Two-Year Cycle Group Calendar

by

Praise Daramola, Praise24

Report submitted on 16 November 2019

EE 333 – Engineering Programming Using Objects

Department of Electrical and Computer Engineering
The University of Alabama at Birmingham

ABSTRACT

This purpose of this project was to explore the object-oriented design process through development of a program which models a real-world system. This report discusses several object-oriented approaches and methods for solving problems. This report also explores several problems encountered and features discovered through the development process.

Table of Contents

INTRODUCTION

The goal of this project is to apply object-oriented design approach to development of a group calendar that works on a two-year cycle. Object-Oriented design is useful for modelling real-life systems on a computer. There are several Object-Oriented Programming languages including C++, Ruby, VB .NET, Python. For this project, I used the JAVA programming language. Also, I demonstrated knowledge of development tools within the NetBeans IDE by making appropriate use of these tools during my development process.

PROJECT DEFINITION

The program to be designed in this project is a calendar which works on a two-year cycle. This calendar will be able to store processes which consist of multiple activities to be performed on specific dates. This calendar will also be able to assign different processes to different processes to different roles. The calendar should also allow the user to extract information related to several processes including the roles assigned (and any sub roles if needed) to that process, the date the process is to be executed, and the description of that process.

Constraints

- CO-01: The system shall be developed using the Java Programming language
- CO-02: The problem shall be solved using the object-oriented programming design approach.
- CO-03: The modelling classes shall not contain any I/O operations

Goals

- GO-01: The program should be user friendly.
- GO-02: The program should be delivered by the due date.
- GO-03: The program should comply with standards associated with object-oriented design.
- GO-04: The program should adhere to rules described in the project description.
- GO-05: The program should allow for explicit creation of activities and roles.
- GO-06: The program should store the history after each run.
- GO-07: The programmed will be debugged and tested using the IDE debugger tools and other debugging methods to ensure the programs functionality.

Features (in MoSCoW list format)

- Must-01: Be able to search and retrieve information about processes
- Must-02: Keep track of Processes.
- Must-03: Allow for the creation of new Processes
- Must-04: Allow for the assignment of processes to different roles.
- Must-05: Allow for the creation and deletion of processes.
- Must-06: Keep track roles related to processes.
- Must-07: Keep track dates for processes to be executed.
- Should-01: Allow for the creation of new roles
- Should-02: Allow for manual addition of new sub roles to existing roles.
- Should-03: Allow the user to edit current existing processes.
- Should-04: Be presented in a meaningful sorted order.
- Could-01: Include a Graphical User Interface.
- Could-02: Inform roles about upcoming processes to be executed.
- Could-03: Inform roles about a new process addition to the calendar.
- Won't-01: Automatically assign sub roles to roles.

Applicable Standards

- STD-01: All files will comply with the acceptable documentation standards outlined on the Javadoc website.
- STD-02: All classes will comply with the format outlined in the EE 333 documentation page.

DESIGN

After receiving the assignment, I read through the entire assignment description. After my initial reading, starting from the beginning of the problem description, I began evaluating several possible design approaches based on my initial understanding of the problem. While reading through, I underlined and highlighted several keywords I believed would be important when working on a solution. I also took note of several potential objects that could be implemented if my solution could be solved using an object-oriented approach.

Next, I evaluated how the problem to be implemented would imitate a real-world calendar. To do this, I examined how I interact with my real-life calendar. I took note of each step I took throughout my interaction with the calendar. Some of the interactions include:

1. Viewing a task associated to a date in my calendar:
    - Opening the calendar
    - Searching for the desired year.
    - Searching for the desired month.
    - Searching for the desired day.
    - Identifying the tasks/events assigned to that day.
    - Exited the calendar.

2. Adding a task to my calendar:
    - Opened my calendar.
    - Searched for desired year.
    - Searched for the desired month
    - Searched for the desired day.
    - Selected the desired day.
    - Added the desired date.
    - Selected the duration of the task.
    - Selected when to be reminded about the task.
    - Exited the calendar.

3. Viewing a date associated with a task.
    - Opened my calendar.
    - Selected the search bar.
    - Typed the title of the task I was interested in.
    - Clicked search.
    - Identified the date [day, month, year] associated with the task.
    - Exited the calendar.

There were other recorded interactions associated with my calendar, however, I found the interactions listed above to be most relevant to this report.

Next, I began to identify features and behaviors that each object would exhibit. This was to identify my potential classes and determine the characteristics of these classes. I also evaluated possible inputs that would be required to create each object and the data that each class would store. I also evaluated the interaction between different classes. This helped me identify other possible data points that will be contained within an object and possible dependencies of each object.

After brainstorming possible classes, I eventually settled on the most relevant classes which include: A Calendar class to keep track of the date, processes associated with each date, role and activity. A Role class to keep track of the existing roles and their sub roles and the activities assigned to each role. An Activity class to keep track of each created activity, the roles associated with that activity, and the date associated with that activity. A Date class

to keep track of the dates assigned to processes (this decision was made to account for date pairings such as “April/May”, “January/February”, etc.). I also evaluated possible commands and queries associated with each class. For example, an ‘addSubRole’ method for adding a sub role to a role, an ‘addActivity’ method for adding an activity to the calendar.

Next, I evaluated possible options for storing Processes: firstly, I determined that I could create a processes class to store a set of activities and assign a process object to each date in the calendar. Secondly, I considered creating an ArrayList named process which stores activities in either my Calendar class or Date class. Also, I decided I was going to create a file which stores the calendar history.

Although several potential approaches were evaluated, these approaches were used only as a blueprint for the final coding process. As the development process progressed, several changes were made to account for roadblocks with respect to code capabilities. Other alternative designs were also evaluated during the development process. These alternatives/changes were reflected on the final deliverable.

Appropriate for Object Oriented Approach

For a program to be appropriate for an object-oriented approach, the model to be represented by the program should be able to be physically represented by objects in the real world. In other words, the model should have certain features, states, and behaviors. Upon evaluation of the problem description, the assignment contained several features that can be represented by a real-world system. For example, a feature of a calendar is that it has several dates-[feature] and can be used to monitor-[behavior] progress-[state], a process-[state] can be added to a physical calendar-[object] manually, an activity-[feature] can be assigned to a role-[object], a role-[feature] can be assigned to a person. As a result of this evaluation, it was determined that an object-oriented approach is appropriate for this project.

Design Decisions

AL-01 How does the program store activities with respect to the calendar?

AL-01A Create a processes object for each date object, activities are stored in processes.

AL-01B Create a process ArrayList in each date object which stores activities

AL-01C Create a process ArrayList in the calendar object which relates to each date. This stores activities.

ALD-01C: My reason for choosing this solution is because it gave the calendar a way to track the activities involved with the calendar. This method simplified the activity search with respect to year and date. This is because it allowed me to filter the results based on the user input without having to create a new object. Since a process is just a set of activities, I realized it was unnecessary to create a process class.

AL-02 How does the user interact with the program?

AL-02A GUI for search/creation of activities, roles, and dates.

AL-02B Commands through the command line.

ALD-02 A I decided to create a GUI for the user to interact with the system. Initially, I assumed this would simplify my input process, however, it made development time significantly longer because I had to learn how to use JavaFX. Although the development time was significantly longer, creation of a GUI for user interaction made it more presentable and user friendly which was one of my goals for the design.

AL-03 Does the program assign ID's to each activity and role?

AL-03A Yes, and gives the user the ability to search by ID.

AL-03B Yes, but only uses the ID internally.

AL-03C No, keep track of and roles using other relationships.

ALD-03 B Although I implemented this feature in the program, there is no real use for the activity ID feature in the current system, however, I decided to implement it in case any modifications are made in the future which requires an ability to track an activity. The role ID feature, however, is used to monitor roles created. (more details discussed in the DEBUG section).

AL-04 How to search for an activity, role or set of processes in the calendar?

AL-04A Overloaded getActivities(…) methods which take in parameters like activity, role, date, year.

AL-04B Single getActivity(…) method which takes in an activity parameter.

ALD-04A: I decided to go with overloaded methods because it allowed for easier management and allowed for multiple ways of accessing the information instead of restricting the system to one method and setting conditions in that one method.

AL-05 How will the history be stored?

AL-05A Use JSON format and use a JSON parser to read it.

AL-05B Use CSV format and create a .csv parser.

ALD-05B: I decided on a CSV format for storing my activities, however, I decided it would be easier for me to create a different format for storing my roles. The format I used for storing the roles is like a JSON format but is proprietary (shown below). I decided on the format because I did not want to use the JSON library for java so I could have more control over hoe the data was stored and read. As the file is parsed, the data is used to automatically create roles and activities that existed from the previous run.

Sample format for Role history file:

{Role 1}
{Role 2}
Sub role 1 for role 2
Sub role 2 for role 2
{Role 3}
Sub role 1 for role 3
Sub role 2 for role 3
Sub role 3 for role 3
{Role 4}
Sub role 1 for role 4
Sub role 2 for role 4
Sub role 3 for role 4
Sub role 4 for role 4
Sub role 5 for role 4

Object Oriented Design

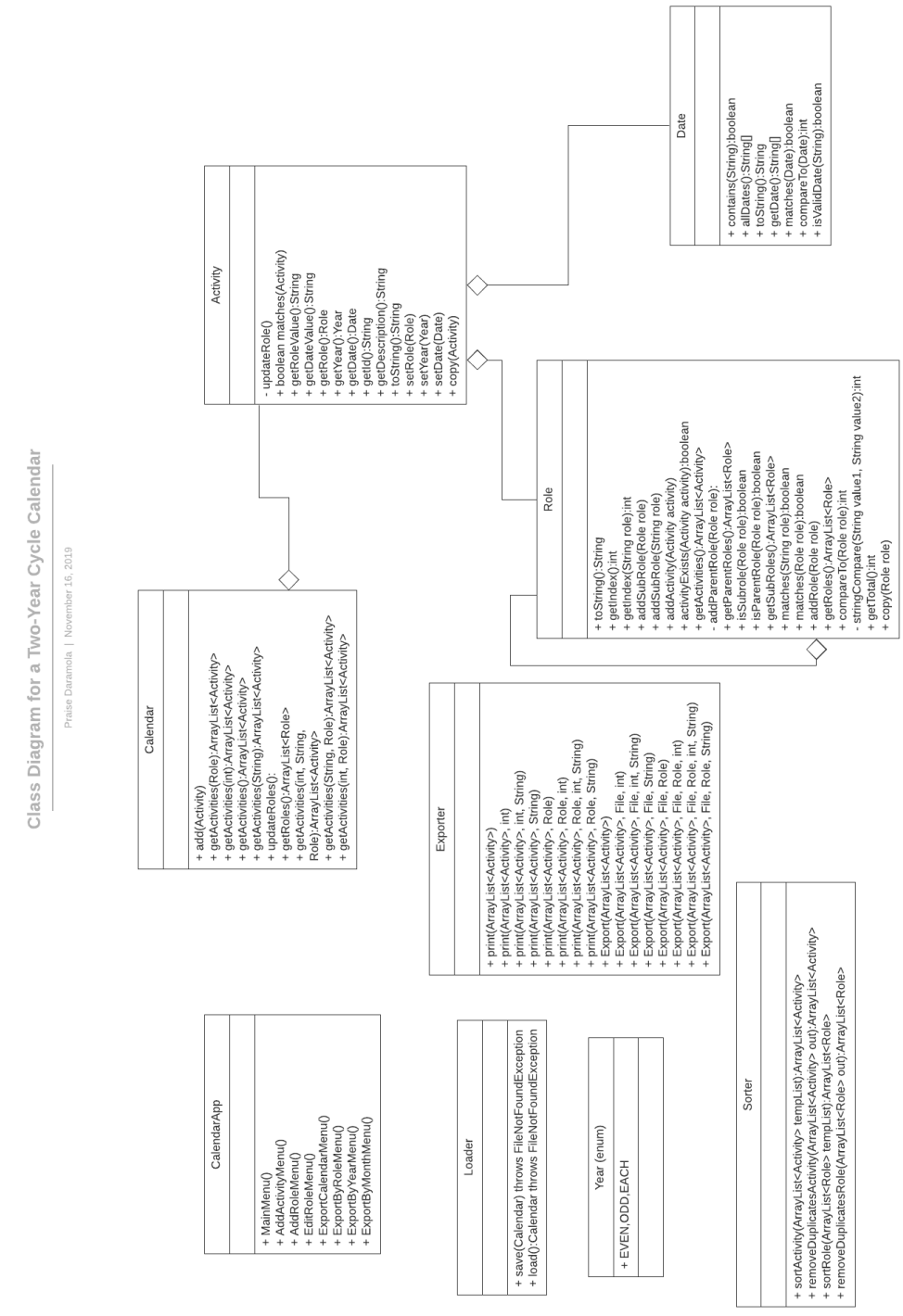


Fig 1. Class Diagram for the two-year cycle Diagram.

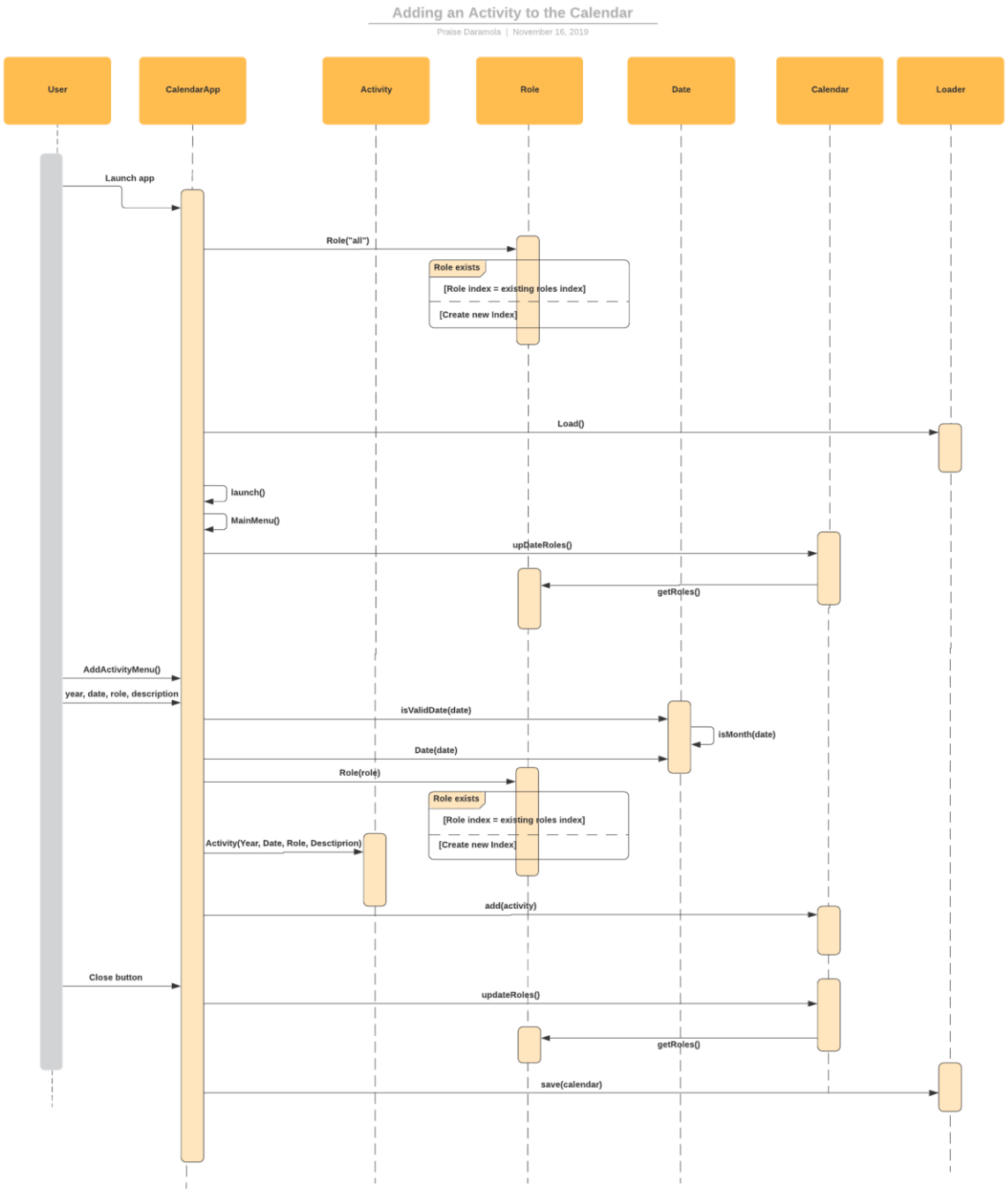


Fig 2. Interaction diagram for adding an activity to the calendar.

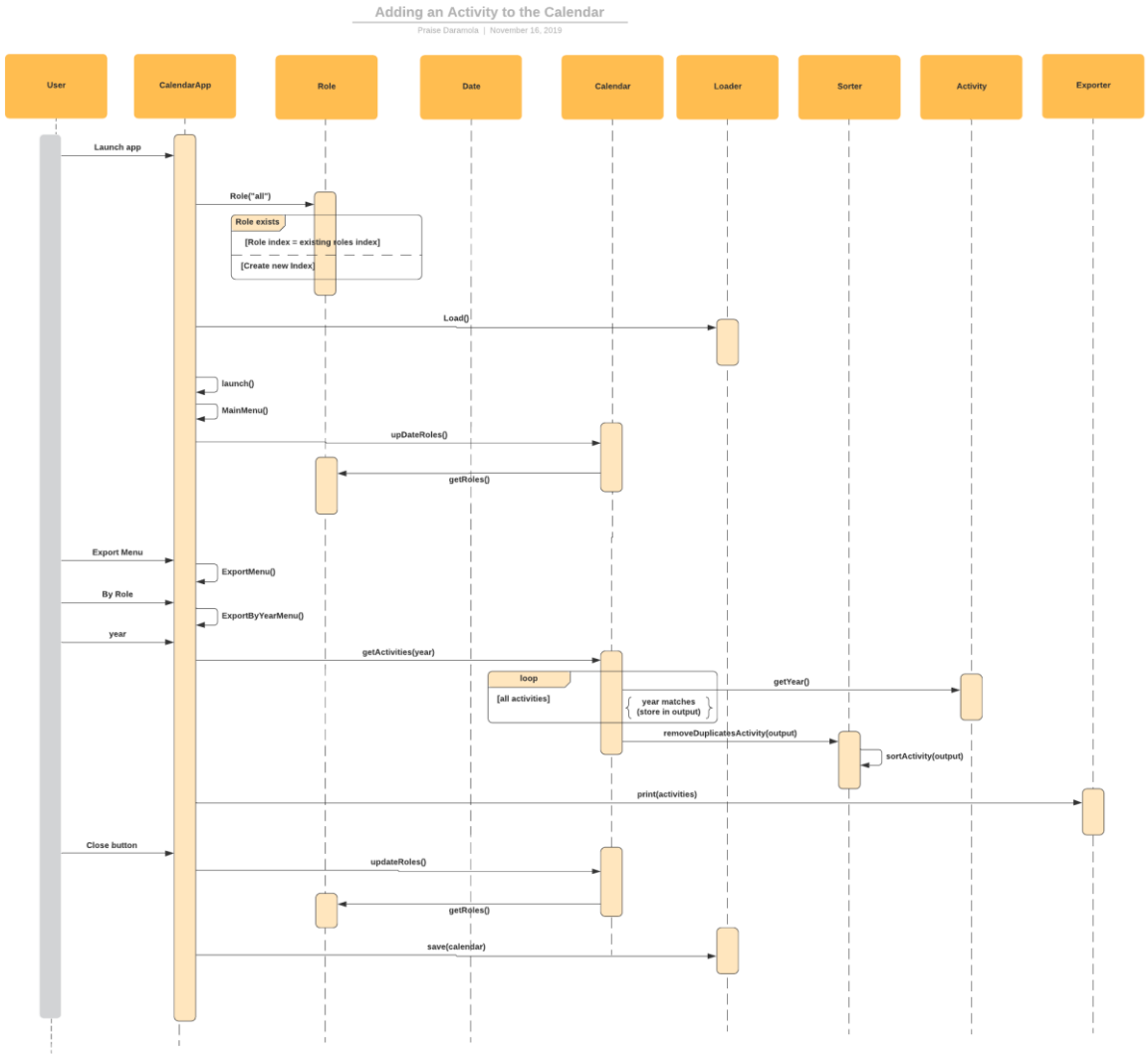


Fig 3. Interaction diagram for searching for an activity by year.

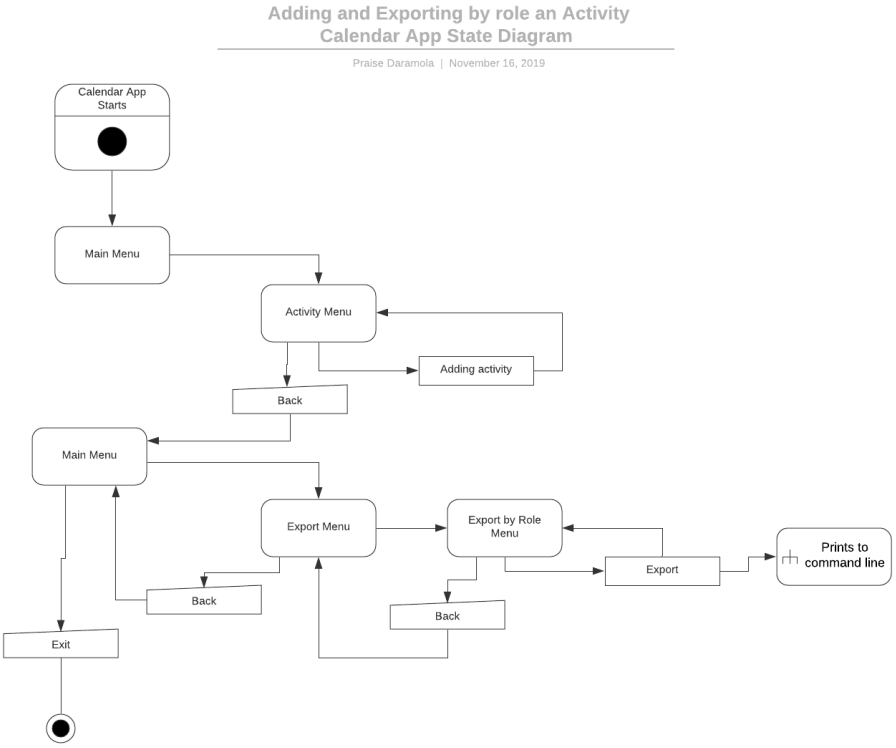


Fig 4. State diagram for the calendar application when adding and exporting activities.

DISCOVERY AND USE OF ONLINE INFORMATION

My decision to create a GUI for the application resulted in me using some online information. Working with JavaFx, I used several resources including the stack overflow and tutorial points website. One of the activities for which these websites was helpful was the creation and placement of elements in the GUI. I used the tutorial points website to learn possible ways in which elements can be placed in a scene, then on a stage. Some of the methods include placing these elements in a HPane, Vpane, GridPane, or Pane, then placing these elements in a scene, then placing this scene in a stage.

A few activities for which I also used online resources are:

1. Understanding how threads work in java.
2. How work with file IO in java.
3. Setting up Git for version control and backup.
4. Looking up references for syntax on the oracle website.
5. Looking up references for error codes.

The activities mentioned above are some but not all of the activities for which online resources were used.

DEBUG

One issue I ran into during development was implementation of the ability for each role to keep track of its sub roles and their activities. This was because any time a new role object is created, even if the role already exists, a new role object was unaware of changes made to an identical role object. Changes such as adding a new role or adding an activity.

To resolve this problem, I initially created an array list which kept track of all roles created, then any time a modification is made to a role, the calendar searched for the roles that had a matching title and applied the changes to these roles. However, I encountered several errors relating to multiple occurrence of the same role in the sub role array list.

I was unsatisfied with this solution, so I decided to make a rough sketch of the interaction between roles and their sub roles. (Sketches shown in the APPENDIX section)

After brainstorming and trying different solutions, I settled on creating a Role ID/index for every role created. This is because if one role has the same name as another role, it is the same role, so it should have the same index in an arrayList of existing roles. This is to ensure that when the role is modified, it's the role at that index of stored existing roles that gets modified. In other words, the calendar allows for creation of multiple role objects, however, if two or more roles have the same name, only the first instance of a role is the gets modified if a sub role is added and only the first instance of the role is referenced in the calendar. This was achieved by having a condition in the constructor which checks if the role title matches an existing role, if it does, the role index/role ID will be the same as the existing role, if it does not match, the program adds the role to an arrayList of existing roles and assigns it an index and increments the index count.

I also implemented a condition which accounted for hierarchy when adding a sub role to a role. For example, A role cannot be added to its sub role. This was because if a role was added to its sub role, this would lead to an infinite loop of sub roles being added to a role. I achieved this implementation by implementing a add and get parent role method to keep track of each roles parent role, if they had any. Through these methods, the role can verify if the sub role being added to it is one of its parent roles and if it is, it does not allow the user to add the sub role.

I also implemented a feature which deleted a role, [X], from a list of sub roles if a new sub role [Y] added is a parent role of the role, [X], already in the list. This is because, since [X] is a sub role of [Y], [X] is not needed for getting activities because my implementation of the getActivities(…) method is recursive with respect to sub roles of a role.

Another minor bug I ran into was forgetting to set an exit condition for my sorter. I was able to identify this bug after about an hour of debugging by printing my final results to the command line.

RESULTS

Overall, I am satisfied with the current version of the program even though there were a few features I could not implement due to time constraints. To test the systems functionality,

I went through different components and menus of the program and extensively reviewed several possible cases for which the system will be used.

In this section, I have attached several images of the menus through the program and demonstration of features that were implemented. During this testing process, I observed and resolved a few bugs which I may not have noticed otherwise. These bugs will be discussed in the discussion section below.

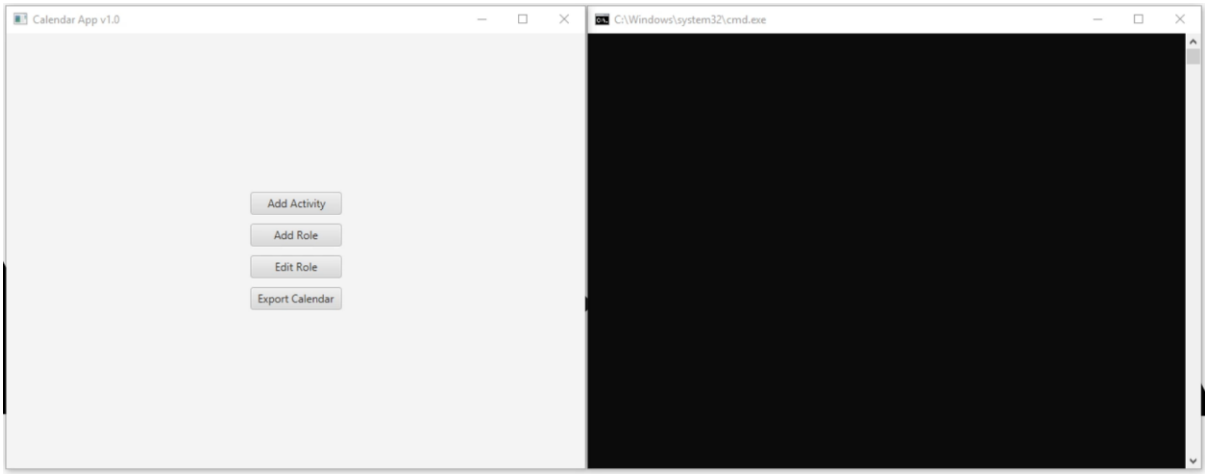


Fig. 1. Main menu and Terminal window for output.

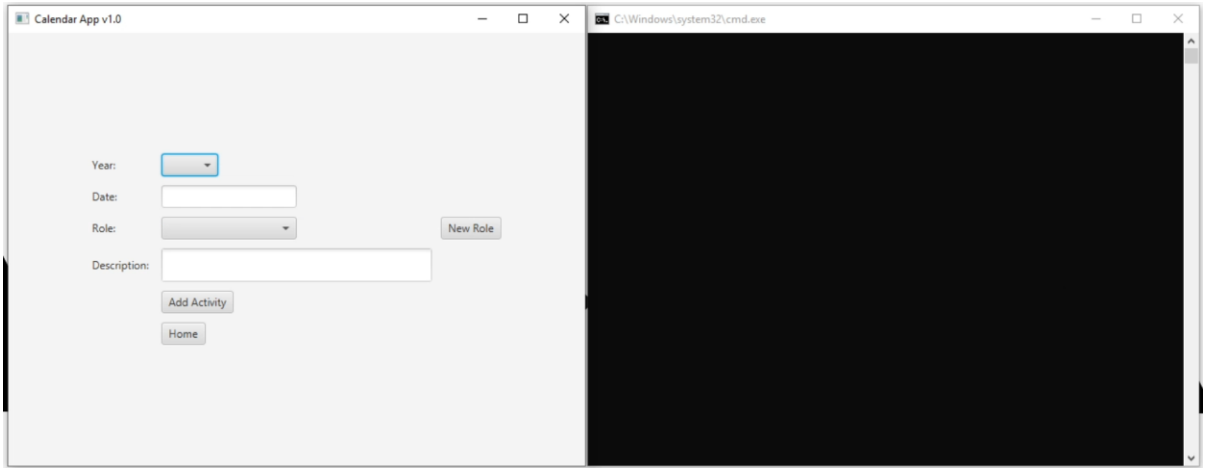


Fig. 2. Add Activity menu.

The “Year” option is a drop-down list with “EVEN”, “ODD”, and “EACH” options. The date box accepts dates in “Month” or “Month1/Month2” formats. If the user inputs an invalid date/date format, the dialog box shown in Fig. 3. pops up.

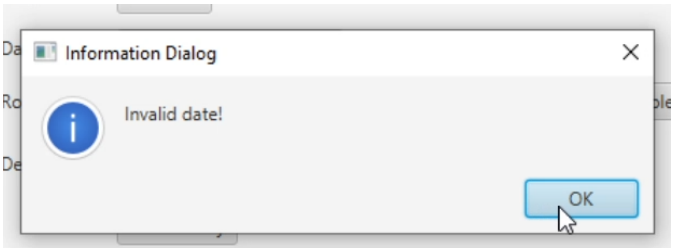


Fig 3. Date verification Dialog box.

The “New Role” button shown in Fig. 2 in the add activity menu redirects the user to the “Add Role Menu” shown in Fig. 4. to allow the user to create a new role it the role needed for the activity does not already exist.

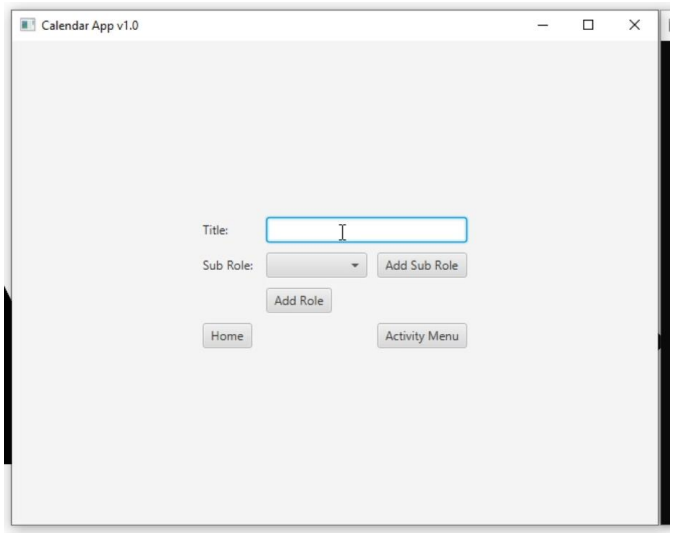


Fig 4. Add Role menu.

The sub role the user decides to add can be selected from a list of preexisting roles. (Excluding the all role, The program prevents the creation/modification of the role called “all”). When the user selects a sub role, they must click the “Add Sub Role” button before selecting the “Add Role” button which creates the new role. The “Activity Menu” redirects to the Activity menu Fig. 2. The “Home” button redirects to the main menu Fig. 1.

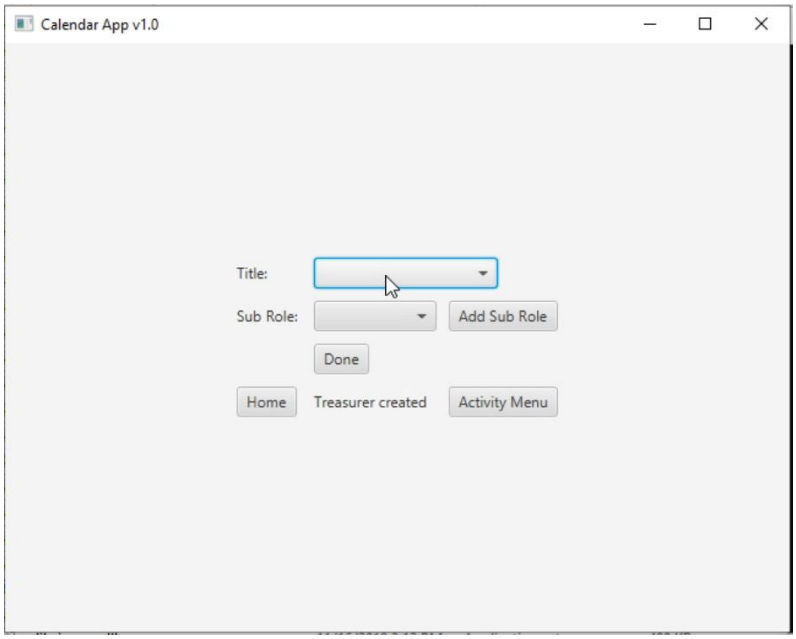


Fig 5. Edit Role Menu

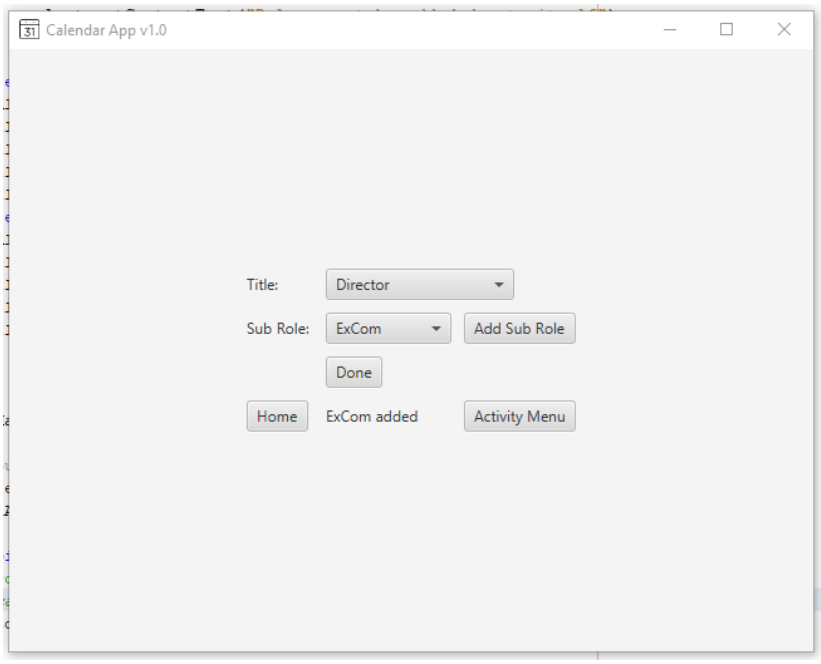


Fig 6. Edit Role Menu for modified role.

The “Title” is a drop-down menu of existing roles and the “Sub Role” is a also drop-down menu of existing roles. The user must click the “Add Sub Role” button before clicking the “Done” button. Once the user selects the “Add Sub Role” button, an alert shown up at the bottom of the page with the name of the role follwed by “added”.

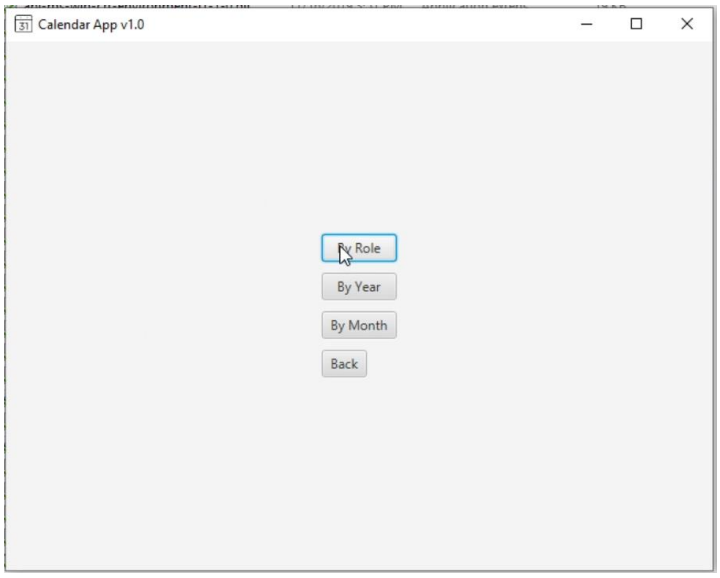


Fig 7. Export Calendar Menu.

The “Export Calendar Menu” shown in Fig 8. contains three buttons that redirect to different menus. The “By Role” button redirects to a menu for exporting/printing by role, shown in Fig 8. The “By Year” button redirects to a menu for exporting/printing by Year only, shown in Fig 10. The “By Month” button redirects to a menu for exporting/printing by Month only, shown in Fig 12.

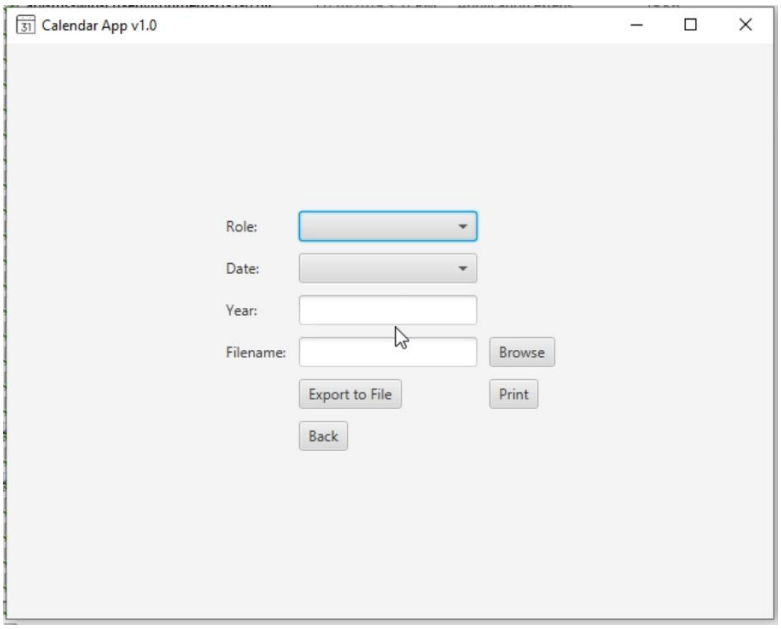


Fig 8. Export by Role menu

If a role is selected from the list of available roles, and the other fields are left blank, the print and export buttons will print the activities for that role to the command line and export the activities to a chosen file respectively. The “Browse” button allows the user to pick the

location and file name to export the activities into. The user can export to TXT of ICS formats. If a date and role is selected from the list of dates, the activities for a role on a certain date is exported. If a role and date are selected, and a year is typed, the activities for the role during that year is exported. If no role is selected, a dialog box with the information "No role selected!" is displayed.

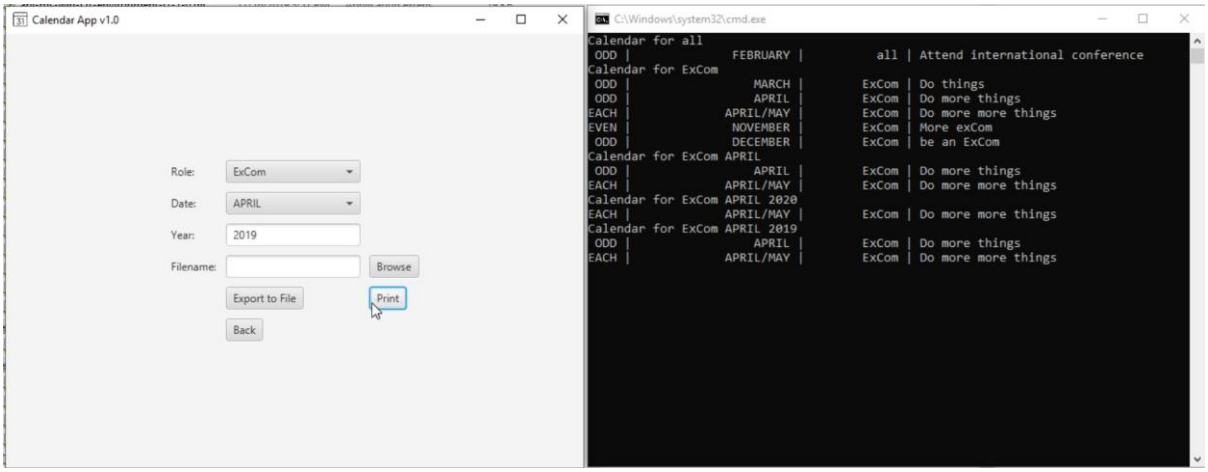


Fig 9. Sample output for Export by Role Menu.

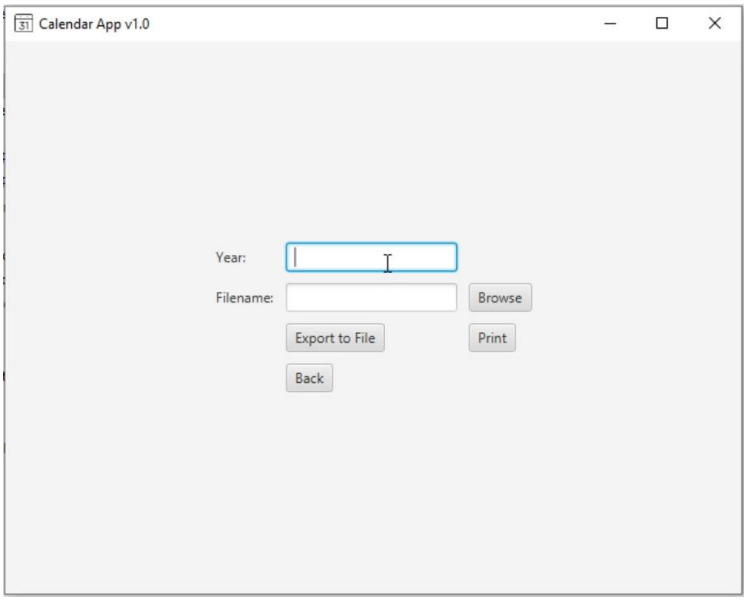


Fig 10. Export by Year Menu

This menu takes in a year number input i.e 2019, 2020 and exports the roles for that year based on the year type [EVEN, ODD, EACH]. The user also has the option to redirect the output to a file.

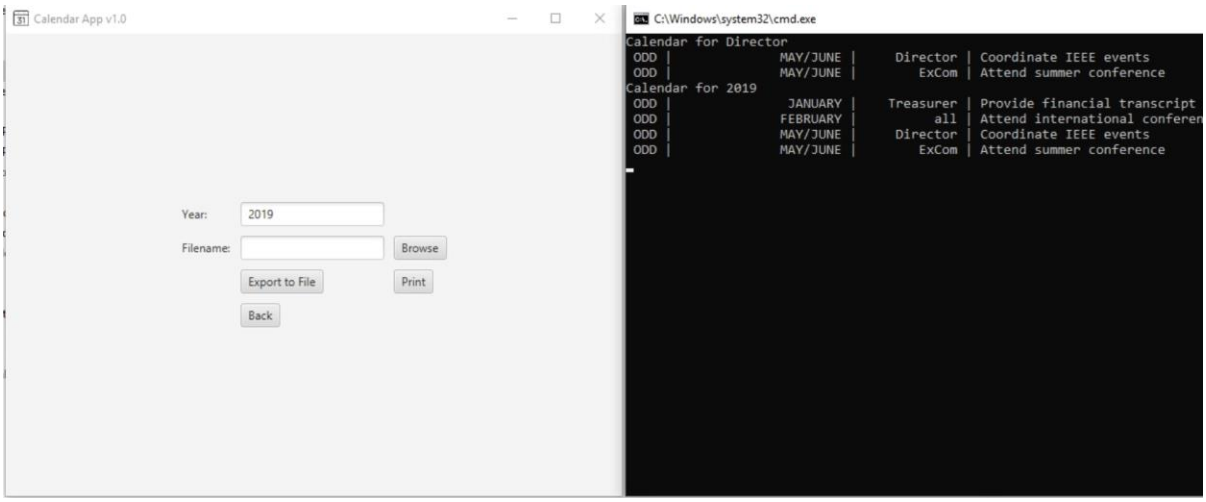


Fig 10. Sample output for exporting by Year.

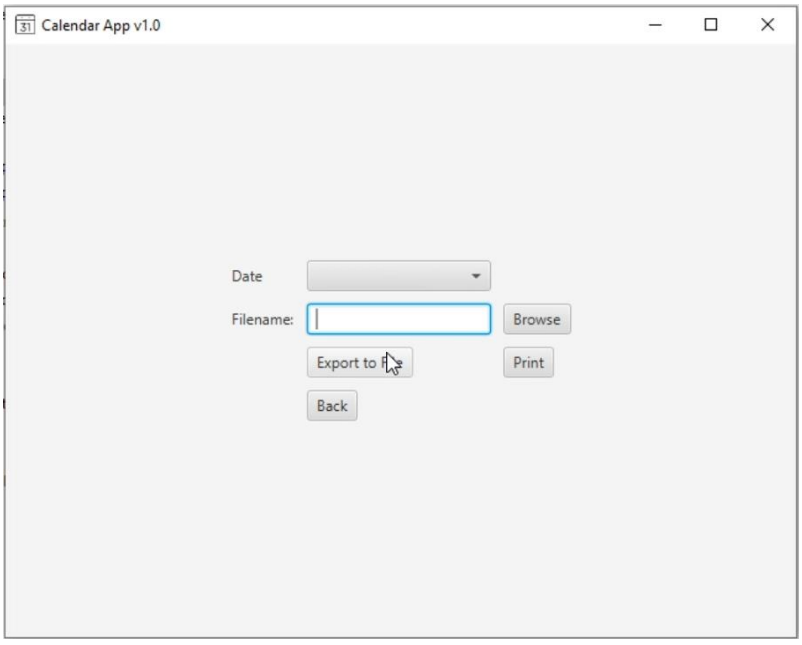


Fig 11. Export by Month Menu

This menu allows the user to select from a drop down list of months and export based on the selected month.

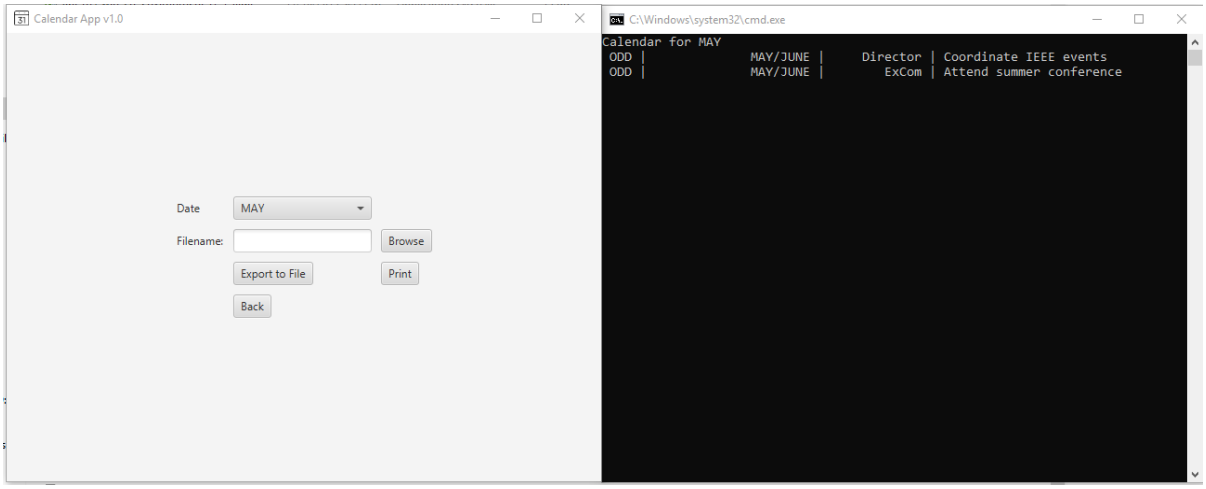


Fig 12. Sample output for Exporting by Month

DISCUSSION

The results were able to satisfy most of the goals, constraints, and standards set for the project. Although I had to make a few modifications to account for unforeseen conditions, the program was able to run and function properly. One aspect of the modification process involved the creation of several classes that were unaccounted for in the design phase. Including:

1. **Loader**: for loading and saving the history
2. **Sorter**: for sorting and removing duplicates from the lists of roles or activities.
3. **Exporter**: For exporting and printing the list of activities.
4. **CalendarApp**: The main app for the program.

There were a few features I was unable to implement due to time constraints, however, I built my classes with those features in mind in case I decide to implement those features later in the development process.

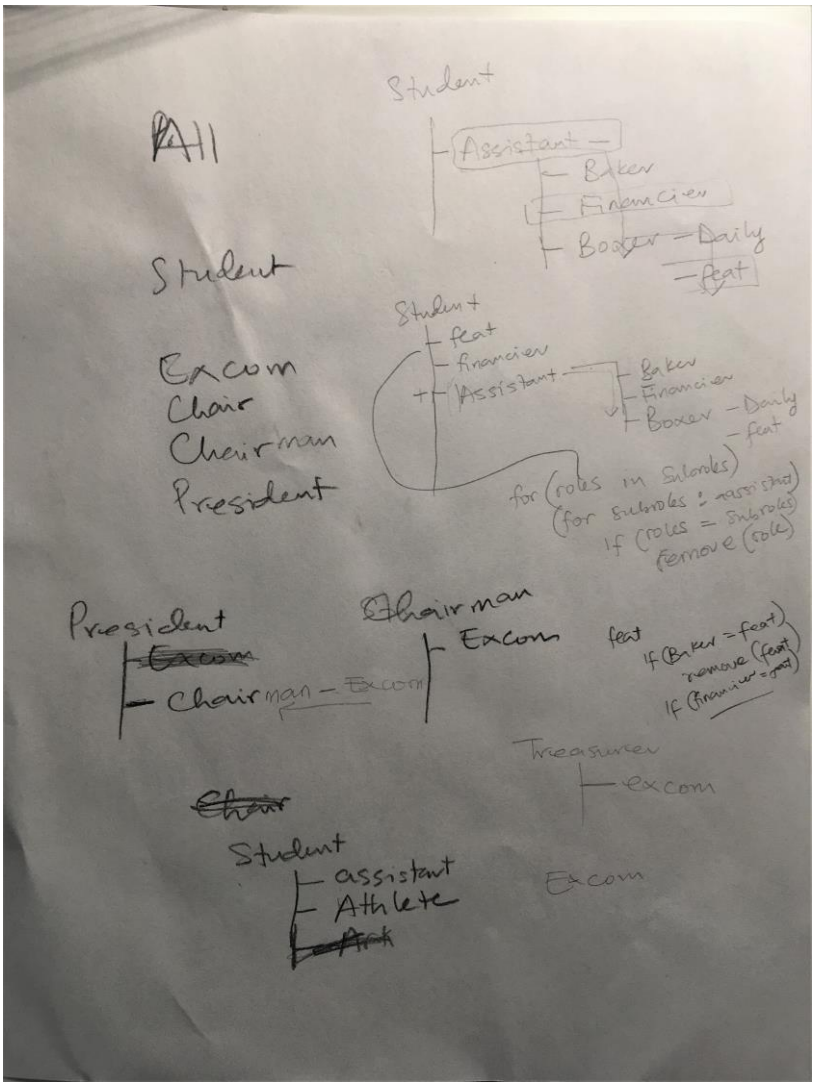
CONCLUSIONS

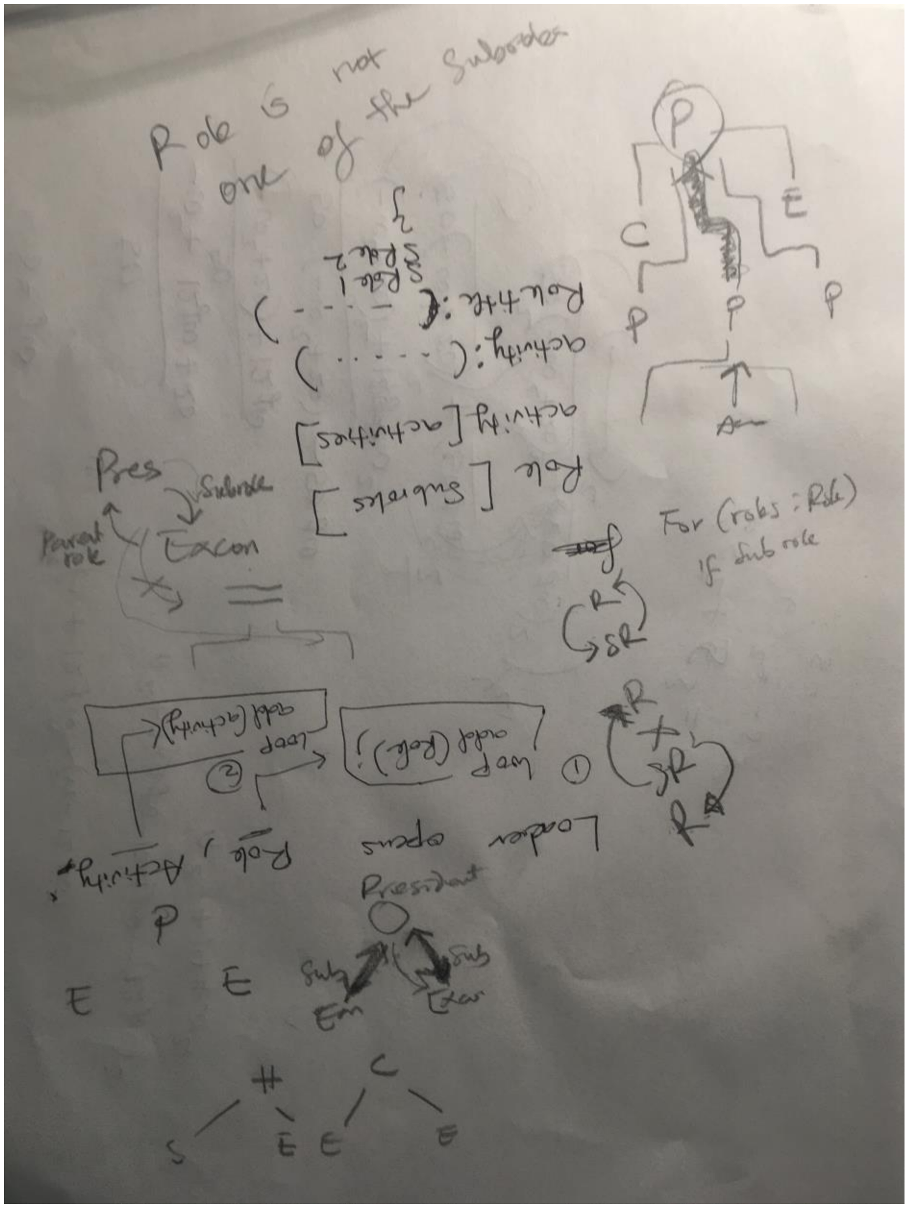
The development of this system was a valuable experience for me. I learned a lot about how the java system works and how to work with GUI. This project also significantly improved my knowledge of the object-oriented design process. My goal is to continue improving the design and try to implement several features in the future. One example of said features is the ability to display the calendar in the app as opposed to displaying it on the command line.

REFERENCES

1. (2006, May, 22) *IEEE Transactions LaTeX and Microsoft Word Style Files* .http://www.ieee.org/portal/pages/pubs/transactions/stylesheets.html

APPENDIX


Student
Assistant
Baker
Financier
Boxer
Daily
feat
All
Student
Excom
Chair
Chairman
President
Student
feat
financier
Assistant
Baker
Financier
Boxer
Daily
feat
for (roles in Subroles)
(for subroles = assistant)
if (roles = subroles)
remove (role)
President
Excom
Chairman – Excom
Chairman
Excom
feat
if (Baker = feat)
remove (feat)
if (financier = feat)
Treasurer
excom
Chair
Student
assistant
Athlete
Excom


Pres
Parent role
Excon
For (roles : Role)
President