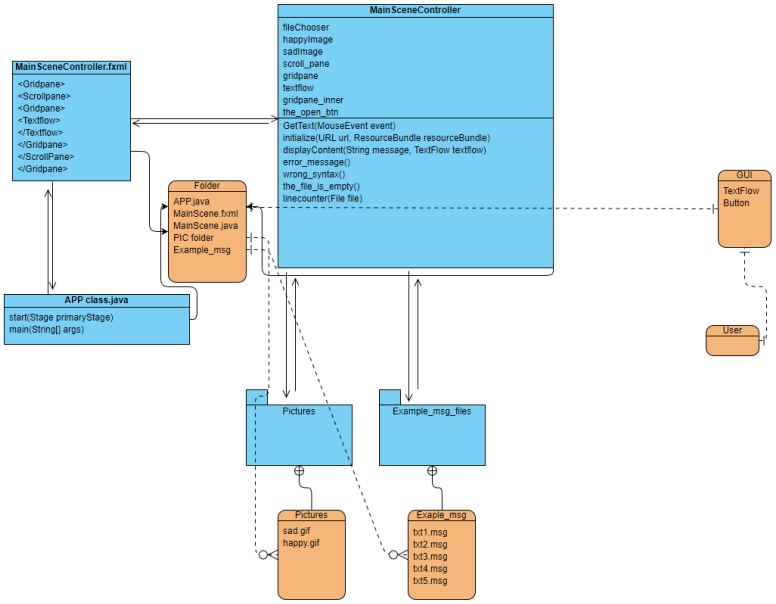
4. The "App.java" file is the application's entrance point. Using the "FXMLLoader" class, it creates an instance of the "MainScene.fxml" file and configures the GUI scene and stage. The scene is then assigned as the stage's root, and the stage is presented to the user.

The JavaFX application is implemented by defining the UI layout and design in the "MainScene.fxml" file, implementing the logic and functionality in the "MainSceneController.java" file,

and creating an instance of the UI and displaying it in the "App.java" file.

To run the application, you would first need to compile the App.java file to generate an "App.class" file. Then, you can run the application by executing the App class with the following command: **java APP** or it is possible to run it using any IDE that the user would like.
This will activate the application and present the user interface. Figure 2 is a more pictorial illustration.

Figure 2: UML design



4.1 From design to implamentation

Differences between the original design and its execution. The original Chat Viewer application design included additional classes and features, such as a “ChatMessage” class with additional attributes, a “ChatViewerController” class that would handle user input and update the display area, and a Save feature that would allow users to save the chat conversation to a file. Nonetheless, it was determined during the implementation phase that these features were superfluous for the project’s scope, and so they were eliminated. The final implementation consisted of only the classes and features necessary for a simple yet functional conversation viewer application.

5 Testing

To ensure a high-quality user experience, the author of this technical report opted for a comprehensive phase of manual testing. By addressing all extreme cases, the author ensured that input errors will not occur. In the unlikely event that an error does occur, the program will display a plain and informative error message to prompt the user to choose a different file.

The phase of testing included the manual evaluation of five distinct files representing a diversity of use cases. These evaluations were conducted to ensure optimal program performance and user satisfaction:

- Verification that there are no double "new lines" between messages, ensuring that the chat messages are displayed correctly.
- Confirmation that the message format is correct, in the form of [TIME] Name : Message, with any deviations resulting in an error message.
- Validation of the font, size, and color, to ensure that the chat display is visually appealing and easy to read.
- Testing for cases where multiple messages are sent by the same user, ensuring that the chat display is easy to follow and understand.
- Verification that the file extension is ".msg" and nothing else, to prevent users from selecting an incompatible file format.
- Evaluation of the UI to ensure that it is easy to use and responsive, promoting an enjoyable user experience.

It is important to note that while only two of the five files passed all tests, the three that did not provided valuable insight into how to improve the code and handle edge cases. This thorough testing process will ultimately lead to a better user experience and more reliable program performance.

6 User Documentation

6.1 Introduction

Chat Viewer is a standalone graphical application that enables the viewing of a file-based chat conversation. The application provides a user-friendly interface that formatted displays conversation messages. This document describes how to use the application Conversation Viewer.

6.2 System Requirements

- Windows 7 or later, macOS or Linux.
- Java 8 or later and java SDK

6.3 Installation

- Download the Chat Viewer application from the provided link—https://github.com/wfgemyd/Java.
- Extract the downloaded zip file to a desired location.
- Double-click the “APP.java” file to start the application.

6.4 Usage

- Launch the Chat Viewer application by running the “APP.java”.
- Click on the “Open” to choose a chat conversation file in the supported format.
- The chat messages will be displayed in the main window of the application.
- one could scroll through the chat messages using the scroll bar on the right side of the window.
- To close the application, click on the “Exit” or “X” in the top right corner.

6.5 Limitations

- The application supports only chat conversation files in the specified format - ".msg".
- The application does not provide any editing or saving options for the chat conversation.

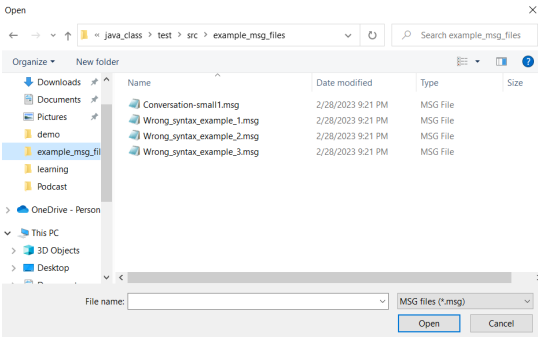
7 Project management

7.1 Scope

The objective of the project is to create an independent graphical application called "Chat Viewer" that displays a chat conversation stored in a particular file format. The user interface of the application should present conversation messages in a formatted manner. The project's specifications include:

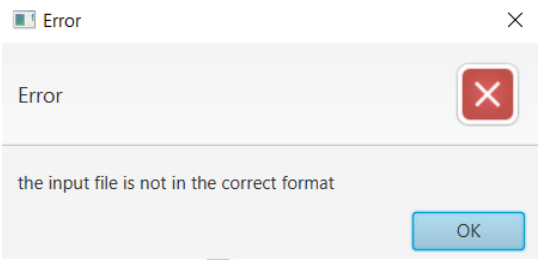
- The application should read chat conversation data from a file formatted in a particular manner.

Figure 3: Reading file



- The application should format the conversation messages before displaying them.

Figure 4: Error



- The application must include buttons for accessing and closing chat files.
- The application must permit the user to navigate the conversation history.
- The application should be scalable and adaptable to accommodate future modifications and enhancements.

Figure 5: Main screen

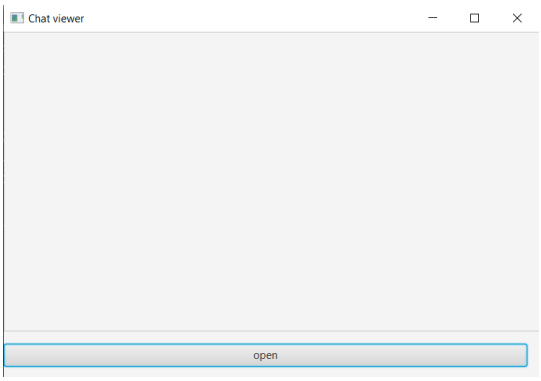
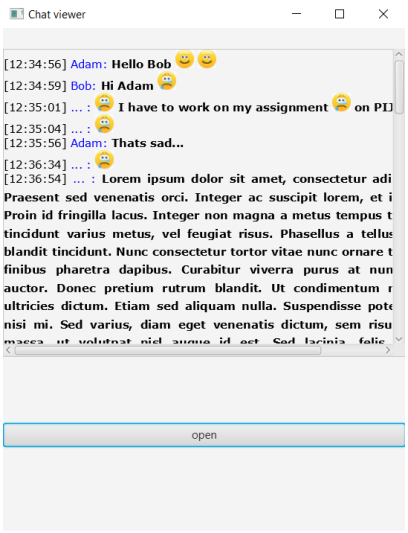


Figure 6: Displaying file



7.2 Constraints

Several constraints must be considered during the development of the Conversation Viewer application, including:

- Java must be used to develop the application.
- The application must utilize the JavaFX library for graphical user interfaces.
- The application must be compatible with Microsoft Windows, Apple macOS, and Linux.
- When viewing chat data, the application must adhere to specific file format requirements.
- The application should be developed within a predetermined timeframe – seven weeks.

7.3 Project Management Tools

To effectively administer the Chat Viewer project, several project management tools can be utilized, including:

- A document that defines the scope, objectives, deliverables, and budget of a project.
- ICA guidelines.
- Time tracker.
- To-do lists with task manager.
- Putting the project on GitHub for future use and community enhancement - https://github.com/wfgemy
  .

7.4 Adapting to Changes

Adaptability is an essential aspect of project administration. Changes may occur in the ChatViewer project due to new requirements, technical issues, or external factors. The following actions can be taken to acclimate to change:

- Conduct regular status meetings (classes) to review project progress, identify issues, and discuss potential changes.
- Inform project stakeholders (teacher) of changes and adjust expectations accordingly.
- By retesting the software with new potential chats, it may be possible to discover new flaws.

8 Suggested improvements

There are some improvements that can be done:

- Adding a settings option that enables the user to customize the app to their liking, such as changing the text font, size, background, etc.
- Adding more functionality and interactivity, including sound effects, animations, and additional game modes to enhance the user experience. Additionally, one could add more icons or controls to the GUI to provide end users with more customization options.
- Making the program a genuine live chat in which two or more individuals could converse in real time, with the option to save the conversations if desired.
- Additional emoticons for the user to select.
- Refactoring the code to be more object-oriented: Object-oriented programming can make the code more modular, easier to understand, and simpler to maintain. This can also make it easier to add new functionality in the future.

9 List of figures

- Figure 1: UML proposed design (Created the author (2023)).
- Figure 2: UML design (Created by author (2023)).
- Figure 3: Reading file (Created by author (2023)).
- Figure 4: Error (Created by author (2023)).
- Figure 5: Main screen (Created by author (2023)).
- Figure 6: Displaying file (Created by author (2023)).

10 Self-reflection

Planning, organization, communication, and problem-solving abilities are required for effective project management. To ensure the success of a project, it is necessary to monitor progress, identify problems, and adapt to change. This project required me to construct a timeline for both the code and the paper. One week was allotted for the code and four days for the paper. While constructing the application step by step, I encountered numerous errors and crashes throughout the duration of the project. While occasionally exasperating, these obstacles taught me valuable lessons about working with Java as a programming language. Although I worked independently on this project, I could foresee enjoying it more as a team effort, particularly if its scope were larger. However, because I do not anticipate using Java in the future, I did not devote an excessive amount of time to the endeavor. Reflecting on the project's performance throughout its duration and after its conclusion, I realized there are opportunities for advancement. This understanding will inform my approach to managing future projects. By employing the project management skills I have acquired, I can continue to hone my abilities and ensure the success of future endeavors.

References

interaction-design.org (June (2021)). “UI”. en. In: URL: https://www.interaction-design.org/literature/topics/ui-design.

javatpoint.com (June (2021)). “Class definition in java”. en. In: URL: https://www.javatpoint.com/class-definition-in-java.

Pawlan, Monica ((2013)). “JavaFX”. en. In: URL: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm.

Team, Great Learning (Nov. (2022)). “Access modifiers”. en. In: URL: https://www.mygreatlearning.com/blog/the-access-modifiers-in-java.

w3schools.com ((2023)). “Java methods”. en. In: URL: https://www.w3schools.com/java/java_methods.asp.