# Final Report

## Planner Application

Created by Cash Hollister, David Smith, Matt Santoro, Andrew Coggins

# Functional Specification

## Application Description:

The application is a daily planner/calendar utilized to add, remove, edit, and organize events assigned by a registered user. A user is required to create an account with the application. The application has a main interface that allows users to track the events of the current day in detail and also view the title of various events occurring in the current week. Users will have the ability to view a larger calendar containing the events of the current month. Users can view and change their username and password through the user profile interface. Events will have a variety of potential inputs giving the user the ability to control the level of detail involved in creating an event.

## 1. User Authentication Function:

**1.1 Log-in:** The user logs in utilizing a username and password.

**1.2 Create Account:**The user creates an account by providing username, password, and password-confirmation.The system verifies that the username created by the user is unique and that both the password and password-confirmation match. If there is an error the user is notified.

## 2. Main Interface Function:

**2.1 Weekly Calendar:** The system displays a sub calendar of the current week containing the titles of "events" assigned to specific days of the current week. Events are selected to display event details. Completed "events" are green and pending "events are blue.

**2.2 Daily Planner:** The system displays a detailed list of "events" assigned to the current day with a description, time, priority level, course, and title. Completed "events" are green and pending "events are blue.

## 3. Add Events Function:

The system displays a combination of text inputs and list boxes for the user to construct an "event" entry:

- Title: A title for the event ( text input )
- Date: The day and month of the "event" ( list box )
- Time: The time the event is occurring that day ( list box )
- Course: The course associated with the event ( text input )
- Description: A brief description of the "event" (text input )
- Priority Level: A level between low, moderate, high ( list box )
- Status: The status of the event such as "Not Started", "In progress", "Complete", ect.

## 4. Event Details Function:

**4.1 Event Details:** The system displays a window containing the details assigned to the "event" selected by the user. The details are assigned by the user at the creation of the event (title, date, time, course, description, and priority level)

**4.2 Edit Event:** The system allows the user to edit the event details from the event specific details window.

**4.3 Remove Event:** The system allows the user to remove an event utilizing the event specific details window.

**4.4 Mark as Complete:** The system allows the user to mark the event as complete from the event specific details window.

**4.4 Reminders:** Each event shall have an option to enable/disable reminders. Users can choose their preferred reminder type for each event and customize the reminder time interval for each event.

## 5. Calendar Function:

The system displays the calendar of the current month. The title of each “event” is displayed on the day it takes place. Event titles are selected to view event details regarding the event. Completed “events” are green and pending “events are blue.

## 6. User Profile Function:

**6.1** The system displays the user’s username and password.

**6.2** The user can delete their account via the user profile page.

## 7. Navigation Function:

**7.1 Task Bar:** The system displays a task bar that allows the user to navigate the planner application utilizing a variety of buttons.

**Buttons:**

- Home: Routes the user to the **Main Interface**.
- User Profile: Routes the user to a page to alter data pertaining to the user.
- Add Events: Routes the user to a page to create calendar “events”.
- Calendar: Routes the user to a page that displays the events of the current month.

## Glossary:

**Event**: A scheduled activity or occurrence, such as a meeting, appointment, class, or task, that the user wants to track and manage within the calendar application.

**Completed Event**: An event that has already taken place or has been marked as finished by the user.

**Pending Event**: An event that is scheduled to occur in the future and has not yet been completed.

**User Profile**: A section within the application where the user can view and manage their personal information, such as username, password, and other preferences.

**Task Bar**: A persistent or contextually appearing navigation element within the application's interface that provides quick access to key functionalities or views, such as the Main Interface, User Profile, Add Events page, and Calendar view.

**Main Interface**: The primary screen or dashboard of the application, providing an overview of the user's schedule, typically including a daily planner and a weekly calendar.

**Daily Planner**: A view within the application that displays the events scheduled for the current day in detail, including their descriptions, times, priority levels, courses, and titles.

**Weekly Calendar**: A view that shows the events scheduled for the current week, typically displaying only the titles of the events on their respective days.

**Monthly Calendar**: A view that presents the entire month's schedule, allowing the user to see the distribution of events throughout the month.

**Reminders**: Notifications sent to the user to alert them of upcoming events or tasks. These can be delivered through various channels, such as email or push notifications.

# Use Cases:

## 0. Initiate Planner Application:

1. The user locates the application file
2. The user double clicks the application file or icon

## 1.1 Log-in:

1. The user carries out **Initiate Planner Application**.
2. The user inputs the username and password associated with their account.
3. The user clicks the “Log-in” button.
4. The application routes the user to the Main Interface.

Variant #1: Incorrect login details.

1.1 Start at Step 2.
1.2 The user enters an incorrect username or password.
1.3 The application displays an error message: "Invalid login details."
1.4 Continue with Step 2 for a retry.

## 1.2 Create Account:

1. The user carries out **Initiate Planner Application**.
2. The user selects “Create Account”.
3. The application routes the user to the Create Account Interface and prompts the user for the following information

   “Username:
   Password:
4. The user enters the requested information and clicks the “create account” button.
5. The application routes the user to the User Authentication Interface.

## 2. View Main Interface:

1. The user carries out **Log-in.**

## 3. Add Event:

1. The user logs into the application.
2. The user clicks the “Add Event” button.
3. The application prompts the user to input event details such as:
    a. Title
    b. Date
    c. Time
    d. Course
    e. Description
    f. Priority Level
    g. Status
    h. Completion Status
    i. Conflict Status
4. The user fills out the form and clicks “Save Event.”
5. The application adds the event to the selected date and time in the calendar

Variant #1: Missing required fields

1.1 Start at Step 4.
1.2 The user leaves one or more required fields blank (not all fields will be a requirement).
1.3 The application displays a message: "Please complete all required fields."
1.4 The user corrects the fields and clicks "Save Event" again.

Variant #2: Event conflicts with an existing event

2.1 Start at Step 5.
2.2 The user selects a date and time for the new event that conflicts with an existing event (overlapping times).
2.3 The application displays a warning: “Event conflicts with an existing event with the same date and time. Do you wish to continue?”
2.4 The user can either:
- Click “Yes” to proceed with the conflict.
- Click “No” to modify the event time.

2.5 Continue with Step 6.

## 4.1 View Event Details:

1. The user logs into the application.
2. The user navigates to the Calendar Interface.
3. The user selects an event from the calendar (daily, weekly, or calendar view).
4. The application displays the details of the selected event, including:
    a. Event Name
    b. Date
    c. Start Time
    d. End Time
    e. Category
    f. Description
    g. If a reminder is set for the event:
        i. Reminder Type/Time
5. The user clicks “Back” to return to the calendar view.

Variant #1. Edit from event details

1.1 Start at Step 5.
1.2 The user clicks "Edit" instead of "Back."
1.3 The application proceeds with the “Edit Event” use case.

## 4.2 Edit Event:

1. The user logs into the application.
2. The user navigates to the Calendar Interface.
3. The user selects the event to be edited.
4. The user clicks the “Edit Event” button.
5. The application displays the event details.
6. The user edits the desired fields and clicks “Save Changes.”
7. The application updates the event and displays a confirmation message.

Variant #1: Abandon changes

1.1 Start at Step 6.
1.2 The user clicks “Cancel” to save changes.
1.3 The application keeps the event unchanged.

## 4.3 Remove Event:

1. The user logs into the application.
2. The user navigates to the Calendar Interface.

3. The user selects the event to be removed (this can be from a daily, weekly, or calendar view).
4. The user clicks the “Remove Event” button.
5. The application removes the event from the calendar

## 5. View Calendar:

1. The user logs into the application.
2. The user navigates to the Calendar Interface.
3. The user selects one of the following views:
    a. Daily View
    b. Weekly View
    c. Monthly View
4. The application displays the calendar with events for the selected view.

Variant #1: No events to display. Add Event.

1.1 Start at Step 4
1.2 The calendar view is empty because there are no events to display
1.4 The user clicks “Add Event” or navigates to a different date.

## 6. Delete Account:

1. The user logs into the application.
2. The user navigates to the User Settings Interface.
3. The user selects “Delete Account.”
4. The application prompts the user to confirm the account deletion: “Are you sure you want to delete your account? This action is irreversible.”
5. The user confirms the deletion by entering their password and clicking “Delete Account.”
6. The application deletes the user account and logs the user out, displaying a confirmation message.

Variant #1. Cancel account deletion

1.1 Start at Step 5.
1.2 The user cancels the action instead of entering the password.
1.3 The application keeps the account active.

# Design Specification

# CRC Cards:

| Class: UserAccount | |
|---|---|
| Responsibilities | Collaborators |
| Manage User Profile:<br>*Store and retrieve user Credentials (username, password).<br>*Allow users to view and edit their profile information.<br>*Store passwords.<br><br>Account Management:<br>*Create a new user account<br>*Delete an existing user account<br>*Validate password and password confirmation match.<br><br>Authentication:<br>*Create a new user account<br>*Handle login errors and provide feedback. | Event: manages events created by the user.<br><br>JTextField and JPasswordField: for username and password input during login.<br><br>JButton: for actions like "Login".<br><br>JLabel: to display labels for input fields and messages to the user.<br><br>JOptionPane: for displaying dialog messages (e.g. error message, confirmation prompts). |

| Class: Event | |
|---|---|
| Responsibilities | Collaborators |
| Manage Event Details:<br>*Store event attributes: title, date, time, course, description, priority level.<br>*Allow editing of event details.<br><br>Event Status Management:<br>*Track Event Status (pending or completed).<br>*Allow marking events as complete or incomplete.<br><br>Reminders:<br>*Set and mange reminder settings (type, time interval).<br><br>Conflict Detection:<br>*Check for time conflicts with other events.<br>*Alert user of potential scheduling | User Login: owner of the event.<br><br>Calendar: displays the event in various calendar views.<br><br>Reminder: manages reminders associated with the event.<br><br>JTextField: for inputting text fields like title and course.<br><br>JTextArea: for multi-line input like the description.<br><br>JComboBox: for selecting date, start time, end time, and priority level.<br><br>JCheckBox: for enabling/disabling reminders.<br><br>JButton: for actions like "Save Event", "Remove Event", "Mark as Complete".<br><br>JLabel: for labels and displaying event information. |

| Class: Calendar | |
|---|---|
| Responsibilities | Collaborators |
| Display Events:<br>*Provide daily, weekly, and monthly views of events<br>*Display event titles on their scheduled dates<br>*Indicate completed (green) and pending (blue) events.<br><br>Event Management:<br>*Add events to the calendar.<br>*Remove events from the calendar<br>*Update event displays when events. are added, edited, or removed.<br><br>Navigation:<br>*Navigate between different dates and views.<br>*Update the display based on user navigation.<br><br>Event Interaction:<br>*Allow users to select events to view details.<br>*Support editing or deleting events dir | Event: contains the events to be displayed.<br><br>User Login: accesses events associated with the logged-in user.<br><br>JPanel: for organizing the calendar layout.<br><br>JList: for displaying lists of events in calendar views.<br><br>JButton: for navigation controls (e.g. next day, previous week).<br><br>JLabel: for displaying the current date, week, or month.<br><br>MouseListener: to detect clicks on events and display event details. |

| Class: Reminder | |
|---|---|
| Responsibilities | Collaborators |
| Manage Reminder Settings:<br>*Store reminder type (email, push notification).<br>*Store reminder time interval before the event.<br>*Enable or disable reminders for an event.<br><br>Send Reminders:<br>*Schedule reminders based on event time and reminder settings.<br>*Dispatch reminders via the chosen method.<br><br>Customization:<br>*Allow users to customize reminder preferences per event.<br>*Support different reminder intervals. | Event: the event associated with the reminder.<br><br>JCheckBox: for selecting reminder types and time intervals.<br><br>Timer (existing Java library class): for scheduling reminders.<br><br>JOptionPane: for displaying reminder notifications. |

| Class: Taskbar | |
|---|---|
| Responsibilities | Collaborators |
| Provide Navigation:<br>*Home: Main Interface<br>*User Profile: User settings<br>*Add Events: Create new events<br>*Calendar: View calendar<br>*Route the user to the selected interface<br><br>Manage Button States:<br>*Highlight the active view.<br>*Enable or disable buttons based on context (e.g. user authentication status).<br><br>User Interaction Handling:<br>*Respond to user clicks on navigation buttons.<br>Update the UI based on navigation actions. | User Login<br><br>Calendar<br><br>Event<br><br>JPanel: serves as the container for the taskbar.<br><br>JButton: represents each navigation button (e.g. User Profile, Add Events)<br><br>ActionListener: handles events when buttons are clicked.<br><br>JLabel: enhances buttons with text labels.<br><br>JOptionPane: for displaying messages or confirmations if needed. |

| Class: MainInterface | |
|---|---|
| Responsibilities | Collaborators |
| *Initialize calendar interface (CalendarFrame, DayViewPanel, WeekViewPanel)<br><br>*Display the interface to the user via the display() method.<br><br>*Switch views between day and week modes in the calendar (collaborates with CalendarFrame)<br><br>*Respond to user actions like navigation (collaborates with JButton, NavigationPanel)<br><br>*Manage user events (collaborates with UserAccount, Event )<br><br>*Handle event displays (e.g., event lists, current date and week display) | CalendarFrame: For managing the view switching between different panels (DayView, WeekView)<br><br>Calendar: To manage and access calendar-specific data<br><br>UserAccount: To manage user credentials and events<br><br>Event: For handling and managing events<br><br>NavigationPanel: For interacting with navigation buttons (next day, previous day, today)<br><br>JButton: enhances buttons with text labels<br><br>DayViewPanel & WeekViewPanel: For displaying day and week views in the calendar interface<br><br>MouseListener: To capture and handle user input actions |

| Class: CreateAccountWindow | |
|---|---|
| Responsibilities | Collaborators |
| *Display UI components for creating an account, such as input fields for username and password<br><br>*Receive and process user input for creating a new account (username and password)<br><br>*Validate the user's input before passing it to the system for account creation<br><br>*Route the user to the LoginWindow after successful account creation<br><br>*Trigger account creation action when the "Create Account" button is pressed | UserAccount: Stores the username and password details for the newly created account<br><br>System: Handles the creation of the user account via createUser(name, pwd) and Validates if the username is unique and performs actions like account deletion<br><br>CreateAccountController: Listens for and handles the "Create Account" button click event and performs the appropriate action<br><br>JTextField: Used for collecting user input (username and password)<br><br>JButton: Interacts with the "Create Account" button to trigger the creation process<br><br>LoginWindow: After successful account creation, the system routes the user to the login window |

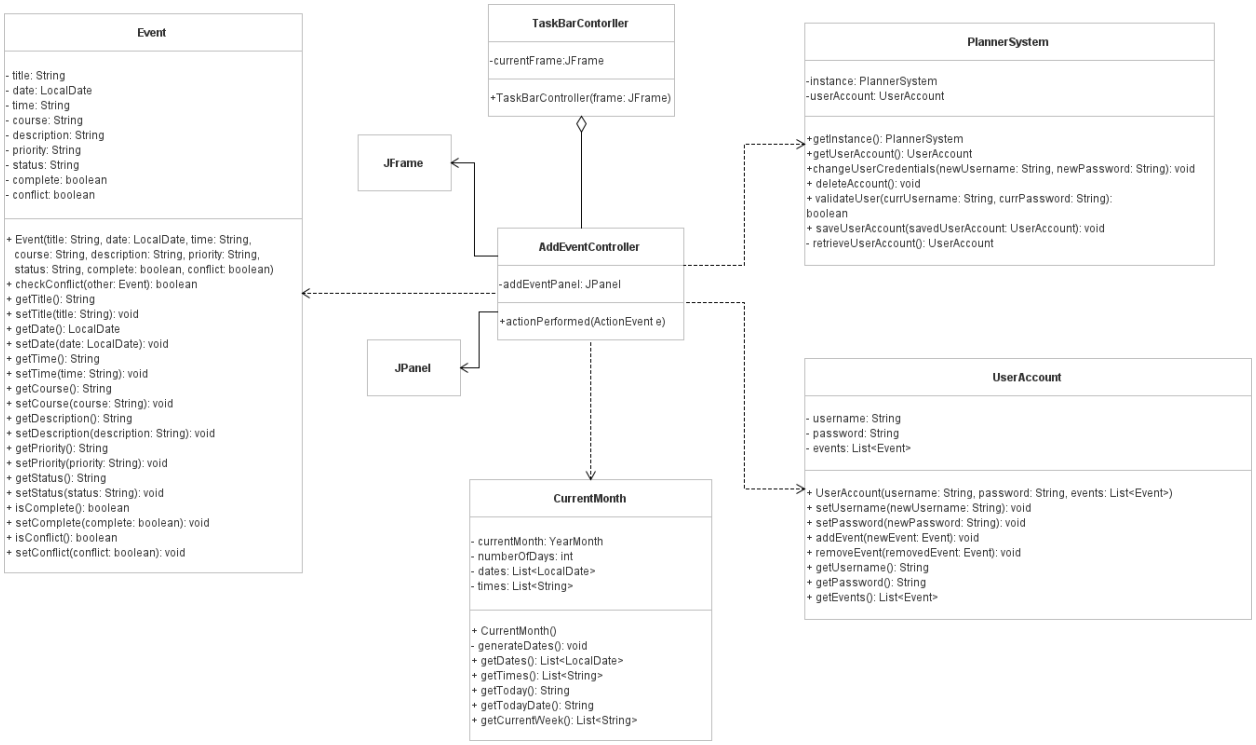
| Class: UserProfileWindow | |
|---|---|
| Responsibilities | Collaborators |
| *Display the user’s current credentials (username and password) in the window<br><br>*Provide input fields for the user to change their username and/or password<br><br>*Handle the "Change Credentials" action, submitting the new credentials to the system<br><br>*Handle the "Delete Account" action, submitting a request to delete the user's account<br><br>*Validate user input for credential changes and account deletion<br><br>*Rout the user to the appropriate page after updating credentials or deleting the account | UserAccount: Stores the username and password details for the newly created account<br><br>System: Handles the creation of the user account via createUser(name, pwd) and validates if the username is unique and performs actions like account deletion<br><br>UserProfileController: Listens for and handles button events, such as changing credentials or deleting the account.<br><br>JTextField: Used for collecting user input (username and password)<br><br>JButton: Interacts with the "Create Account" button to trigger the creation process<br><br>TaskBar: Allows navigation to the UserProfileWindow from other parts of the application |

| Class: AddEventWindow | |
|---|---|
| Responsibilities | Collaborators |
| *Display the UI for creating a new event with input fields (e.g., title, date, time, description, priority)<br><br>*Collect user input for event details such as title, course, description, and priority<br><br>*Handle the "Add Event" button click to submit the new event<br><br>*Validate the entered event information before adding it to the calendar<br><br>*Notify the user if any required fields are missing or invalid | Event: For handling and managing events<br><br>UserAccount: To manage user credentials and events<br><br>AddEventController:Listens for user actions, such as pressing the "Add Event" button, and handles the submission process.<br><br>JTextField, JTextArea, JComboBox, JCheckBox: Collect user input for different attributes of the event (e.g., text for title, description, priority)<br><br>TaskBar: Allows navigation to the UserProfileWindow from other parts of the application<br><br>Timer: Works with reminders to notify the user at specific times. |

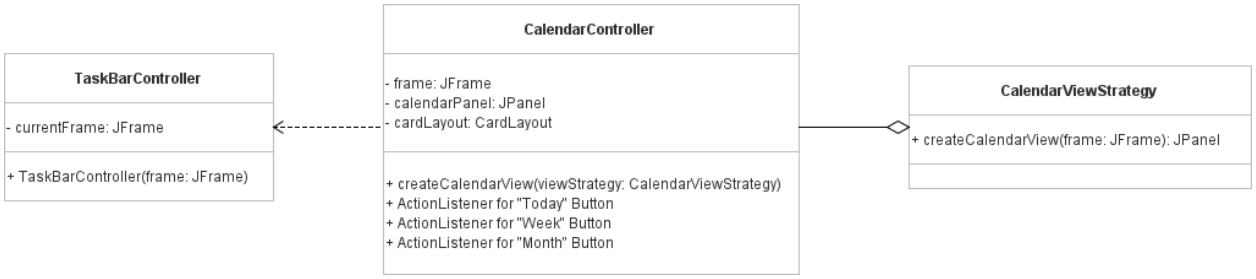
| Class: System | |
|---|---|
| Responsibilities | Collaborators |
| *Manage user authentication (login, create account, verify credentials)<br><br>*Handle event storage, retrieval, and updates<br><br>*Manage user profiles (updating, deleting, etc.)<br><br>*Control application flow between different interfaces (e.g., main interface, user profile, event creation) | UserAccount: To manage user credentials and events<br><br>Event: For handling and managing events<br><br>MainInterface: To handle navigation and display<br><br>CreateAccountWindow: For account creation operations<br><br>UserProfileWindow: For managing user profile information<br><br>AddEventWindow: For event creation and management |

# UML Diagrams:

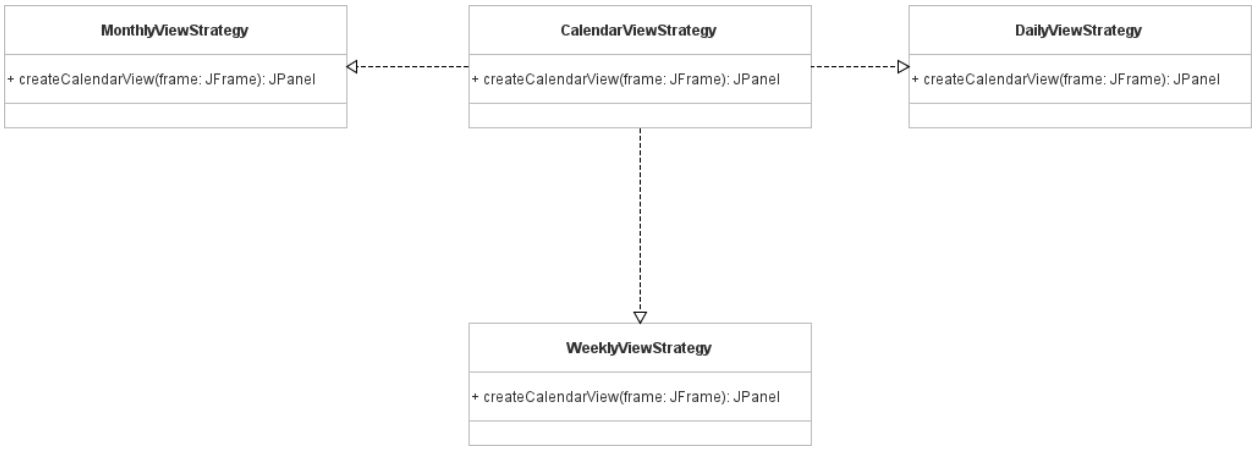
## AddEventController(class diagram)



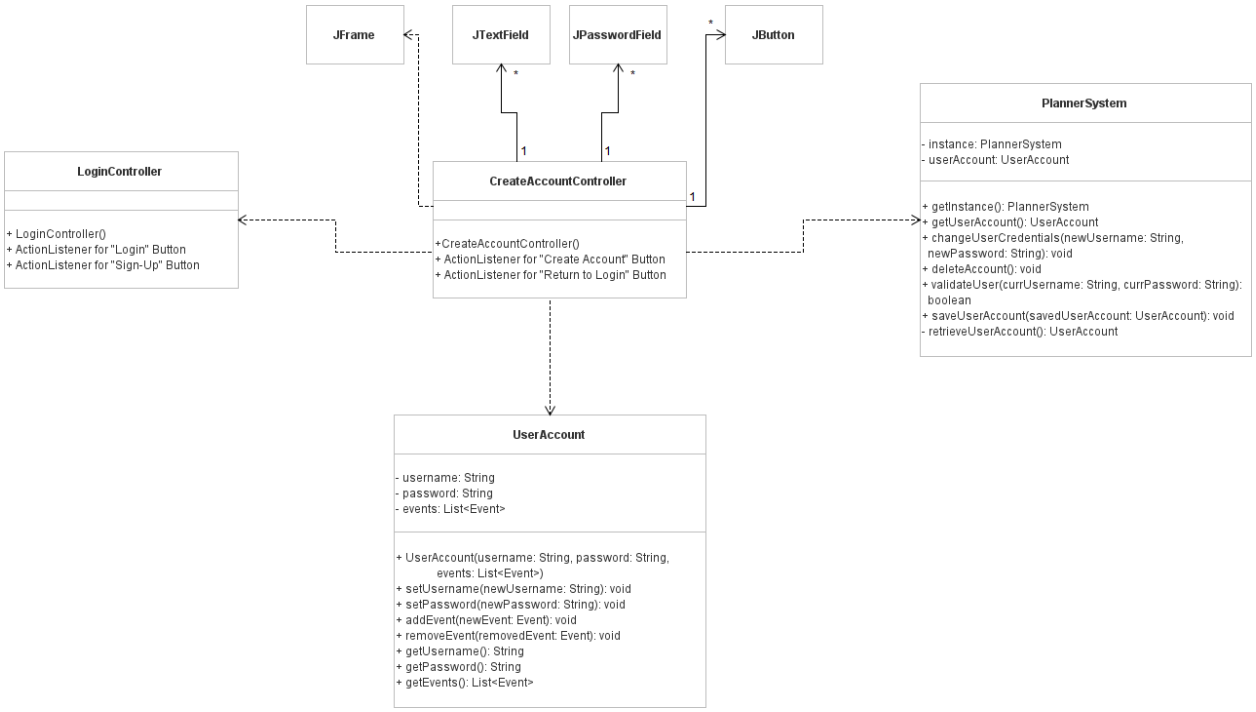
# CalendarController(class diagram)



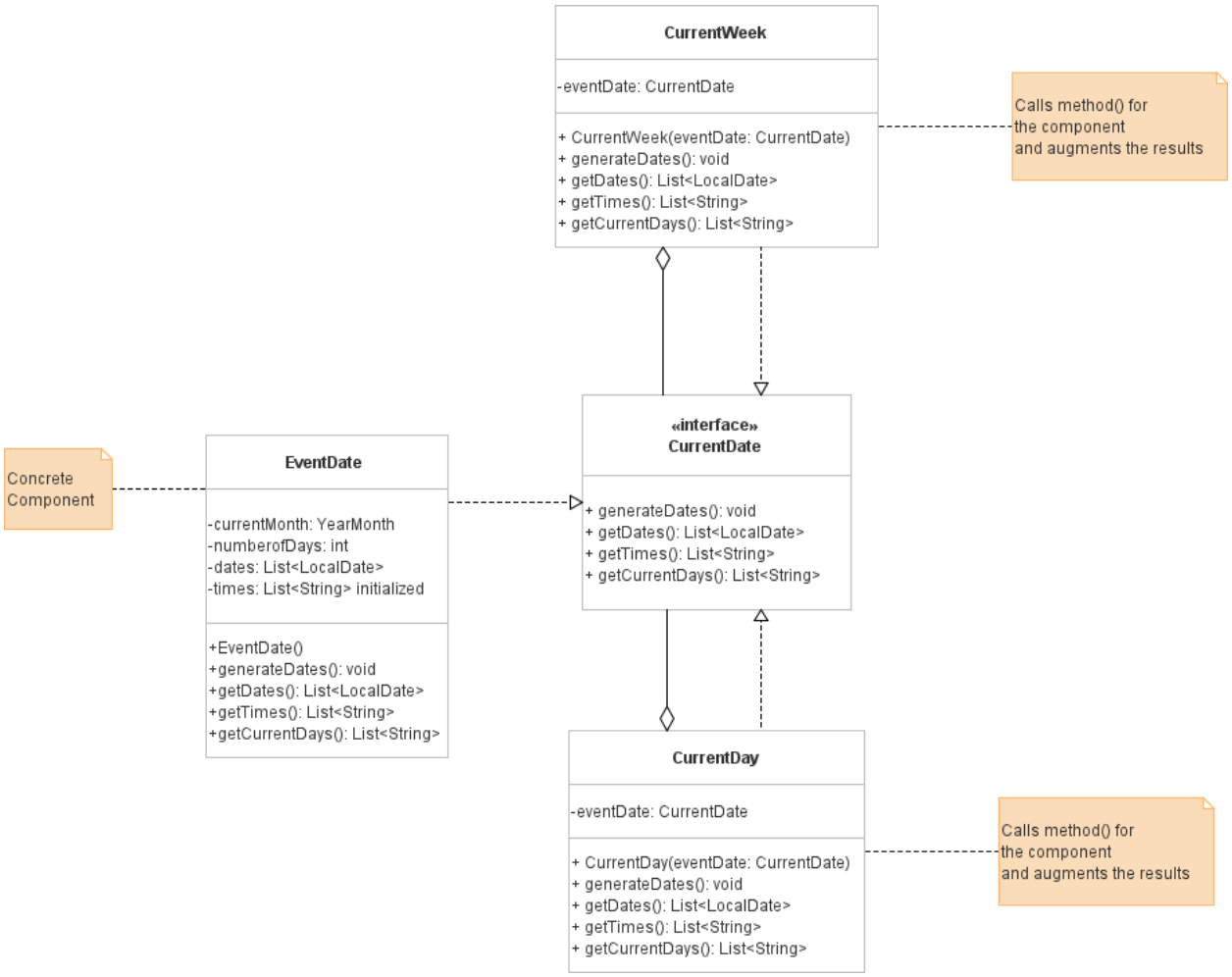

# CalendarViewStrategy(class diagram)



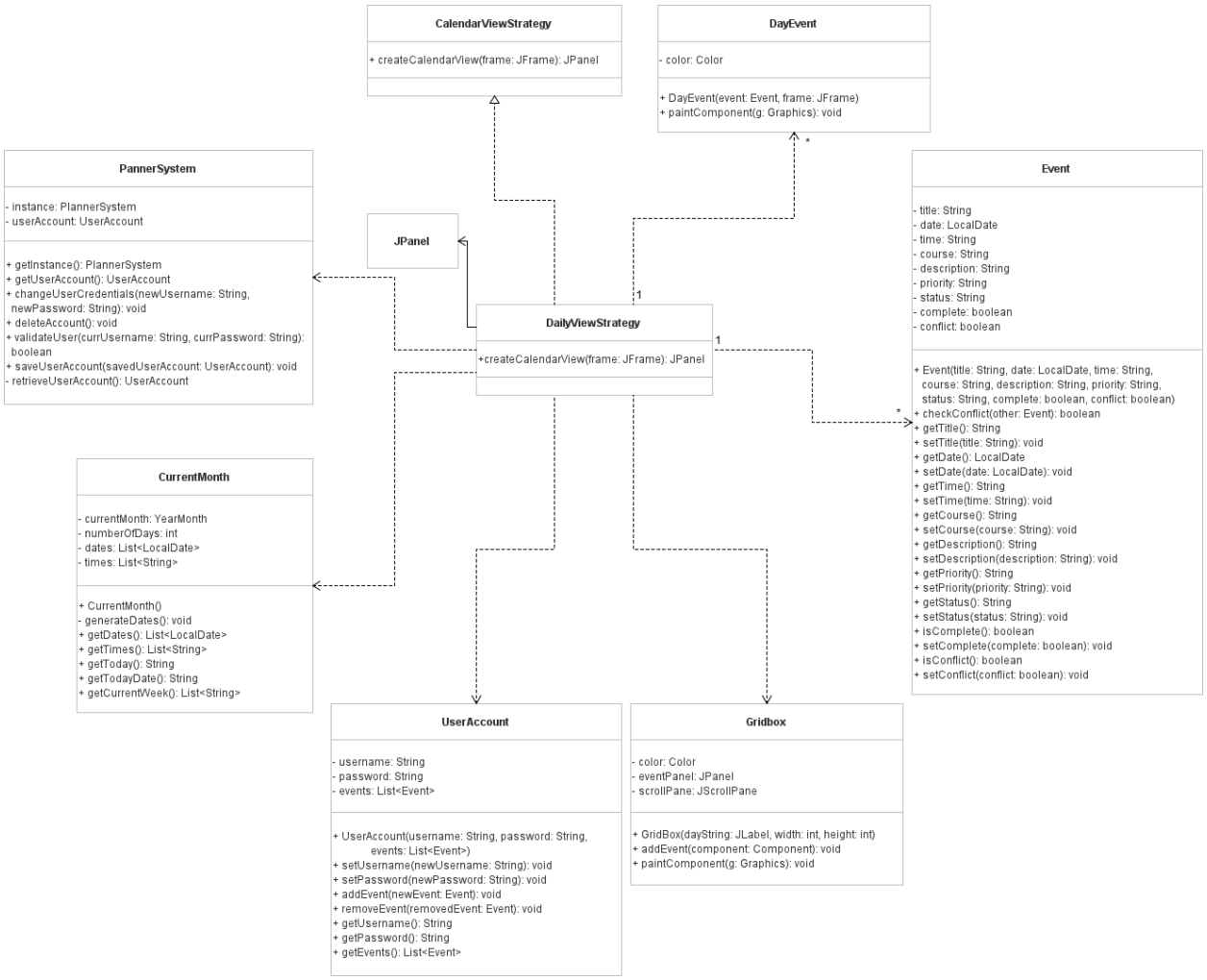
# CreateAccountController(class diagram)



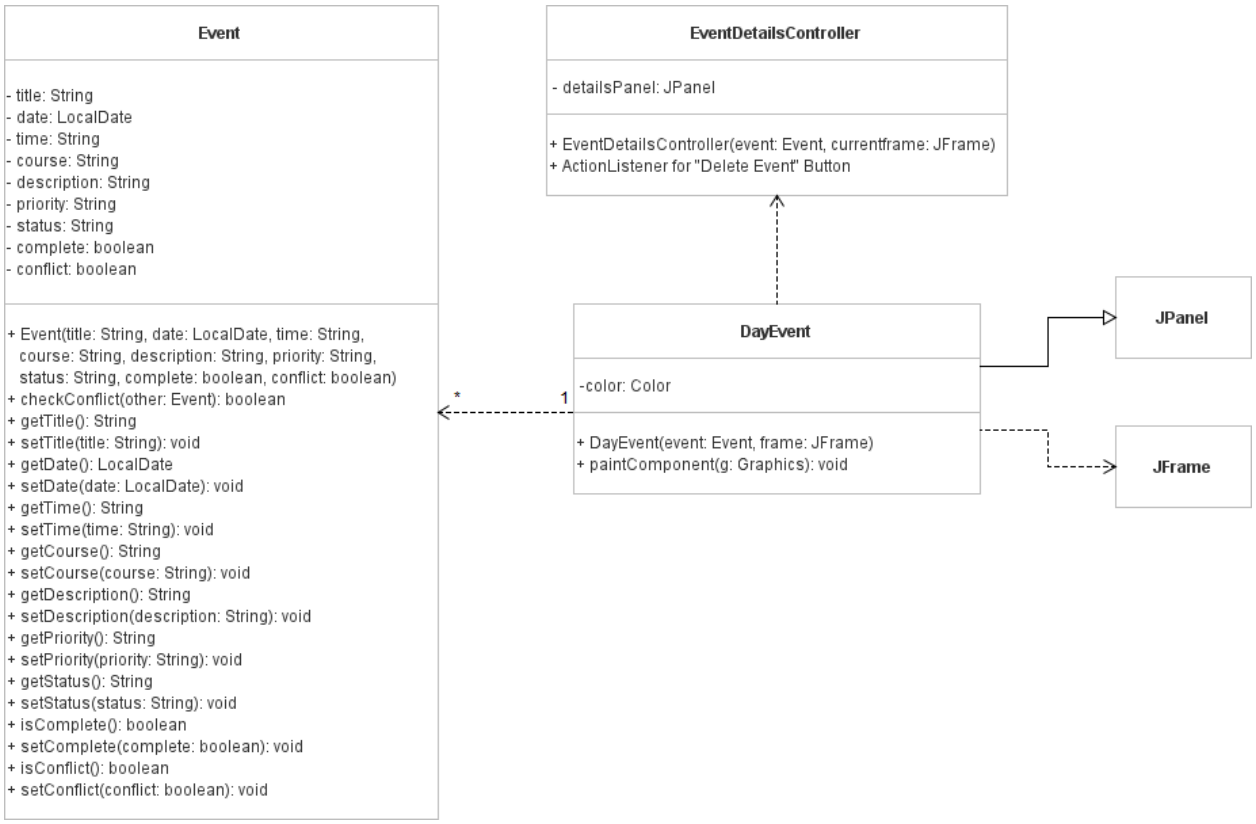
# CurrentDate(class diagram)



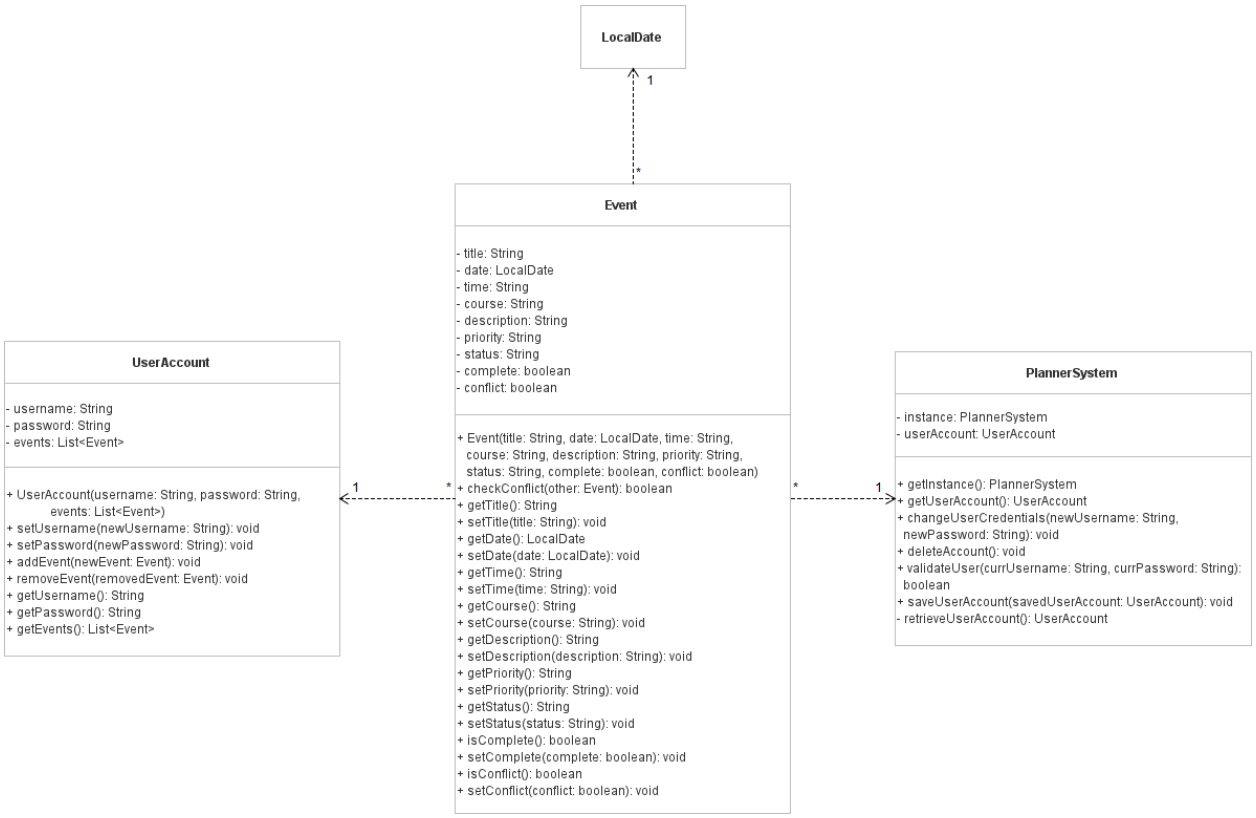
# DailyViewStrategy(class diagram)



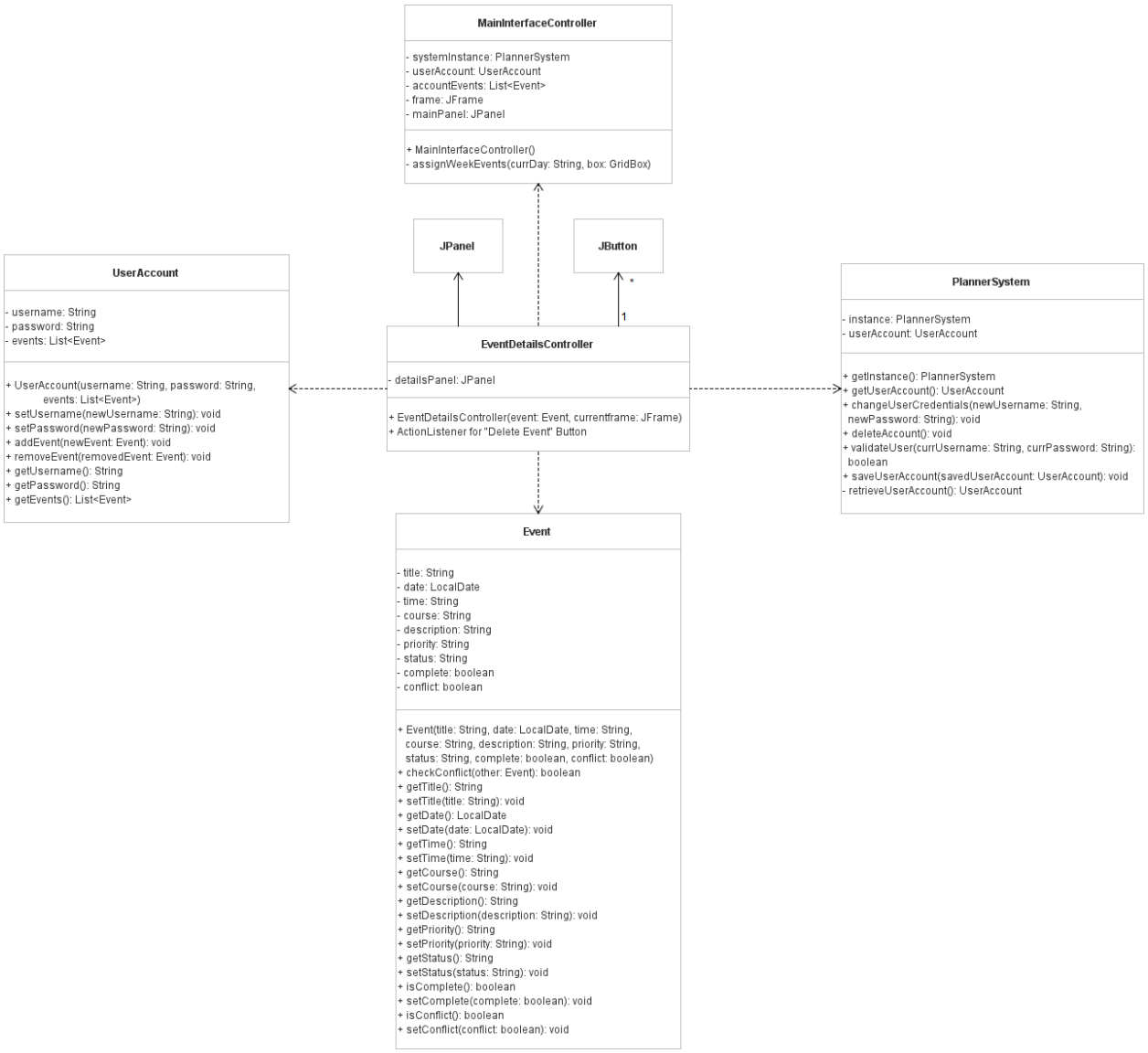
# DayEvent(class diagram)



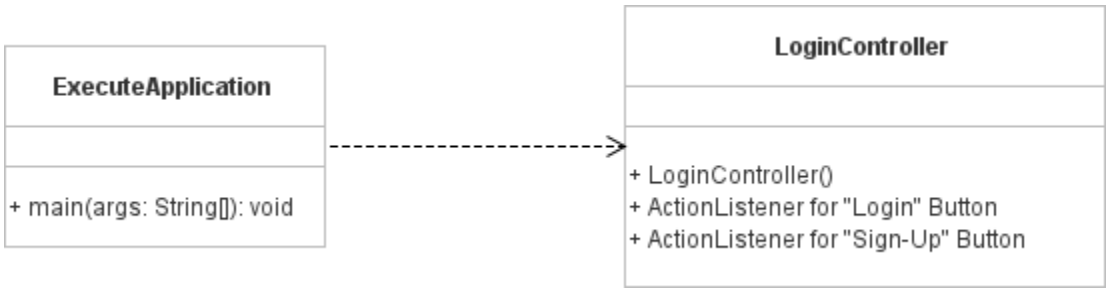
# Event(class diagram)



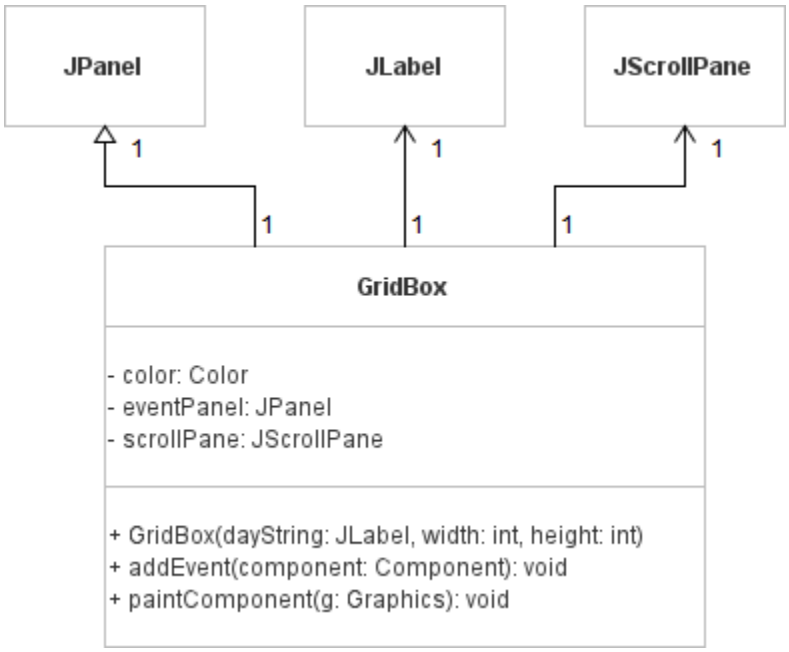
# EventDetailsController(class diagram)



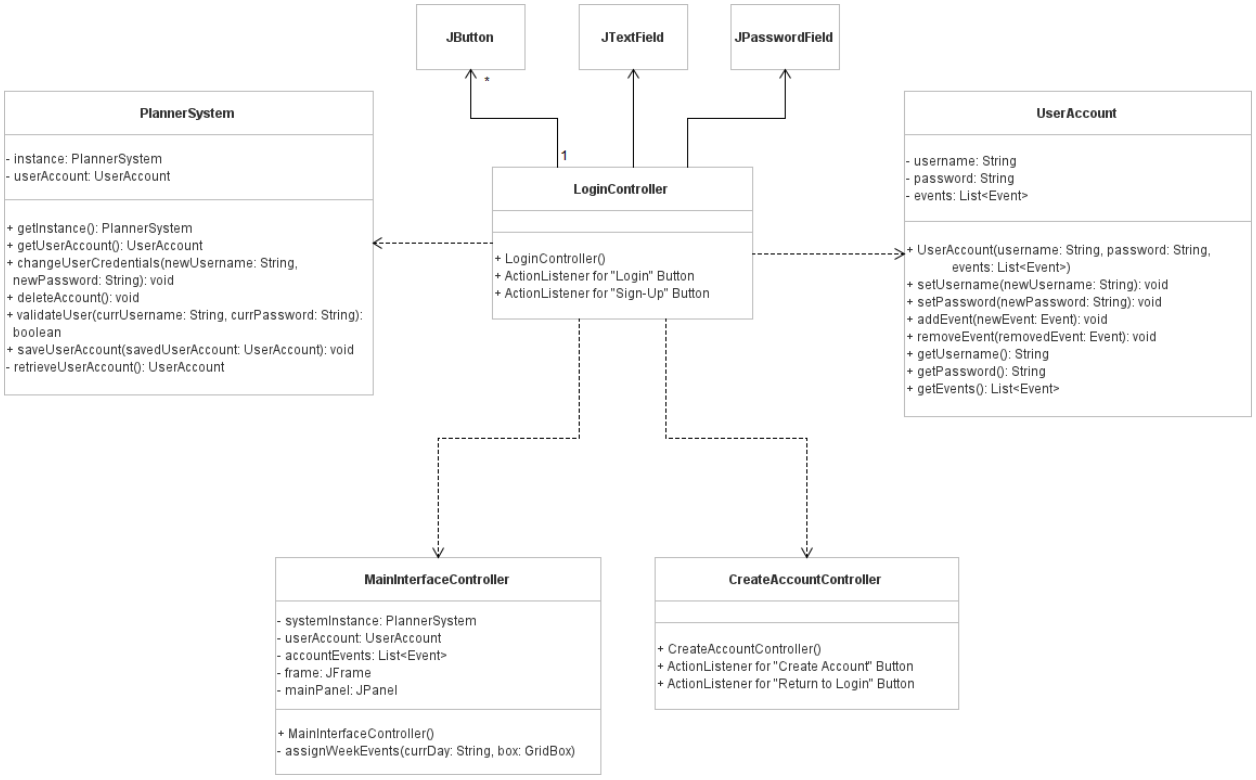

# ExecuteApplication(class diagram)



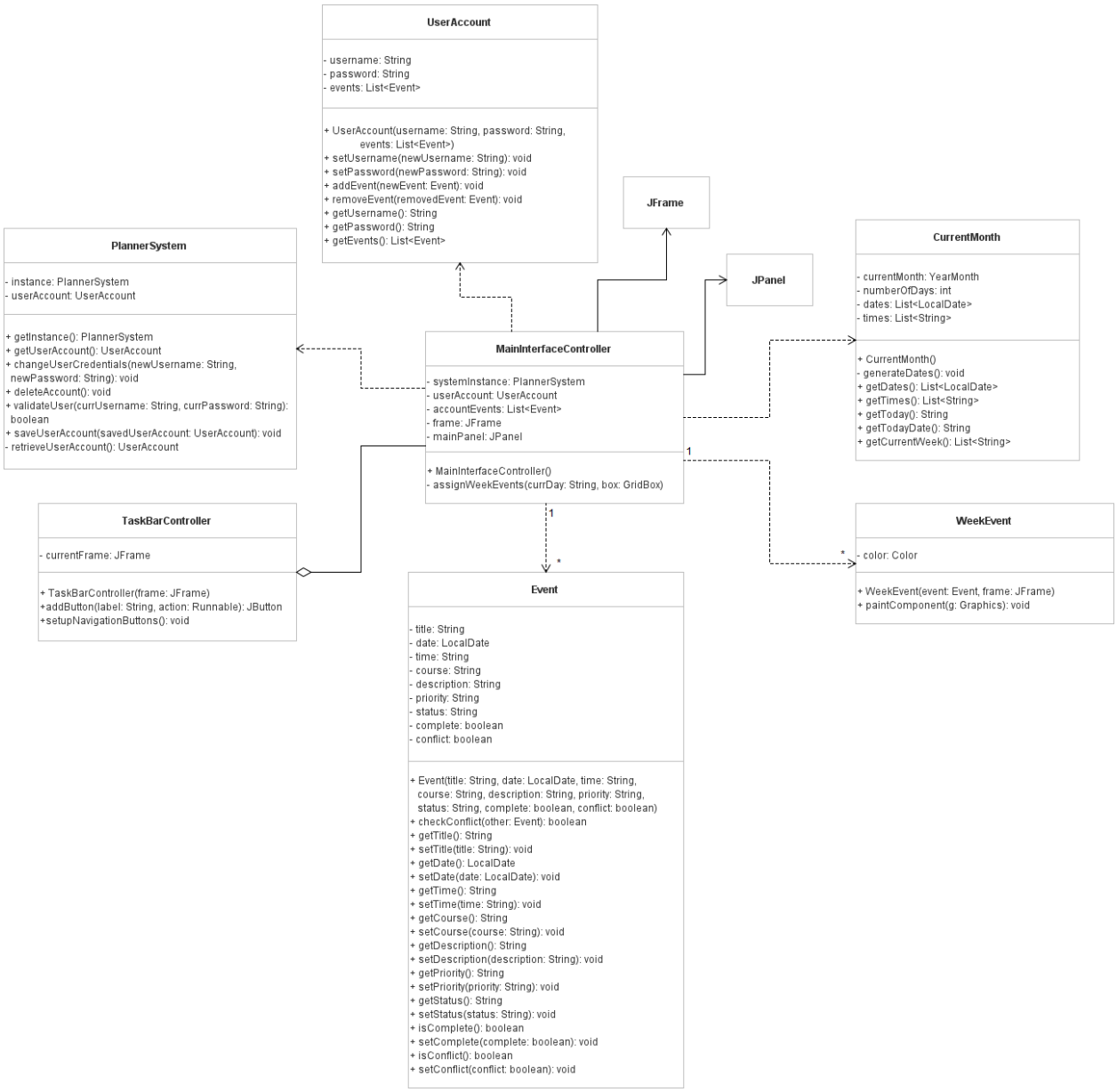
## GridBox(class diagram)



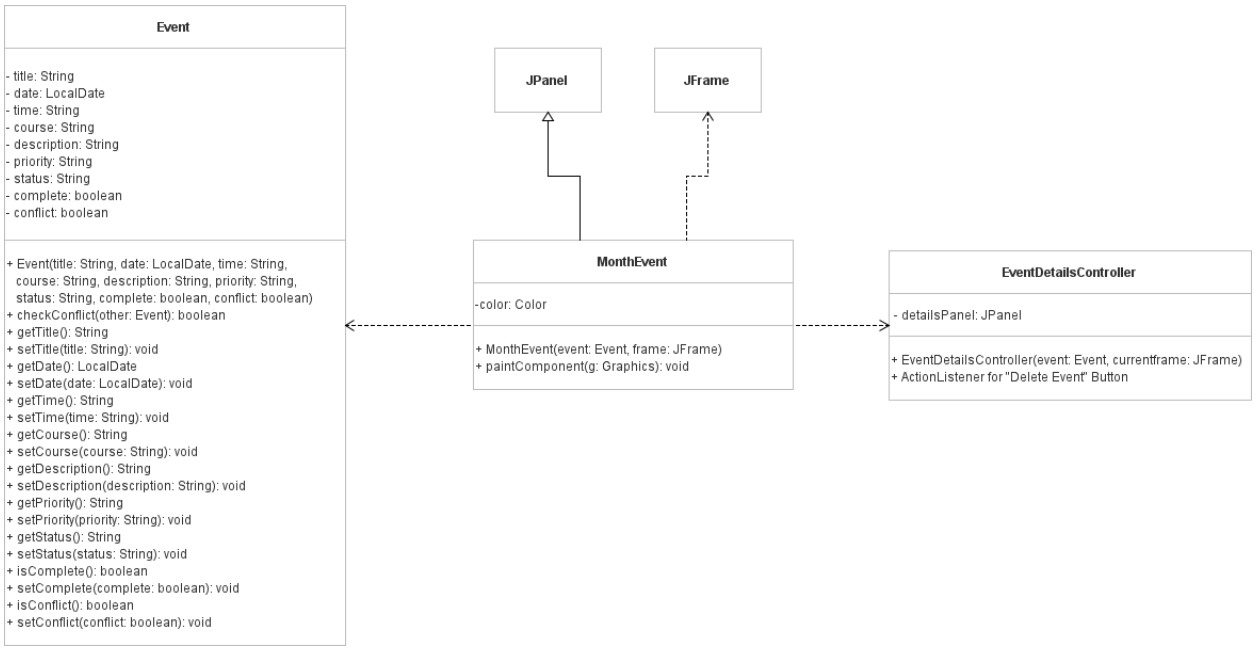

## LoginController(class diagram)



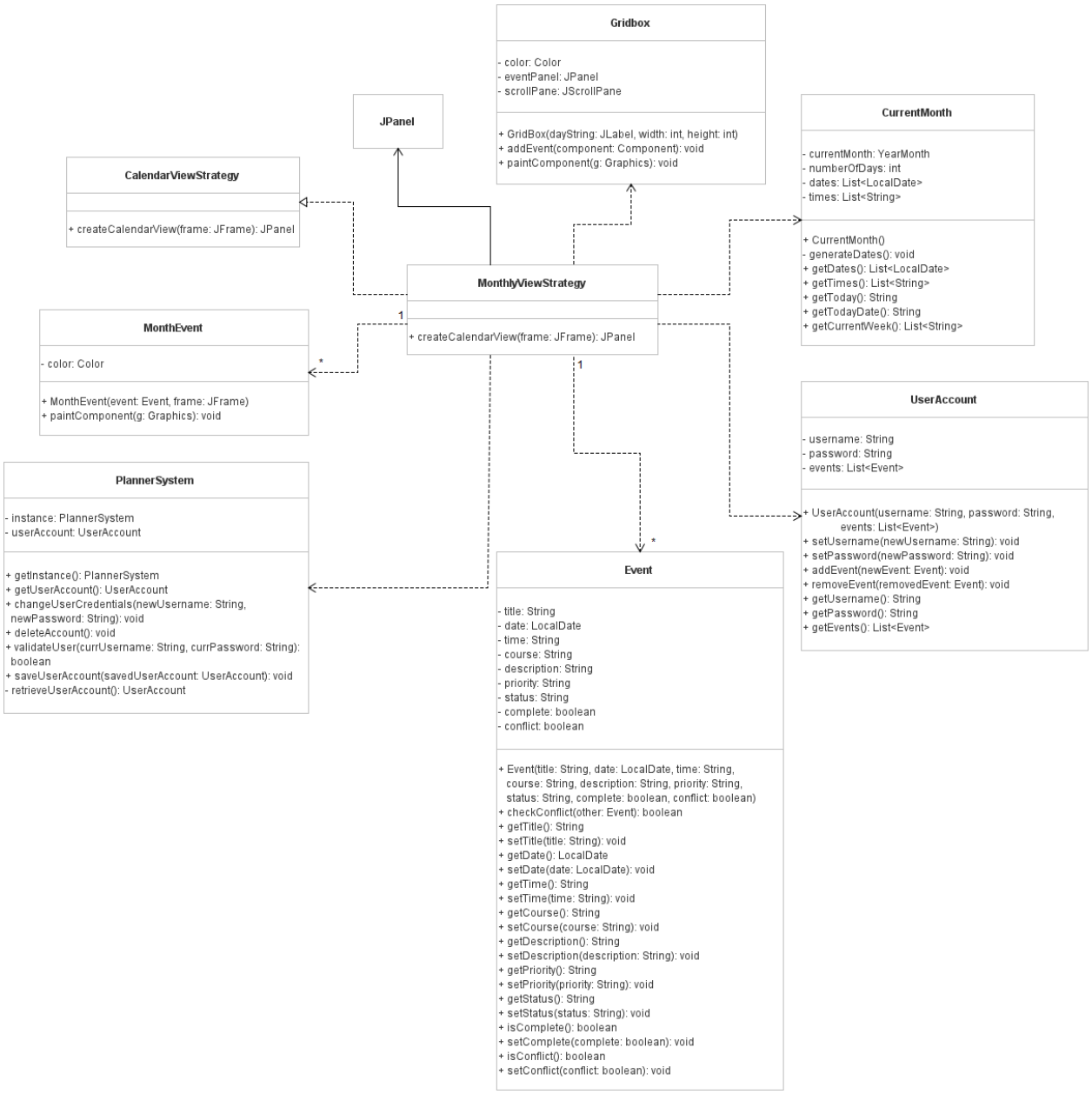
# MainInterfaceController(class diagram)



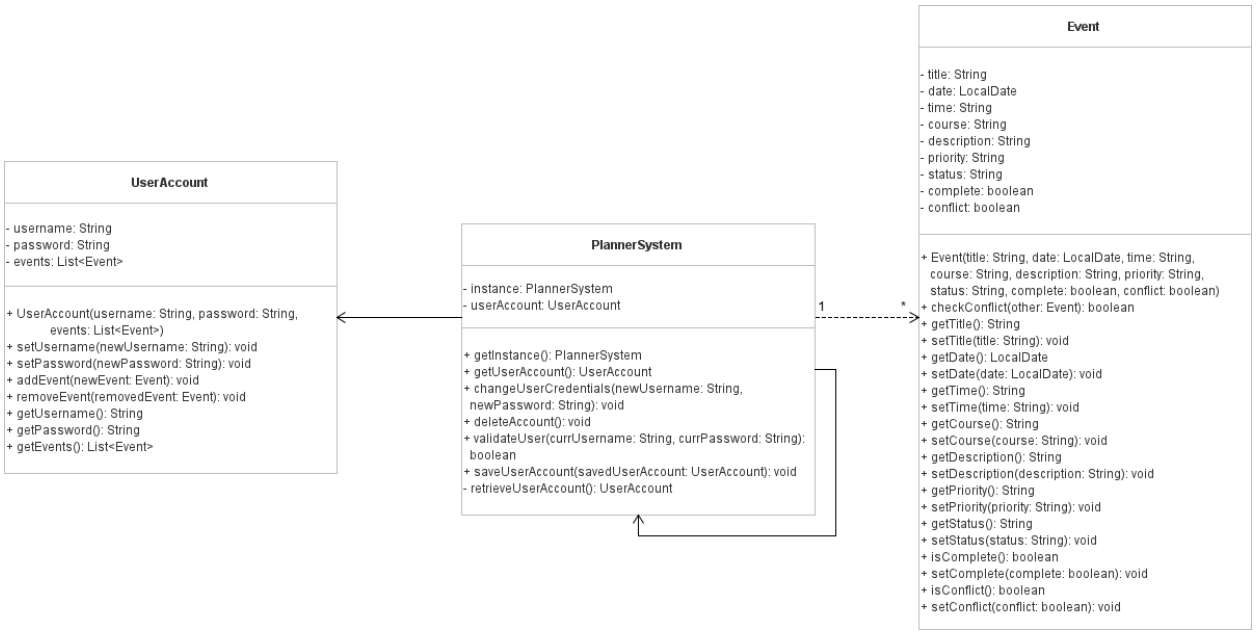
# MonthEvent(class diagram)



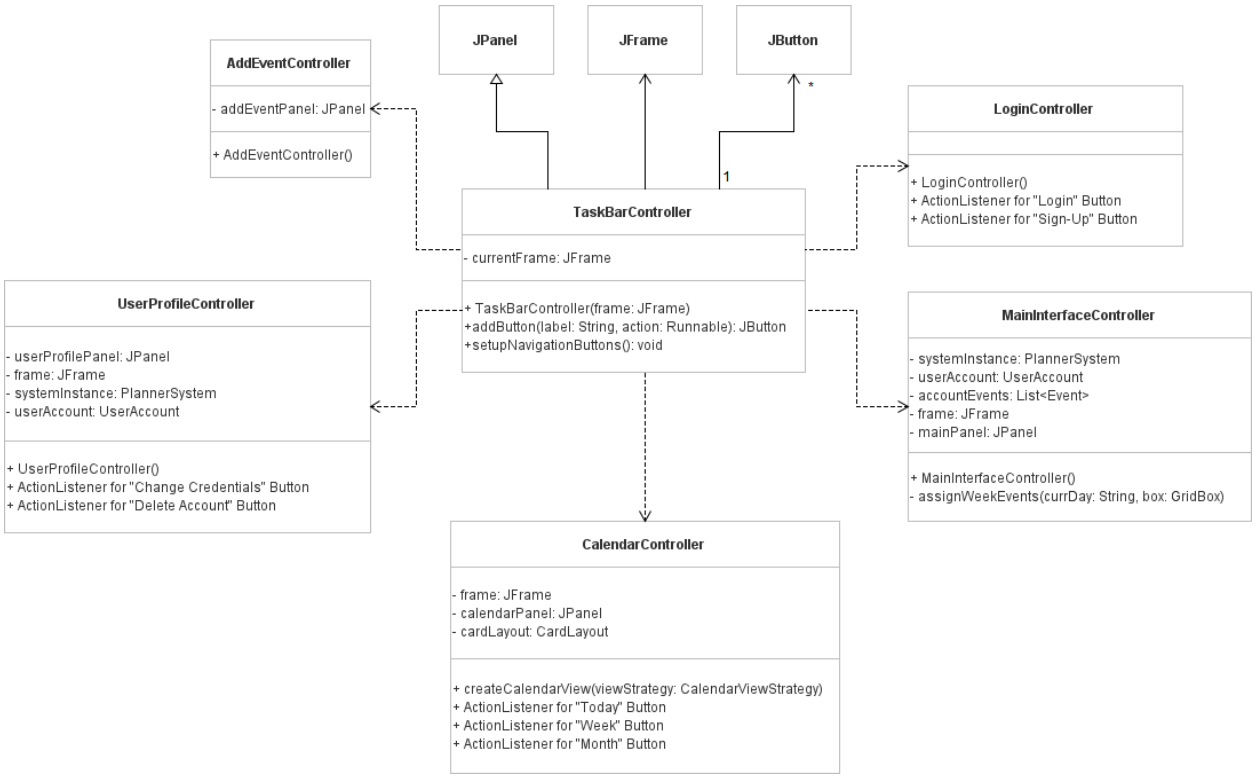
# MonthlyViewStrategy(class diagram)



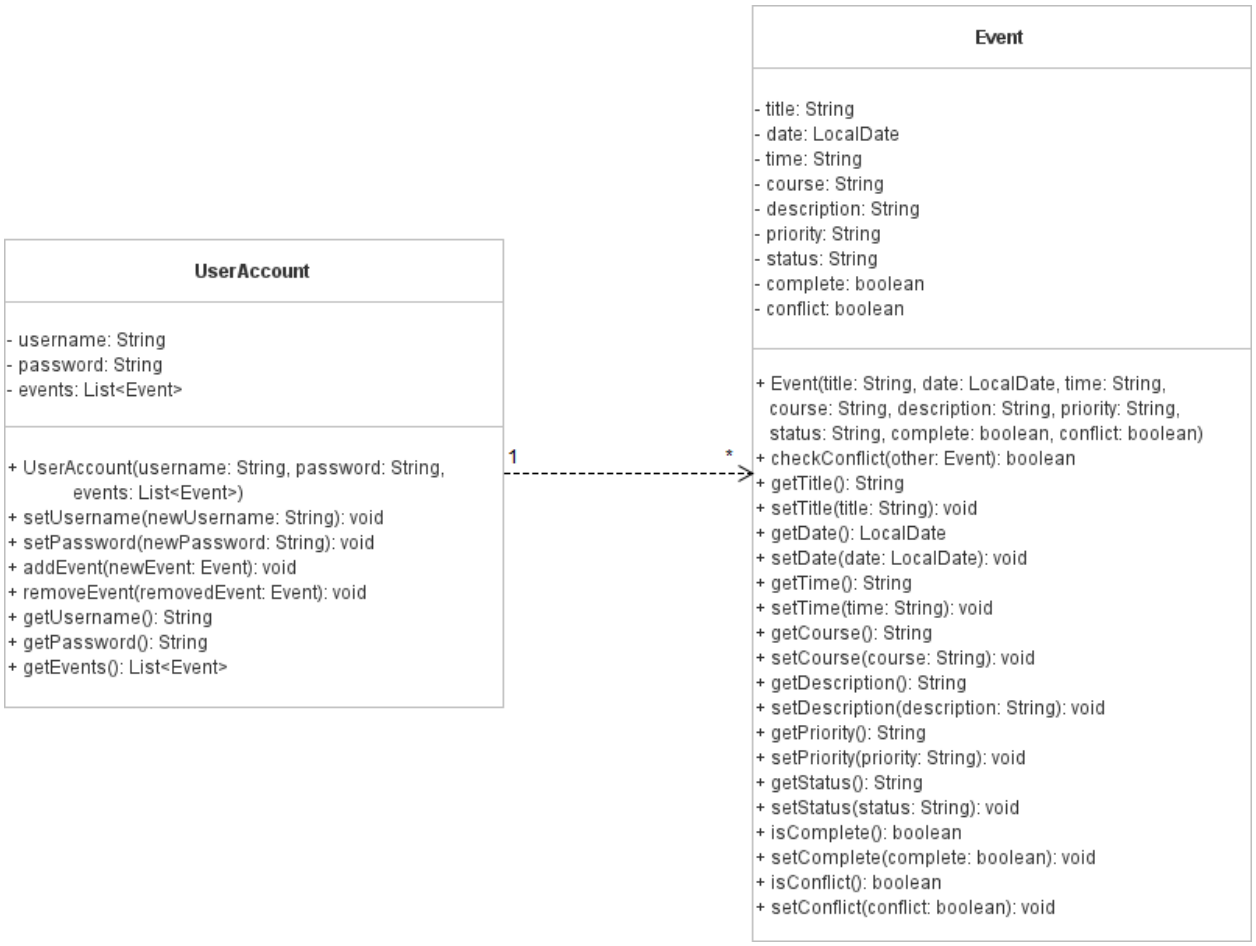
# PlannerSystem(class diagram)



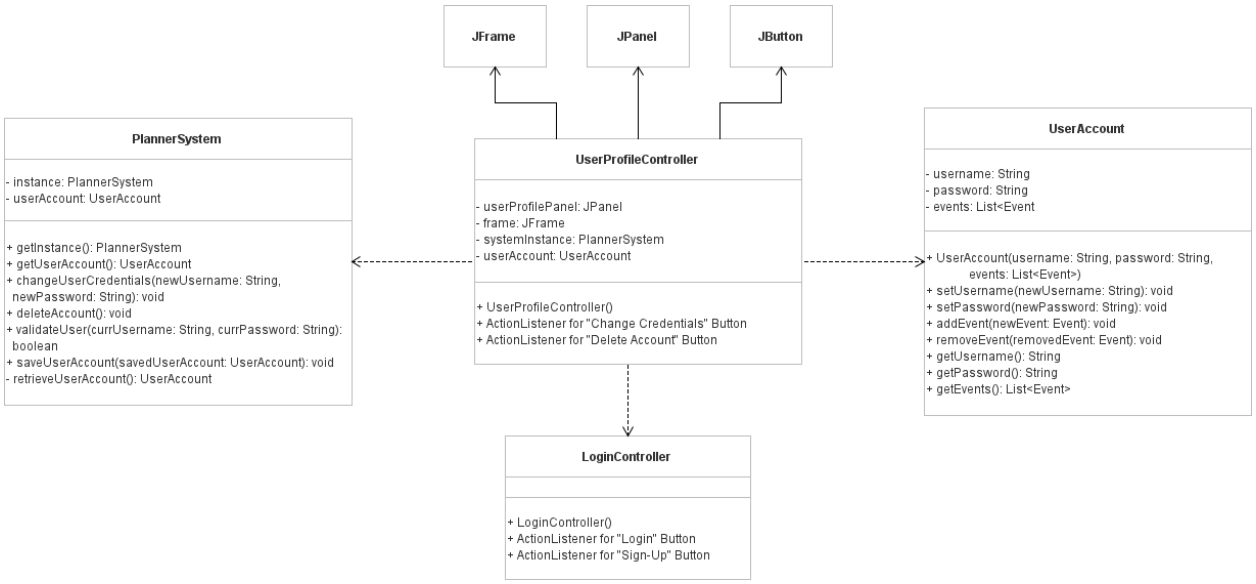

# TaskBarController(class diagram)



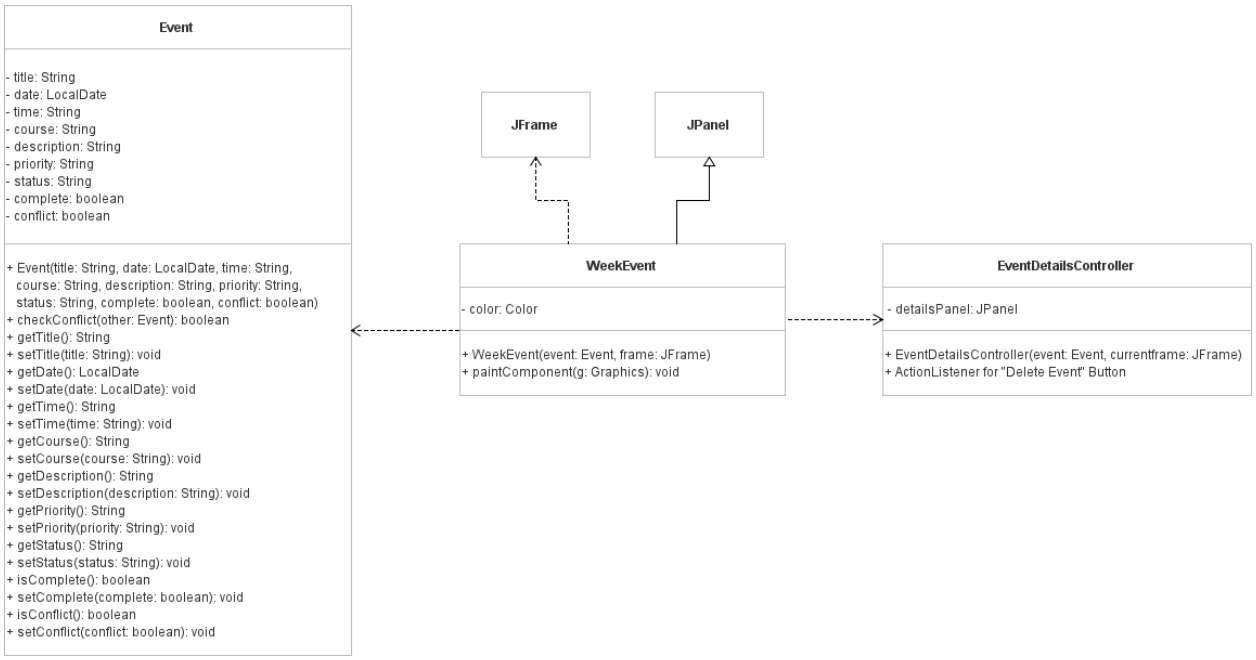
# UserAccount(class diagram)



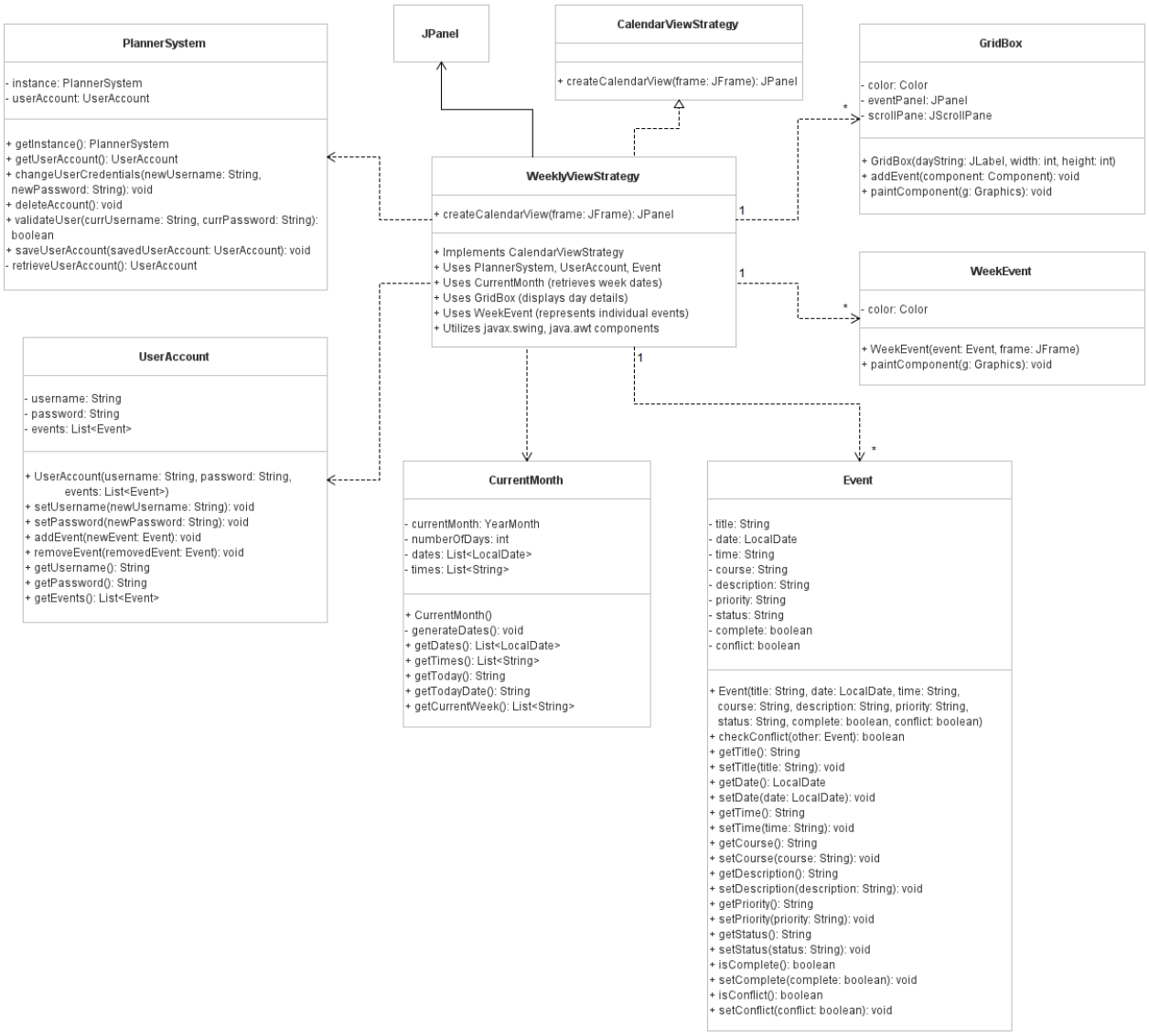

# UserProfileController(class diagram)



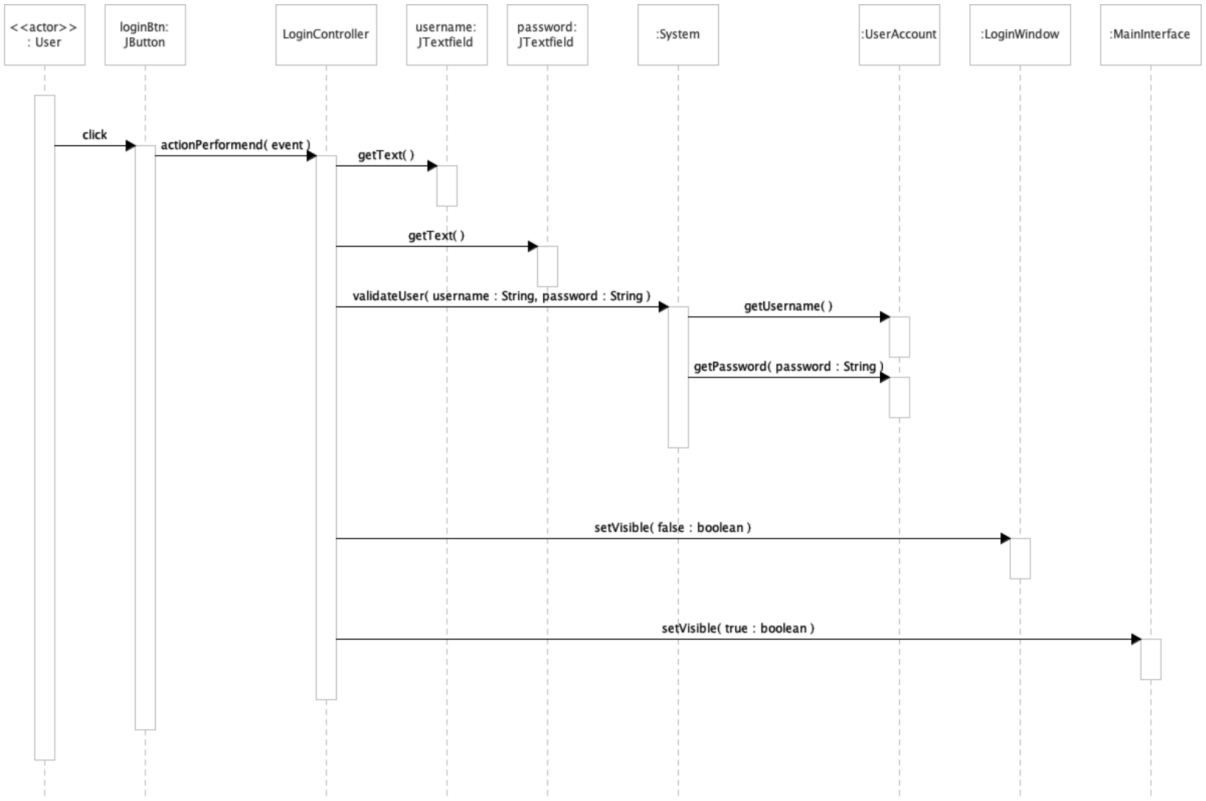
# WeekEvent(class diagram)



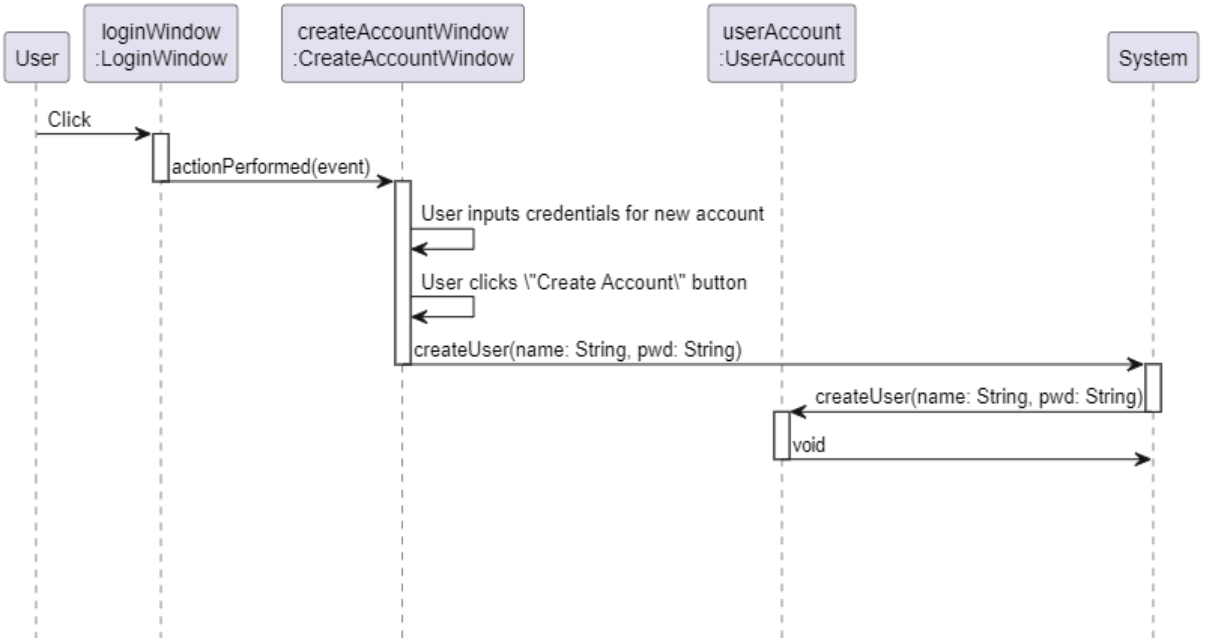
# WeeklyViewStrategy(class diagram)



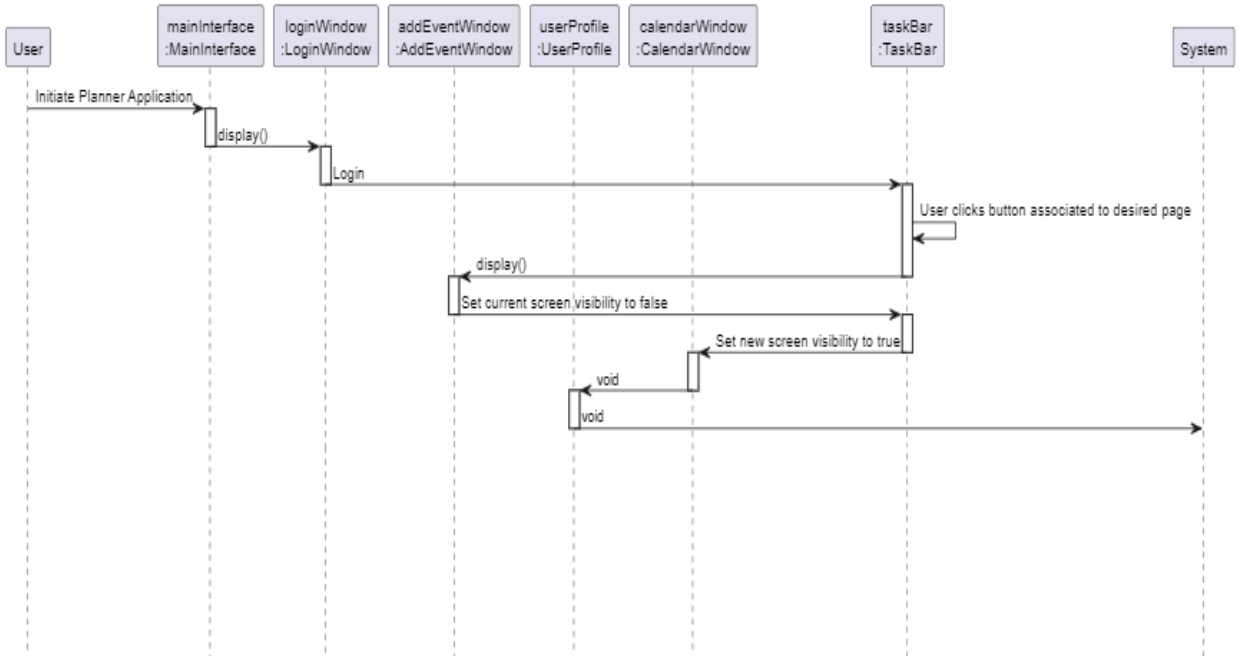
# UserLogin(sequence diagram)



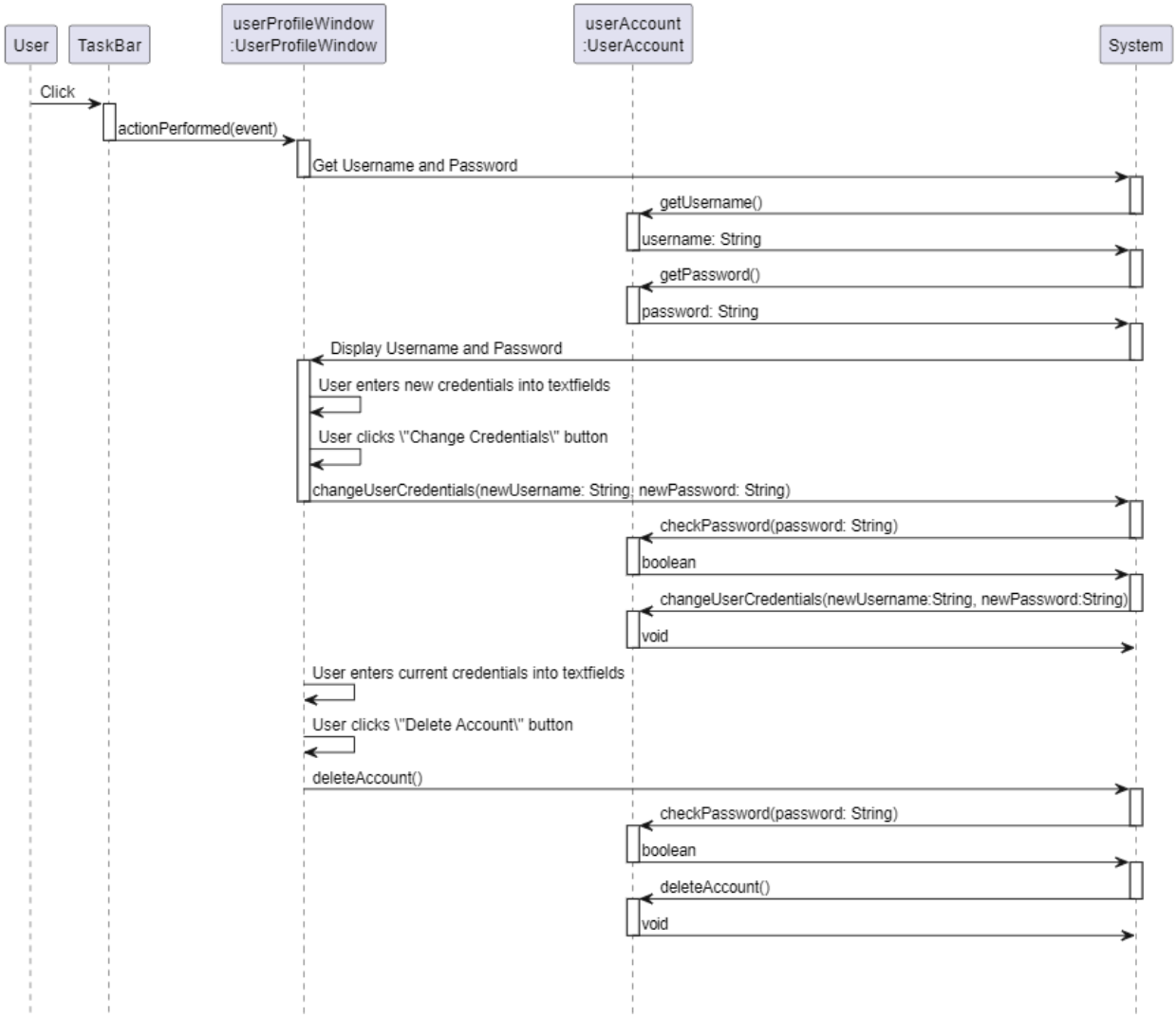
## Create Account (sequence diagram)



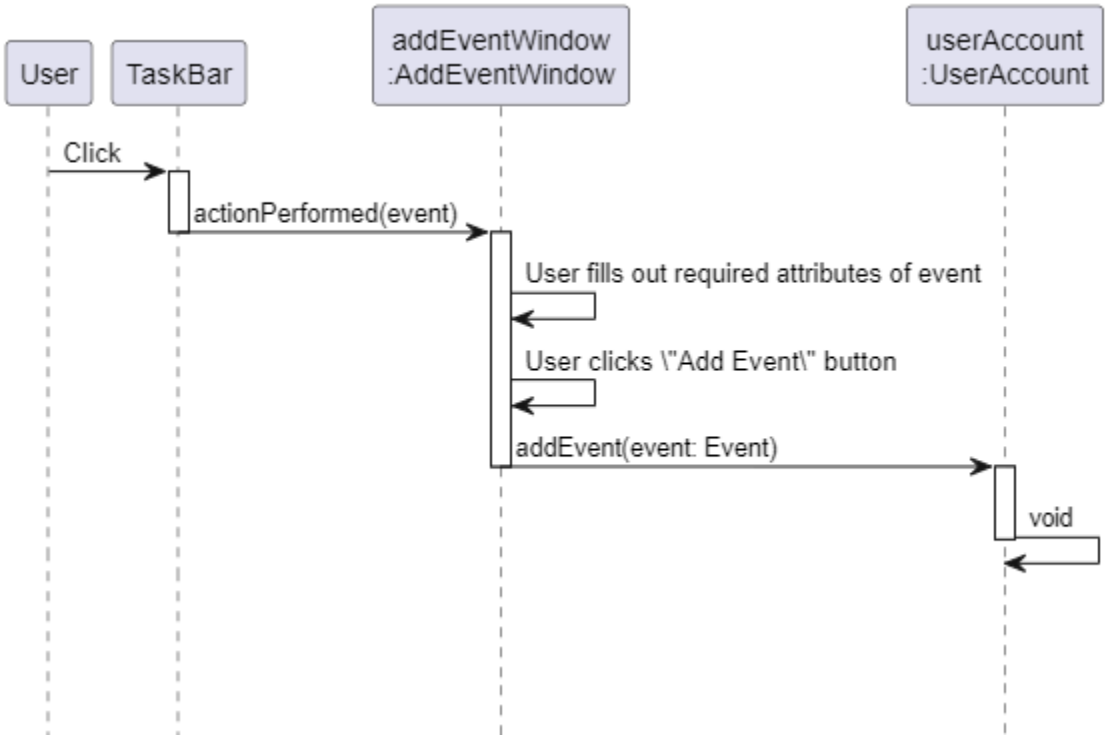

## Task Bar Navigation (sequence diagram)



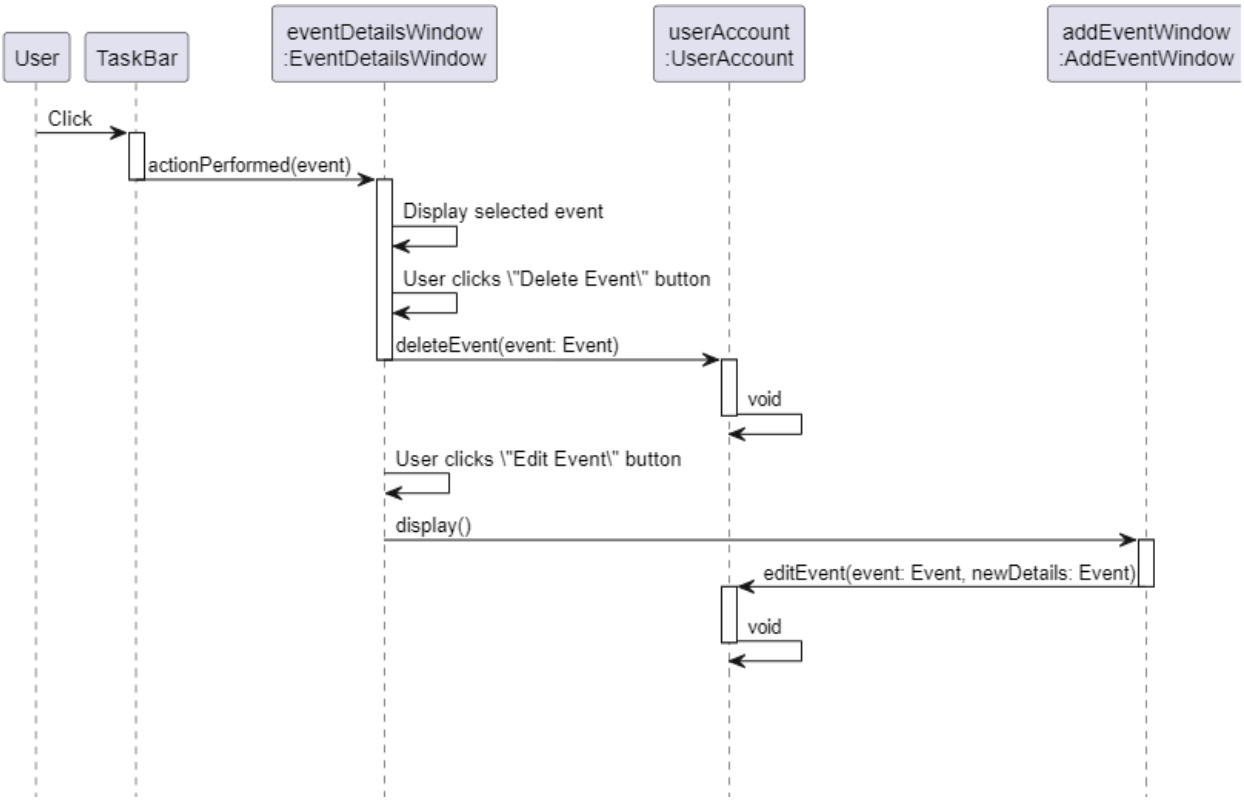
## Change Credentials/Delete Account (sequence diagram)



## Add Event (sequence diagram)




## Delete/Edit Event (sequence diagrams)



# Java Code

## CalendarStrategy/

### CalendarStrategy.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import javax.swing.*;

/**
 * Interface for defining the strategy for creating different calendar views.
 * @author andrewcoggins
 */

public interface CalendarViewStrategy {
    /**
     * Creates and returns a JPanel representing the calendar view.
     *
     * @param frame The JFrame to which the calendar view will be added.
     * @return A JPanel representing the calendar view.
     */
    JPanel createCalendarView(JFrame frame);
}
```

### DailyViewStrategy.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
```

```java
 */
package cop4331.CalendarStrategy;

import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.Event;
import cop4331.CurrentDateDecorator.EventDate;
import cop4331.CurrentDateDecorator.CurrentDate;
import cop4331.CurrentDateDecorator.CurrentDay;
import cop4331.SharedViews.GridBox;

import javax.swing.*;
import java.awt.*;
import java.util.List;


/**
 * Concrete strategy for creating the daily calendar view.
 * @author andrewcoggins
 */

public class DailyViewStrategy implements CalendarViewStrategy {
    /**
     * {@inheritDoc}
     *
     * Creates and returns a JPanel representing the daily calendar view.
     */
    @Override
    public JPanel createCalendarView(JFrame frame) {
        JPanel todayContainerPanel = new JPanel(new FlowLayout());
        JPanel todayCompsPanel = new JPanel(new GridLayout(1, 7));
        todayContainerPanel.add(todayCompsPanel);

        PlannerSystem systemInstance = PlannerSystem.getInstance();
        UserAccount userAccount = systemInstance.getUserAccount();
        List<Event> accountEvents = userAccount.getEvents();

        // decorator pattern to get the String value corrisponding to today
        CurrentDate currentDate = new EventDate();
        CurrentDate eventCurrentDay = new CurrentDay(currentDate);
        String today = eventCurrentDay.getCurrentDays().get(0);

        JLabel dateString = new JLabel(today);
        GridBox box = new GridBox(dateString, 450, 800);
```

```
        String currDate = today.substring(0, 2);
        for (Event event : accountEvents) {
            String eventDate = event.getDate().toString().substring(8,10);
            if (currDate.equals(eventDate)){
                DayEvent dayEvent = new DayEvent(event, frame);
                box.addEvent(dayEvent);
            }
        }

        todayCompsPanel.add(box);
        return todayContainerPanel;
    }
}
```

## DayEvent.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import cop4331.gui.EventDetailsController;
import cop4331.EventComposite.Event;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
```

```
 * Class utilized to display events happening today
 * @invariant layouts will not change
 * @author cashhollister
 */
public class DayEvent extends JPanel {
    Color color = Color.BLACK;

    /**
     * Constructor utilized to create day event view
     * @preconditons existing JFrame and event
     * @postconditiond Event display created for event happening today
     * @param event
     * @param frame
     */
    public DayEvent(Event event, JFrame frame) {
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        // generate labels for each detail
        JLabel title = new JLabel(event.getTitle());
        JLabel course = new JLabel(event.getCourse());
        JLabel time = new JLabel(event.getTime());
        JLabel description = new JLabel(event.getDescription());
        JLabel priority = new JLabel(event.getPriority());
        JLabel status = new JLabel(event.getStatus());


        // generate details button
        JButton detailsButton = new JButton("details");

        // add detail labels
        add(title);
        add(course);
        add(time);
        add(description);
        add(priority);
        add(status);

        // handle complete status
        if (event.getComplete()) {
            color = Color.GREEN;
            JLabel complete = new JLabel("Completed");
            add(complete);
        } else {
            JLabel complete = new JLabel("Not Completed");
```

```
        add(complete);
    }

    // handle conflict status
    if (event.getConflict()) {
        color = Color.RED;
        JLabel conflict = new JLabel("Alert: Conflict with event");
        add(conflict);
    } else {
        JLabel conflict = new JLabel("No Conflicts with event");
        add(conflict);
    }

    // add details button
    add(detailsButton);

    // add space to bottom
    add(Box.createVerticalStrut(20));

    //set color based on conditions of details
    for (Component component : this.getComponents()) {
        component.setForeground(color);
    }

    detailsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            new EventDetailsController(event, frame);
        }
    });
}
/**
 * Public method utilized to pain the grid component
 * @preconditions none
 * @postconditions grid is painted
 */
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;

    // Generate border
    g2.setColor(this.color);
    g2.draw(new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 1));
```

```
    }
}
```

## MonthEvent.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import cop4331.EventComposite.Event;
import cop4331.gui.EventDetailsController;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Rectangle2D;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * Class utilized to display events happening this month
 * @invariant layouts will not change
 * @author cashhollister
 */
public class MonthEvent extends JPanel {
    Color color = Color.BLACK;

    /**
     * Constructor utilized to create month event view
     * @preconditons existing JFrame and event
     * @postconditiond Event display created for event happening this month
     * @param event
```

```
 * @param frame
 */
public MonthEvent(Event event, JFrame frame) {
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

    // generate labels for each detail
    JLabel title = new JLabel(event.getTitle());

    // generate details button
    JButton detailsButton = new JButton("details");

    // add detail labels
    add(title);

    // handle complete status
    if (event.getComplete()) {
        color = Color.GREEN;
    }

    // handle conflict status
    if (event.getConflict()) {
        color = Color.RED;
    }

    // add details button
    add(detailsButton);

    // add space to bottom
    add(Box.createVerticalStrut(20));

    //set color based on conditions of details
    for (Component component : this.getComponents()) {
        component.setForeground(color);
    }

    detailsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            new EventDetailsController(event, frame);
        }
    });
}

/**
```

```
     * Public method utilized to pain the grid component
     * @preconditions none
     * @postconditions grid is painted
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        // Generate border
        g2.setColor(this.color);
        g2.draw(new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 5));
    }
}
```

## MonthlyViewStrategy.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import cop4331.CalendarStrategy.MonthEvent;
import cop4331.CalendarStrategy.CalendarViewStrategy;
import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.Event;

import javax.swing.*;
import java.awt.*;
import java.time.LocalDate;
import java.util.List;
import cop4331.CurrentDateDecorator.CurrentDate;
import cop4331.CurrentDateDecorator.CurrentDay;
import cop4331.CurrentDateDecorator.EventDate;
import cop4331.SharedViews.GridBox;

/**
 * Concrete strategy for creating the monthly calendar view.
 * @author andrewcoggins
```

```
 */

public class MonthlyViewStrategy implements CalendarViewStrategy {
    /**
     * {@inheritDoc}
     *
     * Creates and returns a JPanel representing the monthly calendar view.
     */
    @Override
    public JPanel createCalendarView(JFrame frame) {
        JPanel calendarContainerPanel = new JPanel(new FlowLayout());
        JPanel calendarCompsPanel = new JPanel(new GridLayout(5, 7));
        calendarContainerPanel.add(calendarCompsPanel);

        PlannerSystem systemInstance = PlannerSystem.getInstance();
        UserAccount userAccount = systemInstance.getUserAccount();
        List<Event> accountEvents = userAccount.getEvents();

        CurrentDate currentDate = new EventDate();
        CurrentDate eventCurrentDay = new CurrentDay(currentDate);
        String todayDate = eventCurrentDay.getCurrentDays().get(0).substring(0,2);
        List<LocalDate> monthDates = currentDate.getDates();

        for (int x = 0; x < monthDates.size(); x++) {
            String currDate = monthDates.get(x).toString().substring(8,10);
            JLabel dateString = new JLabel(currDate);

            // change color based on current day
            if (currDate.substring(0,2).equals(todayDate)) {
                dateString.setForeground(Color.BLUE);
            }

            // generate grid box for each day
            GridBox box = new GridBox(dateString, 150, 150);

            currDate = currDate.substring(0, 2);
            for (Event event : accountEvents) {
                String eventDate = event.getDate().toString().substring(8,10);
                if (currDate.equals(eventDate)){
                    MonthEvent monthEvent = new MonthEvent(event, frame);
                    box.addEvent(monthEvent);
                }
            }
```

```
            calendarCompsPanel.add(box);
        }

        return calendarContainerPanel;
    }
}
```

## WeekEvent.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import cop4331.gui.EventDetailsController;
import cop4331.EventComposite.Event;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Rectangle2D;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BoxLayout;
import javax.swing.Box;
import javax.swing.JFrame;

/**
 * Class utilized to display events happening this week
 * @invariant layouts will not change
 * @author cashhollister
 */
public class WeekEvent extends JPanel {
    private Color color;

    /**
     * Constructor utilized to create week day event view
```

```
 * @preconditons existing JFrame and event
 * @postconditiond Event display created for event happening this week
 * @param event
 * @param frame
 */
public WeekEvent(Event event, JFrame frame) {
    this.color = Color.BLACK;
    setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

    // generate labels for each detail
    JLabel title = new JLabel(event.getTitle());
    JLabel course = new JLabel(event.getCourse());
    JLabel time = new JLabel(event.getTime());

    // generate details button
    JButton detailsButton = new JButton("details");

    // add detail labels
    add(title);
    add(course);
    add(time);

    // handle complete status
    if (event.getComplete()) {
        color = Color.GREEN;
        JLabel complete = new JLabel("Completed");
        add(complete);
    } else {
        JLabel complete = new JLabel("Not Completed");
        add(complete);
    }

    // handle conflict status
    if (event.getConflict()) {
        color = Color.RED;
        JLabel conflict = new JLabel("Alert: Conflict with event");
        add(conflict);
    } else {
        JLabel conflict = new JLabel("No Conflicts with event");
        add(conflict);
    }

    // add details button
    add(detailsButton);
```

```
        // add space to bottom
        add(Box.createVerticalStrut(20));

        //set color based on conditions of details
        for (Component component : this.getComponents()) {
            component.setForeground(color);
        }

        detailsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                new EventDetailsController(event, frame);
            }
        });
    }
    /**
     * Public method utilized to pain the grid component
     * @preconditions none
     * @postconditions grid is painted
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        // Generate border
        g2.setColor(this.color);
        g2.draw(new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 1));
    }
}
```

## WeeklyViewStrategy.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CalendarStrategy;

import cop4331.System.PlannerSystem;
```

```java
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.Event;

import javax.swing.*;
import java.awt.*;
import java.util.List;
import cop4331.CurrentDateDecorator.CurrentDate;
import cop4331.CurrentDateDecorator.CurrentDay;
import cop4331.CurrentDateDecorator.CurrentWeek;
import cop4331.CurrentDateDecorator.EventDate;
import cop4331.SharedViews.GridBox;

/**
 * Concrete strategy for creating the weekly calendar view.
 * @author andrewcoggins
 */

public class WeeklyViewStrategy implements CalendarViewStrategy {
    /**
     * {@inheritDoc}
     *
     * Creates and returns a JPanel representing the weekly calendar view.
     */
    @Override
    public JPanel createCalendarView(JFrame frame) {
        JPanel weekContainerPanel = new JPanel(new FlowLayout());
        JPanel weekCompsPanel = new JPanel(new GridLayout(0, 7));
        weekContainerPanel.add(weekCompsPanel);

        PlannerSystem systemInstance = PlannerSystem.getInstance();
        UserAccount userAccount = systemInstance.getUserAccount();
        List<Event> accountEvents = userAccount.getEvents();

        // decorator pattern to get the String value corrisponding to today
        CurrentDate currentDate = new EventDate();
        CurrentDate eventCurrentWeek = new CurrentWeek(currentDate);
        List<String> weekDays = eventCurrentWeek.getCurrentDays();

        // get today's date w/ CurrentDay decorator
        CurrentDate eventCurrentDay = new CurrentDay(currentDate);
        String todayDate = eventCurrentDay.getCurrentDays().get(0).substring(0,2);

        for (int x = 0; x < weekDays.size(); x++) {
            String currDate = weekDays.get(x);
```

```
            JLabel dateString = new JLabel(currDate);

            if (currDate.substring(0,2).equals(todayDate)) {
                dateString.setForeground(Color.BLUE);
            }

            GridBox box = new GridBox(dateString, 120, 800);

            currDate = currDate.substring(0, 2);
            for (Event event : accountEvents) {
                String eventDate = event.getDate().toString().substring(8,10);
                if (currDate.equals(eventDate)){
                    WeekEvent weekEvent = new WeekEvent(event, frame);
                    box.addEvent(weekEvent);
                }
            }

            weekCompsPanel.add(box);
        }

        return weekContainerPanel;
    }
}
```

# CurrentDateDecorator/

## CurrentDate.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CurrentDateDecorator;

import java.time.LocalDate;
import java.util.List;

/**
 * Interface for the CurrentDate Decorator
```

```
 * @invariant the list of times, currentMonth, and numberOfDays remain constant after
object creation
 * @author cashhollister
 */
public interface CurrentDate {

    /**
     * Method utilized to add dates to the dates List
     * @preconditions none
     * @postconditions list of dates generated
     */
    public void generateDates();

    /**
     * Method utilized to get current dates in month
     * @preconditions none
     * @postconditions array list of dates
     * @return dates
     */
    public List<LocalDate> getDates();

    /**
     * Method utilized to get times initialized locally
     * @preconditions none
     * @postconditions list of valid times
     * @return List<String> times
     */
    public List<String> getTimes();

    /**
     * Method utilized to get string representing today
     * @preconditions none
     * @postconditions string consisting of date and day of week
     * @return String today
     */
    public List<String> getCurrentDays();

}
```

## CurrentDay.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CurrentDateDecorator;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

/**
 * Class that acts as a Decorator for the EventDate specifying the current day
 * @author cashhollister
 */
public class CurrentDay implements CurrentDate {
    private CurrentDate eventDate;

    /**
     * Constructor to create EventCurrentDay Object
     * @preconditions none
     * @postconditions EventCurrentDay Object created
     * @param eventDate
     */
    public CurrentDay(CurrentDate eventDate) {
        this.eventDate = eventDate;
    }

    /**
     * Method utilized to add dates to the dates List
     * @preconditions none
     * @postconditions list of dates generated
     */
    @Override
    public void generateDates() {
        eventDate.generateDates();
    }

    /**
     * Method utilized to get current dates in month
     * @preconditions none
     * @postconditions array list of dates
     * @return dates
     */
    @Override
    public List<LocalDate> getDates() {
```

```
        return eventDate.getDates();
    }

    /**
     * Method utilized to get times initialized locally
     * @preconditions none
     * @postconditions list of valid times
     * @return List<String> times
     */
    @Override
    public List<String> getTimes() {
        return eventDate.getTimes();
    }

    /**
     * Method utilized to get string values of current day
     * @preconditions none
     * @postconditions get string values of current week
     * @return List<String> thisWeek
     */
    @Override
    public List<String> getCurrentDays() {
        List<String> weekDays = eventDate.getCurrentDays();
        List<String> currentDay = new ArrayList<>();

        // init todayValue
        int todayValue;
        if (LocalDate.now().getDayOfWeek().getValue() == 7) {
            todayValue = 0;
        } else {
            todayValue = LocalDate.now().getDayOfWeek().getValue();
        }
        String today = weekDays.get(todayValue);
        currentDay.add(today);

        return currentDay;
    }
}
```

## CurrentWeek.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CurrentDateDecorator;

import java.time.LocalDate;
import java.util.List;

/**
 * Class that acts as the Decorator for the EventDate Object for current week
 * @author cashhollister
 */
public class CurrentWeek implements CurrentDate {
    private CurrentDate eventDate;

    /**
     * Constructor to create EventCurrentWeek Object
     * @preconditions none
     * @postconditions EventCurrentWeek Object created
     * @param eventDate
     */
    public CurrentWeek(CurrentDate eventDate) {
        this.eventDate = eventDate;
    }

    /**
     * Method utilized to add dates to the dates List
     * @preconditions none
     * @postconditions list of dates generated
     */
    @Override
    public void generateDates() {
        eventDate.generateDates();
    }

    /**
     * Method utilized to get current dates in month
     * @preconditions none
     * @postconditions array list of dates
     * @return dates
     */
    @Override
```

```
    public List<LocalDate> getDates() {
        return eventDate.getDates();
    }

    /**
     * Method utilized to get times initialized locally
     * @preconditions none
     * @postconditions list of valid times
     * @return List<String> times
     */
    @Override
    public List<String> getTimes() {
        return eventDate.getTimes();
    }

    /**
     * Method utilized to get string values of current week
     * @preconditions none
     * @postconditions get string values of current week
     * @return List<String> thisWeek
     */
    @Override
    public List<String> getCurrentDays() {
        return eventDate.getCurrentDays();
    }
}
```

## EventDate.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.CurrentDateDecorator;

import java.time.LocalDate;
import java.time.YearMonth;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
```

```
 * Class the functions as the concrete component of the CurrentDate Decorator
 * @author cashhollister
 */
public class EventDate implements CurrentDate {
    private YearMonth currentMonth;
    private int numberOfDays;
    private List<LocalDate> dates;
    private List<String> times = Arrays.asList(
            "00:00 AM",
            "01:00 AM",
            "02:00 AM",
            "03:00 AM",
            "04:00 AM",
            "05:00 AM",
            "06:00 AM",
            "07:00 AM",
            "08:00 AM",
            "09:00 AM",
            "10:00 AM",
            "11:00 AM",
            "12:00 PM",
            "01:00 PM",
            "02:00 PM",
            "03:00 PM",
            "04:00 PM",
            "05:00 PM",
            "06:00 PM",
            "07:00 PM",
            "08:00 PM",
            "09:00 PM",
            "10:00 PM",
            "11:00 PM"
    );

    /**
     * Constructor to create CurrentMonth Object
     * @preconditions none
     * @postconditions CurrentMonth Object created
     */
    public EventDate() {
        this.currentMonth = YearMonth.now();
        this.numberOfDays = currentMonth.lengthOfMonth();
        this.dates = new ArrayList<>();
    }

    /**
```

```
     * Method utilized to add dates to the dates List
     * @preconditions none
     * @postconditions list of dates generated
     */
    @Override
    public void generateDates() {
        for (int x = 1; x <= numberOfDays; x++) {
            this.dates.add(currentMonth.atDay(x));
        }
        System.out.println("month " + currentMonth);
    }

    /**
     * Method utilized to get current dates in month
     * @preconditions none
     * @postconditions array list of dates
     * @return dates
     */
    @Override
    public List<LocalDate> getDates() {
        this.generateDates();
        System.out.println("dates " + dates);
        return this.dates;
    }

    /**
     * Method utilized to get times initialized locally
     * @preconditions none
     * @postconditions list of valid times
     * @return List<String> times
     */
    @Override
    public List<String> getTimes() {
        return this.times;
    }

    /**
     * Method utilized to get string values of current week
     * @preconditions none
     * @postconditions get string values of current week
     * @return List<String> thisWeek
     */
    @Override
    public List<String> getCurrentDays() {
        // empty list to contain days of this week
        List<String> thisWeek = new ArrayList<>();
```

```
        // init start date
        LocalDate start;
        if (LocalDate.now().getDayOfWeek().getValue() == 7) {
            // ensures that the day of the week will always start on the most recent sunday
            start = LocalDate.now();
        } else {
            // ensures that the day of the week will always start on the most recent sunday
            // finds current day thurs 28 => walkes back the days based on thursdays day
value (4) => returns sunday 24th
            start = LocalDate.now().minusDays(LocalDate.now().getDayOfWeek().getValue());
        }
        // appends the day of the week to thisWeek until the currDay == end (Sunday)
            LocalDate currDay = start;
            for (int x = 0; x < 7; x++) {
                String dayToAdd = currDay.toString().substring(8, 10) + " - " +
currDay.getDayOfWeek().toString();
                thisWeek.add(dayToAdd);
                currDay = currDay.plusDays(1);
            }

            return thisWeek;

    }
}
```

# EventComposite/

## ConflictsCheck.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
template
 */
package cop4331.EventComposite;

import java.util.ArrayList;

/**
```

```
 * Class utilized to check for conflicts in the user's Planner instance Composite Pattern
Composite
 * @author cashhollister
 */
public class ConflictsCheck implements EventConflict{
    private ArrayList<Event> events;

    /**
     * Constructor that generates an Empty array list
     */
    public ConflictsCheck() {
        this.events = new ArrayList<>();
    }

    /**
     * Public method to add event to Conflict Check list of events
     * @param event
     */
    public void add(Event event) {
        this.events.add(event);
    }

    /**
     * Public method to check for conflicts in planner
     * @preconditions none
     * @postconditions conflict status returned
     * @return Boolean conflict
     */
    @Override
    public boolean getConflict() {
        boolean isConflict = false;
        for (Event event : events) {
            isConflict = event.getConflict();
            if (isConflict) {
                return isConflict;
            }
        }
        return isConflict;
    }
}
```

## Event.java

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.EventComposite;
import java.io.Serializable;
import java.time.LocalDate;

/**
 * Class utilized to store Event data
 * @invariant types of instance variable will be constant
 * @author cashhollister
 */
public class Event implements Serializable, EventConflict {
    private String title;
    private LocalDate date;
    private String time;
    private String course;
    private String description;
    private String priority;
    private String status;
    private boolean complete;
    private boolean conflict;
   /**
    * Constructor utilized to create Event instance
    * @preconditions none
    * @postconditions Event created
    * @param title
    * @param date
    * @param time
    * @param course
    * @param description
    * @param priority
    * @param status
    * @param complete
    * @param conflict
    */
    public Event(String title, LocalDate date, String time, String course, String description,
            String priority, String status, boolean complete, boolean conflict) {
        this.title = title;
        this.date = date;
        this.time = time;
```

```
        this.course = course;
        this.description = description;
        this.priority = priority;
        this.status = status;
        this.complete = complete;
        this.conflict = conflict;
    }

    /**
     * Public method to check for conflicts in Events
     * @preconditions Event input
     * @postconditions returns Boolean based on conflict status
     * @param other
     * @return Boolean
     */
    public boolean checkConflict(Event other) {
        LocalDate otherDate = other.getDate();
        String otherTime = other.getTime();
        return (otherDate.equals(this.date) && otherTime.equals(this.time));
    }

    /**
     * Public method to set title
     * @preconditions newTitle String
     * @postconditions title changed for Event
     * @param newTitle
     */
    public void setTitle(String newTitle) {
        this.title = newTitle;
    }

    /**
     * Public method to set date
     * @preconditions newDate Date
     * @postconditions date changed for Event
     * @param newDate
     */
    public void setDate(LocalDate newDate) {
        this.date = newDate;
    }

    /**
     * Public method to set time
     * @preconditions newTime LocalTime
```

```
     * @postconditions time changed for Event
     * @param newTime
     */
    public void setTime(String newTime) {
        this.time = newTime;
    }

    /**
     * Public method to set course
     * @preconditions newCourse String
     * @postconditions course changed for Event
     * @param newCourse
     */
    public void setCourse(String newCourse) {
        this.course = newCourse;
    }

    /**
     * Public method to set description
     * @preconditions newDescription String
     * @postconditions description changed for Event
     * @param newDescription
     */
    public void setDescription(String newDescription) {
        this.description = newDescription;
    }

    /**
     * Public method to set priority
     * @preconditions newPriority String
     * @postconditions priority changed for Event
     * @param newPriority
     */
    public void setPriority(String newPriority) {
        this.priority = newPriority;
    }

    /**
     * Public method to set status
     * @preconditions newStatus Boolean
     * @postconditions status changed for Event
     * @param newStatus
     */
    public void setStatus(String newStatus) {
```

```
    this.status = newStatus;
}

/**
 * Public method to set complete
 * @preconditions newComplete Boolean
 * @postconditions complete changed for Event
 * @param newComplete
 */
public void setComplete(boolean newComplete) {
    this.complete = newComplete;
}

/**
 * Public method to set conflict
 * @preconditions newConflict Boolean
 * @postconditions conflict changed for Event
 * @param newConflict
 */
public void setConflict(boolean newConflict) {
    this.conflict = newConflict;
}

/**
 * Public method to get title
 * @preconditions none
 * @postconditions title returned
 * @return String title
 */
public String getTitle() {
    return this.title;
}

/**
 * Public method to get date
 * @preconditions none
 * @postconditions date returned
 * @return Date date
 */
public LocalDate getDate() {
    return this.date;
}

/**
```

```
     * Public method to get time
     * @preconditions none
     * @postconditions time returned
     * @return LocalTime time
     */
    public String getTime() {
        return this.time;
    }

    /**
     * Public method to get course
     * @preconditions none
     * @postconditions course returned
     * @return String course
     */
    public String getCourse() {
        return this.course;
    }

    /**
     * Public method to get description
     * @preconditions none
     * @postconditions description returned
     * @return String description
     */
    public String getDescription() {
        return this.description;
    }

    /**
     * Public method to get priority
     * @preconditions none
     * @postconditions priority returned
     * @return String priority
     */
    public String getPriority() {
        return this.priority;
    }

    /**
     * Public method to get status
     * @preconditions none
     * @postconditions status returned
     * @return Boolean status
```

```
     */
    public String getStatus() {
        return this.status;
    }

    /**
     * Public method to get complete
     * @preconditions none
     * @postconditions complete returned
     * @return Boolean complete
     */
    public boolean getComplete() {
        return this.complete;
    }

    /**
     * Public method to get conflict
     * @preconditions none
     * @postconditions conflict returned
     * @return Boolean conflict
     */
    @Override
    public boolean getConflict() {
        return this.conflict;
    }
}
```

## EventConflict.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Interface.java to edit this template
 */
package cop4331.EventComposite;


/**
 * Interface utilized for the Composite Pattern EventConflicts
 * @author cashhollister
 */
public interface EventConflict {
```

```
    /**
     * Public method to get conflict
     * @preconditions none
     * @postconditions conflict returned
     * @return Boolean conflict
     */
    public boolean getConflict();

}
```

## SharedModels/

## UserAccount.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.SharedModels;
import cop4331.EventComposite.Event;
import java.io.Serializable;
import java.util.List;

/**
 * Class utilized to store User Account data
 * @invariant events will be type Event, username and password will be Strings
 * @author cashhollister
 */
public class UserAccount implements Serializable {
    private String username;
    private String password;
    private List<Event> events;

    /**
     * Constructor utilized to create UserAccount Instance
     * @preconditions none
     * @postconditions UserAccount created
     * @param username
     * @param password
```

```
     * @param events
     */
    public UserAccount(String username, String password, List<Event> events) {
        this.username = username;
        this.password = password;
        this.events = events;
    }

    /**
     * Public method to set username
     * @preconditons newUsername type String
     * @postconditions username is changed
     * @param newUsername
     */
    public void setUsername(String newUsername) {
        this.username = newUsername;
    }

    /**
     * Public method to set password
     * @preconditons newPassword type String
     * @postconditions password is changed
     * @param newPassword
     */
    public void setPassword(String newPassword) {
        this.password = newPassword;
    }

    /**
     * Public method to add Event
     * @preconditons event type Event
     * @postconditions event is added
     * @param newEvent
     */
    public void addEvent(Event newEvent) {
        this.events.add(newEvent);
    }

    /**
     * Public method to remove Event
     * @preconditons event type Event
     * @postconditions event is removed from events
     * @param removedEvent
     */
```

```
    public void removeEvent(Event removedEvent) {
        this.events.remove(removedEvent);
    }

    /**
     * Public method to get username
     * @preconditons none
     * @postconditions username is returned
     * @return String username
     */
    public String getUsername() {
        return this.username;
    }

    /**
     * Public method to get password
     * @preconditons none
     * @postconditions password is returned
     * @return String password
     */
    public String getPassword() {
        return this.password;
    }

    /**
     * Public method to get events
     * @preconditons none
     * @postconditions events is returned
     * @return List<Event> events
     */
    public List<Event> getEvents() {
        return this.events;
    }
}
```

## SharedViews/

## GridBox.java

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

package cop4331.SharedViews;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Rectangle2D;

/**
 * Class utilized to create grid box calendar components
 *
 * @author cashhollister, andrewcoggins
 */
public class GridBox extends JPanel {
    private Color color = Color.BLACK;
    private JPanel eventPanel;
    private JScrollPane scrollPane;

    /**
     * Constructor Utilized to create GridBox
     * @preconditions none
     * @postconditions GridBox created
     * @param width
     * @param height
     * @param dayString
     */
    public GridBox(JLabel dayString, int width, int height) {
        setPreferredSize(new Dimension(width, height));
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));

        eventPanel = new JPanel();
        eventPanel.setLayout(new BoxLayout(eventPanel, BoxLayout.Y_AXIS));

        eventPanel.add(dayString);
        eventPanel.add(Box.createVerticalStrut(10));

        scrollPane = new JScrollPane(eventPanel);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

        add(scrollPane, BorderLayout.CENTER);
    }

    /**
     * Public method to add events to the Event Panel
     * @preconditions none
     * @postconditions events added to eventPanel and are able to be scrolled
     * @param component
     */
    public void addEvent(Component component) {
        eventPanel.add(component);
        // re-calcs the dimenstions and locations of items in the eventPanel
        eventPanel.revalidate();
        eventPanel.repaint();
    }

    /**
     * Public method utilized to pain the grid component
     * @preconditions none
     * @postconditions grid is painted
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        // Generate border
        g2.setColor(color);
        g2.draw(new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 1));
    }
}
```

## TaskBarController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
```

```
 */
package cop4331.SharedViews;

import cop4331.CalendarStrategy.MonthlyViewStrategy;
import cop4331.CalendarStrategy.CalendarViewStrategy;
import cop4331.gui.MainInterfaceController;
import cop4331.gui.LoginController;
import cop4331.gui.CalendarController;
import cop4331.gui.AddEventController;
import cop4331.gui.UserProfileController;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JFrame;


/**
 * Class utilized to contain the task bar navigation
 * @invariant the currentFrame never changes after creation
 * @author cashhollister
 */
public class TaskBarController extends JPanel{
    private JFrame currentFrame;

    public TaskBarController(JFrame frame) {
        this.currentFrame = frame;

        // Create the taskbar panel
        JPanel taskbarPanel = new JPanel();
        taskbarPanel.setLayout(new GridLayout(5, 1));

        // Create taskbar buttons
        JButton homeButton = new JButton("Main Interface");
        JButton userProfileButton = new JButton("User Profile");
        JButton addEventsButton = new JButton("Add Events");
        JButton calendarButton = new JButton("Calendar");
        JButton logoutButton = new JButton("Logout");

        // Add buttons to the taskbar panel
        taskbarPanel.add(homeButton);
        taskbarPanel.add(userProfileButton);
        taskbarPanel.add(addEventsButton);
```

```
taskbarPanel.add(calendarButton);
taskbarPanel.add(logoutButton);

this.add(taskbarPanel);

// Add action listeners to the buttons
homeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        currentFrame.dispose();
        new MainInterfaceController();
    }
});

userProfileButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        currentFrame.dispose();
        new UserProfileController();
    }
});

addEventsButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        currentFrame.dispose();
        new AddEventController();
    }
});

calendarButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        currentFrame.dispose();
        CalendarViewStrategy strategy = new MonthlyViewStrategy();
        new CalendarController(strategy);
    }
});

logoutButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        currentFrame.dispose();
        new LoginController();
```

```
        }
    });
  }

}
```

# System/

## PlannerSystem.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.System;

import cop4331.SharedModels.UserAccount;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.io.File;

/**
 * Utilizes singleton pattern to handle system operations
 * @invariant only one instance can be created
 * @author cashhollister
 */
public class PlannerSystem {
    private UserAccount userAccount;
    // create an instance of PlannerSystem that is accessed by users
    private static PlannerSystem instance = new PlannerSystem();

    /**
     * Private Constructor utilized to restrict instance creation
     * @preconditions none
     * @postconditions none
```

```
     */
    private PlannerSystem() {
        this.userAccount = null;
    }

    /**
     * Public method utilized to get instance rather than creating an instance w/
Constructor
     * @preconditions none
     * @postconditions instance returned
     * @return PlannerSystem
     */
    public static PlannerSystem getInstance() { return instance; }

    /**
     * Public Method to get the user account saved in memory
     * @precondition the account must be retrieved after creds are verified
     * @postconditions the account is accessed
     * @return UserAccount
     */
    public UserAccount getUserAccount() {
        return this.userAccount;
    }

    /**
     * Public method utilized to change a user's credentials
     * @preconditions new username as string, new passwd as string, UserAccount loaded
in
     * @postconditions the account username and password are changed
     * @param newUsername
     * @param newPassword
     */
    public void changeUserCredentials(String newUsername, String newPassword) {
        if (this.userAccount != null) {
            this.userAccount.setUsername(newUsername);
            this.userAccount.setPassword(newPassword);
            this.saveUserAccount(this.userAccount);
            System.out.println("User Account credentials changed: "  + newUsername + ":" +
newPassword);
        } else {
            System.out.println("No UserAccount found please login");
        }
    }
```

```java
    /**
     * Public method utilized to delete the stored UserAccount
     * @preconditions a user_account.dat file
     * @postconditions the user_account.dat file is deleted
     */
    public void deleteAccount() {
        File file = new File("user_account.dat");
        if (file.exists()) {
            boolean isDeleted = file.delete();
            if (isDeleted) {
                System.out.println("File deleted successfully.");
                this.userAccount = null; // clear userAccount from System instance
            } else {
                System.out.println("Error: File could not be deleted.");
            }
        } else {
            System.out.println("Error: File does not exist.");
        }
    }

    /**
     * Public method to validate a user login
     * @preconditions username / passwd input and user_account.dat file
     * @postconditions the system returns whether the inputs match saved data and saves
UserAccount to System instance if validated
     * @param currUsername
     * @param currPassword
     * @return validation status Boolean
     */
    public boolean validateUser(String currUsername, String currPassword) {
        UserAccount savedAccount = this.retrieveUserAccount();
        String savedUsername = savedAccount.getUsername();
        String savedPassword = savedAccount.getPassword();

        if (currUsername.equals("") || currPassword.equals("")) {
            return false;
        }
        if (currUsername.equals(savedUsername) &&
currPassword.equals(savedPassword)) {
            this.userAccount = savedAccount; // If validated creds save the UserAccount to
system instance
            return true;
        }
        return false;
```

```
}

/**
 * Public method utilized to save a UserAccount to the user_account.dat file
 * @preconditions valid UserAccount input
 * @postconditions user_account.dat file created with UserAccount
 * @param savedUserAccount
 */
public void saveUserAccount(UserAccount savedUserAccount) {
    try{
        ObjectOutputStream out = new ObjectOutputStream(
            new FileOutputStream("user_account.dat"));
        out.writeObject(savedUserAccount);
        out.close();
        this.userAccount = savedUserAccount;
        System.out.println("Saved userAccount to .dat file ");
    } catch (IOException e) {
        System.out.println("Error: unable to write UserAccount to .dat file ");
    }
}

/**
 * Private helper method to retrieve the UserAccount saved in the .dat file
 * @preconditions created user_account.dat file
 * @postconditions UserAccount is returned
 * @return UserAccount
 */
private UserAccount retrieveUserAccount() {
    try {
        ObjectInputStream in = new ObjectInputStream(
                new FileInputStream("user_account.dat"));
            UserAccount retrievedUserAccount = (UserAccount) in.readObject();
            in.close();
            return retrievedUserAccount;

    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Error: unable to read UserAccount from .dat file ");
        UserAccount empty = new UserAccount("", "", new ArrayList<>());
        return empty;

    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        System.out.println("Error: unable to find UserAccount class ");
```

```
        UserAccount empty = new UserAccount("", "", new ArrayList<>());
        return empty;

    }
  }
}
```

gui/

## AddEventController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cop4331.gui;

import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.Event;
import cop4331.SharedViews.TaskBarController;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.LocalDate;
import java.util.List;
import cop4331.CurrentDateDecorator.CurrentDate;
import cop4331.CurrentDateDecorator.EventDate;

/**
 * This class represents the window for adding new events to the calendar.
 * It provides a user interface with fields for entering event details.
 * @author cashhollister, andrewcogins
 */
public class AddEventController {
    private JPanel addEventPanel;
```

```java
/**
 * Constructs the AddEventWindow and initializes its components.
 */
public AddEventController() {
    JFrame frame = new JFrame();
    frame.setLayout(new BorderLayout());
    frame.setTitle("Add New Event");

    // Add the taskbar panel to the west of the frame
    TaskBarController taskbarPanel = new TaskBarController(frame);
    frame.add(taskbarPanel, BorderLayout.WEST);

    addEventPanel = new JPanel();
    addEventPanel.setLayout(new BorderLayout());

    // Create header panel
    JPanel headerPanel = new JPanel();
    headerPanel.setLayout(new BoxLayout(headerPanel, BoxLayout.Y_AXIS));

    headerPanel.add(Box.createVerticalStrut(20));
    headerPanel.add(Box.createVerticalStrut(20));

    // Create details panel
    JPanel detailsPanel = new JPanel();
    detailsPanel.setLayout(new BoxLayout(detailsPanel, BoxLayout.Y_AXIS));

    // Create and set up separate field panels
    JPanel titlePanel = new JPanel();
    titlePanel.setLayout(new BoxLayout(titlePanel, BoxLayout.X_AXIS));
    JPanel datePanel = new JPanel();
    datePanel.setLayout(new BoxLayout(datePanel, BoxLayout.X_AXIS));
    JPanel timePanel = new JPanel();
    timePanel.setLayout(new BoxLayout(timePanel, BoxLayout.X_AXIS));
    JPanel coursePanel = new JPanel();
    coursePanel.setLayout(new BoxLayout(coursePanel, BoxLayout.X_AXIS));
    JPanel descriptionPanel = new JPanel();
    descriptionPanel.setLayout(new BoxLayout(descriptionPanel, BoxLayout.X_AXIS));
    JPanel priorityPanel = new JPanel();
    priorityPanel.setLayout(new BoxLayout(priorityPanel, BoxLayout.X_AXIS));
    JPanel statusPanel = new JPanel();
    statusPanel.setLayout(new BoxLayout(statusPanel, BoxLayout.X_AXIS));
    JPanel completePanel = new JPanel();
    completePanel.setLayout(new BoxLayout(completePanel, BoxLayout.X_AXIS));
    JPanel conflictPanel = new JPanel();
```

```
conflictPanel.setLayout(new BoxLayout(conflictPanel, BoxLayout.X_AXIS));


// Separate panel components
// Event title
JLabel titleLabel = new JLabel("Title: ");
JTextField titleField = new JTextField();
// Event date
JLabel dateLabel = new JLabel("Date: ");
JComboBox<LocalDate> dateComboBox = new JComboBox<>();
// Event time
JLabel timeLabel = new JLabel("Time: ");
JComboBox<String> timeComboBox = new JComboBox<>();
// Event course
JLabel courseLabel = new JLabel("Course: ");
JTextField courseField = new JTextField();
// Event description
JLabel descriptionLabel = new JLabel("Description: ");
JTextArea descriptionField = new JTextArea();
descriptionField.setLineWrap(true);
// Event priority
JLabel priorityLabel = new JLabel("Priority: ");
JComboBox<String> priorityComboBox = new JComboBox<>();
// Event status
JLabel statusLabel = new JLabel("Status: ");
JComboBox<String> statusComboBox = new JComboBox<>();
// Event completion status
JLabel completeLabel = new JLabel("Complete: ");
JCheckBox completeBox = new JCheckBox();
// Event conflict status
JLabel conflictLabel = new JLabel("Conflict: ");
JCheckBox conflictBox = new JCheckBox();


// init CurrentDate object
CurrentDate currentDate = new EventDate();
// add dates to combo box
List<LocalDate> dateList = currentDate.getDates();
for (LocalDate date : dateList) {
    dateComboBox.addItem(date);
}

// add times to combo box
List<String> timeList = currentDate.getTimes();
```

```
for (String specificTime : timeList) {
    timeComboBox.addItem(specificTime);
}

// add priorities to combo box
priorityComboBox.addItem("Low");
priorityComboBox.addItem("Moderate");
priorityComboBox.addItem("High");

// add statuses to combo box
statusComboBox.addItem("Not Started");
statusComboBox.addItem("In Progress");
statusComboBox.addItem("Test and Analysis");
statusComboBox.addItem("Q and A");
statusComboBox.addItem("Complete");


// add components to specific panels
titlePanel.add(titleLabel);
titlePanel.add(titleField);
datePanel.add(dateLabel);
datePanel.add(dateComboBox);
timePanel.add(timeLabel);
timePanel.add(timeComboBox);
coursePanel.add(courseLabel);
coursePanel.add(courseField);
descriptionPanel.add(descriptionLabel);
descriptionPanel.add(descriptionField);
priorityPanel.add(priorityLabel);
priorityPanel.add(priorityComboBox);
statusPanel.add(statusLabel);
statusPanel.add(statusComboBox);
completePanel.add(completeLabel);
completePanel.add(completeBox);
conflictPanel.add(conflictLabel);
conflictPanel.add(conflictBox);

// add specific panels to details panel
detailsPanel.add(titlePanel);
detailsPanel.add(datePanel);
detailsPanel.add(timePanel);
detailsPanel.add(coursePanel);
detailsPanel.add(descriptionPanel);
detailsPanel.add(priorityPanel);
```

```
detailsPanel.add(statusPanel);
detailsPanel.add(completePanel);
detailsPanel.add(conflictPanel);


// Set maximum size
Dimension fieldSize = new Dimension(400, 30);
Dimension descriptionSize = new Dimension(400, 120);
titleField.setMaximumSize(fieldSize);
dateComboBox.setMaximumSize(fieldSize);
timeComboBox.setMaximumSize(fieldSize);
priorityComboBox.setMaximumSize(fieldSize);
statusComboBox.setMaximumSize(fieldSize);
courseField.setMaximumSize(fieldSize);
descriptionField.setMaximumSize(descriptionSize);


// Submit button
JButton addEvent = new JButton("Add Event");
detailsPanel.add(addEvent);

// Add panels to main panel
addEventPanel.add(headerPanel, BorderLayout.NORTH);
addEventPanel.add(detailsPanel, BorderLayout.CENTER);

// frame settings
frame.setSize(700, 700);
frame.add(addEventPanel);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
frame.setLocationRelativeTo(null);
frame.setVisible(true);

addEvent.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        PlannerSystem systemInstance = PlannerSystem.getInstance();
        UserAccount userAccount = systemInstance.getUserAccount();

        //get event inputs
        String eventTitle = titleField.getText();
        LocalDate eventDate = (LocalDate) dateComboBox.getSelectedItem();
        String eventTime = (String) timeComboBox.getSelectedItem();
        String eventCourse = courseField.getText();
        String eventDescription = descriptionField.getText();
```

```
            String eventPriority = (String) priorityComboBox.getSelectedItem();
            String eventStatus = (String) statusComboBox.getSelectedItem();
            boolean isEventComplete = completeBox.isSelected();
            boolean isEventConflict = conflictBox.isSelected();

            // create Event object w/ inputs
            Event newEvent = new Event(eventTitle, eventDate, eventTime, eventCourse,
eventDescription,
                    eventPriority, eventStatus, isEventComplete, isEventConflict);

            //get current account events
            List<Event> existingEvents = userAccount.getEvents();

            // if conflict update both events
            for (Event event : existingEvents) {
               if (event.checkConflict(newEvent)) {
                  event.setConflict(true);
                  newEvent.setConflict(true);
               }
            }

            // add event to stored UserAccount instance
            userAccount.addEvent(newEvent);

            // save account update to the system
            systemInstance.saveUserAccount(userAccount);

            frame.dispose();

            new MainInterfaceController();
         }
      });
   }
}
```

## CalendarController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
template
 */
```

```java
package cop4331.gui;

import cop4331.CalendarStrategy.CalendarViewStrategy;
import cop4331.CalendarStrategy.DailyViewStrategy;
import cop4331.CalendarStrategy.MonthlyViewStrategy;
import cop4331.SharedViews.TaskBarController;
import cop4331.CalendarStrategy.WeeklyViewStrategy;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class represents the calendar window, providing a user interface
 * for viewing and interacting with the calendar.
 *
 * @author cashhollister, andrewcogins
 */
public class CalendarController {

    private JFrame frame;
    private JPanel calendarPanel;
    private CardLayout cardLayout; // Make cardLayout a class member

    /**
     * Constructs the CalendarWindow with the specified view strategy.
     *
     * @param viewStrategy The strategy to use for creating the initial calendar view.
     */
    public CalendarController(CalendarViewStrategy viewStrategy) {
        frame = new JFrame();
        frame.setLayout(new BorderLayout());
        frame.setTitle("Calendar");

        // Add the taskbar panel to the west of the frame
        TaskBarController taskbarPanel = new TaskBarController(frame);
        frame.add(taskbarPanel, BorderLayout.WEST);

        // Create header panel
        JPanel headerPanel = new JPanel();
        headerPanel.setLayout(new BoxLayout(headerPanel, BoxLayout.X_AXIS));

        // Button panel
```

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1, 3));

// Create button panel components
JButton todayButton = new JButton("Today");
JButton weekButton = new JButton("Week");
JButton monthButton = new JButton("Month");

// Add components to button panel
buttonPanel.add(todayButton);
buttonPanel.add(weekButton);
buttonPanel.add(monthButton);

// Add button panel to header panel
headerPanel.add(buttonPanel);

// Add calendar view to main panel
calendarPanel = new JPanel(new CardLayout());
cardLayout = (CardLayout) calendarPanel.getLayout(); // Initialize cardLayout here
calendarPanel.add(viewStrategy.createCalendarView(frame), "InitialView");

// Frame settings
frame.add(calendarPanel, BorderLayout.CENTER);
frame.add(headerPanel, BorderLayout.NORTH);
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
frame.pack();
frame.setLocationRelativeTo(null);
frame.setVisible(true);

// Action listeners
todayButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Switch to the "Today" view
        CalendarViewStrategy dailyStrategy = new DailyViewStrategy();
        JPanel dailyViewPanel = dailyStrategy.createCalendarView(frame);

        calendarPanel.add(dailyViewPanel, "Today");
        cardLayout.show(calendarPanel, "Today"); // Use the class member cardLayout
    }
});

weekButton.addActionListener(new ActionListener() {
    @Override
```

```
            public void actionPerformed(ActionEvent e) {
                // Switch to the "Week" view
                CalendarViewStrategy weeklyStrategy = new WeeklyViewStrategy();
                JPanel weeklyViewPanel = weeklyStrategy.createCalendarView(frame);

                calendarPanel.add(weeklyViewPanel, "Week");
                cardLayout.show(calendarPanel, "Week"); // Use the class member cardLayout
            }
        });

        monthButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Switch to the "Month" view
                CalendarViewStrategy monthlyStrategy = new MonthlyViewStrategy();
                JPanel monthlyViewPanel = monthlyStrategy.createCalendarView(frame);

                calendarPanel.add(monthlyViewPanel, "Month");
                cardLayout.show(calendarPanel, "Month"); // Use the class member cardLayout
            }
        });
    }
}
```

## CreateAccountController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

package cop4331.gui;

import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
```

```
/**
 * This class represents the window for creating a new user account.
 * It provides a user interface with fields for entering username and password.
 *
 * @author cashhollister
 */
public class CreateAccountController {

    /**
     * Constructs the CreateAccountWindow and initializes its components.
     */
    public CreateAccountController() {
        // Initialize JFrame with GridLayout
        JFrame frame = new JFrame("Planner Application");
        frame.setLayout(new GridLayout(4, 1));

        JLabel prompt = new JLabel("Sign-Up:");


        // Create and set up field panel
        JPanel usernamePanel = new JPanel();
        usernamePanel.setLayout(new BoxLayout(usernamePanel, BoxLayout.X_AXIS));
        JPanel passwordPanel = new JPanel();
        passwordPanel.setLayout(new BoxLayout(passwordPanel, BoxLayout.X_AXIS));

        // create text input components
        JLabel usernameLabel = new JLabel("Username:");
        JTextField usernameField = new JTextField();
        JLabel passwordLabel = new JLabel("Password:");
        JPasswordField passwordField = new JPasswordField(); // Changed to
JPasswordField for passwords

        //add to field panel
        usernamePanel.add(usernameLabel);
        usernamePanel.add(usernameField);
        passwordPanel.add(passwordLabel);
        passwordPanel.add(passwordField);

        // Set maximum size
        Dimension fieldSize = new Dimension(400, 30);
        usernameField.setMaximumSize(fieldSize);
        passwordField.setMaximumSize(fieldSize);
```

```
        // Create and set up button panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));

        // create button components
        JButton createAccountButton = new JButton("Create Account");
        JButton loginButton = new JButton("Return to Login");

        // add to button panel
        buttonPanel.add(createAccountButton);
        buttonPanel.add(loginButton);


        // Add panels to the frame
        frame.add(prompt, BorderLayout.NORTH);
        frame.add(usernamePanel, BorderLayout.CENTER);
        frame.add(passwordPanel, BorderLayout.CENTER);
        frame.add(buttonPanel, BorderLayout.SOUTH);

        // Set frame properties and display
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(500,300);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);


        // Add action listener to the "Create Account" button
        createAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = usernameField.getText();
                char[] passwordChars = passwordField.getPassword(); // Use getPassword() for
JPasswordField
                String password = new String(passwordChars);

                // Save username and password (replace with preferred storage method)
                PlannerSystem systemInstance = PlannerSystem.getInstance();
                UserAccount newAccount = new UserAccount(username, password, new
ArrayList<>());
                systemInstance.saveUserAccount(newAccount);

                System.out.println("Username: " + username);
                System.out.println("Password: " + password);
```

```
            // Close the CreateAccountController and open LoginController
            frame.dispose();
            new LoginController();
        }
    });

    // Add action listener to the "Login" button
    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Close the CreateAccountController and open LoginController
            frame.dispose();
            new LoginController();
        }
    });
  }

}
```

## EventDetailsController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

package cop4331.gui;

import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.Event;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class represents the window for displaying detailed information about an event.
 *
```

```
 * @author cashhollister, andrewcoggins
 */
public class EventDetailsController {
    private JPanel detailsPanel;
    /**
     * Constructor to generate an Event Details Window
     * @precondtions none
     * @postconditions Window generated
     * @param event
     */
    public EventDetailsController(Event event, JFrame currentframe) {
        JFrame frame = new JFrame();
        frame.setLayout(new BorderLayout());
        frame.setTitle("Event Details");

        detailsPanel = new JPanel();
        detailsPanel.setLayout(new BoxLayout(detailsPanel, BoxLayout.Y_AXIS));

        // generate labels for each detail
        JLabel title = new JLabel(event.getTitle());
        JLabel course = new JLabel(event.getCourse());
        JLabel time = new JLabel(event.getTime());
        JLabel description = new JLabel(event.getDescription());
        JLabel priority = new JLabel(event.getPriority());
        JLabel status = new JLabel(event.getStatus());


        // generate details button
        JButton deleteButton = new JButton("Delete Event");

        // add detail labels
        detailsPanel.add(title);
        detailsPanel.add(course);
        detailsPanel.add(time);
        detailsPanel.add(description);
        detailsPanel.add(priority);
        detailsPanel.add(status);

        // handle complete status
        if (event.getComplete()) {
            JLabel complete = new JLabel("Completed");
            complete.setForeground(Color.GREEN);
            detailsPanel.add(complete);
        } else {
```

```
            JLabel complete = new JLabel("Not Completed");
            detailsPanel.add(complete);
        }

        // handle conflict status
        if (event.getConflict()) {
            JLabel conflict = new JLabel("Alert: Conflict with event");
            conflict.setForeground(Color.RED);
            detailsPanel.add(conflict);
        } else {
            JLabel conflict = new JLabel("No Conflicts with event");
            detailsPanel.add(conflict);
        }

        // add details button
        detailsPanel.add(deleteButton);

        // add space to bottom
        detailsPanel.add(Box.createVerticalStrut(20));

        frame.add(detailsPanel);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(300,300);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);


        deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                PlannerSystem systemInstance = PlannerSystem.getInstance();
                UserAccount userAccount = systemInstance.getUserAccount();
                userAccount.removeEvent(event);
                systemInstance.saveUserAccount(userAccount);
                frame.dispose();
                currentframe.dispose();
                new MainInterfaceController();
            }
        });
    }
}
```

## LoginController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */


package cop4331.gui;

import cop4331.System.PlannerSystem;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class represents the login window for the application.
 * It provides a user interface for entering username and password to log in.
 *
 * @author cashhollister, andrewcoggins
 */
public class LoginController {

    /**
     * Constructs the LoginWindow and initializes its components.
     */
    public LoginController() {
        // Initialize JFrame with GridLayout
        JFrame frame = new JFrame("Planner Application");
        frame.setLayout(new GridLayout(4, 1));

        JLabel prompt = new JLabel("Login:");

        // Create and set up field panels
        JPanel usernamePanel = new JPanel();
        usernamePanel.setLayout(new BoxLayout(usernamePanel, BoxLayout.X_AXIS));
        JPanel passwordPanel = new JPanel();
        passwordPanel.setLayout(new BoxLayout(passwordPanel, BoxLayout.X_AXIS));

        // create text input components
        JLabel usernameLabel = new JLabel("Username:");
```

```
        JTextField usernameField = new JTextField();
        JLabel passwordLabel = new JLabel("Password:");
        JPasswordField passwordField = new JPasswordField(); // Changed to
JPasswordField for passwords

        //add to field panel
        usernamePanel.add(usernameLabel);
        usernamePanel.add(usernameField);
        passwordPanel.add(passwordLabel);
        passwordPanel.add(passwordField);

        // Set maximum size
        Dimension fieldSize = new Dimension(400, 30);
        usernameField.setMaximumSize(fieldSize);
        passwordField.setMaximumSize(fieldSize);


        // Create and set up button panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));

        // create button components
        JButton loginButton = new JButton("Login");
        JButton signUpButton = new JButton("Sign-Up");

        // add to button panel
        buttonPanel.add(loginButton);
        buttonPanel.add(signUpButton);


        // Add panels to the frame
        frame.add(prompt, BorderLayout.NORTH);
        frame.add(usernamePanel, BorderLayout.CENTER);
        frame.add(passwordPanel, BorderLayout.CENTER);
        frame.add(buttonPanel, BorderLayout.SOUTH);

        // Set frame properties and display
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(500,300);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);


        // Add action listener to the login button
```

```
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Get username and password from the fields
                String username = usernameField.getText();
                char[] passwordChars = passwordField.getPassword();
                String password = new String(passwordChars);

                PlannerSystem systemInstance = PlannerSystem.getInstance();
                boolean validated = systemInstance.validateUser(username, password);

                // Perform authentication (replace with your actual authentication logic)
                if (validated) {
                    // If authentication is successful, close the LoginController and open
MainInterfaceController
                    frame.dispose();
                    new MainInterfaceController();
                } else {
                    // If authentication fails, display an error message
                    JOptionPane.showMessageDialog(frame, "Invalid username or password",
"Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        });
        signUpButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Close the LoginController and open CreateAccountController
                frame.dispose();
                new CreateAccountController();
            }
        });

    }
}
```

## MainInterfaceController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
template
```

```
 */

package cop4331.gui;

import cop4331.SharedViews.GridBox;
import cop4331.CalendarStrategy.WeekEvent;
import cop4331.EventComposite.Event;
import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;
import cop4331.EventComposite.ConflictsCheck;
import cop4331.SharedViews.TaskBarController;

import javax.swing.*;
import java.awt.*;
import java.util.List;
import cop4331.CurrentDateDecorator.CurrentDate;
import cop4331.CurrentDateDecorator.CurrentDay;
import cop4331.CurrentDateDecorator.CurrentWeek;
import cop4331.CurrentDateDecorator.EventDate;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class represents the main interface of the planner application.
 * It contains the taskbar and a central panel for displaying the current week.
 *
 * @author andrewcoggins Cash Hollister
 */
public class MainInterfaceController {
    private PlannerSystem systemInstance = PlannerSystem.getInstance();
    private UserAccount userAccount = systemInstance.getUserAccount();
    private List<Event> accountEvents = userAccount.getEvents();
    private JFrame frame;
    private JPanel mainPanel; // Main panel to hold the week view

    /**
     * Constructs the MainInterface and initializes its components.
     */
    public MainInterfaceController() {
        frame = new JFrame();
        frame.setLayout(new BorderLayout());
        frame.setTitle("Main Interface");

        // Add the taskbar panel to the west of the frame
        JPanel taskbarPanel = new JPanel(new GridLayout(5, 1));
        TaskBarController taskbar = new TaskBarController(frame);
```

```
        //conflic check buton
        JButton conflictCheck = new JButton("Check For Conflicts");

        // add action listener that notifies the user of any exisiting conflicts
        conflictCheck.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                List<Event> events = userAccount.getEvents();
                ConflictsCheck conflictCheck = new ConflictsCheck();

                for (Event event : events) {
                    conflictCheck.add(event);
                }
                boolean isConflict = conflictCheck.getConflict();

                if (isConflict) {
                    JOptionPane.showMessageDialog(frame, "Conflict Found Check Calendar
Page", "Error", JOptionPane.ERROR_MESSAGE);
                } else {
                    JOptionPane.showMessageDialog(frame, "No Conflicts Found",
"Information", JOptionPane.INFORMATION_MESSAGE);
                }
            }
        });

        taskbarPanel.add(taskbar);
        taskbarPanel.add(conflictCheck);
        frame.add(taskbarPanel, BorderLayout.WEST);

        // Create the main panel
        mainPanel = new JPanel(new FlowLayout());

        // Week container
        JPanel weekContainerPanel = new JPanel(new FlowLayout());
        JPanel weekCompsPanel = new JPanel(new GridLayout(1, 7));
        weekContainerPanel.add(weekCompsPanel);

        // Initialize CurrentDate Object
        // decorator pattern to get the String value corrisponding to today
        CurrentDate currentDate = new EventDate();
        CurrentDate eventCurrentWeek = new CurrentWeek(currentDate);
        // get today's date w/ CurrentDay decorator
        CurrentDate eventCurrentDay = new CurrentDay(currentDate);
        // Populate week components
        List<String> weekDays = eventCurrentWeek.getCurrentDays();
        String todayDate = eventCurrentDay.getCurrentDays().get(0).substring(0,2);
```

```
    for (int x = 0; x < weekDays.size(); x++) {
        try {
            String currDate = weekDays.get(x);
        JLabel dateString = new JLabel(currDate);

        // Change color based on current day
        if (currDate.substring(0,2).equals(todayDate)) {
            dateString.setForeground(Color.BLUE);
        }

        // Generate grid box for each day
        GridBox box = new GridBox(dateString, 150, 800);

        // Assign events to day of week
        assignWeekEvents(currDate, box);

        weekCompsPanel.add(box);
        } catch (Exception e) {
            System.out.println("failed at MainInterface week loop");
        }
    }

    // Add the week view to the main panel
    mainPanel.add(weekContainerPanel);

    // Frame settings
    frame.add(mainPanel, BorderLayout.CENTER); // Add mainPanel to the frame
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}

 /**
 * Assigns events to the appropriate day in the week view.
 *
 * @param currDay The current day's date string.
 * @param box The GridBox representing the day to which events should be assigned.
 */
private void assignWeekEvents(String currDay, GridBox box) {
    String currDate = currDay.substring(0, 2);

    for (Event event : this.accountEvents) {
        String eventDate = event.getDate().toString().substring(8,10);
        if (currDate.equals(eventDate)){
            WeekEvent weekEvent = new WeekEvent(event, frame);
```

```
                box.addEvent(weekEvent);
            }
        }
    }
}
```

## UserProfileController.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

package cop4331.gui;

import cop4331.SharedViews.TaskBarController;
import cop4331.System.PlannerSystem;
import cop4331.SharedModels.UserAccount;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class represents the user profile window for the application.
 * It displays user information and allows for changing credentials or deleting the account.
 *
 * @author cashhollister, andrewcogins
 */
public class UserProfileController {
    private JPanel userProfilePanel;
    private JFrame frame;
    PlannerSystem systemInstance = PlannerSystem.getInstance();
    private UserAccount userAccount = systemInstance.getUserAccount();

    /**
     * Constructs the UserProfileWindow with the given username and password.
     */
    public UserProfileController() {
```

```
frame = new JFrame();
frame.setTitle("User Profile Information");
frame.setLayout(new BorderLayout());

// Add the taskbar panel to the west of the frame
TaskBarController taskbarPanel = new TaskBarController(frame);
frame.add(taskbarPanel, BorderLayout.WEST);

// Initialize the userProfilePanel
userProfilePanel = new JPanel();
userProfilePanel.setLayout(new GridLayout(3, 1));

// Create and set up header panel
JPanel headerPanel = new JPanel();
headerPanel.setLayout(new GridLayout(4, 1));

// header panel components
JLabel usernameLabel = new JLabel("Username: " + userAccount.getUsername());
JLabel passwordLabel = new JLabel("Password: " + userAccount.getPassword());

// add componenets to header panel
headerPanel.add(usernameLabel);
headerPanel.add(passwordLabel);


// Create and set up field panel
JPanel fieldPanel = new JPanel();
fieldPanel.setLayout(new BoxLayout(fieldPanel, BoxLayout.Y_AXIS));

// Create and set up separate field panels
JPanel usernamePanel = new JPanel();
usernamePanel.setLayout(new BoxLayout(usernamePanel, BoxLayout.X_AXIS));
JPanel passwordPanel = new JPanel();
passwordPanel.setLayout(new BoxLayout(passwordPanel, BoxLayout.X_AXIS));

// field panel components
JLabel newUsernameLabel = new JLabel("New Username: ");
JTextField usernameField = new JTextField();
JLabel newPasswordLabel = new JLabel("New Password: ");
JTextField passwordField = new JPasswordField();

// add componenets to field panel
usernamePanel.add(newUsernameLabel);
usernamePanel.add(usernameField);
```

```
        passwordPanel.add(newPasswordLabel);
        passwordPanel.add(passwordField);

        // add separate panels to field panel
        fieldPanel.add(usernamePanel);
        fieldPanel.add(passwordPanel);

        // Set maximum size
        Dimension fieldSize = new Dimension(400, 30);
        usernameField.setMaximumSize(fieldSize);
        passwordField.setMaximumSize(fieldSize);


        // Create and set up button panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));

        JButton changeCredentialsButton = new JButton("Change Credentials");
        buttonPanel.add(changeCredentialsButton); //Name changed for clarity

        JButton deleteAccountButton = new JButton("Delete Account");
        buttonPanel.add(deleteAccountButton); //Name changed for clarity

        // Change UserAccount credentials based on text field input
        changeCredentialsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Change account credentials
                String newUsername = usernameField.getText();
                String newPassword = passwordField.getText();
                systemInstance.changeUserCredentials(newUsername, newPassword);
                frame.dispose();
                new UserProfileController();
                JOptionPane.showMessageDialog(userProfilePanel, "New Username:Password
is " + newUsername  + ":" + newPassword, "Information",
JOptionPane.INFORMATION_MESSAGE);
            }
        });

        // Delete UserAccount
        deleteAccountButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                systemInstance.deleteAccount();
```

```
            frame.dispose();
            new LoginController();
        }
    });
    // Add panels to the main panel
    userProfilePanel.add(headerPanel);
    userProfilePanel.add(fieldPanel);
    userProfilePanel.add(buttonPanel);

    // frame settings
    frame.setSize(800, 450);
    frame.add(userProfilePanel);
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);

  }
}
```

run/

## ExecuteApplication.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package cop4331.run;

import cop4331.gui.LoginController;
import javax.swing.SwingUtilities;

/**
 * Class utilized to kick off the Planner application
 * @invariant always executes the LoginController
 * @author cashhollister
 */
public class ExecuteApplication {
```

```
    /**
     * Main method utilized to run the Planner Application
     * @preconditions none
     * @postconditions application runs
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new LoginController());
    }
}
```