This work is mine unless otherwise cited. - Kai’lani Woodard

Overview

My final course project for Data Abstraction is the Digital Calendar and Planner program. This program will serve as a digital representation of the traditional Gregorian calendar that also has a built-in planner with the capability to write, update and delete events across multiple sessions.

This project is an extension of my final course project from Computational Expression in the Fall semester. Therefore, my primary focus for this project has been optimizing some of the existing methods and functions with concepts that I learned in Data Abstraction. In addition to that, I plan to implement several new methods to ensure that the program is as frictionless as can be.

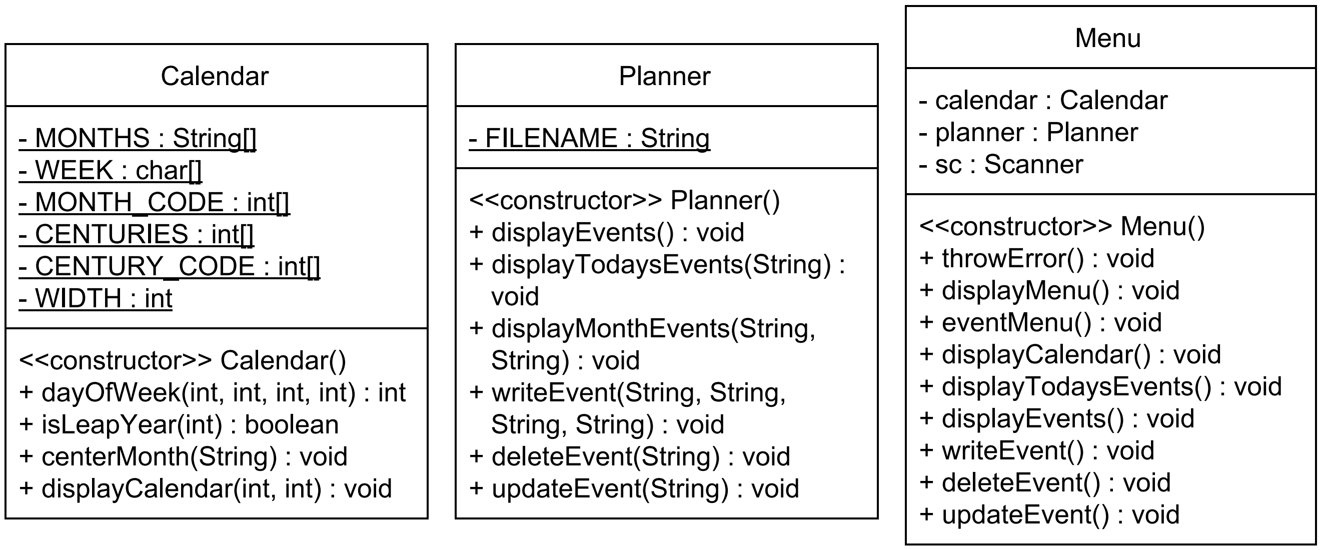
Motivation

The purpose of this program is to create a simple digital alternative to traditional physical calendars and planners. I personally struggle with keeping up with physical calendars or planners, so I felt that the creation of this program would be useful for people that are similarly more digitally inclined versus those who prefer analog methods. The program also runs locally on the user’s machine and stores each of their events within a text file, keeping it relatively light regarding the program’s size. I was also motivated to pursue this project because I wanted to challenge myself by basing the program’s foundation on a concept I was not familiar with at the time. As mentioned previously, the first implementation of this program was in Computational Expression in the previous Fall semester. At the time, I was quite unfamiliar with how to properly read from and write to a file. This project challenged me by being heavily founded upon the relationship between the program and outer files and also interested me because I also fit the demographic of its potential user base.

Background

One of the most popular digital calendars that inspired this program was Google Calendar. Google Calendar goes beyond the scope of my digital calendar and planner by having an interface and also integration between Google applications. Google Calendar also has the ability to push the user notifications to remind them of their events. However, I feel that this program is more akin to the technological level of the PDA, which was invented in 1993 and popular throughout the 90’s and early 2000’s.

Data Structure



```
Calendar
- MONTHS : String[]
- WEEK : char[]
- MONTH_CODE : int[]
- CENTURIES : int[]
- CENTURY_CODE : int[]
- WIDTH : int
<<constructor>> Calendar()
+ dayOfWeek(int, int, int, int) : int
+ isLeapYear(int) : boolean
+ centerMonth(String) : void
+ displayCalendar(int, int) : void
```

```
Planner
- FILENAME : String
<<constructor>> Planner()
+ displayEvents() : void
+ displayTodaysEvents(String) :
  void
+ displayMonthEvents(String,
  String) : void
+ writeEvent(String, String,
  String, String) : void
+ deleteEvent(String) : void
+ updateEvent(String) : void
```

```
Menu
- calendar : Calendar
- planner : Planner
- sc : Scanner
<<constructor>> Menu()
+ throwError() : void
+ displayMenu() : void
+ eventMenu() : void
+ displayCalendar() : void
+ displayTodaysEvents() : void
+ displayEvents() : void
+ writeEvent() : void
+ deleteEvent() : void
+ updateEvent() : void
```

The program is composed of four classes: `Calendar`, `Planner`, `Menu` and `Main`. Displayed above are UML diagrams for three of the four classes. There is no UML diagram included for the `Main` class because it only contains the `Main` method.

The purpose of the `Calendar` class is to contain methods that help calculate the calendar's properties and display the calendar. It is a recreation of the original program's `CalendarDisplay` class. The original version of the class imported the `File`, `IOException` and `Scanner` classes to assist in the functions of the methods. The current version of the class imports no classes. This can be accredited to the complete overhaul of the original structure of the methods.

The `centerMonth()` and the `displayCalendar()` methods are the only two original methods that remained in this edition of the program out of the first edition's four. The other two original methods were not not included in this edition of the program because they were created with the program's previous structure in mind.

The program was previously structured to rely heavily upon a spreadsheet called `2019Months.csv`, from which information was pulled from to structure the calendar. While this method worked well for 2019, it was incredibly ineffective in structuring any other calendar displays from outside of the realm of 2019. This is because the comma-separated value sheet only contained properties of the months in the year 2019, creating an incredibly niche calendar that can only be used in the context of 2019.

The `Calendar` class now has two new methods that lay the foundation of this edition of the program: `dayOfWeek()` and `isLeapYear()`. The `dayOfWeek()` method is structured upon an equation that can be used to calculate the day of the week for any day throughout history beginning at year 1582 and ending at year 9999. It is restricted to this range because the Gregorian calendar was invented in 1582, therefore its structure is nonexistent until that point and is held maximum at 9999 because the proceeding number is five digits. The equation is formulated to only handle four digits, therefore, it must cap its max year at 9999.

The method takes four parameters, which are `m`, `d`, `c`, and `y`, representing month, day, century and year, respectively.

However, in this context, century refers to the first two digits of a year and year refers to the last two digits of the year. (This is calculated `year / 100` and `year % 100`, respectively.)

These values are used to calculate specific codes in the following calculation that determines the day of the week:

```
(yearCode + monthCode + centuryCode + day - leapYearCount) % 7
```

> Year Code is determined by taking the last two digits of a year, in this case represented by the `y` variable, and adding it to `y / 4` and then modulating the sum by 7.
>
> Month Code is determined by an array of numbers that correlate with the months in order, which are `{0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5}`, respectively.
>
> Century Code is determined in a similar fashion by utilizing the pattern: `4, 2, 0, 6, 4, 2, 0...` These numbers are associated with centuries, starting at 1700 simply because Great Britain, and therefore the United States, did not put the Gregorian calendar in 1752. However, the pattern will work for previous years despite the Gregorian calendar not being in full effect previous to those years.
>
> Day simply represents the day that you are trying to figure out the day of the week for.
>
> Leap Year Count either takes the values `0` or `1` to represent whether a year is a leap year or not and is only accounted for in the equation when calculating January or February of a leap year.

This equation serves as the foundation for the program because it determines what the first day of the month will be for any month and year pair within history (and accurately within the logical range) and displays such in a calendar formation. This equation fundamentally changed the structure of the program and made it exponentially more powerful.

The `dayOfWeek()` method uses the `isLeapYear()` method within it to determine the value of the Leap Year Code or `lyc` variable. It simply takes a four digit year as a parameter and puts it through three tests:

> Does the year evenly divide by four?
>
> If it does, does that year evenly divide by one hundred and does that quotient evenly divide by four hundred? If it does, the year is a leap year.
>
> If it does not evenly divide by one hundred by does evenly divide by four, it is also a leap year.

This can be accomplished quite simply with a series of if-else statements.

While the `isLeapYear()` method may not be as foundational as the `dayOfWeek()` method, it also brings incredible amounts of power into the program. Without this method, the `dayOfWeek()` equation would be inaccurate and therefore the program would be non-functional.

All of these methods, `dayOfWeek()`, `isLeapYear()`, and `centerMonth()` come into play when it comes to the `displayCalendar()` method. The `displayCalendar()` method takes all of these calculations into consideration when displaying the calendar to the console.

The sole purpose of `centerMonth()` method is to center the month within the width of the calendar, which is represented by the `WIDTH` variable, which equals `25`.

The `dayOfWeek()` method helps determine how many blank spaces will be placed into the calendar before printing the first day into the calendar. It also helps determine where the rest of the days in the month will be placed.

The `isLeapYear()` method determines the amount of days on one condition, that condition is if there is a leap year and the month being displayed is February.

The `displayCalendar()` method thereon uses a series of control flow and conditional statements to determine what will be printed to the console. It bases its logic upon calculated spacing and the output of the aforementioned methods. The `displayCalendar()` method will print output that looks like the following:

```
               APRIL

     S   M   T   W   T   F   S

                 1   2   3   4

     5   6   7   8   9   10  11

     12  13  14  15  16  17  18

     19  20  21  22  23  24  25

     26  27  28  29  30
```

The purpose of the `Planner` class is to contain methods that help write to events to the `events.txt` file, update and delete those events, and display those events. It is a recreation of the original program's `CalendarEvent` class. The original version of the class is fairly similar to the latest edition of the class, with a few exceptions.

The major changes within the `Planner` class consist of adding the `deleteEvent()` and `updateEvent()` methods and changing a few of the parameters to accept integer arguments versus the previous edition's String arguments.

The `Planner` class consists of the following methods: `displayEvents()`, `displayTodaysEvents()`, `displayMonthEvents()`, `writeEvent()`, `deleteEvent()` and `updateEvent()`.

These methods all find their foundation upon the `BufferedWriter`, `FileWriter`, `File` and `Scanner` classes. These four basic classes ensure that each method is able to read and write to the file when necessary and most importantly access the file.

The `displayEvents()`, `displayTodaysEvents()` and `displayMonthEvents()` methods all have the same basic structure but they display different sets of events. The `displayEvents()` displays every event from the `event.txt` file in chronological order. The `displayTodaysEvents()` method only displays events marked with the same tag as the system's current date. The `displayMonthEvents()` method is intended to be used alongside the `Calendar` class method `displayCalendar()` to display the events that coincide with the given month. All of these methods use the

`BufferedWriter`, `FileWriter` and `Scanner` classes to read through the files and resort the `eventList` ArrayList using `Collections`. Essentially, these methods are different renditions of one another that all serve their own unique purposes.

The `writeEvent()`, `deleteEvent()` and `updateEvent()` methods are slightly similar in structure but also slightly different.

The `writeEvent()` method relies on four String parameters to create an entry for the `events.txt` file. It checks if the given Strings for month and day and if they are single digit numbers and if they are, it adds a zero to the front to maintain consistent formatting chronologically. After that, it concatenates all of the String parameters like so:

```
month + “/” + day + “/” + year + “: ” + event
```

It saves this concatenation to a String variable named `entry` and writes it to the file using `BufferedWriter`.

The `deleteEvent()` and `updateEvent()` methods are slightly different in structure. Both of these methods rely on taking one String parameter, `keyword`, and searching the event list for potential matches. Once potential matches are found, it lists them to the user and the user chooses an option from the numbered list to either delete or update. Afterward, the methods cycle through control flow statements to determine which element of the ArrayList to delete or update and rewrites the newly updated ArrayList to the file.

In this edition, I added a new class called the `Menu` class. This class essentially exists to create ease within the main class. The `Menu` class easily has the most methods out of all of the classes but that is simply because it contains wrapper methods from both the `Calendar` class and the `Planner` class; it contains several methods that contain large bodies of code so when they are placed in the `Main` class, it reads less cluttered and gives more ease to rearrange the methods when they are only a single line of code versus being large blocks.

Results

Because of the nature of the Digital Calendar and Planner program, depicting results on graphs or tables is not very efficient in displaying the data. Rather, I felt that giving examples of the console output would help understand the potential that the program has while also demonstrating its functions.

```
--------------------------
What would you like to do?
(D)isplay the calendar
(S)ee today's events
(V)iew all events
(M)ore options...
(Q)uit
--------------------------
D

What month would you like to see? (1-12)
4

From what year? (YYYY)
2020

--------------------------

          APRIL

S   M   T   W   T   F   S

            1   2   3   4

5   6   7   8   9   10  11

12  13  14  15  16  17  18

19  20  21  22  23  24  25

26  27  28  29  30

--------------------------
04/29/2020: CR/NC Exemptions Due
04/29/2020: CS 101 Final Project and Report Due
04/30/2020: CS 101 Final Exam
```

Above is an excerpt of the output the code has produced from a test session. It demonstrates the `displayMenu()`, `displayCalendar()` and `displayMonthEvents()` methods. This is only one potential glimpse into the capabilities of what the program can do. However, I felt that displaying the output that the program produces is the best way of conveying my results in hopes of them being understood.

Conclusion

At this time of submission, I feel that my final course project has yielded successful results. I have learned a lot from this project but my largest takeaway is that using wrapper classes to declutter one's `Main` method is incredibly useful. Ultimately, the most challenging part about developing this program was the adaptation of the equation for the `dayOfWeek()` method. Before its final form, I had previously researched two separate equations to yield the day of the week a date occurs that would not work for each date inputted. Adapting an equation that is complex in nature to the program was extremely difficult considering all of the additional properties of the final equation. However, with the extreme adversity faced, the greater the reward was when the equation was fully functional. It was extremely rewarding to expand the capabilities of this program exponentially and create something I am proud of.

All in all, I have accomplished what I had seeked to before going into this project and *that* is what was most rewarding.