Assignment 1

Sonica Gupta (guptaso)

April 8, 2018

Contents

1 Program Description

1.1 Main Program's Task

After running the Calendar Main, the current dates events are displayed. So, since today is 4/5/2018, the Calendar tells me that I have a Birthday Party today at 3:30pm and my class to 2:30pm is resechduled. It then tells me tomorrow's (4/6/2018) tasks which is visiting my parents. Then an appointment at 5/-1/2018 is not valid, probably because -1 is not a valid date. Then I get a recap of todays events, so not including tomorrow's events. So, it displays again infor about the resechduled class and the Birthday party.

1.2 Each Class's Task

There are 7 classes utilized in this program.

1. **CalendarMain**
   Description: CalendarMain is the main class of the program and is

what handles the program's main task by utilizing the other classes. For instance, when it creates an appointment, it uses the Appt class to set the minute, month, year, title, and description. The exact data (minute, hour, day, etc.) is specified in this class then the other classes like DataHandler would be used to read and write this information to the calendar.

Source Line of Code: (using the NCSS given by JavaNCSS): 77

Number of Methods: This class has one public method and no private methods.

2. **Appt**
   Description: Appt holds one appointment, so this class is instantiated in Calendar main whenever an appointment needs the be created. Then when an instance of this class is created, the appointment's information is also filled in like the start hour, minute, day, month, and year, alone with title, description, and email address. This class also checks to make sure that the date and times aren't out range, for instance the minutes can't be less than 0 or greater than 59 as well as the month can't be less than 1 or greater than 12.

   Source Line of Code: (using the NCSS given by JavaNCSS): 146

   Number of Methods: This class has 36 total methods, 30 which are public and 6 which are private.

3. **CalendarUtil**
   Description: CalenderUtil provides methods that could be utilized by the Calendar. So, the Calendar needs to know how many days in a month, this class provides an array with this information. However, February could vary based on whether it is leap year or not. This class checks to see if based on the year, if it is leap year or not.

   Source Line of Code: (using the NCSS given by JavaNCSS): 18

   Number of Methods: There are 2 total methods of this class, both of which are public.

4. **XmlParserErrorHandler**
   Description: XmlParserErrorHandler deals with the errors that might occur with the XML parser and is called from the DataHandler class because that is where the reading takes place. Along with handling the

errors that might occur with parsing, this class also displays a string describing the error that occured.

Source Line of Code: (using the NCSS given by JavaNCSS): 16

Number of Methods: There are 4 total methods of this class, 3 of which are public and 1 is private.

5. **CalDay**
   Description: CalDay stores all the apointments of a single day. For instance, in the main program's tasks, there were two appointments created for today, the Birthday party and the rescheduled class. This class would group those two appointments together because they occur on the same day. This is then used in the CalendarMain class to link the appointments together and then display it back to the user.

   Source Line of Code: (using the NCSS given by JavaNCSS): 97

   Number of Methods: There are 16 total methods of this class, 12 which are public and 4 which are private.

6. **DateOutOfRangeException**
   Description: This class is called when the dates are out of range or invalid. This class is called from DataHandler since that is when the data is processed.

   Source Line of Code: (using the NCSS given by JavaNCSS): 4

   Number of Methods: There are 2 total methods of this class, both of which are public.

7. **DataHandler**
   Description: Since there is data being stored into the Calendar, the class DataHandler deals with all data that being both read and written. So, since CalDay is used for linking a day's worth of information, the class is clalled within here to bind together the data. Also, if there are errors with the information, then the class DateOutOfRangeException is called to handle that. Similarly, if there is an error with the parsing then the class XmlParserErrorHandler will be called to handle that.

   Source Line of Code: (using the NCSS given by JavaNCSS): 249

   Number of Methods: There are 15 total methods of this class, 7 of which are public and 8 which are private.

1.3 Pre-Conditions and Post-Conditions

- **Appt Class**
  - **public Appt(...)**
    Description: There are two constructors for this class, one that is called when there is a start time on the day specified and one without the start time. But other than that, they both create an appointment given a day, month, year, title, descrition, and email address. The first also adds in a start hour and start minute.
    Pre-Condition: The startDay is between 1 and 31, startMonth is between 1 and 12, startYear is greater than 0, and title, description, email-address is valid. If startHour and startMinute is present for the first constructor: startHour is between 0 and 23 and startMinute is betweeen 0 and 59.
    Post-Condition: A new Appointment has then been created.
  - **isValid(...)**
    Description: This method checks if the appointment values are valid. So, the month is between 1 and 12, the hour is between 0 and 23, the startMinute is between 0 and 59, and the startYear is greater than 0. It then checks the days according to the month to see if it is correct, so like February can't have day 30 because there are never 30 days in February.
    Pre-Condition: There are values passed in for startMonth, startHour, startMinute, startYear, and startDay.
    Post-Condition: Returns true if all the values of the appointmnet are valid, sets false if the values are not vaild.
  - **setRecurrence(...)**
    Description: If an appoint recurrs, this method is called to set the days that it does reoccur with the correct information.
    Pre-Condition: Has already been determined that the event reoccurs since the setting happens within the method.
    Post-Condition: The days of reoccurance now contains the appointment.
  - **setRecurDays(...)**
    Description: This method is called by the setRecurrence method. This method checks the days that the event will occur on.

Pre-Condition: Called by setReurrence and so contains some value for recurDays.
Post-Condition: recurDays contains the days the appointment will re-occur on.

- **CalDay Class**
  - **public CalDay(...)**
    Description: There are two constructors for CalDay, a default and a non-default. If the default constructor is called then it an invalid object. But if the non-default is called, a new CalDay is created with appointments addend to it from the calendar with info: day, month, year, appointments.
    Pre-Condition: A valid GregorianCalendar with day, month, year. Months are valid for numbers between 0-11.
    Post-Condition: A CalDay is created and appointments can be added to it.
  - **addAppt(...)**
    Description: Above, a CalDay is created and is ready to add appointments to the linked list, this methods adds the appointments to this object. This also orders the appointments by the time they occur
    Pre-Condition: Appointments occuring on the same day as the CalDay object are passed in.
    Post-Condition: Appointments added to linkedlist of the CalDay object.
- **CalendarUtil Class**
  - **IsLeapYear(...)**
    Description: This function determines whether a given year is a leap year. If the year is a multiple of 400 then the year is determined to be a leap year, otherwise it is not.
    Pre-Condition: year greater than 0
    Post-Condition: returns true if year is leap year, otherwise false
- **DataHandler Class**
  - **getApptRange(...)**
    Description: This method is used to give the range of time between

two appointments. So, two calendar appointments are passed in, the first appointment and then the second appointment. If the appointments are passed in in reverse order, an error will be given. Also, an exception would be thrown if there is an error with the dates of the given appointments.
Pre-Condition: The appoinments are already placed on the calendar.
Post-Condition: Either an integer giving the value of days between the two appointments is displayed or an exception indicating that the appointments were passed in reverse order or that the appointment's dates were invalid will be displayed.

- **getApptOccurence(...)**
  Description: This method gets all the occurances of a particular appointment given the first and the last day.
  Pre-Condition: The appointment, first day, and last day of occurance are passed in. If first day is actually given last day, then the method returns an empty result.
  Post-Condition Returns a linked list of all the occurances within the range specified.

- **getNextApptOccurence(...)**
  Description: This method gets the next occurance of a given appointment. This method also returns null if the appointment does not occur again or there is some issue with the date.
  Pre-Condition: An appointment is passed in along with a calendar, they both need to be created and the appointment should contain some data.
  Post-Condition: Returns the next appointment if found, if no other recurrence then it returns null or if there is some issue with calculating the date, it returns null.

- **saveAppt(...)**
  Description: This method saves an appointment to the XML data tree. This method starts with creating a new empty element of the tree, and then fills out the the fields of the element with the information passed in with the appointment.
  Pre-Condition: Passes in an appointment with correct data, since it is going to be added to the XML data tree.
  Post-Condition: If this method executed correctly, returns true,

else returns false. Additionally, if returns true, then the appointment was added into the tree.

2 Bugs Description

1. **isValid(...)** from the Appt.java

   The isValid method checks to see if an appointment is valid. This method was supposed to only show one method is invalid and that is the one that is displaying the date as -1. However, for this method I switched the logic around. So, for lines 171, 173, 175, 177, and 181 of Appt.java, I changed the 'false' to 'true'. Also, for line 183, I changed the 'true' to 'false'. Then the result gave me 2 invalid appointments, one of which is supposed to be invalid. However, the other one that came out invalid: '4/8/2018 at 4:30pm, Visit, Visiting my parents!', which is a valid entry. This would be hard catch because the logic was flipped around which made the valid entry invalid. The other invalid entry was probably deemed invalid before reaching this method and therefore was not affected and switched to valid.

2. **getNextApptOccurrence(...)** from the DataHandler.java

   The getNextApptOccurence gets the date of the next occurance of a specified appointment. So, to change this, on line 339 I subtracted 1 from the value of the recurrence. Now, if the appointments were reoccur on a monthly basis, it would reoccur on a weekly basis and if it were to be yearly, it would become monthly. Moreover, if the events would reoccur on a weekly basis, as they are supposed to be here, nothing would happen as 0 wouldn't be a case and that is what happened to the 'Visit, Visiting my parents!', it ended up not reoccuring because it ended up not matching a case.

3. **representationApp()** from the Appt.java

   The representationApp method is used to display the string of the appointment back to the user. So, all the strings that I see displayed use this method. So, for this method on line 360, I swapped out the

greater than 11 to less than 11. This was used to determine whether the time needed an 'am' or a 'pm'. Since all the times were displayed with a 'pm' and I reversed the sign, all the times were displayed in terms of 'am' which is incorrect.

4. **toString()** from the Appt.java

   The toString method is used to display the string back to the user all concatenated together. Additionally, the method is used to display if the string is valid or not. On line 377, I changed if (!getValid()) to the opposite if (getValid()). This made it so that every string displayed was then not valid. Moreover, the string that was not valid ended up being displayed as valid.

5. **isValid()** from the CalDay.java

   The isValid method returns whether or not the CalDay object is valid. I changed line 97 to instead of returning valid, it would return not valid. So, then when the function was called towards the end, an error was generated causing the DataHandler method to throw exceptions. So, instead of displaying the 'number of appointments between 04/07/2018 and 04/08/2018' it displayed the expceptions about getting CalDay.getFullInformationApp. This would make sence because since the data was not validated, it was not able to store and therefore there is trouble when retrieving. This would then make it hard to locate the problem to be in the isValid method.