Changes

App as a Whole

Addition: Dependencies listed below

java.time.LocalDate, java.time.LocalDateTime, java.time.ZoneId, java.time.ZonedDateTime, java.time.format.DateTimeFormatter, java.time.format.DateTimeParseException, java.util.ArrayList, java.util.List, java.util.Scanner, java.util.Date, java.util.Calendar, java.util.GregorianCalendar.

Why: These aided in giving the calendar date and time functionalities without having to write everything from scratch and helped me get input from the user directly.

Why Unavoidable: If I would have not added these classes the application would have gone much out of the scope of the assignment or I would be unable to properly display data and take input from the user.

Addition: Private Attributes

Why: This would help me both debug and ensure that data is managed correctly.

Why Unavoidable: The issue with public attributes on a lot of these things would be that data might be overwritten when it is not intended to be overwritten. For this reason it was much easier and safer to make all attributes that were not interrelated be private.

App Class

Addition: main

Why: The main function is the entry point for the program and allows the user to access the menu and create their account without it there would be no user input.

Why Unavoidable: This gives users the ability to interact with the program at the start and access a crucial step in the program which is create or login into their account.

Addition: loginUser

Why: According to the requirements and uml class it is meant to be able to handle multiple users and share calendars or events with multiple users. Without a method to login a user would not be able to differentiate and distinct.

Why Unavoidable: A user would have otherwise not been able to be distinct or would need to manually validate that they were the right user to access a calendar or event.

Addition: logoutUser

Why: Since there are multiple users the user needs a way to switch between accounts and logging out of the application would be necessary. Additionally the application does not have persistent memory so it can't be closed and then logged into a different user since all data would have been lost.

Why Unavoidable: Without being able to logout the application would not have been able to support multiple users. There would have been no other way to switch between accounts on the application especially since there is no persistent memory.

Addition: printUsers

Why: This is a helper function that is used to validate that the user is logged into the correct account.

Why Unavoidable: The user would otherwise have no way of knowing if they did login to the correct account other than checking their events.

Addition: findUsers

Why: This is a helper function which will retrieve the user if the user is found.

Why Unavoidable: This function is necessary in order to ensure that users that are logging in have in fact created their account and are not just random users.

Addition: setTimeZones

Why: The user needs to be able to set time zones as per the requirements.

Why Unavoidable: With the previous implementation there was no method to do this and it was a key requirement of the implementation. Time zones should be able to be updated. There was no other way to do this for the whole user as a whole and no better place than the App class since this is the overarching entry point for the user on entrance to the application.

User Class

Addition: Constructor

Why: A user would need data to be imputed in order to function properly a constructor was key for this class.

Why Unavoidable: The constructor is the only method for creating multiple class instances.

Addition: userNavigation

Why: In order to allow the user to navigate through their account and select what action they would like to take on their account or calendars.

Why Unavoidable: In order to improve design, enhance modularity, and efficiency.

Addition: viewCalendar

Why: This function allows the user to select how they want to see their events displayed for today, the week, the month, or the entire year.

Why Unavoidable: It is unavoidable due to following the requirements of the assignment and taking input from the user.

Addition: findCalendar

Why: A helper function that searches through the calendar list and returns the found calendar.

Why Unavoidable: The helper function simplifies the tasks, improves modularity, and improves readability.

Addition: printCalendar

Why: A helper function that prints all calendars the user has.

Why Unavoidable: The helper function simplifies the task, improves modularity, and improves readability.

Addition: updateCalendar

Why: A helper function that updates the calendar from any edits the user made

Why Unavoidable: The helper function is an unavoidable change because it is needed if the user needs to update the calendar such as adding, removing, and editing events.

Addition: changeTimeZone

Why: This function helps change all the events in calendars to the timeZone that the user has changed to.

Why Unavoidable: In order to follow the requirements of the assignment, this change is needed to update any events of any timezone changes that the user has made.

Calendar Class

Addition: Constructor

Why: The constructor is a key method that allows a calendar to both be created and obtain the initializing information for a calendar such as the name, event list, public setting and owner.

Why Unavoidable: The constructor is the only method for creating multiple class instances.

Addition: calendarNavigation

Why: In order to allow the user to navigate through their calendar actions such as creating, removing, or viewing events.

Why Unavoidable: In order to improve design, enhance modularity, and efficiency.

Addition: printEvents

Why: A helper function that prints out all events

Why Unavoidable: It is unavoidable because it allows quick and simplifies the task, improves modularity, and improves readability.

Addition: setEventTimes

Why: A helper function that returns a list of the user's chosen start and end times.

Why Unavoidable: This helper function helps simplify addEvent() by improving readability, shortening addEvent() code, and encouraging modularity.

View Class

Addition: printFilteredEvents

Why: It filters and displays events within a specific date range for different views.

Why Unavoidable: It allowed me to make the code much more modular and kept me from repeating code causing more possible points of failure for the program. This made debugging easier and implementation easier.

Addition: isSameMonth

Why: It helps determine if an event falls within the same month as the current date.

Why Unavoidable: Without the visualize monthly events method would lack a way to correctly filter events by month. Putting this logic in it's own function allowed me to better focus on one small more complication step rather than the whole scheme of things making the code much more modular.

Addition: printEvent

Why: This helper function allows the user to find the information of the event.

Why Unavoidable: It is unavoidable because it allows quick and simplifies the task, improves modularity, and improves readability.

Event Class

Addition: Constructor

Why: The constructor is a key method that allows a calendar to both be created and obtain the initializing information for a calendar such as the name, event list, public setting and owner.

Why Unavoidable: The constructor is the only method for creating multiple class instances.

Addition: eventNavigation

Why: In order to allow the user to navigate through their event and execute actions on their selected event such as changing start and end times.

Why Unavoidable: In order to improve design, enhance modularity, and efficiency.