IB Computer Science IA — Attendance System for the Marching Band

Jacob Samurin

February 8, 2023

Contents

1 Criterion A: Planning

1.1 Defining the Problem

The problem of Mr. Todd Fessler (my client) is that for our marching band class there is no good and efficient way of taking attendance. The way of taking attendance right now is that the "Drum Majors" who are the overall leaders in the marching band, go around and ask each row for their attendance, and it took a very long time to take attendance.

1.2 Rational for Proposed Solution

My solution will make it possible for the leaders of each row to take attendance then the Drum Majors won't have to go row by row. This will also skip the Drum Major step completely since my client will have direct accesses to the app. This will make it easier for everyone and faster, so we can have a longer rehearsal times.

1.3 Success Criterion

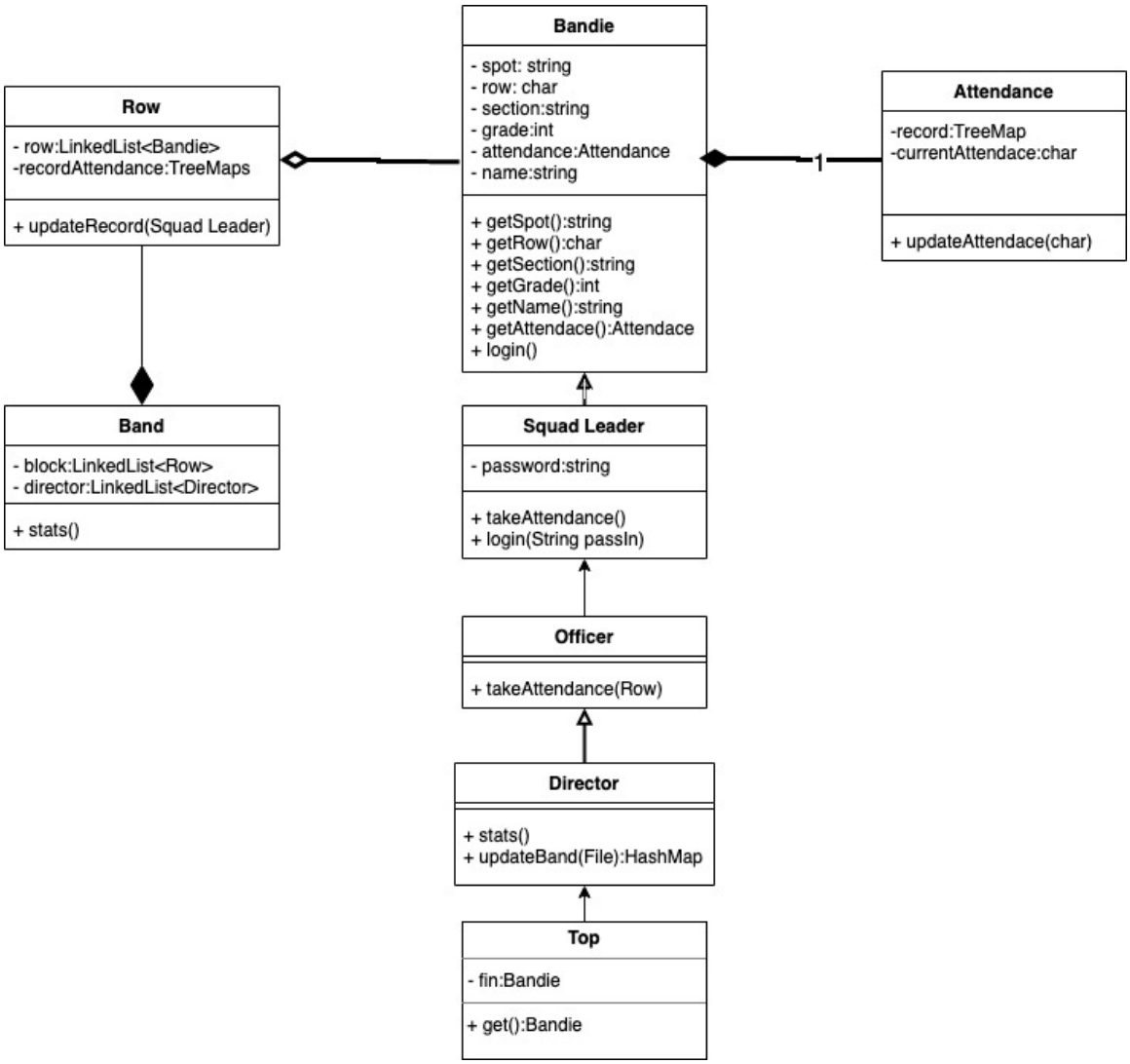
Different classes each level of leaders
A web app will be created add the different levels can edit different kinds of pages
Make it accessible from a phone
Make it accessible from an iPad
Make it accessible from a laptop/computer
Have the sever running the web app set up

2 Criterion B: Solution Overview

2.1 Sketch

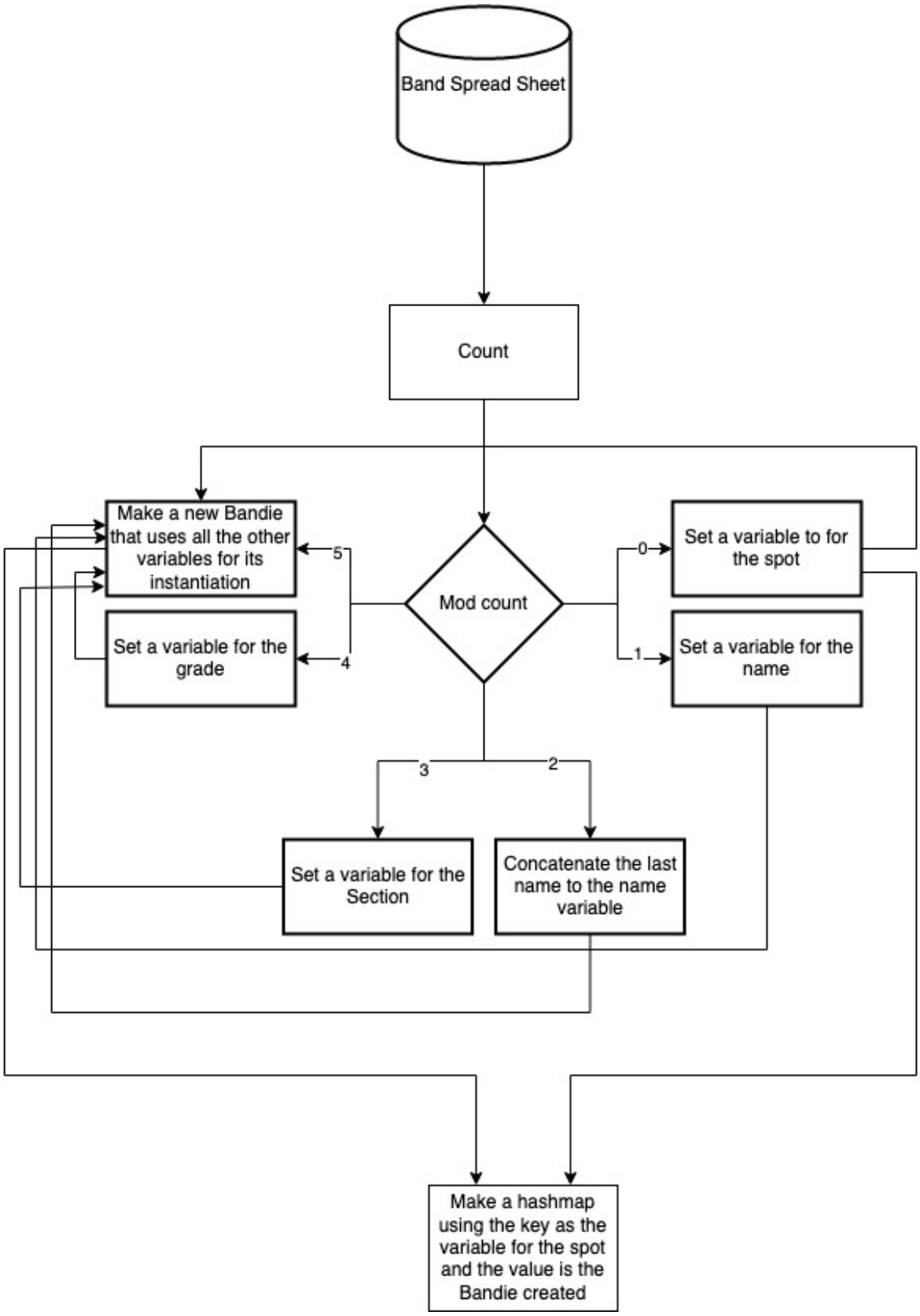
2.2 UML Diagram

This is the basic outline of the different classes and data types



2.3 Flowchart

This is the basic flow of the `updateBand` method


Band Spread Sheet
Count
Make a new Bandie that uses all the other variables for its instantiation
Set a variable for the grade
Mod count
5
4
0
1
Set a variable to for the spot
Set a variable for the name
3
2
Set a variable for the Section
Concatenate the last name to the name variable
Make a hashmap using the key as the variable for the spot and the value is the Bandie created

2.4 Pseudocode

For the pseudocode I will be writing out the `updateBand` method

```
item = read in a file for every section that stops in a comma
while there is a next item
if tem%6 equals 0
spot = item
if its k-row
row = first 2 letter
else
row = first letter
if temp%6 equals 1
name = item + " "
if temp%6 equals 2
name += item
if temp%6 equals 3
section = item
if temp%6 equals 4
grade = item
if temp%6 equals 5
new object t of type Top(spot, row, name, section, grade, item)
temp++
```

2.5 Development Plan

This is a plan for how to create the final product

- Create the `Bandie` class and the other levels of authorization all the way to Director/Admin
- Write the Row class with the hash table for the whole row's attendance and a linked list for the Bandies in the Row
- Create the Band class with the linked list for the directors and another linked list for the rows
- Create the Attendance class to store the past and current attendance for each `Bandie`
- Be able to upload a new CSV file to upload the structure of the band
- Store past attendance for each `Bandie` in a CSV file
- Make a web app that will ...
    1. Have a login screen where any level above `Bandie` has a password
    2. Have the different screens for Bandies, Squad Leaders, and Drum Majors.
    3. For Directors, they will also have a Statistics page with a tagline at the top(refer to the Sketch above)

2.6 Test Cases

Case	Outcome
Logging into a higher level account	• An incorrect password will send errors to the user. • A correct password will let the user through. • An invalid password(i.e. sending in the wrong data type) will send an error to the user
Uploading a spreed sheet file	• If the file is not a CSV file then will pass an error to the user "Improper File type" • If the file is a CSV but is not properly formatted it will pass an error to the user "Improper formatted" • If the file is a CSV and the first line is properly formatted it will upload the file

2.7 Record of Tasks

Task Number	Planned Action	Planned Outcome	Time Estimated (Minutes)	Target Completion Date	Criterion
1	Brainstorming with my client	An idea for the project	30	May 5, 2022	A
2	Interview	Get to know what my client wants	8.5	May 24, 2022	A
3	Write a draft proposal	proposed to teacher	60	May 26, 2022	A
4	Write a proposal	Re-proposed to teacher	75	Aug 22, 2022	A
5	make a Record of Tasks	Organize a timeline	15	Aug 23, 2022	B
6	Second Interview	Get a better Idea from my client	7	Sep 1, 2022	A
7	Create UML diagram	Get a UML diagram	90	Oct 20, 2022	B
8	Working on criterion B	Complete the outline of the criterion B	20	Oct 21, 2022	B
9	Third Interview	Get a better idea of the statistics section	6	Oct 26, 2022	A
10	Create drawing UI	To put my idea on to paper	45	Nov 5, 2022	B
11	Re-setup the document	Make it easier to work on the document	180	Nov 8, 2022	A/B
12	added a case to the Test Cases section	Work on a part needed for the finishing Criterion B	45	Nov 9, 2022	B
13	added a case to the test case's section	work on a part needed for the finishing criterion b	15	Nov 9, 2022	B
14	Fix the UML and finish Flowchart and Pseudocode	fixed UML and finished Flowchart and Pseudocode	90	Dec 19, 2022	B
15	writing the hash table section	started work on the writing part of criterion C	30	Jan 8, 2023	C
16	Coding to complete criterion c	complete the coding for criterion c	240	Feb 4, 2023	C

3 Criterion C: Development

3.1 Program Structure

The main parts of my program are the `App.java`, `Bandie.class`, `Row.java`, and `Band.java`. The `App` class, this has my main method and the details for my basic Terminal User Interface(TUI). The `Bandie` class is used as a base for my inheritance for every member of the band. `Row.java` is using a LinkedList("LinkedList", 2021) of `Bandie` to simulate what row would look link in real life. The final class `Band` is working like a HashTable("HashTable", 2021) because it contains a LinkedList("LinkedList", 2021) of `Row`. This make `Band.java` act like a band block would even have a dynamic amount of people on each row by using the `Row` object.

```
.
├── src
│   └── main
│       └── java
│           └── com
│               └── jzdoot
│                   └── IA
│                       ├── App.java
│                       ├── Attendance.java
│                       ├── Band.java
│                       ├── Bandie.java
│                       ├── Director.java
│                       ├── KRow.java
│                       ├── Officer.java
│                       ├── Row.java
│                       ├── SquadLeader.java
│                       └── Top.java
└── config
    ├── block.csv
    └── rows.txt
```

3.2 Techniques Used

- Inheritance
- Polymorphism
- Encapsulation
- File Reading and Writing
- HashTable("HashTable", 2021)
- LinkedList("LinkedList", 2021)
- HashMaps("HashMap", 2021)
- Singleton Class

3.2.1 Inheritance

I'm using inheritance as a way to easily have the same base for all my users. Each user (except Directors) are inheriting aspects from `Bandie` and other objects. The major point of using this was to make is easy to instantiate any of the users in my program. Here is an example from my `Director` class.

```
1  // Director.java
2        Bandie current = new Bandie(); // This is setting up a default
3        Director direct = new Director();
4        LinkedList<Row> newBlc = new LinkedList<Row>();
5        boolean dir = false;
6        Scanner s = new Scanner(f);
7        String name = "";
8        s.useDelimiter(",");
9        int count=0,realCount=0;
10       if(topLine)
11          s.nextLine();
12       while(s.hasNext()){
13          String item = s.next();
14          switch(count%6){
15             case 0:
16                switch(item){ //Here is were we actualy where we can make this a any other
                      type of user
17                   case "director":
18                      dir=true;
19                      break;
20                   case "officer":
21                      current = new Officer(); //This will change current into an Officer
22                      break;
23                   case "leader":
24                      current = new SquadLeader();//This will change current into an Squad
                           Leader
25                      break;
26                   case "bandie":
27                      current= new Bandie();//This will keep current the same as the default
                           of Bandie
28                      break;
29                }
30                break;
```

On line 2 we set a default and the left-hand side of the instantiate, down at line 16 we have a switch case which well make the `current` on line 2 into any other type of user from lines 21 to 27.

3.2.2 Polymorphism

Polymorphism is when you have different classes that know what they are and what they can do. I'm using this when I'm calling the `takeAttendance` and the `takeRowAttendance` between when `Director`, `Officer`, or `SquadLeader` call it.

```
1  //SquadLeader.java
2  public void takeRowAttendance(char[] a){
3     Band.getRow(super.getRow()).updateRecord(this);
4     for (int i = 0; i < Band.getRow(super.getRow()).size(); i++) {
5        takeAttendance((""+super.getRow())+(i+1), a[i]);
6     }
7  }
8  public void takeAttendance(String spott, char a
9     Band.getBandie(spott).setAttendance(a);
10 }
11
```

```
12 //Office.java
13 public void takeRowAttendance(char row, char[] a){
14    Band.getRow(row).updateRecord(this);
15    for (int i = 0; i < Band.getRow(row).size(); i++) {
16       super.takeAttendance((""+row)+(i+1), a[i]);
17    }
18 }
19
20 //Director.java
21 public void takeRowAttendance(char row, char[] a){
22    Band.getRow(row).updateRecord(this);
23    for (int i = 0; i < Band.getRow(row).size(); i++) {
24       takeAttendance((""+row)+(i+1), a[i]);
25    }
26 }
27 public void takeAttendance(String spott, char a){
28    Band.getBandie(spott).setAttendance(a);
29 }
```

This example is showing the difference between the `takeRowAttendance` method in `SquadLeader` on line 1 and `Offcer/Director` on line 12 and 20. The `takeRowAttendance` on line 2 us more so used when the row is predefined like in `SquadLeader.java` where they can only change their row. But the `takeRowAttendance` on line 13 and 21 can change any row that is asked.

3.2.3 Encapsulation

Encapsulation is the practice of keeping everything contained in one object and using accessor methods to get values. I use this in the `Row` class because I don't have access to the LinkedList("LinkedList", 2021) it's self, but I can get the size trough the `size()` method. Here is that example

```
1  //Attendance.java
2  package com.jzdoot.IA;
3  import java.util.Date;
4  import java.util.HashMap;
5
6  public class Attendance{
7     private HashMap<Date,Character> record;
8     //NOTE the currentAttendace is now in the bandie class
9
10    public void updateAttendance(char attendance){
11       record.put(new Date(), attendance);
12    }
13 }
```

In this example the HashMap("HashMap", 2021) on line 7 is private and the method `updateAttendance` on line 10 is used to set the HashMap("HashMap", 2021) instead of changing the HashMap("HashMap", 2021) directly.

3.2.4 File Reading and Writing

File reading and writing is a way for a program to read in a file and output out to a file. I'm using it in one main way, in the `Director` class there is a method called `updateBand` it is used to update the block structure easily by using a `.csv` file because it's stored in plain text but can be edited by any spreadsheet software. I use this to store the band and to reset it when the program gets closed.

To store this copy this file to a new file in a saved directory(Thilo, 2011).

3.2.5 HashTable

HashTables("HashTable", 2021) are arrays of LinkedLists("LinkedList", 2021). I'm using this differently I made a LinkedList("LinkedList", 2021) of `Row`, but `Row` is a LinkedList("LinkedList", 2021) of `Bandie` this works in the same general way as a HashTable("HashTable", 2021). The way I'm using this "HashTable" is to store the structure of the band and each row will be able to be dynamic and so will the number of rows which is why I didn't use just a normal HashTable("HashTable", 2021).

3.2.6 LinkedList

A LinkedList("LinkedList", 2021) is a list where each item is a linked node which have two values. One of the values is the value you give the item. The other is a link to the next item. I use this throughout my project in almost every class. Here are some examples:

```
1  //Band.java
2  private static LinkedList<Row> block;
3  public static void init() throws FileNotFoundException{
4     if(instance != null)
5        System.out.println("Error: instance already exists");
6     else{
7        instance = new Band();
8        int kCount = 0;
9        Scanner s = new Scanner(new File("config/rows.txt"));
10       while(s.hasNext()){
11          String item = s.nextLine();
12          char r = item.charAt(0);
13          //TODO add exception for krow
14          if(r != 'k')
15             addRow(new Row(r));
16          else{
17             kCount++;
18          }
19       }
20       LinkedList<LinkedList<Bandie>> kr = new LinkedList<LinkedList<Bandie>>();
21       for(int i=0;i<=kCount;i++)
22          kr.add(new LinkedList<Bandie>());
23       addRow(new KRow(kr));
24    }
25 }
26 public static void addRow(Row r){
27       if(block.size() > 0)
28          for(int i=0; i<block.size();i+=0){
29             if(block.get(i).compareTo(r)<0)
30                i++;
31             else if(block.get(i).compareTo(r)==0){
32                System.out.println("already exists");
33                break;
34             }else if(block.get(i).compareTo(r)>0){
35                block.add(i, r);
36                break;
37             }else
38                block.add(r);
```

```
39          }
40       else
41          block.add(r);
42  }
43  public static Row getRow(char row){
44     for(Row arr : block){
45        if(arr.getLetter()==row)
46           return arr;
47     }
48     return null;
49  }
```

Here we use a LinkedList("LinkedList", 2021) to store the band block, and we write the left-hand side of the instantiate of the LinkedList on line 2. I put in two other methods in that use this LinkedList `init` and `addRow`. This benefits from using a LinkedList instead of a ArrayList("ArrayList", 2021) because the LinkedLists are better at iterate and getting items out. ArrayList are better for just storing items.

3.2.7 HashMaps

HashMaps("HashMap", 2021) are a set of "keys" and "values". Keys can't repeat but values can. you can use the keys to accesses the values. I use HashMaps in the `Attendance` and `Row` classes. Here is an example:

```
1   //Attendance.java
2   private HashMap<Date,Character> record;
3   public Attendance(){
4      record=new HashMap<Date,Character>();
5   }
6   public void updateAttendance(char attendance){
7      record.put(new Date(), attendance);
8   }
9   //Row.java
10  private Map<Date, Top> recordAttendance;
11  public void updateRecord(Top sl){
12     recordAttendance.put(new Date(), sl);
13  }
```

We start the instatation on line 2 and call it `record`. We update `record` on line 7. We use the HashMap on line 10 the same way it is just the best way to store this kind of data.

3.2.8 Singleton Class

A Singleton Class is a style of writing a class that makes sure there is only one instance of the object. I use this style of writing a class in my `Band` class. The most important part is the `init()` method I wrote. This method will start by making sure there is an instance of `Band` and creating all the rows for the block. The usefulness of the Singleton Class structure here is that I can call that same `Band` instance from anywhere in the program. This makes it much easier to modify this object in all scenarios.

4 Criterion E: Evaluation

5 Appendix A: Interview

Interview conducted in person Interview 1 initial information gathering

Student - Basically my plan right now for the for the program is there's going to be different tiers. So, there's going to be a squad leader tier, a drum major tier, and then a director/admin tier. Squad leaders will be able to change attendance within the row, for Drum Major will be able to change attendance for the entire band and then for directors they can change the attendance for the entire band and on top of that there's some statistics like what rows haven't taken attendance yet, who turned into attendance for which row, and then maybe some row of the week stuff like most improved week to week, move most improved within the week, or best overall within the week or for the entire year. I'll be able to collect all that kind of data just from attendance. Client – That's awesome. So would it be like you're thinking, like you said squad leaders have control over their rows. Student - and then drum majors the entire band and then you guys can take this for the entire band and then you guys can do attendance for the entire band and then on top of that you can look at statistics and stuff Client - Is there any way once like for example of this squad leaders have submitted their attendance for the day is there a way for them to update it? Student - I mean I could definitely implement that. Client - or like a way for them to update it until the end of class. Somebody shows up that's the issue I think yeah if somebody shows up late on excused then they're marked absent for the whole day, but we need to know if they were late that day or if they were asking because that changes Student –I can probably set up a window of time like every morning from like 7:30 to 8:30 maybe even earlier I can set it to like 7 like to whenever time the period ends nowadays. Talking about some basic structure

6 Appendix B: References

References

Arraylist (17th ed.) [Computer software manual]. (2021). Oracle. Retrieved from https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

Hashmap (17th ed.) [Computer software manual]. (2021). Oracle. Retrieved from https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/HashMap.html

Hashtable (17th ed.) [Computer software manual]. (2021). Oracle. Retrieved from https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Hashtable.html

Linkedlist (17th ed.) [Computer software manual]. (2021). Oracle. Retrieved from https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/LinkedList.html

Thilo. (2011, January 10). *How do i move a file from one location to another in java?* Retrieved from https://stackoverflow.com/a/4645271