|  |  |
| --- | --- |
| Full Names: | Jan Kristoffer Cheng, Kurt Neil Aquino, Francesco Salceda, John Israel Caingles |
| Section: | S17 |

**Calendar App**

You found out that your friend has an implementation of a Swing-based calendar that you urgently need for a project that you need to complete. You immediately download the code and run the download code with your group mates and are excited to integrate the code because it’s exactly what you need.

Upon downloading the code and running it, you realize that the said calendar application contained a lot of flaws. Some of the flaws that were discovered were the following:

* There is no clear way of trying to add an event into the calendar
* When you try to add events, changing the month will make the events disappear
* It cannot be integrated because the program uses methods and classes which are native to it.

With the aforementioned limitations specified, perform the following tasks so that you will be able to implement the calendar application to your project (and future projects that will require such application):

1. Inspect and critique the code on how the calendar application was designed. Be able to identify areas of the design of the existing code that are poor and justify why it is so.

* Some attributes instantiated in class CalendarProgram are declared as public

/\*\*\*\* Day Components \*\*\*\*/

public int yearBound, monthBound, dayBound,

   yearToday, monthToday;

        /\*\*\*\* Swing Components \*\*\*\*/

        public JLabel monthLabel, yearLabel;

public JButton btnPrev, btnNext, btnAddSMS, btnAddFB;

     public JComboBox cmbYear;

public JFrame frmMain;

public Container pane;

public JScrollPane scrollCalendarTable;

public JPanel calendarPanel;

        /\*\*\*\* Calendar Table Components \*\*\*/

public JTable calendarTable;

        public DefaultTableModel modelCalendarTable;

* Calendar logic and User interface are merged into one class
  + - logic between the methods in the user interface and other classes (when modified) are tightly coupled
    - user interface holds methods out of its responsibility
    - the method refreshCalendar and the different ActionListeners(logic)  should be put in another class since CalendarProgram is only responsible over the GUI
    - adding new features to the program forces the programmer to make modifications
* the ActionListeners of the buttons are separated into different classes
  + - they should have been implemented within a single method in the GUI class
    - every time a button is added, a separate class is also created for its ActionListener
* Overall functionality of the program itself is limited
  + - does not preview event from file imports
    - cannot add new events into the calendar
    - is not able to preview event notifications with the use of user interfaces

1. Provide a class diagram to illustrate the design of the calendar app for your project so that it is able to:
   1. Create different types of events using different colors on the calendar.
   2. Send notifications on the date of the event. Notification libraries for Facebook and SMS are provided. You CANNOT modify these classes.
   3. Import existing calendar events on a CSV-file format:

Date of event (MM/DD/YYYY), event name, color

Date of event (MM/DD/YYYY), event name, color

…

Date of event (MM/DD/YYYY), event name, color

* 1. Import existing calendar events in a pipe-delimited-file format

Event name, Date of event (MM/DD/YYYY), color

Event name, Date of event (MM/DD/YYYY), color

…

Event name, Date of event (MM/DD/YYYY), color

1. Implement your design.
2. Justify how you have improved the design that better interfaces with other applications and why your design is reusable.

* Features:
  + - Can import events from files to the calendar
    - User may add events to the calendar by selecting the table cell of the desired day
    - User may view event notifications on different interfaces
    - More than one event may be displayed in a single cell with their corresponding color coding
* Design Patterns:
  + - A Strategy Pattern has been implemented for different file parsers. When a new file type has been added (other than the given csv and psv files) for the importing of events, our code will be open for extension.
    - An Adapter Pattern has been implemented for the different view APIs so that updating their notifications has become uniform. If another API is to be added, all we have to do is to implement the main Adapter class to the new API’s corresponding Concrete Adapter.
    - An Observer Pattern has been implemented for previewing event notifications on different user interfaces. With the monitor class (Grabber) implementing the Subject interface’s methods: register, unregister, and update, our program is able to add and remove multiple observers as well as update and sync their notification outputs.
* Reusability:
  + - It is easy to add new observers with existing API’s. Only one class is to be created to implement an interface and extend the current API.
    - Adding the functionality of reading from another file extension is also easier because of the implementation of the Strategy pattern. Only one class is to be added and its sole purpose it to read from a specific file and return the needed information.
    - If another specification is to be added, like notifying observers of newly added events, it is made much easier with the help of the Grabber that holds all the events and the observers to notify them. Since all observers only implement one interface, only one method will be called to update them of events.