# Criterion A:

## The Scenario:

The client, I, has found themselves in a to-do list limbo. There already exist hundreds of to-do list applications and time managers, but none fit the way that the client thinks. Most of those programs were designed for professionals, not students.

The client has found that having a running list of tasks—one wherein all tasks are always visible, regardless of due date—is the most useful way to organize tasks. Paper does this quite well, but it’s bulky and “finished tasks” stay on the page when crossed out. Furthermore, paper is not always on one’s person, but the client always carries a Windows laptop.

Hence, I proposed a software solution that will have a running list of tasks, and, additionally, will include a calendar that can be used.

## The Rationale for the Solution:

This solution addresses the problem because it will be a To-Do manager that fits in the Client’s niche. The Solution should run on Windows, as that is the client’s main operating system. This project will be a standalone application to improve the ease of use for the client. It is intended to be eventually ported for use on an Android phone, so Java was an appropriate language to code in. Additionally, I have been working with mostly Java for most of my previous projects.

## Success Criteria:

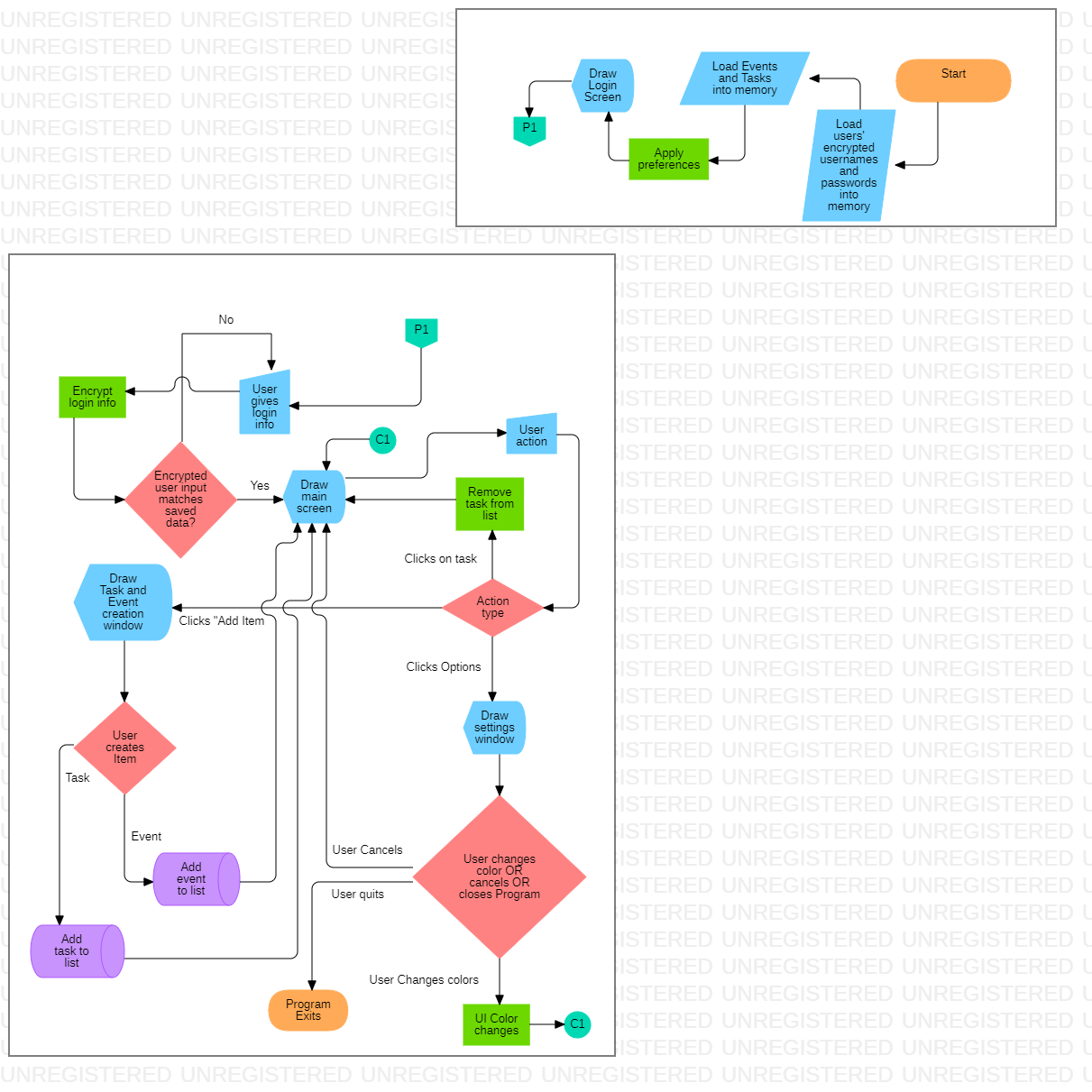
1. Client can add and remove tasks and events with ease.
2. Client can add due dates to tasks.
3. Client can add reminders for events and tasks.
4. Client can add priority levels to their tasks.
5. Client can see completed tasks.
6. Client can change the proportion of their UI used by the calendar and task list.
7. Client can change the colors of their UI.

# Criterion B:

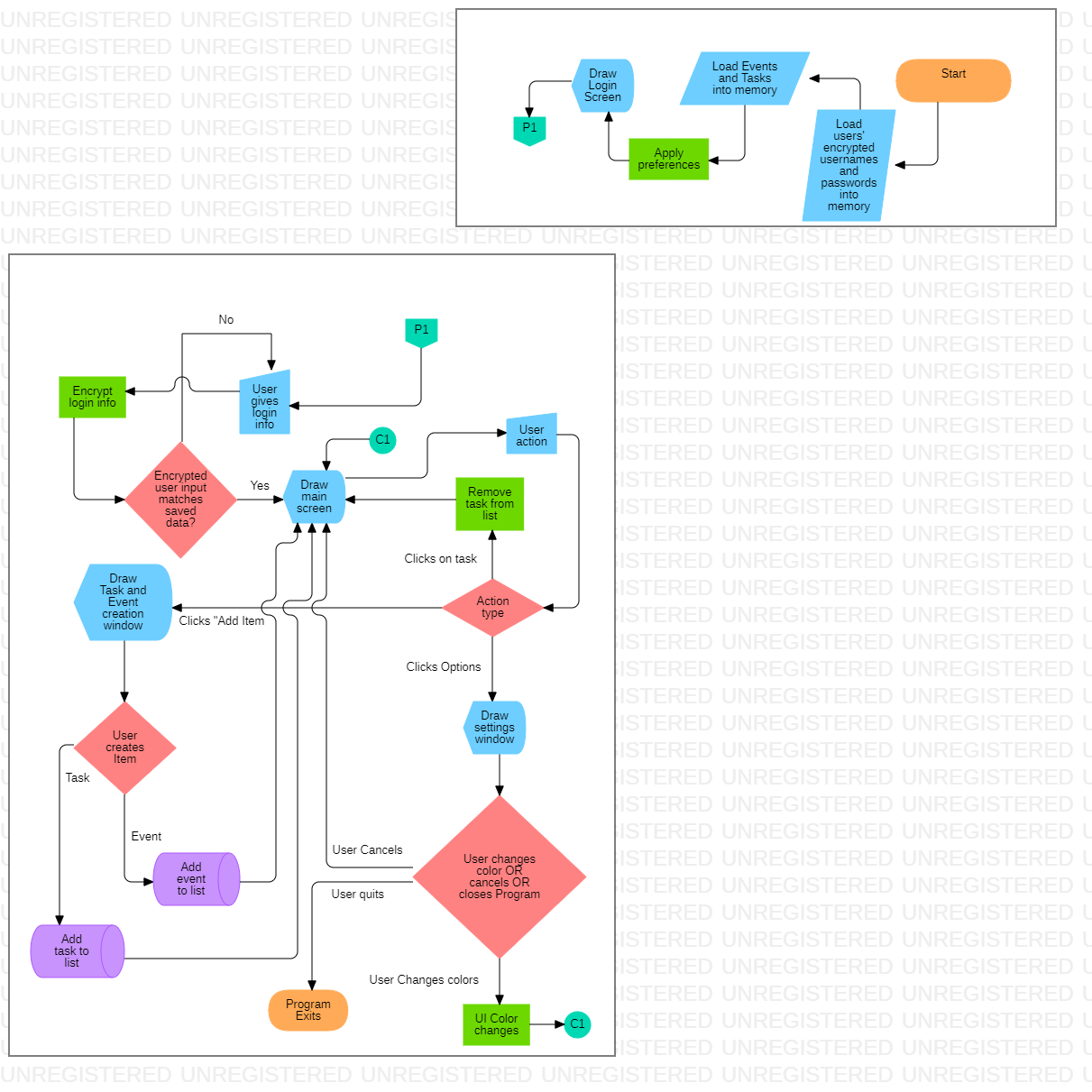
## Tests:

|  |  |  |
| --- | --- | --- |
| Criteria | Test | Result |
| 1. Can the client add and remove tasks and events with ease? | Check that tasks and events can be added with ease. | Pressing the add button brings up a menu where tasks are added to the task list. |
| 2. Can the client add due dates to tasks? | Check that tasks can have due dates added to them. | Clients can be notified when tasks are due. |
| 3. Can the client add reminders to events and tasks? | Check that events and tasks can be given reminders. | In the “Add item” dialog there is a place to add reminders. |
| 4. Can the client add priority levels to their tasks? | Check that tasks can be given priority levels. | In the “Add item” dialog there is a JSlider to choose priority levels. |
| 5. Can the client see completed tasks? | Check that completed tasks are still accessible to the client. | Clients can press a button and be shown completed tasks. |
| 6. Can the client change the proportion of their UI used by the calendar and task list? | Check that the calendar and task lists are resizable. | The JSplitPane has a built in resize bar. |
| 7. Can the client change the colors of their UI? | Check that UI colors are user mutable. | The “Options” dialog has color settings. |

## Flow Charts

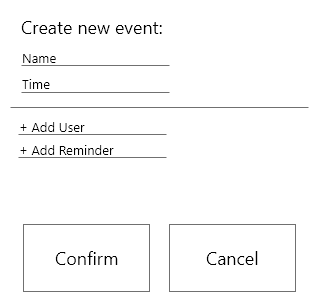


## Flow Charts

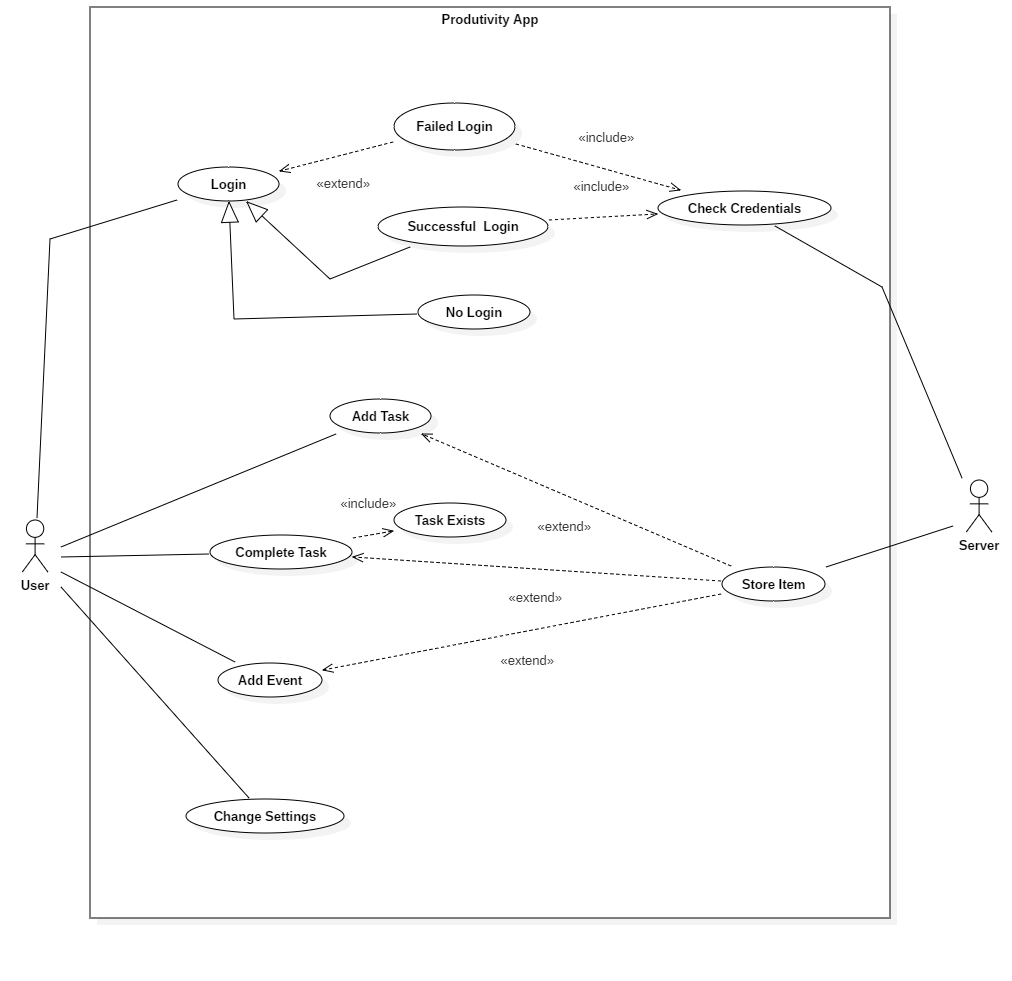


## Planned Graphics

## 



## Planning Documents



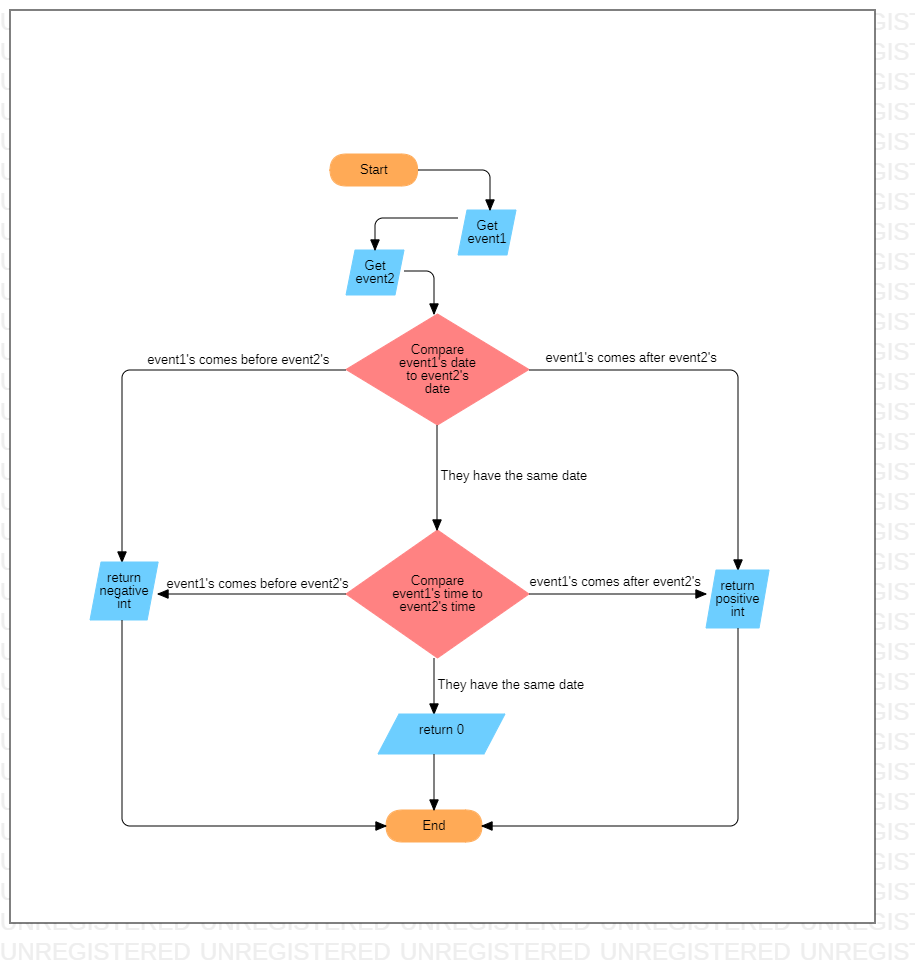
## Key Algorithms

TaskComparator

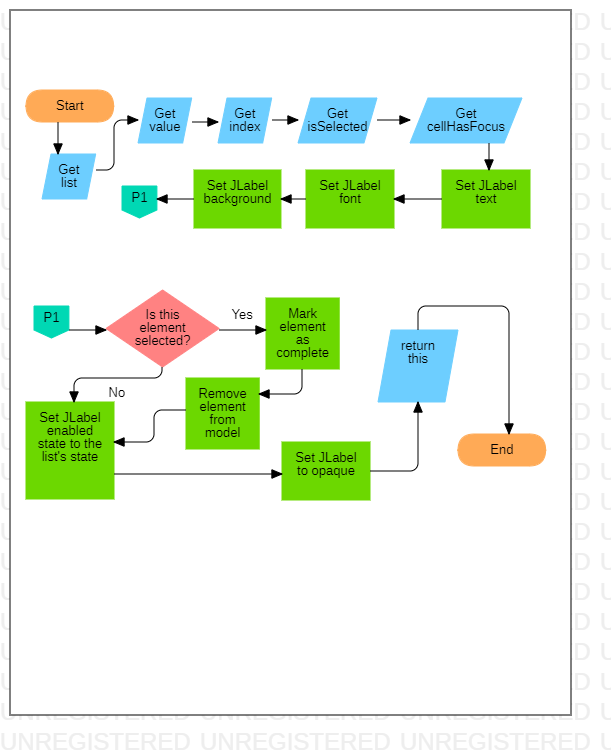
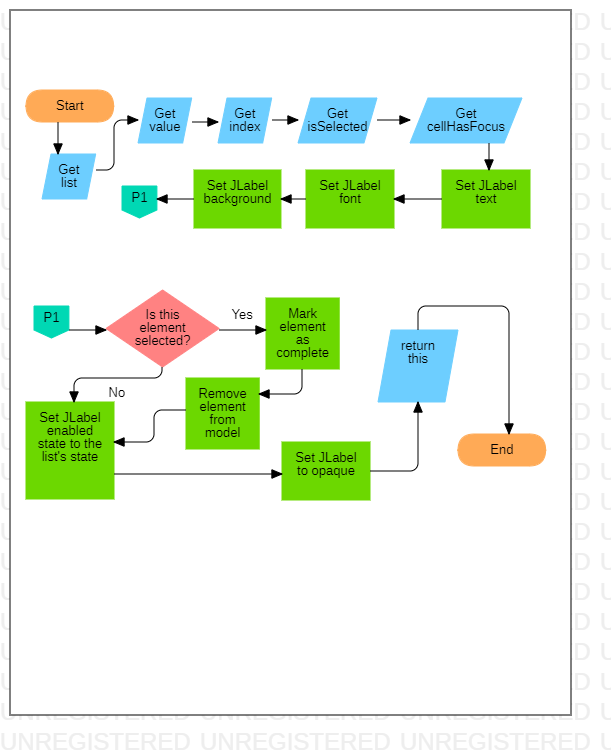
I had to implement a comparator for my custom “Task” class so I could order my tasks by priority in Java Swing. This comparator compares task priorities and then, as a last resort, to settle differences, task names.

### TaskComparator

EventComparator  
I had to implement a comparator for my custom “Event” class so I could order my events by date in Java Swing. This comparator compares event dates and then, as a last resort, to settle differences, event times. This ensures the proper ordering of events.

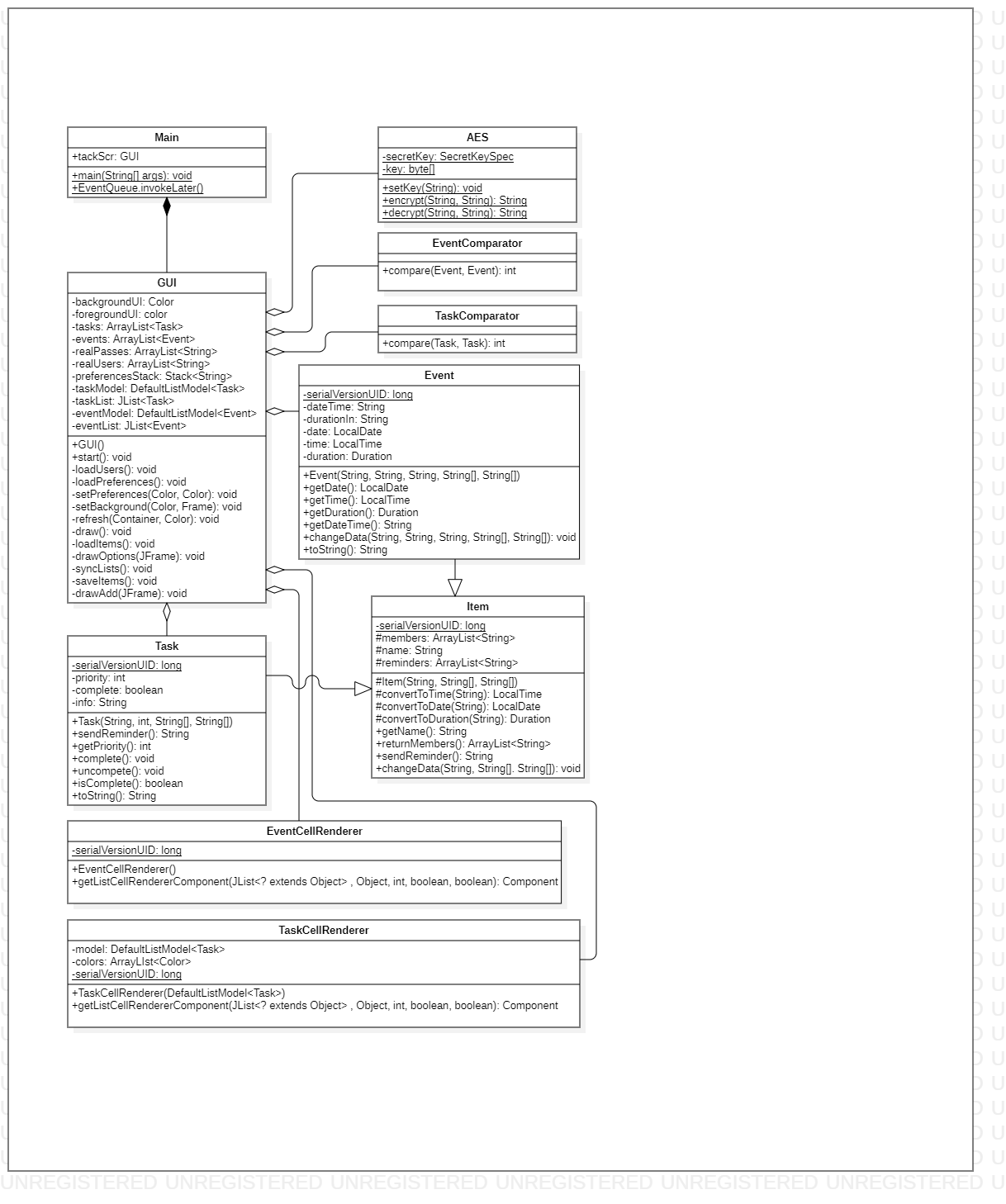


### TaskCellRenderer

In order to draw tasks in Java swing, I had to create a cell renderer. This extends JLabel so I can use the rendering methods and properties that exist in the JLabel class. This renderer essentially just takes the Task’s name and displays it in a JLabel. However, it does have an important functionality where, when the cell is selected, that Task is removed from the JList model (this removes it from the list of drawn tasks, but not from the task list or drive. 

# Criterion C:

## UML Diagram



## Complex Coding Techniques

### Stacks

On lines 52, 142–150 of GUI.java, I used the Stack collection type to store preferences data from the preferences.txt file. I used stacks because each preference only needs to be loaded once and because the speed of stacks’ get and pop methods allow future expansion of the preferences.

### Serialization

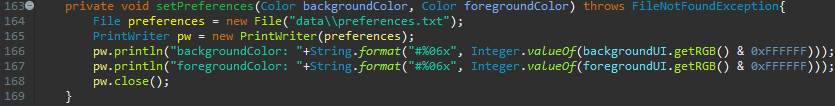
### 

### 

### 

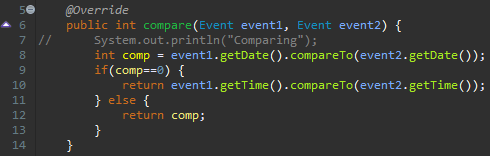
On lines 396–420, 534–561 of GUI.java, I used serialization to store and retrieve Task and Event objects. This was enabled in the class declaration of Task.java and Event.java by implementing Serializable. While I could have written to files in a non-serial manner and then tried to decrypt that stored information, using such a method is considered bad practice.

### Files

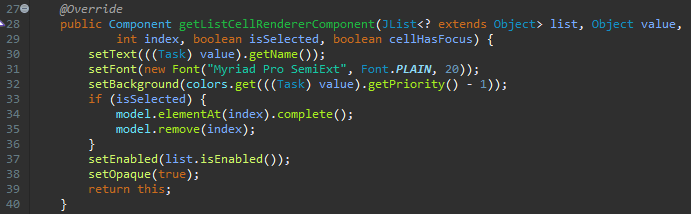
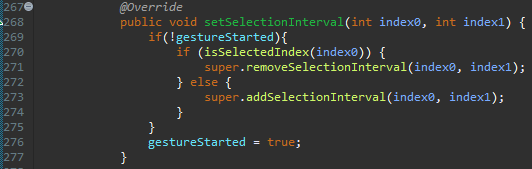


In the loadUsers(), loadPreferences(), setPreferences(), loadItems(), and saveItems() methods of GUI.java, I used files to store and retrieve data. Files were applicable because I needed a way to keep information outside of runtime—your responsibilities do not disappear when the program closes (I wish they would), so their data should not either.

### Method Overriding



### Method Overriding



On line 267 of GUI.java, in TaskComparator.java, in TaskCellRenderer.java, in EventComparator.java, in EventCellRenderer.java, *et. al* I used method overriding to implement the logic to render Tasks and Events. I also used an overridden method written by FuryComputers to prevent an error I encountered where tasks would be continually deleted if you held down left click in the task list.

### Comparators





### Comparators

In order to sort tasks in my task list and events in my event list I used the Comparator interface in both TaskComparator.java and EventComparator.java. This interface was necessary to enable the use of the Collections.sort() method in GUI.java.

### Java Swing

I used Java Swing for all my UI rendering. It was useful because it has so many unique components with pre-implemented functions.

### Recursion

### Recursion

I used recursion for setting the background color of all components in GUI.java because it was the only way to do it simply and effectively.

### Try-Catch

I used Try-Catch when trying to access files in order to handle the error thrown when files aren’t found. And to handle improper input into message dialogs. Both uses are seen in GUI.java

### Inheritance

I used inheritance to consolidate code between Task.java and Event.java. I also used it to enable the proper rendering of Tasks and Events.

### Implementation

I used implementation to enable the use of existing classes in the Java libraries and allow the overriding the relevant methods.

## Development Sources

FuryComputers. (2012). “Fixing over selection”. StackOverflow.com. Available at: <https://stackoverflow.com/questions/2528344/jlist-deselect-when-clicking-an-already-selected-item> [Accessed 15 Mar. 2020].

Lauener, D. “Pattern Title.” RegExLib.com. Available at: <http://www.regexlib.com/REDetails.aspx?regexp_id=1038> [Accessed 20 Mar. 2020].

R. L. Rodrigues, C. “Pattern Title.” RegExLib.com. Available at: <http://regexlib.com/REDetails.aspx?regexp_id=981> [Accessed 20 Mar. 2020].

Gupta, L. “Java AES Encryption Decryption Example.” HowToDoInJava. Available at: <https://howtodoinjava.com/security/java-aes-encryption-example/> [Accessed 19 Mar. 2020].

# Criterion E:

## Meeting Success Criteria

As shown in Criterion D and B, the client can add and remove tasks and events with ease. The client can add due dates and priority levels to tasks. They can also add reminders to tasks and events. Clients can see completed tasks with the click of a button. The client can easily change the proportion of the UI used by the calendar and task list, and they can change the colors of their UI.

## Client Opinions

The product impressed the client, me. I thought that it appropriately filled the niche I was looking for. I really like the way that tasks are just in a running list, but when removed don’t stay there. The immutability of paper was its main weakness for me. I also like how simple your tasks can be. I’ve found that, when I use my default productivity app, I must fill out too much information for tasks that I can recall with a mere name. With TuDu I can simply add a task with a name and a priority. Also, the events list is nice because I can keep track of events without changing applications or screens.

However, despite its utility, I think it could be improved. For example, I don’t really like the current UI, it reminds me too much of early 2000s computing. I also think that in order to reach my standard for daily use the app should allow you to see Item metadata. Further, without the implementation of reminders, I must rely on my own responsibility to check up on the task list—which I don’t trust.

Overall, with the short time budget, I think TuDu was a successful product. I have already seriously considered making it my main To-Do application even if just for its wonderful running list—it just fills my niche perfectly.

## Future Improvements

After receiving feedback from myself I have pinpointed three main improvements and additions that can be made to the program:

### Rework the User Interface

Reworking the user interface would be an important step in improving the user experience. Clean, modern designs have overtaken the traditional look and feel of default Java programs. With a little research into Java “Look and Feel” I could easily apply a broad update to all relevant Java Swing components. I have already experimented with setting development defaults, but I think changing the entire UI with Java’s “Look and Feel” would be a very effective method for beautifying the program. In the options pane, I could add options to change the look of specific components background colors and text colors alongside the UI overhaul.

### Implement Reminders

To reach the complexity level of many modern to-do managers, I would need to implement reminders. I see two ways of doing this: putting a timer on one thread that counts down to the specified time or scheduling an email be sent at the specified time. The first way would probably use Java’s Timer class, or another class uncovered by further research. The second way would possibly require me to set up an SMTP server and possibly work with API’s. I have some experience with API’s so that could be the more effective option.

### Make metadata viewable and editable

To take this solution past the barebones, I should implement methods to make metadata viewable and editable. I would most likely do this by changing the behavior of my Task’s JList selection handler. I would make selecting an event show a JOptionDialog with metadata, a complete button, and a cancel button. All the metadata would be editable within this dialog. It may even use a slightly modified “Add item” dialog. This change is the simplest and could probably be done in a little over an hour.

# Appendix B: Resources

## Tools

### Window Builder

I used the window builder extension for the Eclipse IDE to build my JSwing frames and panels.

### StarUML

I used StarUML to make UML diagrams, flowcharts, and UML use case diagrams.

### Adobe XD

I used Adobe XD for the initial UI mockup.

### Regexr.com

I used regexr.com to try and understand regular expressions.

## References

Diana’s Programming Tutorials (2015). JFrames, JPanels, JButtons, and Drawing with the Java Swing Class. YouTube. Available at: https://www.youtube.com/watch?v=zG8CrISqPpU [Accessed 20 Mar. 2020].

Docs.oracle.com. 2020. Oracle Help Center. [online] Available at: <https://docs.oracle.com/en/> [Accessed 20 March 2020].

GeeksforGeeks. (2018). Java Swing | JDialog with examples. [online] Available at: https://www.geeksforgeeks.org/java-swing-jdialog-examples/ [Accessed 26 Mar. 2020].

Horstmann, C. [Insert Year of Publication], Big Java: Early Objects, Interactive Edition, 6th Edition, [Insert City of Publication]. Available from: VitalSource Bookshelf.

ProgrammingKnowledge (2014). Java Eclipse GUI Tutorial 1 # Creating First GUI Project in Eclipse. YouTube. Available at: https://www.youtube.com/watch?v=r8Qiz9Bn1Ag [Accessed 19 Mar. 2020].

Vasudev, R., 2014. Java Comparator Explained In 10 Minutes.. [online] YouTube. Available at: <https://www.youtube.com/watch?v=dRX6qO46l44> [Accessed 20 March 2020].

Vogel, L. (n.d.). Regular expressions in Java - Tutorial. [online] www.vogella.com. Available at: https://www.vogella.com/tutorials/JavaRegularExpressions/article.html.