ADSA Project Report

On

Calendar

Submitted in partial fulfillment of the

requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

B.Tech

Session 2023-24

in

Computer Science and Engineering

By

Tarique Anwar 22SCSE1010364

Lucky Upadhyay 22SCSE1010334

Pranjal Singh 22SCSE1010333

Under the guidance of

Dr. Vimal Kumar

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

INDIA

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

TABLE OF CONTENTS

1. INTRODUCTION

The JavaFX Calendar Application is a simple yet functional graphical user interface (GUI) application that allows users to view a monthly calendar. The application is built using JavaFX, a framework for creating Java applications with rich visual interfaces. The purpose of this project is to provide a user-friendly tool for displaying and navigating through a calendar.

Features

2.1 Month and Year Selection

- Users can select a specific month and year using ChoiceBoxes.

2.2 Dynamic Calendar Display

- The application dynamically generates a calendar grid based on the selected month and year.
- The calendar grid displays weekdays as columns and days of the month as rows.

2.3 Update Functionality

- Users can update the calendar display by clicking the "Update" button after selecting a new month or year.
- The application responds dynamically to user inputs, providing an updated calendar view.

Technology Stack

3.1 Programming Language

- Java (JDK 8 or later)

3.2 GUI Framework

- JavaFX: A framework for creating cross-platform desktop applications with rich graphical user interfaces.

3.3 Libraries

- Java Standard Libraries: Utilized for various functionalities such as date handling and collections.
- JavaFX Libraries: Used for creating GUI components, layout management, and event handling.

Project Structure

4.1 Main Class

- **Calendar.java:** The main class that extends **Application** and serves as the entry point for the application. Manages the primary stage and the landing page.

4.2 Calendar GUI Components

- **ChoiceBox:** Allows users to select the month and year.
- **Button:** The "Open Calendar" button on the landing page and the "Update" button in the calendar window.
- **TableView and TableColumn:** Used for displaying the calendar in a tabular format.
- **StackPane, HBox, GridPane:** Layout containers for organizing UI elements.

4.3 Calendar Logic

- **LocalDate:** From the Java Date and Time API, used for date-related calculations.
- **ObservableList:** Used for creating dynamic and observable lists for the calendar data.

Code Organization

5.1 Modular Design

- The code is organized into methods for better modularity and readability.
- Clear separation of concerns: Methods handle specific tasks such as UI creation, calendar population, and event handling.

5.2 Comments

- Comments are added throughout the code to explain complex logic, the purpose of variables, and enhance code readability.

Future Enhancements

- **UI Styling:** Implement enhanced styling for a more visually appealing interface.
- **Internationalization (I18N):** Extend the application to support multiple languages and date formats.
- **Reminder Functionality:** Add the ability for users to set and view reminders for specific dates.

2. LIBRARIES

1. javafx.application.Application

- **Purpose:** This class is part of the JavaFX framework and is used as the base class for JavaFX applications. It provides the **start** method that is called when the application is launched.

2. javafx.beans.property.SimpleStringProperty

- **Purpose:** This class is part of the JavaFX Beans API and represents a simple property wrapping a String value. It's commonly used in JavaFX applications for data binding.

3. javafx.collections.FXCollections

- **Purpose:** This class provides utility methods for creating observable collections (e.g., **ObservableList**). Observable collections can be used in JavaFX to automatically update UI elements when the underlying data changes.

4. javafx.geometry.Pos

- **Purpose:** This enumeration represents the position or alignment of a node within its parent. It's used here to set the alignment of UI elements in the **StackPane** and **HBox** to the center.

5. javafx.scene.Scene

- **Purpose:** The **Scene** class is part of JavaFX and represents the content for a stage. It is the container for all graphical content in a JavaFX application.

6. javafx.scene.control.*

- **Purpose:** This package provides a variety of pre-built UI controls that can be used in JavaFX applications. In your code, it includes classes like **Button**, **ChoiceBox**, **TableView**, **TableColumn**, etc., which are used to create buttons, choice boxes, and tables.

7. javafx.scene.layout.*

- **Purpose:** This package provides layout-related classes in JavaFX. In your code, it includes classes like **StackPane**, **HBox**, and **GridPane**, which are used to organize and layout UI elements in a structured way.

8. javafx.stage.Stage

- **Purpose:** Represents the top-level container for a JavaFX application. It is the main window or stage where your application's UI is displayed.

9. java.time.LocalDate

- **Purpose:** Part of the Java Date and Time API, **LocalDate** represents a date without a time component. It's used here to handle date-related operations, such as determining the number of days in a month.

10. java.time.format.TextStyle and java.util.Locale

- **Purpose:** These classes are part of the Java Date and Time API and are used to format and display dates. **TextStyle** is used to specify the full name of a month, and **Locale** is used to determine the locale-specific details for formatting.

11. java.util.*

- **Purpose:** This package provides the basic utility classes. In your code, it's specifically used for the **Locale** class to get locale-specific month names during the initialization of **ChoiceBox** for months.

3. SOURCE CODE

```
package com.example.calendar;

import javafx.application.Application;

import javafx.beans.property.SimpleStringProperty;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*;

import javafx.scene.layout.*;

import javafx.stage.Stage;


import java.time.LocalDate;

import java.time.format.TextStyle;

import java.util.Locale;


public class Calendar extends Application {


    private Button updateButton; // Add the missing variable


    public static void main(String[] args) {

        launch(args);

    }


    @Override

    public void start(Stage primaryStage) {
```

```
        primaryStage.setTitle("Month Calendar");

        // Create a button to open the calendar
        Button openCalendarButton = new Button("Open Calendar");
        openCalendarButton.setOnAction(event -> openCalendar(primaryStage));

        StackPane landingPane = new StackPane(openCalendarButton);
        landingPane.setAlignment(Pos.CENTER);
        Scene landingScene = new Scene(landingPane, 300, 200);

        primaryStage.setScene(landingScene);
        primaryStage.show();
    }

    private void openCalendar(Stage primaryStage) {
        // Create ChoiceBoxes for month and year
        ChoiceBox<String> monthChoiceBox = new ChoiceBox<>();
        for (int i = 1; i <= 12; i++) {
            monthChoiceBox.getItems().add(LocalDate.of(2000, i,
1).getMonth().getDisplayName(TextStyle.FULL, Locale.getDefault()));
        }

monthChoiceBox.setValue(LocalDate.now().getMonth().getDisplayName(TextStyle.FULL
, Locale.getDefault()));

        ChoiceBox<String> yearChoiceBox = new ChoiceBox<>();
        for (int i = 2000; i <= 2100; i++) {
            yearChoiceBox.getItems().add(String.valueOf(i));
        }
        yearChoiceBox.setValue(String.valueOf(LocalDate.now().getYear()));
```

```
        // Create an "Update" button
        updateButton = new Button("Update");

        // Create an HBox for ChoiceBoxes and the "Update" button
        HBox navigationBar = new HBox(10);
        navigationBar.setAlignment(Pos.CENTER);
        navigationBar.getChildren().addAll(monthChoiceBox, yearChoiceBox,
updateButton);

        // Create a GridPane to hold the navigation bar and the calendar
        GridPane calendarPane = new GridPane();
        calendarPane.setAlignment(Pos.CENTER);
        calendarPane.setVgap(10);

        // Add the navigation bar to the GridPane
        calendarPane.add(navigationBar, 0, 0);

        // Create the calendar table
        TableView<ObservableList<String>> calendarTable =
createCalendarTable(LocalDate.now(), monthChoiceBox, yearChoiceBox);
        calendarPane.add(calendarTable, 0, 1);

        Scene calendarScene = new Scene(calendarPane, 600, 400);
        Stage calendarStage = new Stage();
        calendarStage.setTitle("Calendar");
        calendarStage.setScene(calendarScene);

        calendarStage.show();
```

```
    }


    private TableView<ObservableList<String>> createCalendarTable(LocalDate
currentDate, ChoiceBox<String> monthChoiceBox, ChoiceBox<String> yearChoiceBox)
{

        TableView<ObservableList<String>> calendarTable = new TableView<>();

        calendarTable.setEditable(false);


        // Add column headers (Weekdays)

        String[] weekdays = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

        for (int i = 0; i < weekdays.length; i++) {

            final int columnIndex = i;

            TableColumn<ObservableList<String>, String> column = new
TableColumn<>(weekdays[i]);

            column.setCellValueFactory(param -> new
SimpleStringProperty(param.getValue().get(columnIndex)));

            calendarTable.getColumns().add(column);

        }


        // Add rows (Week numbers and days)

        int daysInMonth = currentDate.lengthOfMonth();

        int dayOfWeek = currentDate.withDayOfMonth(1).getDayOfWeek().getValue();
// 1 = Monday, 7 = Sunday

        int currentDay = 1;


        for (int i = 1; i <= 5; i++) {

            ObservableList<String> row = FXCollections.observableArrayList();


            for (int j = 1; j <= 7; j++) {

                if (currentDay <= daysInMonth || (i == 1 && j < dayOfWeek)) {
```

```
                    if (i == 1 && j < dayOfWeek) {
                        // Add empty cells for days before the first day of the
month
                        row.add("");
                    } else {
                        // Add days of the month
                        row.add(String.valueOf(currentDay));
                        currentDay++;
                    }
                } else {
                    // Add empty cells for days after the last day of the month
                    row.add("");
                }
            }

            calendarTable.getItems().add(row);
        }

        // Update ChoiceBoxes when the "Update" button is clicked
        updateButton.setOnAction(event -> {
            int selectedMonth =
monthChoiceBox.getSelectionModel().getSelectedIndex() + 1;
            int selectedYear = Integer.parseInt(yearChoiceBox.getValue());
            updateCalendarTable(LocalDate.of(selectedYear, selectedMonth, 1),
calendarTable, monthChoiceBox, yearChoiceBox);
        });

        return calendarTable;
    }
```

```
    private void updateCalendarTable(LocalDate currentDate,
TableView<ObservableList<String>> calendarTable, ChoiceBox<String>
monthChoiceBox, ChoiceBox<String> yearChoiceBox) {

        // Clear the existing table

        calendarTable.getItems().clear();

        calendarTable.getColumns().clear();



        // Call createCalendarTable with the updated date

        createCalendarTable(currentDate, monthChoiceBox, yearChoiceBox,
calendarTable);

    }



    private void createCalendarTable(LocalDate currentDate, ChoiceBox<String>
monthChoiceBox, ChoiceBox<String> yearChoiceBox,
TableView<ObservableList<String>> calendarTable) {

        // Add column headers (Weekdays)

        String[] weekdays = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

        for (int i = 0; i < weekdays.length; i++) {

            final int columnIndex = i;

            TableColumn<ObservableList<String>, String> column = new
TableColumn<>(weekdays[i]);

            column.setCellValueFactory(param -> new
SimpleStringProperty(param.getValue().get(columnIndex)));

            calendarTable.getColumns().add(column);

        }

        // Add rows (Week numbers and days)

        int daysInMonth = currentDate.lengthOfMonth();

        int dayOfWeek = currentDate.withDayOfMonth(1).getDayOfWeek().getValue();
// 1 = Monday, 7 = Sunday

        int currentDay = 1;



        for (int i = 1; i <= 5; i++) {
```

```
            ObservableList<String> row = FXCollections.observableArrayList();
            for (int j = 1; j <= 7; j++) {
                if (currentDay <= daysInMonth || (i == 1 && j < dayOfWeek)) {
                    if (i == 1 && j < dayOfWeek) {
                        // Add empty cells for days before the first day of the
month
                        row.add("");
                    } else {
                        // Add days of the month
                        row.add(String.valueOf(currentDay));
                        currentDay++;
                    }
                } else {
                    // Add empty cells for days after the last day of the month
                    row.add("");
                }
            }
            calendarTable.getItems().add(row);
        }
        // Update ChoiceBoxes when the "Update" button is clicked
        updateButton.setOnAction(event -> {
            int selectedMonth =
monthChoiceBox.getSelectionModel().getSelectedIndex() + 1;
            int selectedYear = Integer.parseInt(yearChoiceBox.getValue());
            updateCalendarTable(LocalDate.of(selectedYear, selectedMonth, 1),
calendarTable, monthChoiceBox, yearChoiceBox);
        });
    }
}
```

4. PROJECT EXPLANANTION

1. Calendar Class

- **Purpose:** The main class that extends **Application** and serves as the entry point for the JavaFX application.
- **Key Components:**
  - **updateButton**: A button used to trigger the update of the calendar when clicked.

2. main Method

- **Purpose:** Launches the JavaFX application by calling the **launch** method.
- **Key Components:**
  - **launch(args)**: Initiates the JavaFX application.

3. start Method

- **Purpose:** Initializes the primary stage with a button to open the calendar.
- **Key Components:**
  - **openCalendarButton**: A button that, when clicked, opens the calendar.
  - **landingPane**: A stack pane containing the open calendar button.
  - **landingScene**: The scene for the landing page.

4. openCalendar Method

- **Purpose:** Creates and displays the calendar window with month and year selection.
- **Key Components:**
  - **monthChoiceBox** and **yearChoiceBox**: ChoiceBoxes for selecting the month and year.
  - **updateButton**: Button to update the calendar.
  - **navigationBar**: An HBox containing the choice boxes and the update button.
  - **calendarPane**: A GridPane to hold the navigation bar and the calendar table.
  - **calendarTable**: TableView to display the calendar.
  - **calendarScene**: The scene for the calendar window.

5. createCalendarTable Method

- **Purpose:** Creates the calendar table with columns for weekdays and rows for days.
- **Key Components:**
  - **weekdays**: Array of weekday names.
  - **calendarTable**: TableView for displaying the calendar.
  - **currentDate**: The current date used to determine the days in the month.

- **updateButton**: Configured to update the calendar when clicked.

6. updateCalendarTable Method

- **Purpose:** Clears the existing calendar table and updates it with a new date.
- **Key Components:**
  - **calendarTable**: The TableView to be updated.
  - **currentDate**: The new date for updating the calendar.

7. createCalendarTable (Overloaded) Method

- **Purpose:** Creates the calendar table with columns for weekdays and rows for days (used for updating).
- **Key Components:**
  - Similar to the **createCalendarTable** method without overloading.

8. TableView and TableColumn

- **Purpose:** Displaying the calendar in a table format with columns for weekdays and rows for days.
- **Key Components:**
  - **calendarTable**: The main table to display the calendar.
  - **column**: TableColumn instances representing individual weekdays.

9. Date Calculations

- **Purpose:** Determines the number of days in a month, the day of the week for the first day, and populates the calendar.
- **Key Components:**
  - **daysInMonth**: Calculated based on the length of the current month.
  - **dayOfWeek**: Determines the day of the week for the first day of the month.
  - **currentDay**: Tracks the day being processed during calendar population.

10. Event Handling

- **Purpose:** Handles events such as button clicks to update the calendar.
- **Key Components:**
  - **updateButton.setOnAction**: Defines the action to be taken when the update button is clicked.

11. Comments

- **Purpose:** Comments are used to explain complex logic, the purpose of variables, and to enhance code readability.
- **Key Components:**

- Comments have been added throughout the code to provide explanations.

12. General Recommendations

- **Purpose:** Suggestions for potential improvements such as variable naming, UI enhancements, and code organization.
- **Key Components:**
  - Recommendations for improving code quality and user experience.

5. ALGORITHMS & DATA STRUCTURES

The JavaFX Calendar Application involves user interface (UI) design and date-related operations, and while it doesn't involve complex algorithms or intricate data structures, there are certain aspects worth mentioning:

Algorithms:

1. **Date Calculations:**

- **Purpose:** Calculating the number of days in a month, determining the day of the week for the first day of the month, and populating the calendar.
- **Algorithm:** Utilizes the **java.time.LocalDate** API to perform date-related calculations. For example, the length of the month is obtained using **currentDate.lengthOfMonth()**, and the day of the week for the first day of the month is determined using **currentDate.withDayOfMonth(1).getDayOfWeek().getValue()**.

2. **Update Calendar Table:**

- **Purpose:** Updating the calendar table when the user selects a new month or year.
- **Algorithm:** The existing table is cleared, and the **createCalendarTable** method is called with the updated date. This ensures that the calendar is re-generated based on the user's selection.

Data Structures:

1. **ObservableList:**

- **Purpose:** Used to create dynamic and observable lists for the calendar data.
- **Usage:** Each row in the calendar table is represented by an **ObservableList<String>**. The **TableView** is populated with these observable lists, allowing for automatic updates in the UI when the underlying data changes.

2. **TableView and TableColumn:**

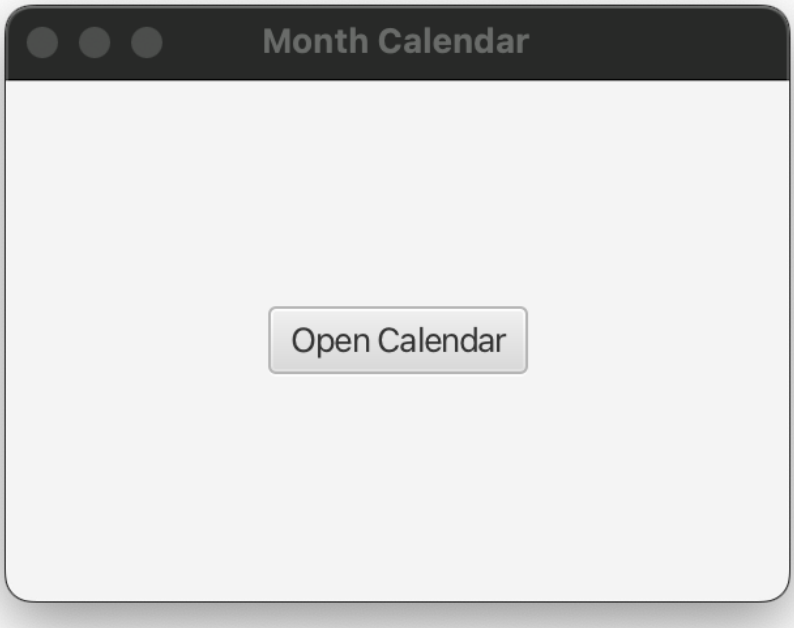
- **Purpose:** Representing the calendar in a tabular format.
- **Usage:** The **TableView** is the main container for the calendar, and each day is represented by a **TableColumn**. This structure allows for the organization and display of data in a grid format.

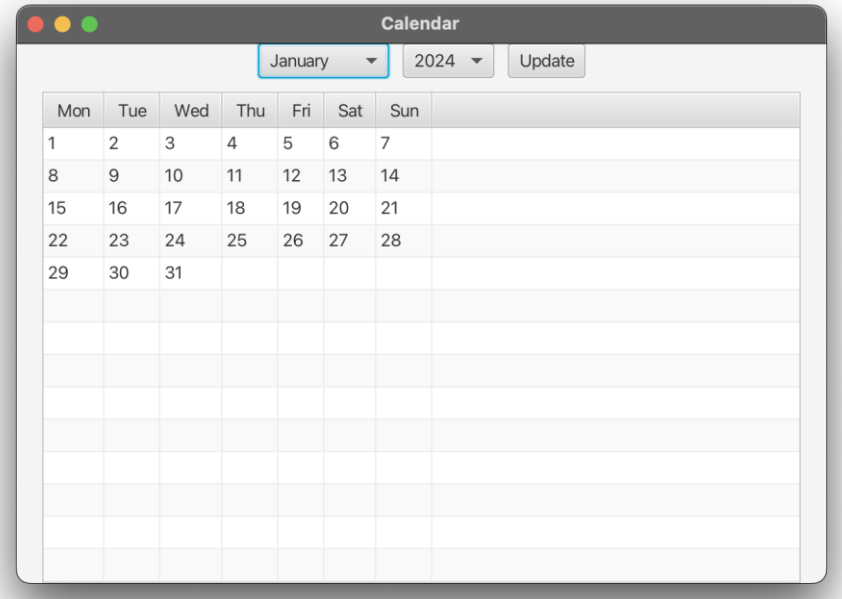
Event Handling:

1. **Button Click Events:**

- **Purpose:** Handling events when buttons are clicked (e.g., opening the calendar or updating it).
- **Usage:** Utilizes the **setOnAction** method to define the actions to be taken when the buttons are clicked. For example, clicking the "Update" button triggers the recalculation and display of the calendar with the new date.

6. OUTPUT


Month Calendar
Open Calendar


Calendar
January
2024
Update
Mon Tue Wed Thu Fri Sat Sun
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31