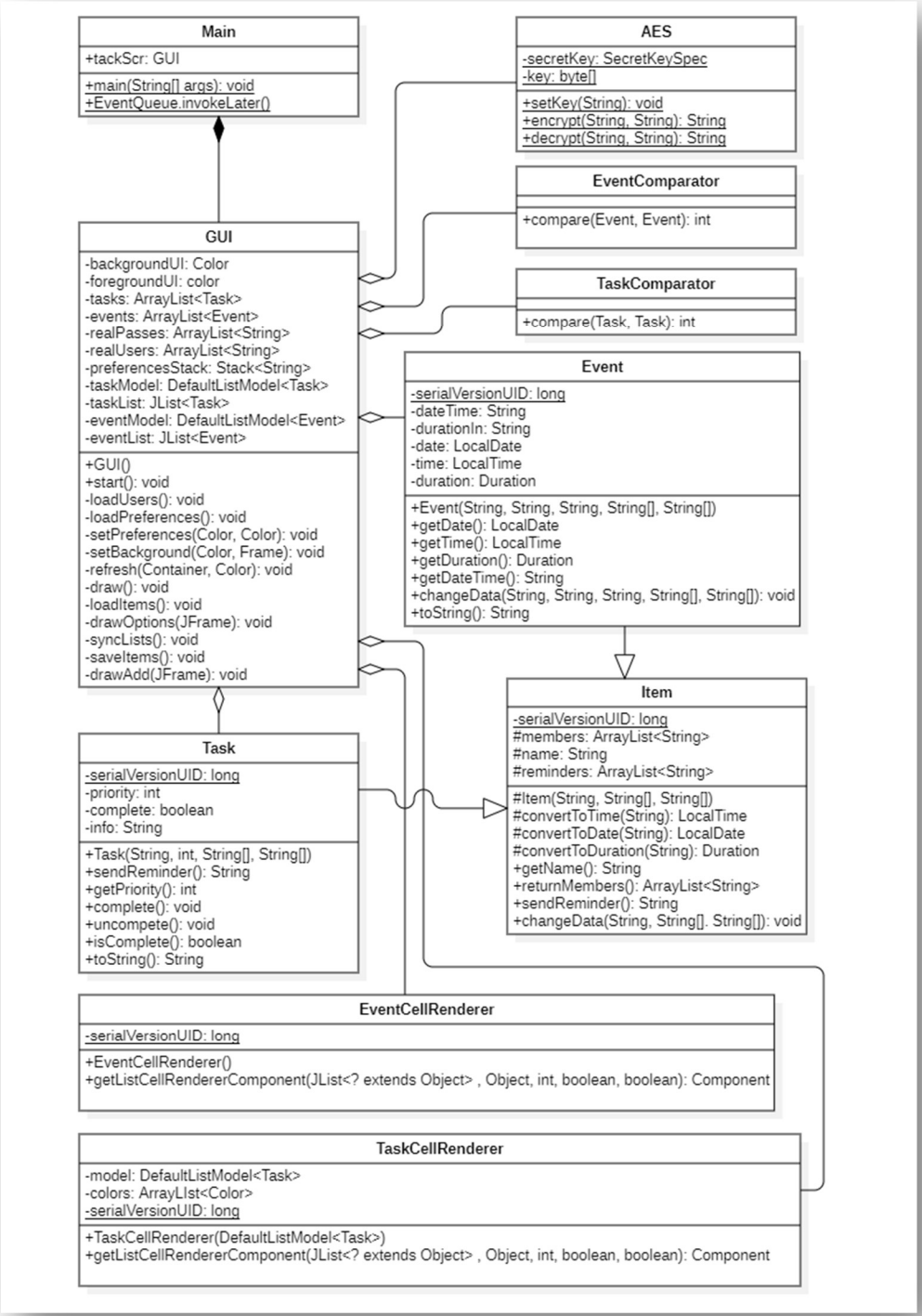
Criterion C:

UML Diagram



Complex Coding Techniques

Stacks

```
52      private Stack<String> preferencesStack = new Stack<String>();
```

```
142             File preferences = new File("data\\preferences.txt");
143             Scanner sc = new Scanner(preferences);
144             preferencesStack.clear();
145             while(sc.hasNextLine()){
146                 preferencesStack.push(sc.nextLine());
147             }
148             sc.close();
149             foregroundUI = Color.decode(preferencesStack.pop().substring(17,24));
150             backgroundUI = Color.decode(preferencesStack.pop().substring(17,24));
```

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

```
396             File taskFile = new File("data\\taskFile.txt");
397             File eventFile = new File("data\\eventFile.txt");
398             FileInputStream tIn = new FileInputStream(taskFile);
399             ObjectInputStream taskIn = new ObjectInputStream(tIn);
```

```
410             FileInputStream eIn = new FileInputStream(eventFile);
411             ObjectInputStream eventIn = new ObjectInputStream(eIn);
```

```
 6  public class Task extends Item implements Serializable{
 7      /**
10      private static final long serialVersionUID = 1L;
```

```
 9  public class Event extends Item implements Serializable{
10      /**
13      private static final long serialVersionUID = 1L;
```

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

```
112             File user = new File("data\\users.txt");
113             Scanner sc  = new Scanner(user);
114             int i=0;
115             while(sc.hasNextLine()) {
116                 if( i%2==0) {
117                     realUsers.add(sc.nextLine());
118                 }else if(i%2!=0){
119                     realPasses.add(sc.nextLine());
120                 }
121                 i++;
122             }
```

```
163     private void setPreferences(Color backgroundColor, Color foregroundColor) throws FileNotFoundException{
164         File preferences = new File("data\\preferences.txt");
165         PrintWriter pw = new PrintWriter(preferences);
166         pw.println("backgroundColor: "+String.format("#%06x", Integer.valueOf(backgroundUI.getRGB() & 0xFFFFFF)));
167         pw.println("foregroundColor: "+String.format("#%06x", Integer.valueOf(foregroundUI.getRGB() & 0xFFFFFF)));
168         pw.close();
169     }
```

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

```
5       @Override
6       public int compare(Event event1, Event event2) {
7   //      System.out.println("Comparing");
8           int comp = event1.getDate().compareTo(event2.getDate());
9           if(comp==0) {
10              return event1.getTime().compareTo(event2.getTime());
11          } else {
12              return comp;
13          }
14      }
```

Method Overriding

```
27      @Override
28      public Component getListCellRendererComponent(JList<? extends Object> list, Object value,
29              int index, boolean isSelected, boolean cellHasFocus) {
30          setText(((Task) value).getName());
31          setFont(new Font("Myriad Pro SemiExt", Font.PLAIN, 20));
32          setBackground(colors.get(((Task) value).getPriority() - 1));
33          if (isSelected) {
34              model.elementAt(index).complete();
35              model.remove(index);
36          }
37          setEnabled(list.isEnabled());
38          setOpaque(true);
39          return this;
40      }
```

```
267             @Override
268             public void setSelectionInterval(int index0, int index1) {
269                 if(!gestureStarted){
270                     if (isSelectedIndex(index0)) {
271                         super.removeSelectionInterval(index0, index1);
272                     } else {
273                         super.addSelectionInterval(index0, index1);
274                     }
275                 }
276                 gestureStarted = true;
277             }
```

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

```
3   public class TaskComparator implements Comparator<Task>{
```

```
3   public class EventComparator implements Comparator<Event> {
```

```
515             Collections.sort(tasks, new TaskComparator());
```

```
523             Collections.sort(events, new EventComparator());
```

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

```
624         panel.add(nameField);
625
626         panel.add(memberLabel);
627
628         panel.add(membersList);
629
630         memberButton.addActionListener(new ActionListener() {
631             @Override
632             public void actionPerformed(ActionEvent e) {
633                 String entry = JOptionPane.showInputDialog(panel, "Member: ");
634                 if(!entry.equals(null) && !entry.equals("")) {
635                     memberList.add(entry);
636                 }
637                 membersList.setText(memberList.toString());
638             }
639         });
640
641         panel.add(memberButton);
```

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

Recursion

```
177     private void refresh(Container parent, Color col){
178
179         for(Component c : parent.getComponents()){
180
181             if(c instanceof Container){
182
183                 if(c instanceof JComponent){
184
185                     c.setBackground(col);
186 //                  c.updateUI();
187
188                 }
189                 refresh((Container) c, col);
190             }
191         }
192     }
```

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

```
459         @Override
460         public void actionPerformed(ActionEvent e) {
461             try {
462                 backgroundUI=Color.decode(backHex.getText());
463                 backHex.setBackground(backgroundUI);
464                 setPreferences(backgroundUI, foregroundUI);
465             } catch(Exception ex) {
466                 JOptionPane.showMessageDialog(options, "Invalid Hex Code [#XXXXXX]");
467             }
468         }
```

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

```
11 class TaskCellRenderer extends JLabel implements ListCellRenderer<Object>{
```

```
6 public class Task extends Item implements Serializable{
```

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

```
3 public class TaskComparator implements Comparator<Task>{
```

```
11 class TaskCellRenderer extends JLabel implements ListCellRenderer<Object>{
```

```
9 public class Event extends Item implements Serializable{
```

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].