**Jeff Angius, Jeffrey Nguyen**

**Jonathon Chenvert, Arthur Hoang**

**CIS-044 Project Report**

**Spring 2016**

**TABLE OF CONTENTS**

[**Introduction**](#Introduction)

[**Body**](#Body)

[**Conclusion**](#Conclusion)

**Introduction**

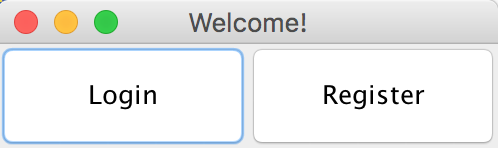
This team’s collaboration led to an idea of developing some sort of event-based to-do list application. We wanted to create something that could, not only connect to a certain date and/or time, but also allow the user to search through the events log for a certain event. In this inspiration, we began to heavily research in event-based programming, Java GUI modeling, and the amalgamation of these core parts to the data structures that we have learned in class, the heart of the program. In doing so, we learned a great deal in those capacities.

First, we began with the Java GUI model, endeavoring to understand the framing, the panels, the layouts, and all of the pertaining components. As we explored those interfaces, we also explored event programming, programming based on object that represents a user's interaction with a GUI component that can be "handled" to create interactive elements. Once the event occurs, it necessitated the listener, an object that waits for events and responds to them. For example, to handle an event, we attach a listener to a component, and the listener will be notified when the event occurs.

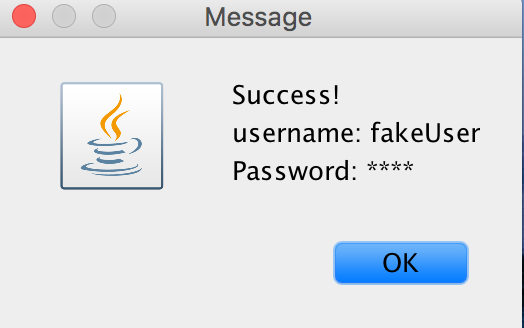
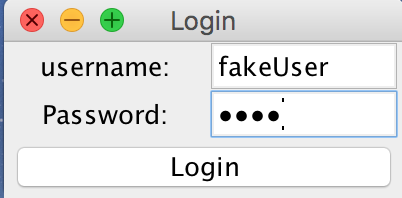
Most importantly, we examined our lessons to modify the data structure codes so that we could implement it into our program. The data structure we chose that best suited our needs was the Linked Bag system. In implementing it, we required a means of sorting through the user input data. For this task, we realized that the search algorithm best suited for our requirements was the Merge Sort process.

**Body**

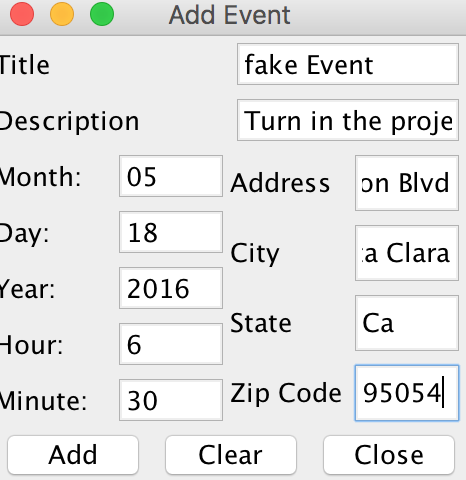
We built the application with the explicit focus on the user. The program initiates by displaying a window with two button options, Register or Login.



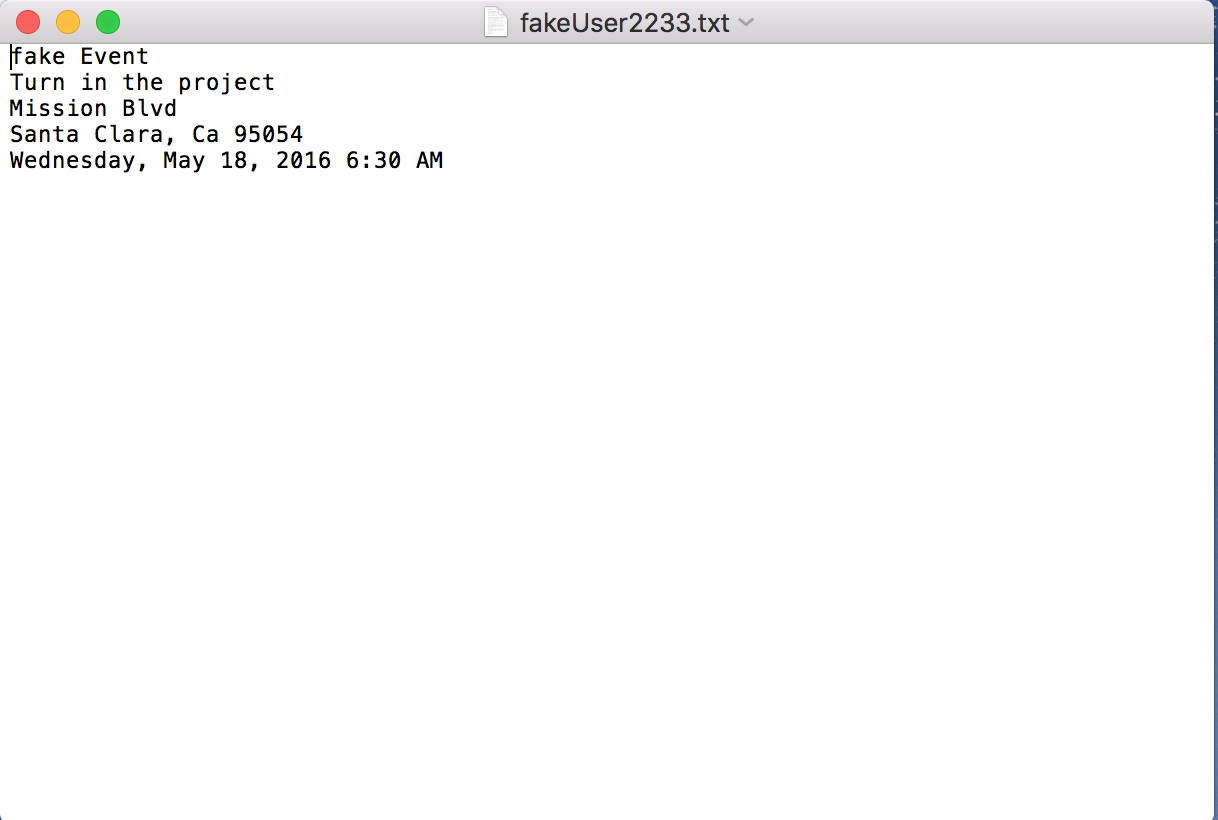
This window was formed with the Java GUI (Java graphical user interface). From the Swing libraries, we were able to create the button components Login and Register and place them in a JFrame. Once the components were settled into their containers, we used the ActionLIstener Interface to check whether an action or event (which could be a mouse click or the press of a button) has occurred. If it has, then ActionListener will accept it as an event variable. This variable is then checked in an if statement parameters to see if it is valid. If so, it then begins a try / catch condition. In the try block, three variables are created. Two of the variables are there to hold the user input of username and password, and the third variable holds the combination of the username and password and creates a .txt file of that. The catch block, checks whether or not the user has inputted valid information. If not, the catch block throws a warning.

The same code structure was constructed for the Login button. Only now we are not making a file, but scanning for it. If the user input is correct, the program would continue. If the user input was incorrect, it would trigger a pop-up window to the user stating the invalidity. Once the login was successful, the user is prompted with the window to create an event he or she wants recorded. 

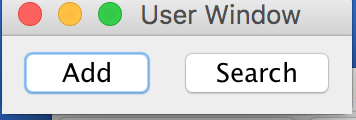
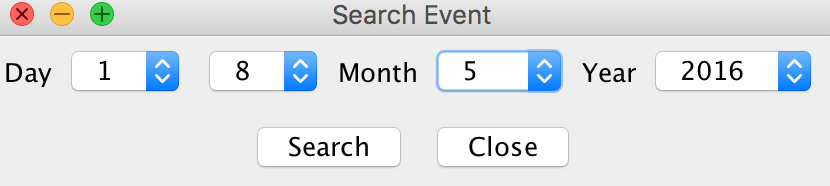
The Event Window allows the user to add unique events to his/her collection of Events. To add an event properly the user must type in all required information in the Event Window’s TextFields, and then click the “Add” button. When the “Add” button is clicked, an ActionEvent is fired to gather all the data in the EventWindow’s TextFields, store that data in a new Event object, and finally add that Event object to the user’s linkedBag. When the user wants to clear all of the EventWindow’s TextFields, he/she must click the Clear button. Lastly, when the user wants to print his/her linkedBag to their TextFile, the user must click the close button, which will fire an ActionEvent to print the entire linkedBag using the method LinkedUserFile.print().



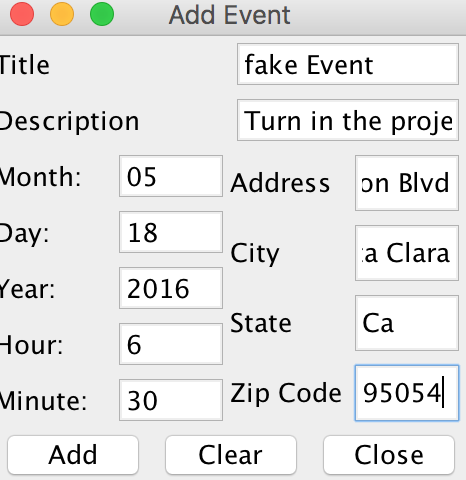
For the event class itself, we had variable types that are typical to an event: the title of the event, the description, the month, day, year, hour, minute, address, city, state, and zip. The class has typical set and get methods, though there are three methods that set it to be different: getMilitaryTime(), compareTo(), and equals() method. The equals() method allows us to be able to compare all the event dates within a given bag and see if they are of equal values. This helps since we have a method within LinkedBag that relies on this method (arrayTransfer). The getMilitaryTime() method returns to us the actual military time by concatenating both the hour and the minute integers. Our compareTo() method compares the military time of both event objects, and checks to see whether the military time is greater than or less than. Thus, the user has created their first .txt file storage of data.



Now, in order for the user to sift through its events on some specific dayand maybe add some more reminders, we needed to create a search and add window. The Search Window allows the user to search for specific dates in the user’s TextFile that holds his/her Events. To search for a date the user must select integers from the fields, “Day”, “Month”, “Year” and then click the button “Search”. The “Search” button fires an ActionEvent that gathers the integer values of “Date” fields, “Month” fields and “Year” fields. The ActionEvent then passes that information to the method, LinkedUserFile.arrayTransfer(int day, int month, int year). The arrayTransfer method searches through the user’s TextFile of events and finds every Event that matches the date that was passed to the arrayTransfer method. As the arrayTransfer method finds Events that match, it adds them to an array of Events and returns that array to the user. Finally, the array of Events that the arrayTransfer method returns is passed to the method, LinkedUserFile.sort(Event[]) method. After the array of Events is sorted, the array is printed in the toDoListWindow().

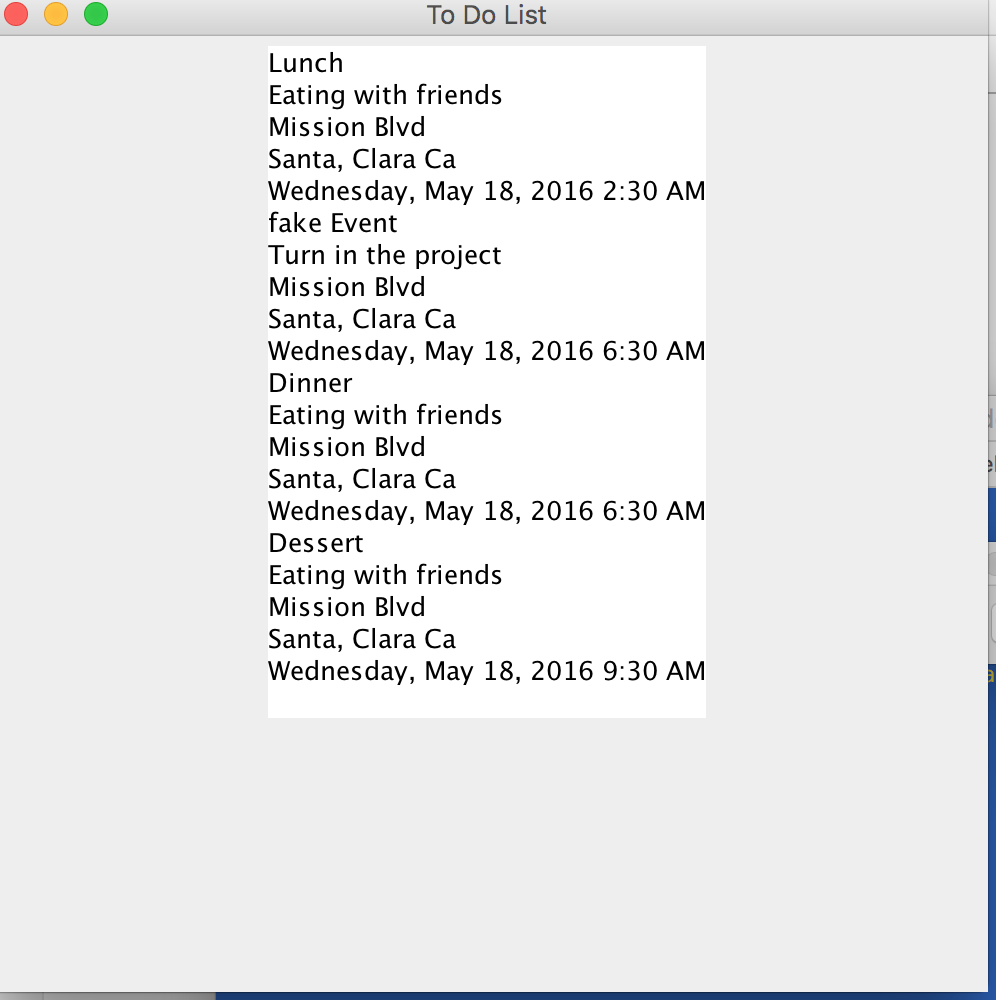


The Add window allow the person to add or search data that is being implemented to a LinkedBag which it converts into a text file using it’s print method. The add button fires an action event which opens the Event window, which allows the user to input their data. In this case, it would be the: Title (name of event), Description (describing the event), Month (adds current month/past/future events from 1 to 12), Day (from 1-31 days), Year (from 1976 to any near future), Street (your current valid address), City (any valid city), State (any state, 50 states), and ZipCode (a valid 5-digit number). This will be stored into a LinkedBag in a text file



            Within the LinkedBag class, everything is the same, except for the arrayTransfer() method. The arrayTransfer() method accepts parameters for month, day, and year, which is all given via user input. This then becomes a new event object that can be used to check and see if all events in the bag are equal to the user’s search for the newly created event object. If it is, then the event object in the bag gets written to an event array, ultimately returning the event array that has the same event dates as the search criteria given. In this fashion, we are able to search through the inputted dates of the given events.

            In trying to sort through the data, the compareTo() method was utilized in the event class to be able to sort with the Merge Sort method that we worked on earlier in the semester to be able to sort the event array. With the event array from arrayTransfer(), the array can be thrown to be sorted, and return completely sorted. We searched through the created To-Do List.



The toDoListWindow accepts an array of Events and prints them to a JTextArea. It is the window that allows the user to see the chronological order of events that correspond to a given date.

**Conclusion**

Hence, we have a number of plans in store for the future of this project. One of our intentions is to implement a remove button within the GUI interface, and have a remove method that consists with the button. This method should remove a specified method that the user requests to be removed. Another plan for the future of this project is to make the GUI look a lot better, and be ready for other platforms, such as Android and iOS. Before porting to a different platform, however, we need to have two things: have the option to edit the event in question should the user need to change something about the event, and the program should also be able to throw notifications to the user, letting them know that their event is coming up soon, or be a modifiable notification that the user can preset to send them a reminder when the event will take place. For the user, we also would like to show them when their event was created, and when the event was modified as well. Lastly, we would like to store the information that the user inputs into an LDAP server, and modify our program to search from the LDAP server. We will also work on securing our LDAP server to prevent people from hacking into our server. These are, essentially, our plans for future modification of our project before we have an official beta.