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

Introduction

Since the inception of my project, I have hit quite a few milestones towards my program. As I have mentioned before, this project is an extension of my final project in Computational Expression from last semester, so my focus has mainly been on optimizing some of the existing functions while also adding several new functions to make the program as frictionless as it can be.

Progress Breakdown

Most of the work that I have focused on for the project is the restructuring of the `Calendar` class. This class imitates the previous program's `CalendarDisplay` class. The `CalendarDisplay` class originally imported the `File`, `IOException` and `Scanner` classes to create its methods. The improved `Calendar` class imports none thus far. This is because the previous implementation of the program was structured around pulling information from a comma-separated value sheet to get its data to run the program. While this was a good demonstration of knowledge in file operations, the execution of such was not nearly as efficient as it could be. The previous version could only pull information from months in 2019 and relied on concrete data to run while the newest version is now flexible within any time period from the beginning of time to far into the future. This was made possible with the `dayOfWeek()` method.

The `dayOfWeek()` method takes four parameters, which are all integers, and uses them in an equation to calculate what day of the week a specific day will take place. These parameters are `m`, `d`, `c` and `y` representing month, day, century and year respectively. However, in this context, the century variable refers to the first two digits of a year while the year variable refers to the last two digits of the year. It has been done this way in order to make the calculations possible. The equation is adapted from two equations that calculate the specific day of a date using numbers to represent the days (0 representing Sunday, 1 representing Monday and so on…). These equations are by Dr. Bill and Dr. Peterson from mathforum.org and Alex Lopez-Ortiz from the computer science department of U of Waterloo. However, due to the complex nature of the equation the input has to change a little bit in order to provide the correct answer. This is referring to March as the first month of the year and January as the thirteenth and February as the fourteenth respectively. While I do not understand why this changes the calculations, changing the input to reflect that provides the correct answer every time.

The next new method added into my program was `isLeapYear()`. This method is fairly simple in concept, as it is only composed of three nested if-else conditional statements to determine whether or not a provided year is a leap year. This method just follows basic rules in calculating whether or not a year is a leap year. It primarily uses the modulus function to determine whether year modulus specific numbers (4, 100 and 400) yield 0. These specific numbers were chosen based on some basic math theory in determining whether or not a year was a leap year.

The other two methods in the `Calendar` class are based upon their legacy methods into the previous program and these methods have been adapted to fit the newer version of the program better while also being more efficient in nature. The `centerMonth()` method simply prints a version of a month title in the center while the `displayCalendar()` method prints a layout of the calendar to the terminal like so:

```
          DECEMBER

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  31
```

The primary use of the `centerMonth()` method is within `displayCalendar()`. As it can be seen above, it simply centers the month title within the calendar.

Between the previous version of the program and the current version of the program, `displayCalendar()` has undergone many changes. In the first edition, `displayCalendar()` relied on the previously mentioned comma-separated value sheet for its data. This made the code longer and overall less efficient. The current implementation uses a combination of arrays and the `isLeapYear()` and `dayOfWeek()` methods to get a more customized and also more efficient result.

In addition to working on the `Calendar` class, I have also worked on the `Planner` class. Now, from this implementation and the previous implementation, not much has changed. The `Planner` class imitates the previous program's `CalendarEvents` class. Both versions of the class import `BufferedWriter`, `File`, `FileWriter`, `IOException`, `ArrayList`, `Collections` and `Scanner`. The classes also have a lot of methods in common, including the `displayEvents()`, `displayTodaysEvents()`,

`displayMonthEvents()` and `writeEvent()` methods. The primary difference in this implementation regarding these classes is the addition of a `deleteEvent()` method and potentially `updateEvent()` and `searchEvent()` methods. Adding these methods give the program nearly as much functionality as other digital calendars and planners. It also makes it more user-friendly. I only say that I would potentially add the `updateEvent()` and `searchEvent()` methods because of the potential complexities that these would add to the program. Certainly in any future implementations, these would be an essential addition but with regards to time, I am currently unsure of whether or not they will be implemented in this version of the project.

These additional methods all have similar structure but they all heavily rely on `BufferedWriter`, `FileWriter` and `Scanner`. Each of these imported classes are incredibly powerful in file operations and give the ability to store the user input for each event created or modified. This is an improvement from the previous implementation which relied on the user to manually delete an event from the text file if they needed to, which is not efficient. Therefore, the addition of these events makes the program more powerful overall.

Regarding the `Main` class, there are still structural changes to be made. It will operate similarly to other programs that have multiple options and stay encased within a `while` loop. It will have a menu that prompts the user to select from a list of options and takes action from there. However, I have yet to fully implement the final version of the main class because I am still testing methods.

Conclusion

I am still continuing to work to finish the program. Thus far, there have been no major issues, only a minor issue with understanding the equation within `dayOfWeek()`. I feel that the aspiration of this project is to ultimately create a program that I have the need for, in order to add the best-fitted methods possible to it. I also feel that the combination of file operations and dynamic list structures reflect the content that has been taught in Data Abstraction well.