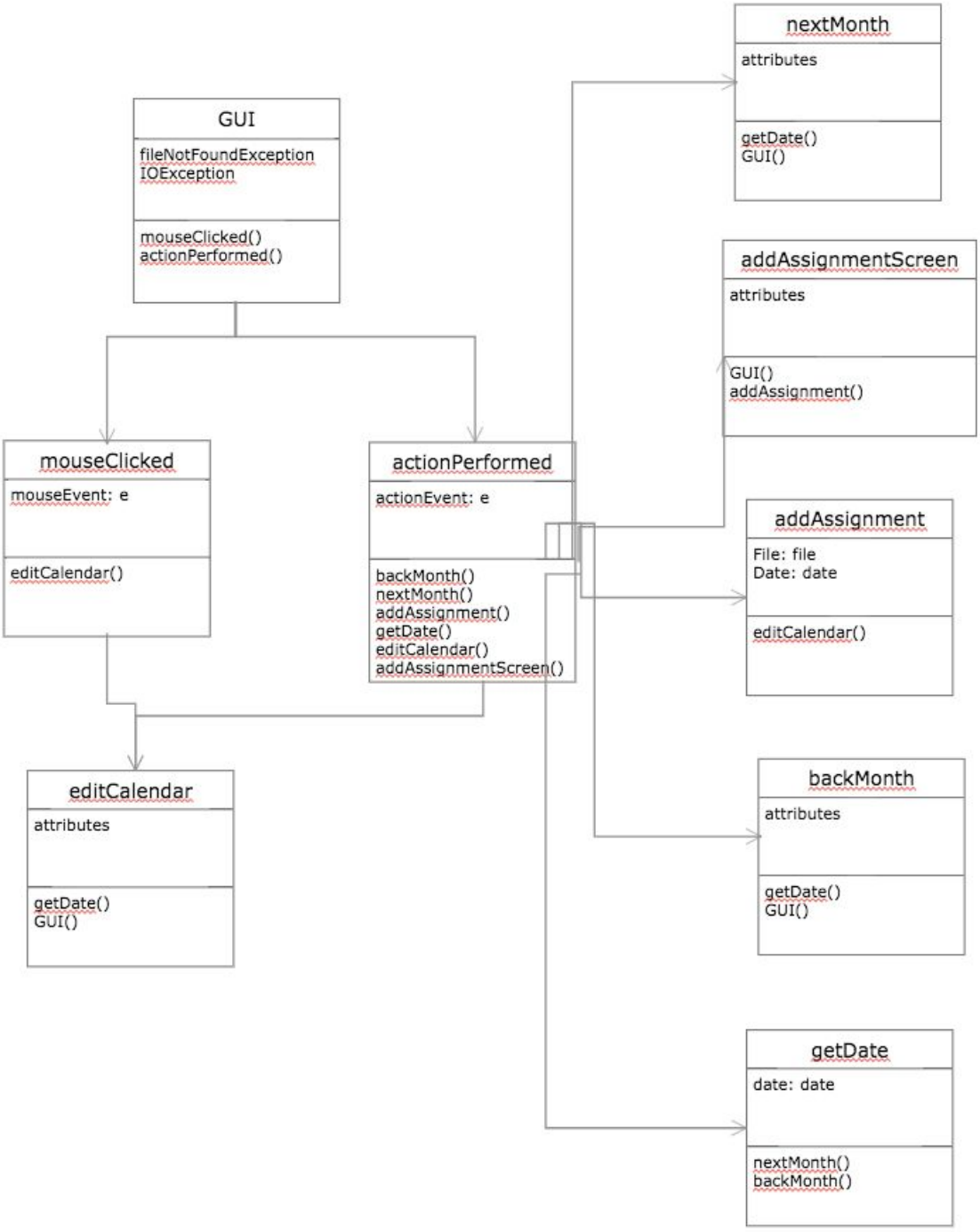
Part C:

UML Diagram:



Complex Code:

File Object:

```
3   public class file {
4       private String date;
5       private String title;
6       private String category;
7
8
9       public file(String date, String title, String category){
10          this.date = date;
11          this.title = title;
12          this.category = category;
13      }
14
15      public String getDate() {
16          return date;
17      }
18
19      public void setDate(String date) {
20          this.date = date;
21      }
```

I used a file object in my program to help me store and access all of the assignments that were added by the user. I do this by first creating a class named file with the private strings “date”, “title”, and “category” as these were the three details of an assignment I wanted a user to be able to save. I then create a “public file” as seen on line 9, which takes in the three string parameters of date, title and category. This allows me to create “file” objects with the parameters date, title and category. I also have getters and setters for each to allow me to get or set parameters of the objects.

File arrayList:

```
90      ArrayList<file> filesArray = new ArrayList<>();
```

```
177         BufferedReader br = new BufferedReader(new FileReader("src//texts//assignments"));
178
179         String line = br.readLine();
180         while(line != null){
181             filesArray.add(new file(line,br.readLine(), br.readLine()));
182             line = br.readLine();
183         }
184         for(int i = 0; i<filesArray.size(); i++){
185             System.out.println(filesArray.get(i).getDate());
186             System.out.println(filesArray.get(i).getTitle());
187             System.out.println(filesArray.get(i).getCategory());
188         }
```

Arraylists are very useful as they are able to hold an unspecified number of objects. This is very useful in my case where I am allowing a user to enter as many assignments as he or she wishes, and I do not have to put a maximum limit on the number the user is allowed to have. This is also useful because the arraylist can store objects, which is what I am using to save the assignment details, and then access those objects later. This code above allows me to read the text file where I am saving the files, and create new file objects each time the program is launched to ensure that my arraylist is filled with all the users assignments. This works by first creating a buffered reader to read the “assignments” text file. It then assigns string “line” to the first line of the text file through the command “br.readLine();” Then it creates a while loop for when line is not null, meaning that there is still more text meaning more assignments, and creates a new file object with the attributes “line, br.readLine(), br.readLine()”, and then adds this new file to the arraylist. This new file is created with the line string, meaning the first line of the text file, and then the subsequent two lines of the text file for the second and third attributes. It then reassigns “line” to the next line in the file, meaning the first attribute in the next pair of three attributes, or it is blank in which case the while loop ends.

Saving with text files:

```
249         public void addAssignment() throws IOException, ParseException{
250             String name = titleOfAssignment.getText();
251             String type= categoryBox.getSelectedItem().toString();
252             String dateEntered = dueDate.getText();
253 
                BufferedWriter bw = new BufferedWriter(new FileWriter("src//texts//assignments", true));
255 
256             bw.write(dateEntered);
257             bw.newLine();
258             bw.write(name);
259             bw.newLine();
260             bw.write(type);
261             bw.newLine();
262             bw.close();
263 
264             filesArray.clear();
265 
266             BufferedReader br = new BufferedReader(new FileReader("src//texts//assignments"));
267 
268             String line = br.readLine();
269             while(line != null){
270                 filesArray.add(new file(line,br.readLine(), br.readLine()));
271                 line = br.readLine();
272             }
```

This segment of code deals with how I used text files to save user entered assignments in between program launches. This code is executed on the “finish adding assignment” button press, meaning the user has indicated that he or she wishes to add the assignment with the given characteristics. The first step is to get the details the user has entered and save them into strings. This can be seen in lines 250 to 252, where I get the text from two JTextArea’s and get the selected text from a JComboBox. I then

use a bufferedWriter that is writing to the “assignments” text file with append set to true. This allows me to always be writing to the next line rather than overwriting previous data. I then write the three strings to the text file with a newLine in between each, before closing the bufferedWriter. After this is done, I clear the arrayList holding all of the files and repeat the process explained earlier, where I create new file objects from the text file and add them to the arrayList.

Displaying the assignments through mouseClicked:

```
481     public void mouseClicked(MouseEvent e) {
482         assignmentLBL.clear();
483         viewPanel.removeAll();
484         String line = null;
485 
486         for(int x = 0; x<6; x++){
487             for(int y = 0; y<7; y++){
488                 if(e.getSource() == gridArray[x][y] && numbering[x][y].getForeground() == Color.RED){
489                     for(int i = 0; i<dateArr.size()-2; i++){
490                         if(month + 1 == Integer.parseInt(dateArr.get(i+1)) && year == Integer.parseInt(dateArr.get(i+2))){
491                             if(Integer.parseInt(numbering[x][y].getText()) == Integer.parseInt(dateArr.get(i))){
492                                 for(int z = 0; z<filesArray.size(); z++){
493                                     if(filesArray.get(z).getDate().equals(numbering[x][y].getText() + "/" + (month+1) + "/" + year)){
494                                         assignmentLBL.add(new JLabel(filesArray.get(z).getDate(), JLabel.CENTER));
495                                         assignmentLBL.add(new JLabel(filesArray.get(z).getTitle(), JLabel.CENTER));
496                                         assignmentLBL.add(new JLabel(filesArray.get(z).getCategory(), JLabel.CENTER));
497                                     }
498                                 }
500                                 for(int d = 0; d<3; d++){
501                                     viewPanel.add(assignmentLBL.get(d));
502                                 }
                                viewPanel.updateUI();
                            }
```

This segment of code was used to display the assignment of any given day when that day was clicked on. It starts off by clearing the arrayList filled with the JLabels of the assignment being displayed, and removing everything from the panel “viewPanel” so that nothing stays from a previous click. It then loops through the “gridArray” array and “numbering” array. Then it checks for if a click was on a panel and that panels numbering was red, indicating that there is an assignment that day. It then loops through “dateArr”, an arrayList filled with all of the assignment dates split into the three basic segments: month, day and year. It then checks if the current month (“month + 1”) is equal to any of the months of an assignment, and does the same with year as seen on line 489. If this passes, it then checks if the day for that assignment matches the day that was clicked on. If this is true, then it loops through the arrayList of objects, “filesArray”. It then checks if any of the file objects dates are equal to the string “(the day number clicked on)/(month)/(year). If this is true, then it creates a new JLabel with the objects date, title and category and adds those JLabels to an arrayList. It then adds the JLabels onto the viewPanel for the user to see the assignment for any day the user clicks on.

Word Count: 907

Citations:

Docs.oracle.com. (2018). Calendar (Java Platform SE 7). [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html [Accessed 2 Mar. 2018].

Docs.oracle.com. (2018). Date (Java Platform SE 8). [online] Available at: https://docs.oracle.com/javase/8/docs/api/java/util/Date.html [Accessed 2 Mar. 2018].

Docs.oracle.com. (2018). MouseListener (Java Platform SE 7). [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html [Accessed 5 Mar. 2018].