Ani Megerdichian
INF122 - Garcia
14 January, 2021

Assignment 1-1 | Calendars

Class Descriptions

This section describes each class and its main methods. Attributes and methods in the UML diagram have self describing names, but those that need further explanation are described below. Note that setters and getters are implied, but not explicitly described.

class CalendarsApp

- The CalendarsApp class creates and holds all of the users and the settings of Calendars in ArrayLists.
- **Main Methods:**
    - **add/removeUser:** These methods will be used to add or remove a User from the users ArrayList.
    - **addSetting:** This method will be used to add a new Setting to the settings ArrayList.
    - **updateSetting:** This method will be used to update a Setting in the settings ArrayList using the parameter of the Setting name.

abstract class Setting

- This is an abstract class that will be used to add more setting configurations in the future through inheritance. Currently, the only setting accounted for in the design is the theme.

class User

- Each user has a userId, a collection of Calendar objects, and a specified time zone.
- **Main Methods:**
    - **addCalendar:** This method will be used to create and add a Calendar object to the user's ArrayList of Calendars. The method takes a CalendarType enum as a parameter.
    - **remove/update:** These methods are used to remove or update a Calendar from the calendars ArrayList using the calendar ID as a parameter.
    - **searchCalendars:** This method takes a String keyword as a parameter and finds events that have that keyword in their name. Note that each calendar has a search method as well, which would be used in this method to search each individual Calendar.
    - **changeTimeZone:** takes a ZoneId as a parameter and will be used to update the timeZone attribute. Note that this method would also call the updateTimes method of each Calendar in its collection to update the time of their events to correspond to the new time zone.
        - **ZoneId Documentation:** https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/ZoneId.html
    - **filterCalendars:**These methods take in an ArrayList of calendarIds, along with self-described integer parameters in order to filter Calendar events based on date and time criteria. These methods will call the filter methods of the Calendar class.

abstract class CalendarItem

- CalendarItem is an abstract class that holds the attributes and methods that are shared by all items that can be added to a calendar. This class allows for more calendar items to be easily added through inheritance in further iterations.
- **Main Method:**
    - addItemAddition: This method will be used to add an ItemAddition to the additions ArrayList.

abstract class ItemAddition

- This class is used to hold additional components of calendar items. For example, a photo to accompany an event is an ItemAddition, and can be added to the design through inheritance.

class Event

- Event is a class that holds all the information specific to events such as start and end dates. This class inherits from the CalendarItem class. The start and end attributes are of type ZonedDateTime, which is from the java.time package.
  Documentation for ZonedDateTime:
  https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/ZonedDateTime.html

class Deadline

- Deadline is a class that inherits from the Event class and has attributes and methods that are specific to events that have deadlines and require extra functionality such as a countdown timer. Because of these extra requirements, I chose to create a separate Deadline class that inherits from Event instead of simply adding deadline functionality to the Event class. My approach is the more modular option and would more easily allow the expansion of event options in the future. The Deadline class has a timer attribute that is of type Timer from the java.util package.
  Documentation for Timer:
  https://docs.oracle.com/javase/7/docs/api/java/util/Timer.html
  Example Timer:
  https://docs.oracle.com/javase/8/docs/technotes/guides/lang/Countdown.java
- **Main Methods:**
    - **timeLeft:** method that returns the time remaining, formatted as a string.
    - **stopTimer:** a method to stop the timer.
    - **startTimer:** a method to start the timer.

class Year, class Month, class Week, class Day

- These four classes are similar in structure in that they are compositions of each other. Their attributes are named appropriately, and are self explanatory. Each class implements a display method which would print out a formatted version of each class.

abstract class Calendar

- Calendar is an abstract class that holds the attributes and methods necessary for a basic calendar. By having such a class, it is possible to easily add in different calendar types through inheritance. Furthermore, each new calendar type class will have the functionality of a Calendar object, in addition to any new attributes or methods that are unique to the calendar type. This enhances the extensibility of the design.

- **Main Methods:**
    - **add/remove/update Item & add/remove/update Event:** These methods are used for events and other calendar items. The remove and update methods take the ID of the event or item as a parameter in order to be able to reference the correct event/item.
    - **shareWith:** This method takes a user's ID as a parameter, and would add the ID to the calendar's sharedWith ArrayList.
    - **search:** This method takes a keyword as a string and then searches the ArrayList of events for event names that contain the keyword.
    - **filterBy methods:** There are several filter methods, each of which takes three integers as parameters. These methods will return an ArrayList of events that contain the date or time specified by the parameter.
    - **updateTimes:** This method will be used when the user changes the timezone, which then would require that all event times be updated to correspond with the new timezone.

class GregorianCalendar

- GregorianCalendar is a class that inherits from the Calendar class. This class has 2 attributes that are necessary to create Gregorian calendar that is formatted correctly:
    - **startingMonth:** the month that this calendar starts with; an enum
    - **startingDayOfTheWeek:** the day of the week the calendar starts with; an enum

interface Displayable

- This interface will be used by all classes that need to be able to print formatted information to the console. It includes one method called display which performs this task.

Flexibility in Design

This design is able to support further changes to Calendars. These design features have been discussed in the section above, but they will be reiterated here:

1. **Addition of other types of calendars**

   The Calendar class is an abstract class that holds the basic attributes and methods common between all types of calendars. Any type of calendar can be easily added to the design through inheritance and instantiated through polymorphism.

2. **The ability to add simple to-do lists that interface with calendars**

   The CalendarItem class is an abstract class whose attributes and methods can be inherited to add to-do lists to each calendar, in addition to any other items that would be beneficial to add to a calendar.

3. **Allowing users to add images to represent calendars or events**

   The ItemAddition class would allow adding images to represent calendars or events through inheritance.

4. **The ability to add notes or a description for a particular event or calendar**

   The CalendarItem class is an abstract class whose attributes and methods can be inherited to add notes and descriptions to each calendar or event, in addition to any other items that would be beneficial to add to a calendar.

5. **Expansion of the settings or configurations of the app**

   The Setting class in the current design allows for the inclusion of the theme setting, but also would allow the addition of more settings and configurations in the future.

6. **Saving or importing calendars, events, or users**

   The Importable interface includes methods that would allow these elements to be imported and saved in further iterations.

Calendars
UML Diagram

Deadline
- timer : Timer

+ timeLeft() : String
+ stopTimer() : void
+ startTimer() : void

Event
- start : ZonedDateTime
- end : ZonedDateTime

+ display() : void

1 — 1

CalendarItem <<*abstract*>>
- itemId : String
- name : String
- additions : Array<ItemAddition>

+ addItemAddition() : void
+ display() : void

0..* — 1

ItemAddition
- description : String

+display() : void

0..*

Displayable <<interface>>

+ display() : void

Year
- yearId : String
- yearNum : int
- months : ArrayList<Month>

+ display() : void

1 — 1

Month
- monthId : String
- monthName : String
- numDays : int
- numWeeks : int
- monthNum : int
- yearNum : int
- weeks : ArrayList<Week>

+ display() : void

1..* — 1

Week
- weekId : String
- weekNum : int
- yearNum : int
- monthName : String
- days : ArrayList<Day>

+ display() : void

1..* — 1

Day
- dayId : String
- year : int
- weekNum : int
- dayNum : int
- dayOfWeek : int

+ display() : void

1..*

Calendar<<*abstract*>>
- calendarId : String
- daysPerWeek : int
- private : boolean
- sharedWith : ArrayList<String>
- items : ArrayList<CalendarItem>
- events : ArrayList<Event>
- year : Year
- ownerId : String

+ addItem() : boolean
+ removeItem(itemId : String) : boolean
+ updateItem(itemId : String) : boolean
+ addEvent() : boolean
+ removeEvent(eventId : String) : boolean
+ updateEvent(eventId : String) : boolean
+ shareWith(userid : String) : boolean
+ search(keyword : String) : ArrayList<Event>
+ filterByStartDate(day : int, month : int, year : int) : ArrayList<Event>
+ filterByEndDate(day : int, month : int, year : int) : ArrayList<Event>
+ filterByStartTime(hour : int, min : int, sec : int) : ArrayList<Event>
+ filterByEndTime(hour : int, min : int, sec : int) : ArrayList<Event>
+ updateTimes() : boolean

1 — 1

0..*

<<interface>> Importable

+import() : void
+save() : void

CalendarsApp
- users : ArrayList<Users>
- settings : ArrayList<Setting>

+ addUser() : void
+ removeUser(userId : String): void
+ addSetting() : void
+ updateSetting(sName : String): void

1 — 0..*

Setting <<abstract>>
- name : String

1 — 0..*

User
- userId : String
- calendars : ArrayList<Calendar>
- timezone : ZoneId

+ addCalendar(type : CalendarType) : boolean
+ removeCalendar(calID : String) : boolean
+ updateCalendar(calID : String) : boolean
+ searchCalendars(key : String) : ArrayList<Event>
+ filterCalendarsByStartDate(cals : ArrayList<String>,day : int, month : int, year : int) : ArrayList<Event>
+ filterCalendarsByEndDate(cals : ArrayList<String>, day : int, month : int, year : int) : ArrayList<Event>
+ filterCalendarsByStartTime(cals : ArrayList<String>, hour : int, min : int, sec : int) : ArrayList<Event>
+ filterCalendarsByEndTime(cals : ArrayList<String>, hour : int, min : int, sec : int) : ArrayList<Event>
+ addEvent(cal : Calendar) : boolean
+ changeTheme(t : ThemeType) : void
+ displayCalendar(calId : String) : void
+ changeTimeZone(t : ZoneId) : void

1 — 0..*

<<enum>> ThemeType

Light
Dark

<<enum>> CalendarType

Gregorian

<<enum>> GregorianMonth

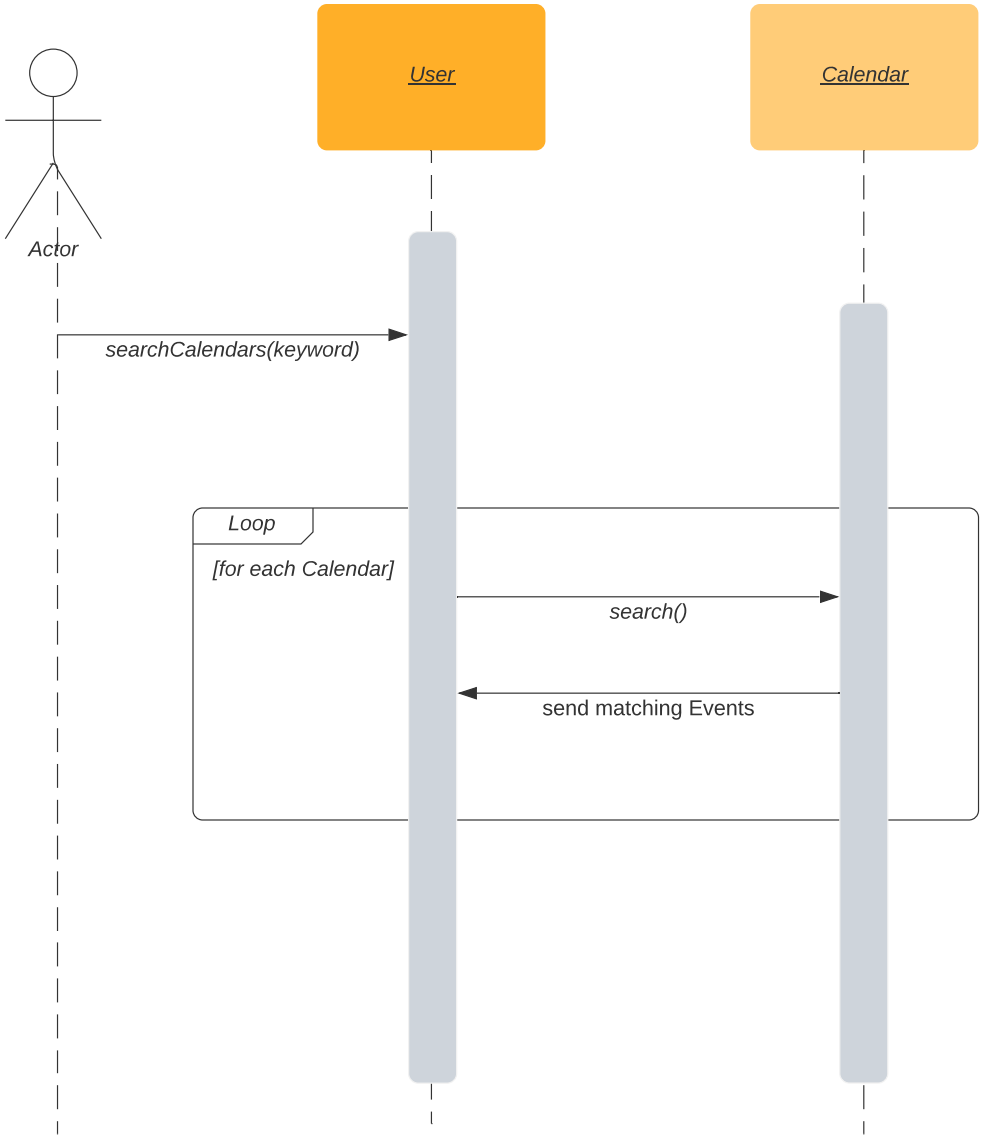
// January is the 1st month of the year
// January has 31 days
January(1, 31)
February(2, 28)
// Fill in for rest of the months

GregorianCalendar
- startingMonth : GeorgianMonth
- startingDayOfTheWeek : GeorgianDay

<<enum>> GregorianDay

// Sunday is first day of the week
Sunday(1)
Monday(2)
// Fill in for rest of the days

Use Case: User searches all of their calendars for a keyword.



Use Case: User changes their time zone.