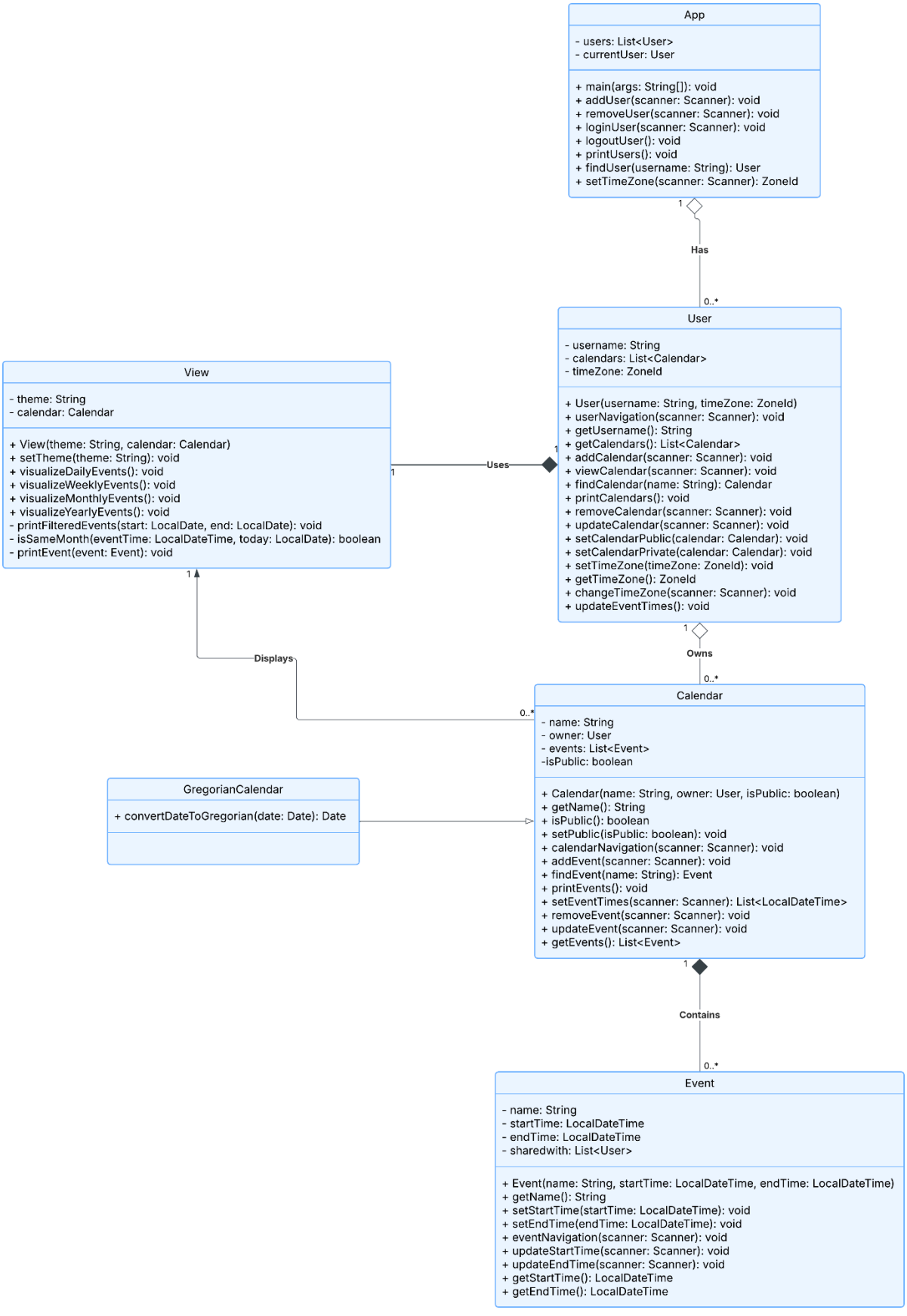
App
- users: List<User>
- currentUser: User
+ main(args: String[]): void
+ addUser(scanner: Scanner): void
+ removeUser(scanner: Scanner): void
+ loginUser(scanner: Scanner): void
+ logoutUser(): void
+ printUsers(): void
+ findUser(username: String): User
+ setTimeZone(scanner: Scanner): ZoneId
1
Has
0..*
User
- username: String
- calendars: List<Calendar>
- timeZone: ZoneId
+ User(username: String, timeZone: ZoneId)
+ userNavigation(scanner: Scanner): void
+ getUsername(): String
+ getCalendars(): List<Calendar>
+ addCalendar(scanner: Scanner): void
+ viewCalendar(scanner: Scanner): void
+ findCalendar(name: String): Calendar
+ printCalendars(): void
+ removeCalendar(scanner: Scanner): void
+ updateCalendar(scanner: Scanner): void
+ setCalendarPublic(calendar: Calendar): void
+ setCalendarPrivate(calendar: Calendar): void
+ setTimeZone(timeZone: ZoneId): void
+ getTimeZone(): ZoneId
+ changeTimeZone(scanner: Scanner): void
+ updateEventTimes(): void
1
Uses
1
View
- theme: String
- calendar: Calendar
+ View(theme: String, calendar: Calendar)
+ setTheme(theme: String): void
+ visualizeDailyEvents(): void
+ visualizeWeeklyEvents(): void
+ visualizeMonthlyEvents(): void
+ visualizeYearlyEvents(): void
- printFilteredEvents(start: LocalDate, end: LocalDate): void
- isSameMonth(eventTime: LocalDateTime, today: LocalDate): boolean
- printEvent(event: Event): void
1
Displays
0..*
1
Owns
0..*
Calendar
- name: String
- owner: User
- events: List<Event>
-isPublic: boolean
+ Calendar(name: String, owner: User, isPublic: boolean)
+ getName(): String
+ isPublic(): boolean
+ setPublic(isPublic: boolean): void
+ calendarNavigation(scanner: Scanner): void
+ addEvent(scanner: Scanner): void
+ findEvent(name: String): Event
+ printEvents(): void
+ setEventTimes(scanner: Scanner): List<LocalDateTime>
+ removeEvent(scanner: Scanner): void
+ updateEvent(scanner: Scanner): void
+ getEvents(): List<Event>
GregorianCalendar
+ convertDateToGregorian(date: Date): Date
1
Contains
0..*
Event
- name: String
- startTime: LocalDateTime
- endTime: LocalDateTime
- sharedwith: List<User>
+ Event(name: String, startTime: LocalDateTime, endTime: LocalDateTime)
+ getName(): String
+ setStartTime(startTime: LocalDateTime): void
+ setEndTime(endTime: LocalDateTime): void
+ eventNavigation(scanner: Scanner): void
+ updateStartTime(scanner: Scanner): void
+ updateEndTime(scanner: Scanner): void
+ getStartTime(): LocalDateTime
+ getEndTime(): LocalDateTime

Explanation of UML

The above figure displays a uml that has been refined from the one provided to be for implementation.In my implementation I prioritized modularity and maintainability. My implementation has notably more methods than the original. This is mainly due to helper functions and the necessity for user input via the terminal.

The App class has the main functionality for managing user and user class specific functionality at a higher level. For this reason the App class has two attributes: a list of users and a current user initialized to null. Since this application is meant to be all users on the same machine this implementation is most suitable. One user is signed in at one time and for that reason this is the most efficient and straightforward. The main function takes in user input of strings creating the opening for the program as a whole. From multiple functions are in existence the addUser, removeUser, loginUser, printUsers, findUser, and setTimeZone. Each of these functions are key to implement within the app class due to their direct effect on Users as a whole. Although the App class must have 0 or more Users.

The User class is mainly tasked with managing all the assets of a specific user such as Calendars and Events. In order to track these assets we would need the username, a list of calendars, and the ZoneId of the timeZone. For that reason there are respective attributes for each of those attributes. A user of course has a constructor User which populates each of these fields. userNagvigation is a high point of interest in this class since it is sort of a router that takes in user input via the scanner and calls the respective functions to accomplish the desired functionality. There are a few helper functions such as getUsername, getCalendars, findCalendar, printCalendar, getTimeZone, and updateEventTimes. All playing a crucial role to the full list of methods. The first thing a user might want to do is create a Calendar.

The calendar class is designed to manage user created calendars within the application allowing users to add, remove, and navigate through events. Each calendar object is associated with a specific user, ensuring ownership and access control if at the very least superficially for now. The class maintains a list of event objects and includes a visibility tag called isPublic to determine the visibility of a calendar; this visibility can be either private or public. The class provides a calendarNavigation method which is the sort of the hub directing action behind the scenes based on user input through the leveraging of the scanner. Event creation involves prompting the user for a unique event name and both a start and end time that is validated. The findEvent and printEvent methods are meant to facilitate the lookup and display that has been implemented through other classes. More on that later. Some other helper functions are notably setPublic, removeEvent, and updateEvent each which is involved in modifying the attributes of a calendar. The calendar although very functional needs to use the View class in order to display the events for the user.

The View class is tasked with providing a visual of all the events of a particular calendar. It maintains a reference to a calendar for that reason and a theme attribute so it knows how to display said calendar. The view class is capable of giving the user a view of their events for the day, week, month, or year. To do this there are respective methods such as visualizeDailyEvents, visualizeWeeklyEvents, visualizeMonthlyEvents, and visualizeYearlyEvents. Along with these functions I have also implemented helper functions in order to remove redundant code which might be pain points of debugging such as

printFilteredEvents which is tasked with printing to the user rather than having each visualization method have to print for themselves, isSameMonth to simplify the logic of comparing dates, and printEvent which is capable of only printing a single event. Of course a View class needs a Calendar to display but per the requirements the Calendar must be implemented as a Gregorian Calendar.

The GregorianCalendar class is a utility class that is meant to handle the conversion between a date imputed and the Gregorian calendar format. For this it has but one function converDateToGregorian which utilizes the java.util.Calendar rather than my calendar class to get the standardized format. Each calendar can contain zero or more events.

The Event class represents individual events within a user's calendar, and utilizes key attributes such as name, startTime, endTime, and a list of sharedWith users. Each event has a unique name and a defined start and end time, ensuring proper scheduling within the application. The class provides methods to retrieve and modify event details, including setStartTime and setEndTime, which allow users to update event timings while ensuring that the end time does not precede the start time. The eventNavigation method facilitates an interactive interface where users can modify an event's start or end time through a command-line menu. To enforce proper time formatting, the class employs DateTimeFormatter and LocalDateTime.parse in the updateStartTime and updateEndTime methods, ensuring user inputs adhere to the required format. Additionally, the class includes basic validation checks to prevent incorrect input and ensure logical event durations.