Yabi Gruga and Dexter Kong
May 19, 2023
IS 413

In our project, JavaFX is used to create a simple phone contact organizer. Users can display name, address, and email as well as edit all of the columns in the application, as well as add and delete entries. Along with splitting columns for primary and secondary emails, we will also add another column for email. A minimum of two different style sheets will be available, and the application will have persistent storage. In addition to maintaining the bare minimum of functionality, our goal is to design a user-friendly and aesthetically pleasing interface.
The Model Class:

In our project, a simple phone contact organizer is made using JavaFX. Users can add and delete entries, edit all the columns in the application, and display name, address, and email in addition to displaying them. We will divide columns for primary and backup emails and add an additional column for email. The application will support persistent storage and at least two different style sheets. Our objective is to create an interface that is both user-friendly and aesthetically pleasing while maintaining the bare minimum of functionality.

Model and View Classes:

In the development of the project application, Yabi primarily focused on the implementation of the Model and View classes. The Model class encapsulates the data management logic, providing methods for saving, updating, and deleting DataEntry objects. The dexter dedicated their efforts to designing and implementing this class, ensuring efficient data handling and integration with the application's user interface.

Simultaneously, Yabi also worked on the View class, which constitutes the graphical user interface (GUI) of the application. By creating and arranging various UI components such as labels, text fields, buttons, and a table view, they provided a user-friendly environment for data entry and display.

Controller Class and Data Entry:

Dexter took responsibility for developing the Controller class, a crucial component that acts as a mediator between the Model and View. The Controller class enables communication and coordination between user actions in the GUI and the underlying data management operations in the Model. They implemented event handlers and methods that respond to user interactions, ensuring seamless data flow and synchronization between the View and Model.

Additionally, Dexter worked on the DataEntry class, which represents the individual entries within the application. Collaboratively, both partners defined the structure and attributes of the DataEntry class, ensuring it accurately represents the data being stored and managed. Dexter played a significant role in implementing the necessary fields, constructors, and getter methods to encapsulate the data related to each entry.

Persistent Storage Implementation: Both partners actively collaborated on the implementation of persistent storage, which allows the application to save and retrieve data between sessions. Working together, they decided to utilize file-based storage to achieve this functionality. The Model class was modified to incorporate file operations, enabling the saveDataToFile() method to write the data to a specified file location. Both partners contributed to this implementation, ensuring the data entries were correctly written and retrieved from the file

- The Model class serves as an intermediary between the user interface and the underlying data. It encapsulates the logic and operations related to data management in the application. Let's explore the key components and methods of this class.

Class Variables and ImportsThe code begins with importing necessary packages, including the javafx.collections package for ObservableList functionality. It also defines a class-level constant FILE_PATH, representing the path to the file where the data will be saved.
ObservableList and Initialization:

- The Model class maintains an ObservableList of DataEntry objects, named dataEntries. This list enables real-time data binding with JavaFX UI components, ensuring automatic updates when the data changes. In the constructor, the dataEntries list is instantiated using the FXCollections.observableArrayList() method, initializing an empty list.
- The getDataEntries() method allows external classes or UI components to access the dataEntries list. By returning the ObservableList, other parts of the application can retrieve and manipulate the data stored within the Model.
- The saveDataToFile(DataEntry dataEntry) method is responsible for persisting the provided DataEntry object to a file. It uses the PrintWriter and FileWriter classes to write the data to the specified file (FILE_PATH). Each field of the DataEntry object is written to a new line in the file, preceded by an appropriate label. The "try-with-resources" statement ensures the proper closure of the resources and handles potential IOExceptions that may occur during file writing.

Updating Data Entry:

- The updateDataEntry(DataEntry oldEntry, DataEntry newEntry) method enables updating an existing data entry. It searches for the oldEntry within the dataEntries list using the indexOf() method. If the oldEntry is found, it is replaced with the newEntry object, effectively updating the data. This method ensures the model is kept up-to-date with any changes made to the data.
- The deleteDataEntry(DataEntry dataEntry) method facilitates the removal of a specific data entry from the dataEntries list. It utilizes the remove() method of the ObservableList to eliminate the provided dataEntry object from the list.

View Class:

The View class is responsible for creating and managing the graphical user interface (GUI) elements in the application. It focuses on the visual representation of data and user interactions. Let's delve into the key aspects of the View class:

UI Components:

1. The View class instantiates and configures various UI components, such as labels, text fields, buttons, and a table view. These components provide the visual elements that users interact with, enabling data entry, display, and manipulation.

Layout Organization:

2. The View class arranges the UI components within a layout, typically a VBox (vertical box) or other suitable containers. This organization ensures a structured and visually appealing interface for users. Components are added to the layout in a desired order, defining the flow and positioning of elements.

Configuration and Styling:

3. The View class applies configuration and styling to the UI components to enhance the user experience. This includes setting preferred dimensions, applying CSS styles for aesthetics, and defining event handlers for user interactions.

Accessor Methods:

4. To facilitate interaction with the UI components, the View class provides accessor methods. These methods allow other parts of the application, such as the Controller class, to access and manipulate the UI elements. Examples of such methods include getNameTextField(), getTableView(), and getSaveButton().

Controller Class:

The Controller class acts as the intermediary between the View and Model classes. It handles user interactions, responds to events triggered by the View, and coordinates data flow and manipulation. Let's explore the key aspects of the Controller class:

Event Handling:

1. The Controller class defines event handlers for user interactions with the UI components. These event handlers capture user actions, such as button clicks or text field updates, and respond accordingly. They trigger specific methods in the Controller to update the data and communicate with the Model.

Data Validation and Manipulation:

2. The Controller class is responsible for validating and manipulating the data entered by the user. It ensures that the input adheres to predefined rules or constraints and performs any necessary data transformations or calculations. This validation and manipulation ensure the integrity and accuracy of the data being processed.

Communication with the Model:

3. The Controller class communicates with the Model class to perform data management operations. It invokes methods from the Model to save, update, or delete data entries based on user actions. This interaction ensures that the data stays consistent across the application and the underlying data storage.

Update of the View:

4. Upon changes to the data, the Controller class updates the View to reflect the modified state. It triggers the necessary UI updates, such as refreshing the table view or displaying error messages, to provide real-time feedback to the user.

The View and Controller classes in the project JavaFX application play integral roles in creating a functional and interactive user interface. The View class focuses on UI component creation, layout organization, and visual presentation of data. On the other hand, the Controller class handles event handling, data validation, manipulation, and communication with the Model class. Through their collaboration, these classes ensure a seamless user experience, facilitating data entry, display, and management in the application