|  |
| --- |
| Merrimac state high school |
| Calendar Application |
| Year 11 IPT Major Assignment 2014 |
|  |
| **Damon Murdoch & Gareth Watson** |
| **8/6/2014** |

|  |
| --- |
|  |

Table of Contents

[1.0 - Problem Definition 10](#_Toc398226660)

[**1.1** **- Application features** 10](#_Toc398226661)

[1.2 - System requirements 11](#_Toc398226662)

[2.0 - Proposed User Interface 11](#_Toc398226663)

[3.0 – System Storyboard 13](#_Toc398226664)

[4.0 Object Diagrams 14](#_Toc398226665)

[4.1 – Calendar.java 14](#_Toc398226666)

[4.1.1 – btnNext ActionListener 14](#_Toc398226667)

[4.1.2 – cmbYear ActionListener 14](#_Toc398226668)

[4.1.3 – btnNext ActionListener ActionPerformed 15](#_Toc398226669)

[4.1.4 – btnPrev ActionListener 15](#_Toc398226670)

[4.1.5 – btnPrev ActionListener ActionPerformed 16](#_Toc398226671)

[4.1.6 – cmbYear ActionListener ActionPerformed 17](#_Toc398226672)

[4.1.7 – getIntMonth(String month) 18](#_Toc398226673)

[4.1.8 – infoBob(String infoMessage, String title) 19](#_Toc398226674)

[4.1.9 – RefreshCalendar(int month, int year) 20](#_Toc398226675)

[4.1.10 – mouseEventDemo MouseListener 22](#_Toc398226676)

[4.1.11 – mouseEventDemo MouseListener MousePressed 22](#_Toc398226677)

[4.1.12 – Class Calendar 23](#_Toc398226678)

[4.1.13 – Class Calendar Constructor 24](#_Toc398226679)

[4.2 – Editor\_Schedule.java 26](#_Toc398226680)

[4.2.1 – Class Editor\_Schedule 26](#_Toc398226681)

[4.2.3 – btnSaveAction ActionListener 26](#_Toc398226682)

[4.2.2 – Class Editor\_Schedule Constructor 27](#_Toc398226683)

[4.2.4 – TypeAddActionListener ActionPerformed 30](#_Toc398226684)

[TypeActionListener ActionPerformed (Bottom Left) 31](#_Toc398226685)

[TypeActionListener ActionPerformed (Bottom Right) 32](#_Toc398226686)

[4.2.5 – Checkbox CheckChanged 33](#_Toc398226687)

[4.2.6 – Function CheckScheduleDataBase(String type, String name, String start, String end) 34](#_Toc398226688)

[Function CheckScheduleDataBase(String type, String name, String start, String end) (Bottom) 35](#_Toc398226689)

[4.2.7 - Function DragUpDown(int change) 35](#_Toc398226690)

[4.2.8 – Editor\_Schedule imports 36](#_Toc398226691)

[4.2.9 – itemStateChanged CheckChanged hasChanged 37](#_Toc398226692)

[itenStateChanged CheckChanged hasChanged (bottom) 38](#_Toc398226693)

[4.3 – Editor.java 39](#_Toc398226694)

[4.3.1 – Class Editor 39](#_Toc398226695)

[4.3.2 – Class Editor Constructor 40](#_Toc398226696)

[4.3.3 – btnBack ActionListener 41](#_Toc398226697)

[4.3.4 – btnBack ActionListener ActionPerformed 41](#_Toc398226698)

[4.3.5 – btnDelete ActionListener 42](#_Toc398226699)

[4.3.6 – btnDelete ActionListener ActionPerformed 42](#_Toc398226700)

[4.3.7 – btnExpenses ActionListener 43](#_Toc398226701)

[4.3.8 – btnExpenses ActionListener ActionPerformed 44](#_Toc398226702)

[4.3.9 – btnSchedule ActionListener 44](#_Toc398226703)

[4.3.10 – btnSchedule ActionListener ActionPerform 45](#_Toc398226704)

[4.3.11 – btnSchedule ActionListener ActionPeformed 45](#_Toc398226705)

[4.3.12 – btnView ActionListener 46](#_Toc398226706)

[4.3.13 - btnView ActionListener ActionPeformed 46](#_Toc398226707)

[4.3.14 – Function CheckIllegalCharacters(String check) 47](#_Toc398226708)

[4.3.15 – DoubleCheck(String check) 47](#_Toc398226709)

[4.3.16 – Function ItemPressed(int number) 48](#_Toc398226710)

[4.3.17 - UncheckIllegalCharacters 50](#_Toc398226711)

[4.3.18 – TimeCheck(String check) 51](#_Toc398226712)

[4.3.19 – Editor Imports 51](#_Toc398226713)

[4.4 – Editor\_ClientCreate 52](#_Toc398226714)

[4.4.1 – Class Editor\_ClientCreate 52](#_Toc398226715)

[4.4.2 – btnAddClient ActionListener 52](#_Toc398226716)

[4.4.3 – Class Editor\_ClientCreate Constructor 53](#_Toc398226717)

[4.4.4 – btnAddClient ActionListener PerformedAction 54](#_Toc398226718)

[btnAddClient ActionListener PerformedAction (right branch) 55](#_Toc398226719)

[btnAddClient ActionListener PerformedAction (left branches) 56](#_Toc398226720)

[4.4.5 –Function DragUpDown 57](#_Toc398226721)

[4.4.6 – Editor\_ClientCreate Imports 57](#_Toc398226722)

[4.5 – Editor\_TypeCreate.java 58](#_Toc398226723)

[4.5.1 – Class Editor\_TypeCreate 58](#_Toc398226724)

[4.5.2 – btnAddType ActionListener 58](#_Toc398226725)

[4.5.3 – Class Editor\_TypeCreate Constructor 59](#_Toc398226726)

[4.5.4 – Public AddType ActionListener ActionPerformed (left) 60](#_Toc398226727)

[Public AddType ActionListener ActionPerformed (right) 62](#_Toc398226728)

[4.5.5 - Editor\_TypeCreate Imports 63](#_Toc398226729)

[4.5.6 - Function DragUpDown(int change) 63](#_Toc398226730)

[4.6 – Menu.java 64](#_Toc398226731)

[4.6.1 – btnSwitch ActionListener 64](#_Toc398226732)

[4.6.2 – btnExit ActionListener 64](#_Toc398226733)

[4.6.3 – btnExit ActionPeformed 65](#_Toc398226734)

[4.6.4 – btnSwitch ActionPeformed(top) 66](#_Toc398226735)

[btnSwitch ActionPerformed (bottom left) 67](#_Toc398226736)

[btnSwitch ActionPerformed (bottom right) 68](#_Toc398226737)

[4.6.5 – Class Menu 69](#_Toc398226738)

[4.6.6 – Function ValueUpdate() 69](#_Toc398226739)

[4.6.7 – Menu Constructor 70](#_Toc398226740)

[4.6.8 – Menu Imports 71](#_Toc398226741)

[4.7 Notes.java 71](#_Toc398226742)

[4.7.1 – Class Notes 71](#_Toc398226743)

[4.7.2 - Function hoursToDouble(String time) 71](#_Toc398226744)

[4.7.3 – Function InsertData() (right) 72](#_Toc398226745)

[Function insertData() (Left) 73](#_Toc398226746)

[4.7.4 – Function Round(double value, int places) 74](#_Toc398226747)

[4.7.5 – Notes Imports 74](#_Toc398226748)

[4.7.6 – Class Notes Constructor 75](#_Toc398226749)

[4.8 – Task.java 76](#_Toc398226750)

[4.8.1 – Class Task 76](#_Toc398226751)

[4.8.2 – Main Program Loop (String[] args) 76](#_Toc398226752)

[4.8.3- Function ScreenRatio(int value, char axis) 77](#_Toc398226753)

[4.8.4 – Task Imports 77](#_Toc398226754)

[4.9 – View.java 78](#_Toc398226755)

[4.9.1 – Class view 78](#_Toc398226756)

[4.9.2 – Function checkbox(JComboBox Box, String test) 78](#_Toc398226757)

[4.9.3 – Class View Constructor 79](#_Toc398226758)

[4.9.4 - Function AddViewitem(ViewItem[] Array, ViewItem Element) 80](#_Toc398226759)

[4.9.5 – View Imports 80](#_Toc398226760)

[4.10 – ViewItem.java 81](#_Toc398226761)

[4.10.1 – Class ViewItem 81](#_Toc398226762)

[4.10.2 – Function AddThis() 81](#_Toc398226763)

[4.10.3 – Class ViewItem Constructor 82](#_Toc398226764)

[4.10.4 – Function setBounds() 82](#_Toc398226765)

[4.10.5 – ViewItem imports 83](#_Toc398226766)

[4.10.6 – ViewItem MouseListener 83](#_Toc398226767)

[4.10.7 – ViewItem MouseListener MousePressed 83](#_Toc398226768)

[5.0 – Application Objects 84](#_Toc398226769)

[5.1 – Calendar.java 84](#_Toc398226770)

[5.2 – Editor.java 85](#_Toc398226771)

[5.3 Editor\_ClientCreate.java 86](#_Toc398226772)

[5.4 – Editor\_Schedule.java 87](#_Toc398226773)

[5.5 – Editor\_TypeCreate.java 90](#_Toc398226774)

[5.6 – menu.java 91](#_Toc398226775)

[5.7 – notes.java (redundant) 92](#_Toc398226776)

[5.8 – task.java 92](#_Toc398226777)

[6.0 – Test Report 93](#_Toc398226778)

[6.1 – Schedule data input test report 93](#_Toc398226779)

[6.2 – Client data input test report 95](#_Toc398226780)

[6.3 – Schedule data input test report 96](#_Toc398226781)

[6.4 – Third-Party Reviews 97](#_Toc398226782)

[6.4.1 – Review 1: Steffi Tan 97](#_Toc398226783)

[6.4.2 – Review 2: Jerry Lo 97](#_Toc398226784)

[7.0 – Evaluation (Damon Murdoch) 97](#_Toc398226785)

[7.1 – Performance 97](#_Toc398226786)

[7.2 – Reliability 98](#_Toc398226787)

[Features Implemented: 98](#_Toc398226788)

[Missing Features: 98](#_Toc398226789)

[7.3 – User-Friendliness 99](#_Toc398226790)

[7.4 – Problems 101](#_Toc398226791)

[7.5 – Improvements 101](#_Toc398226792)

[8.0 – Evaluation (Gareth Watson) 103](#_Toc398226793)

[Appendixes 104](#_Toc398226794)

[9.0 – Screenshots 104](#_Toc398226795)

[9.1 – Main Menu 104](#_Toc398226796)

[9.2 – Schedule Editor Window 105](#_Toc398226797)

[9.3 - Schedule Editor Window (cont’d) 106](#_Toc398226798)

[9.4 - Client Edit Window 107](#_Toc398226799)

[Client Edit Window (cont’d) 108](#_Toc398226800)

[9.5 – Job Type Edit Window 109](#_Toc398226801)

[9.6 – View Menu: 110](#_Toc398226802)

[10.0 – Source Code 111](#_Toc398226803)

[10.1 – CalendarApp.package 111](#_Toc398226804)

[10.1.1 – Calendar.java 111](#_Toc398226805)

[10.1.3 – Editor.java 130](#_Toc398226806)

[10.1.4 – Editor\_ClientCreate.java 143](#_Toc398226807)

[10.1. 5 – Editor\_Schedule.java 149](#_Toc398226808)

[10.1.6 - Editor\_TypeCreate.java 161](#_Toc398226809)

[10.1.7 - Menu.java 167](#_Toc398226810)

[10.1.8 - Notes.java 173](#_Toc398226811)

[10.1.9 View.java 177](#_Toc398226812)

[10.1.10 - ViewItem.java 181](#_Toc398226813)

[10.2 – databasetut.package 184](#_Toc398226814)

[10.2.1 – DataBaseControl.java 184](#_Toc398226815)

[10.2.2- LocalClientitem.java 203](#_Toc398226816)

[10.2.3 – LocalScheduleItem.java 204](#_Toc398226817)

[10.2.4 – LocalTypeItem.java 205](#_Toc398226818)

# 

# 1.0 - Problem Definition

This application is being developed for the purpose of giving business owners and employees an effective method of storing and accessing data relating to booked appointments, job types, and their clients. It is expected the primary users of this application will be employees or business owners whose employment is based around taking appointments with clients, such as hairdressers, doctors, physicians, etc. These users would benefit most from this application because it allows them to easily arrange and organise their appointments, and also allows users to track the capital they’ve earned over a set time period based upon the appointments and expenses present in the database.

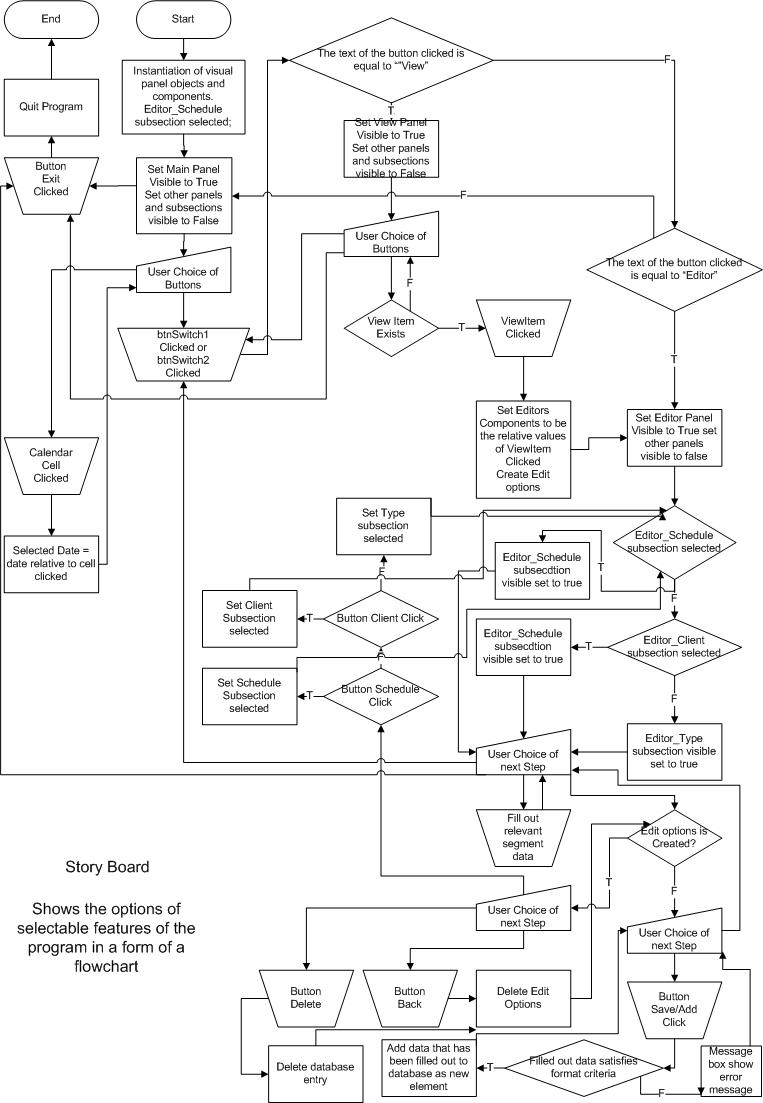
* 1. **- Application features**
* Utilisation of a calendar feature for selecting dates
* Highlighting the current date and weekends on the displayed calendar
* Multiple forms – Main form, view form and edit form.
* Ability for the user to store additional information on selected dates as notes
* Ability for the user to create client, schedule and job type data.
* Ability for the user to log expenses data, including a snapshot of the service receipt to be used as proof of purchase.
* Utilisation of an SQL database to store data, in which the entries can be later accessed for editing or deletion by the user of the application.
* Ability for the user to view and edit client, schedule and job type data after creation
* Migrating the application to android devices
* Ability for android versions to utilise the camera on the user’s device to take service receipt snapshots

## 1.2 - System requirements

This application is coded using Java 8.1, which means it is capable of running on Mac, Linux and Windows operating systems, as long as the user has the java client installed. The hardware requirements for usage of this application are a mouse or touchpad for form interaction along with a keyboard for entering data and a monitor for viewing the application’s Graphical User Interface (GUI). It is expected that the application will operate effectively on any system which has a windows, mac or linux based operating system and Java installed.

# 2.0 - Proposed User Interface

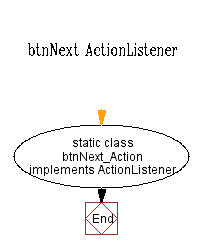
# 3.0 – System Storyboard



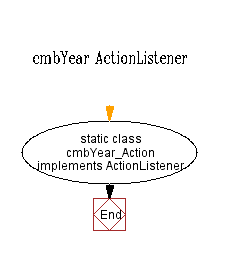
# 4.0 Object Diagrams

## 4.1 – Calendar.java

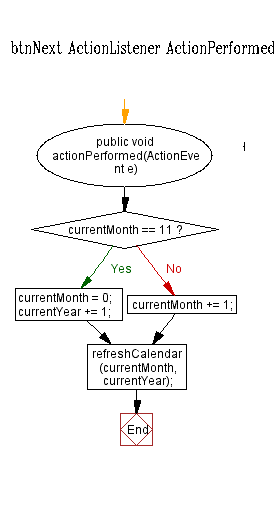
### 4.1.1 – btnNext ActionListener

****

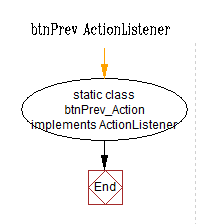
### 4.1.2 – cmbYear ActionListener

****

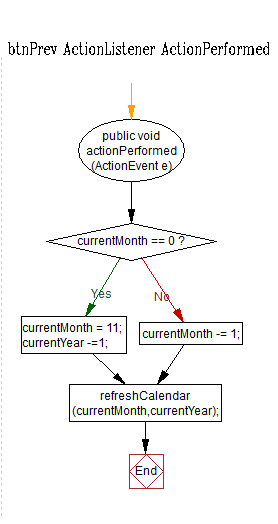
### 4.1.3 – btnNext ActionListener ActionPerformed

****

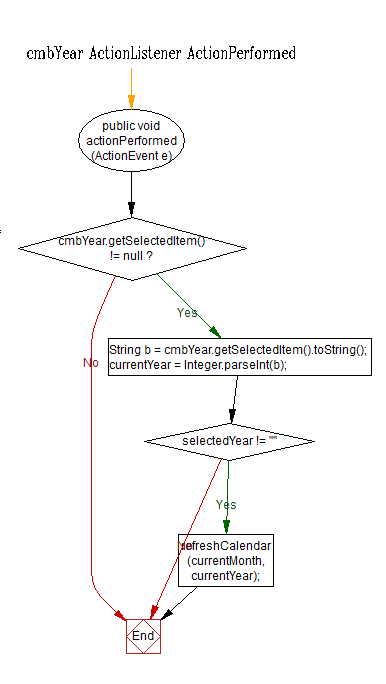
### 4.1.4 – btnPrev ActionListener

****

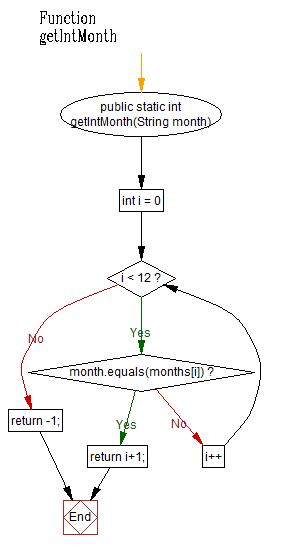
### 4.1.5 – btnPrev ActionListener ActionPerformed

****

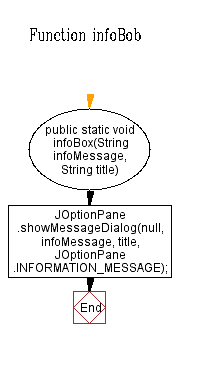
### 4.1.6 – cmbYear ActionListener ActionPerformed

****

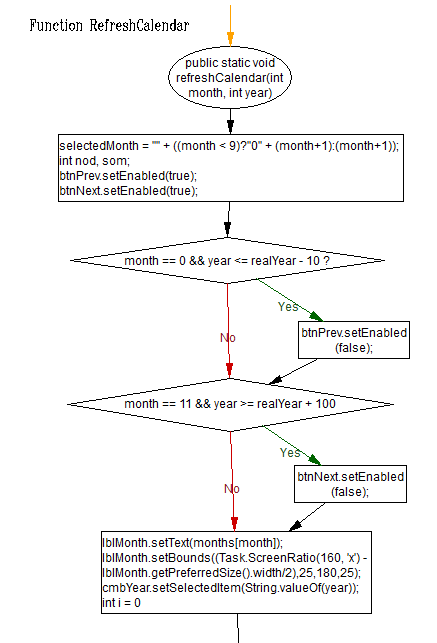
### 4.1.7 – getIntMonth(String month)

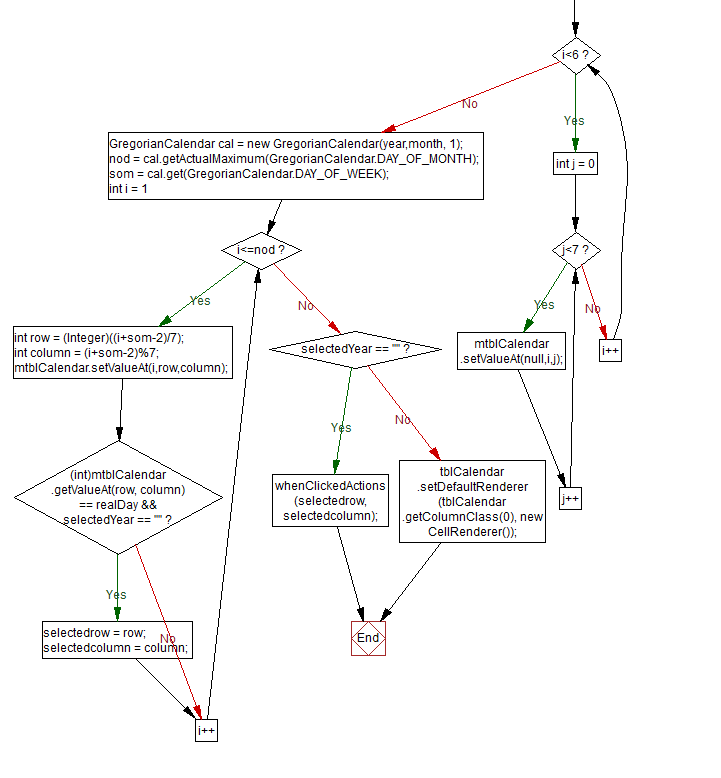
****

### 4.1.8 – infoBob(String infoMessage, String title)

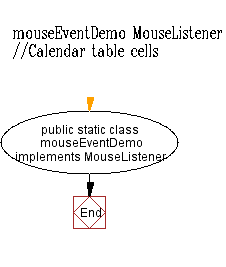
****

### 4.1.9 – RefreshCalendar(int month, int year)

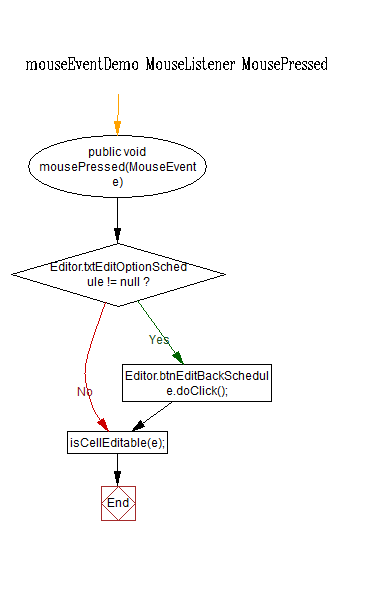
****

****

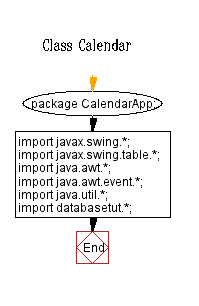
### 4.1.10 – mouseEventDemo MouseListener

****

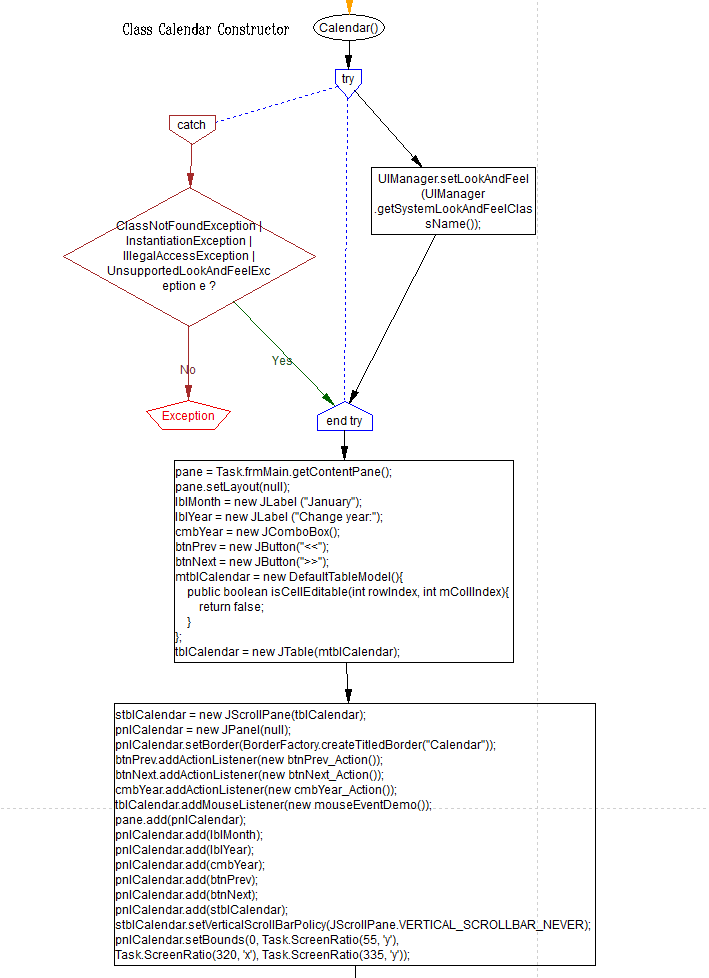
### 4.1.11 – mouseEventDemo MouseListener MousePressed

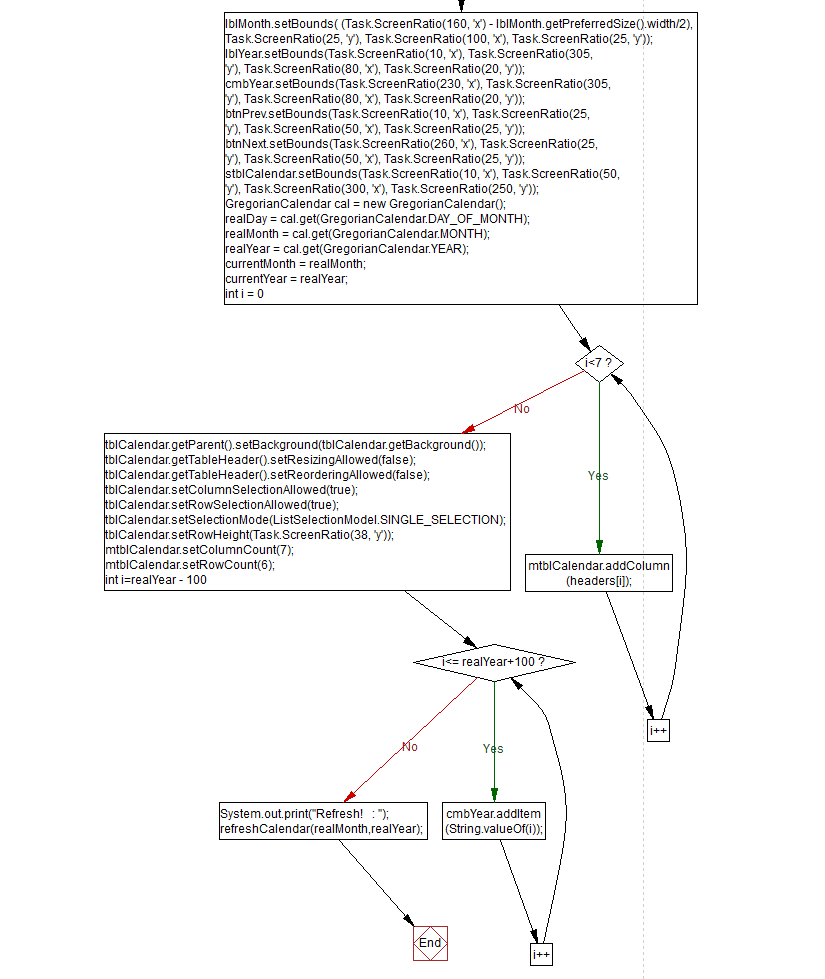
****

### 4.1.12 – Class Calendar

****

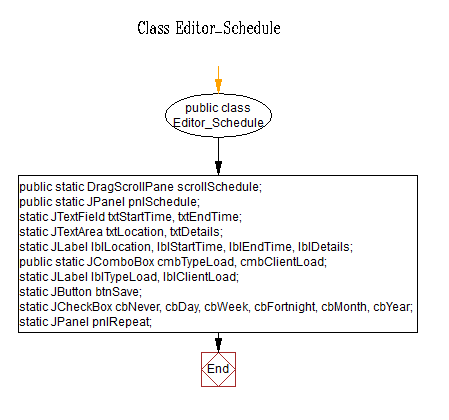
### 4.1.13 – Class Calendar Constructor

****

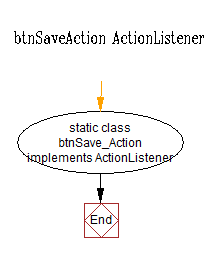
****

## 4.2 – Editor\_Schedule.java

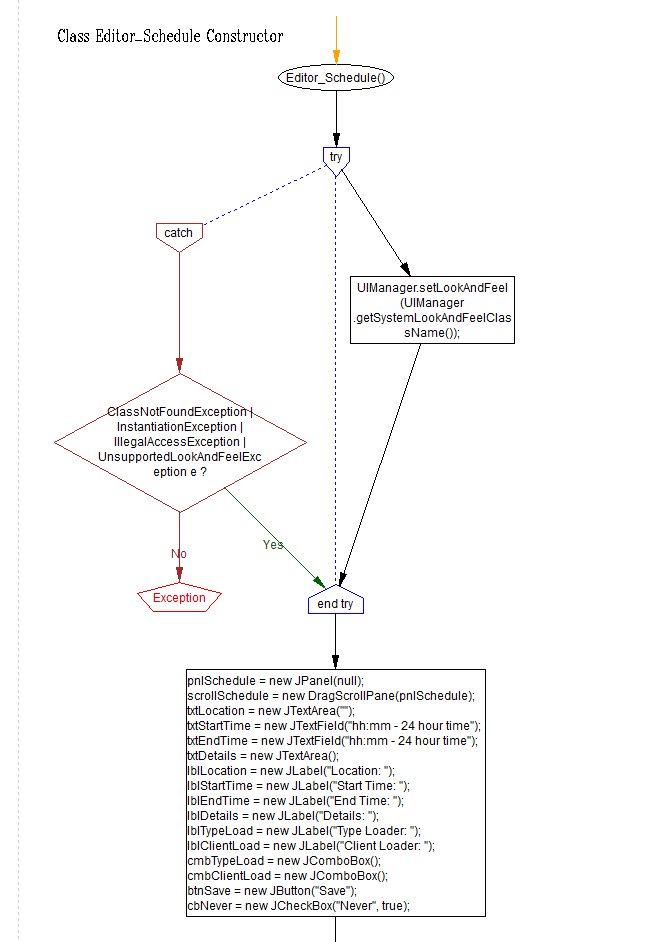
### 4.2.1 – Class Editor\_Schedule

****

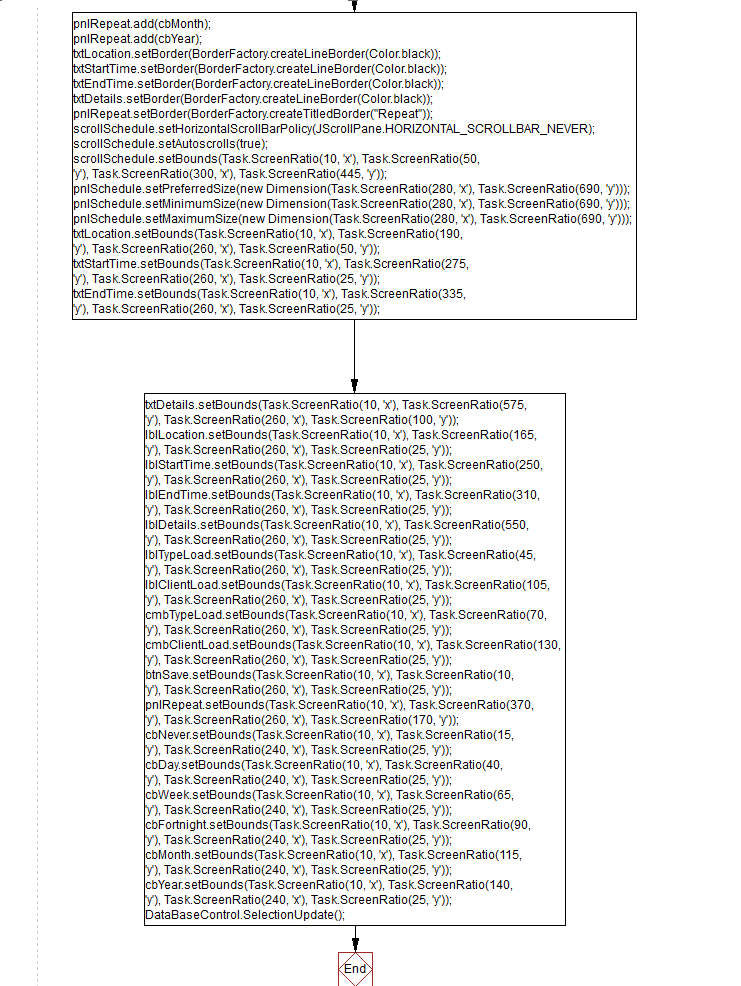
### 4.2.3 – btnSaveAction ActionListener

****

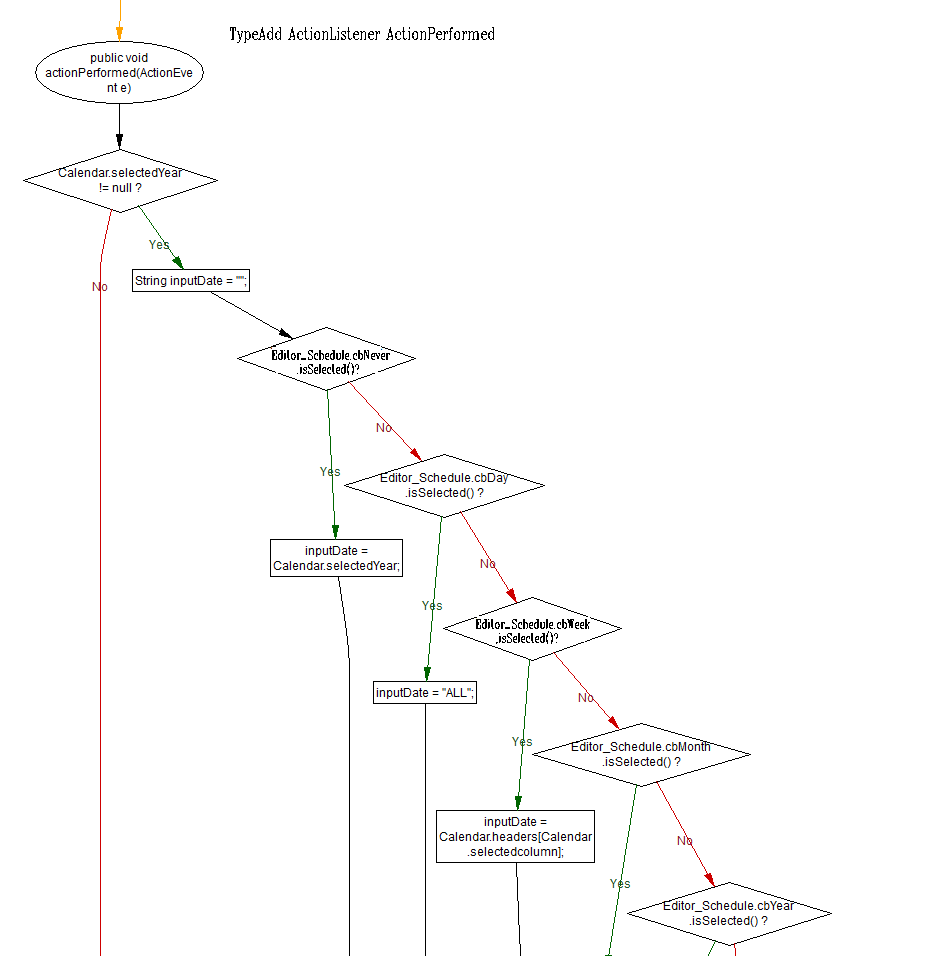
### 4.2.2 – Class Editor\_Schedule Constructor

****

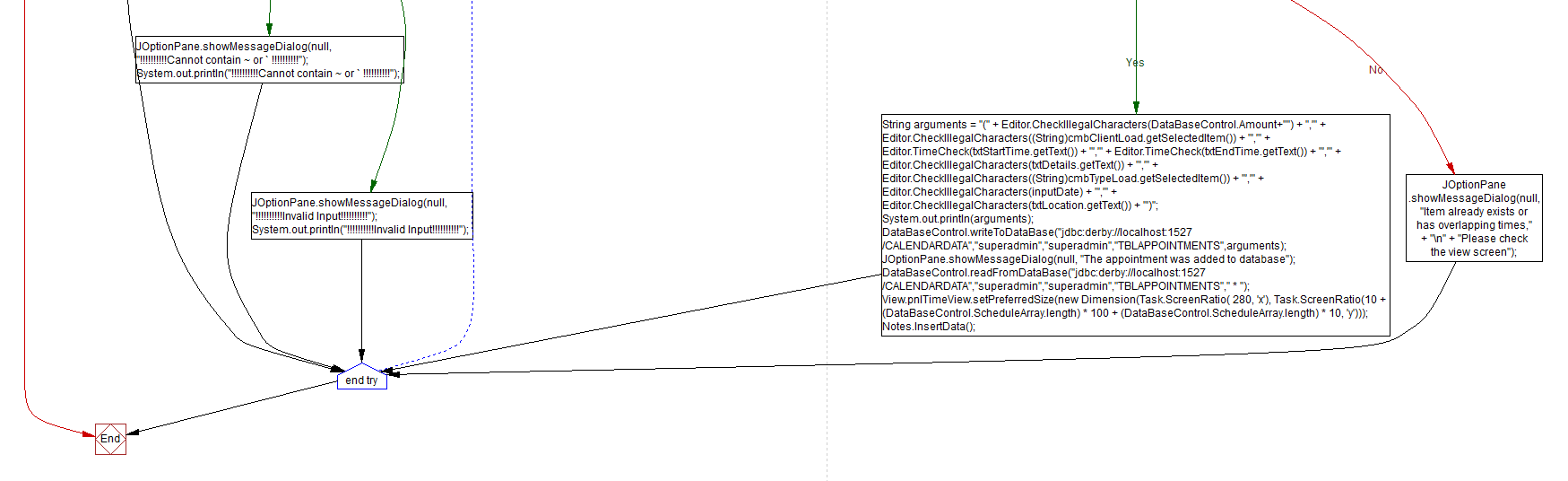
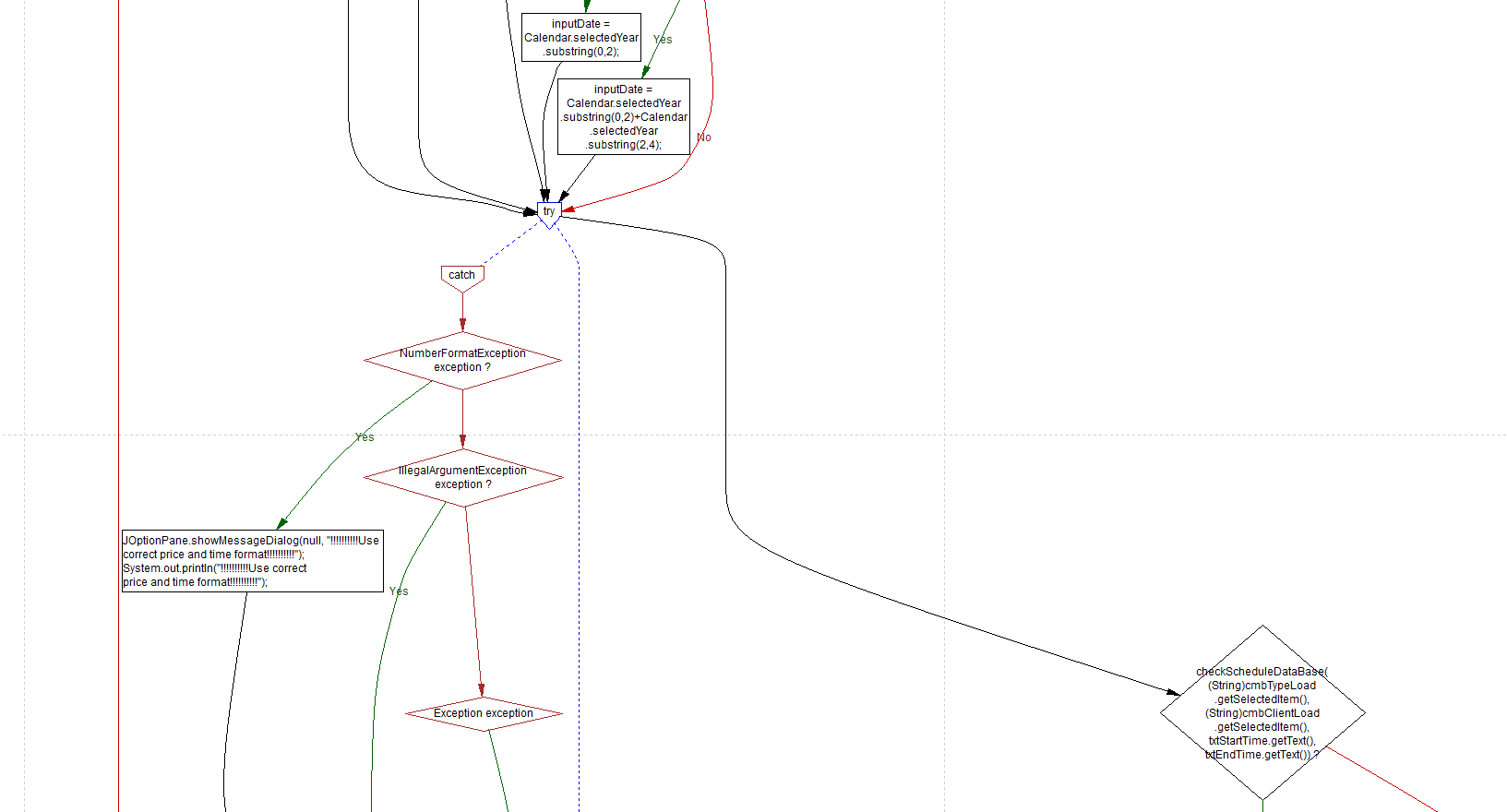
****

****

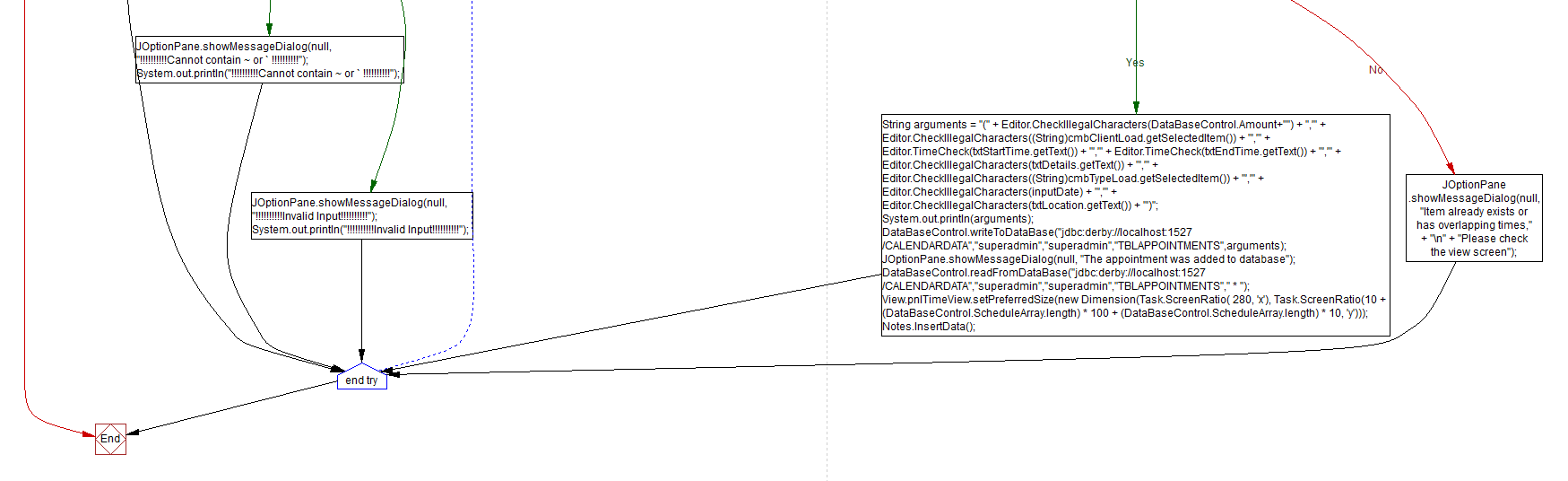
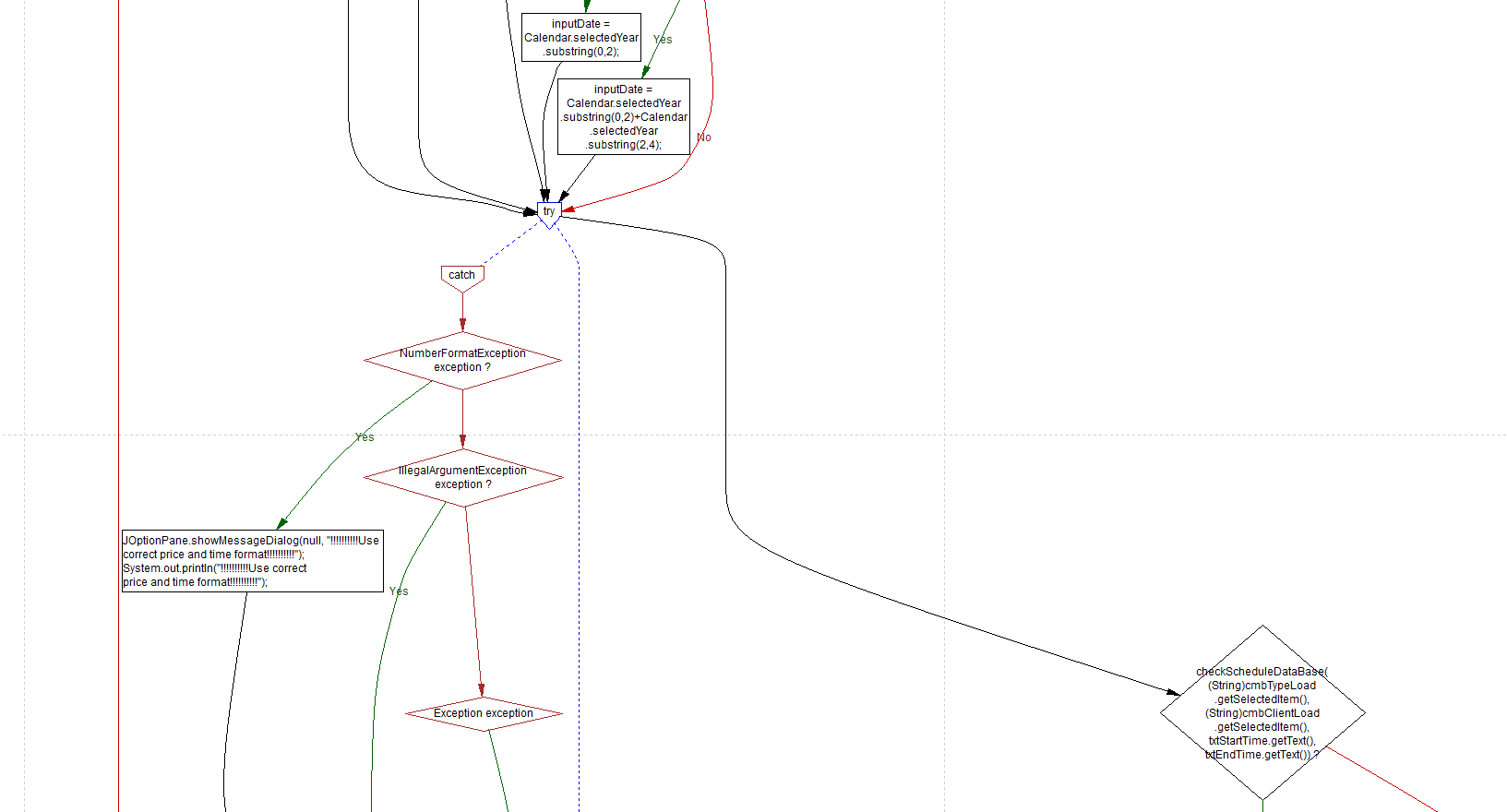
### 4.2.4 – TypeAddActionListener ActionPerformed

****

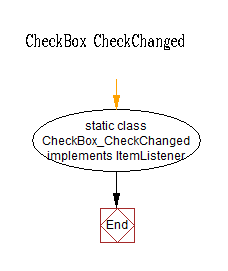
### TypeActionListener ActionPerformed (Bottom Left)

****

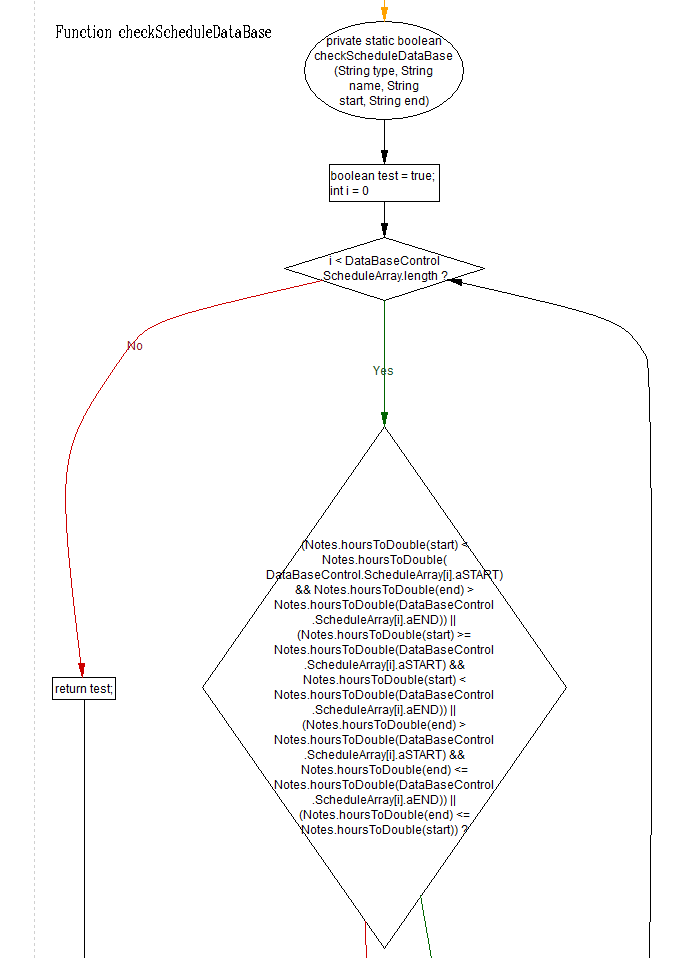
### TypeActionListener ActionPerformed (Bottom Right)

****

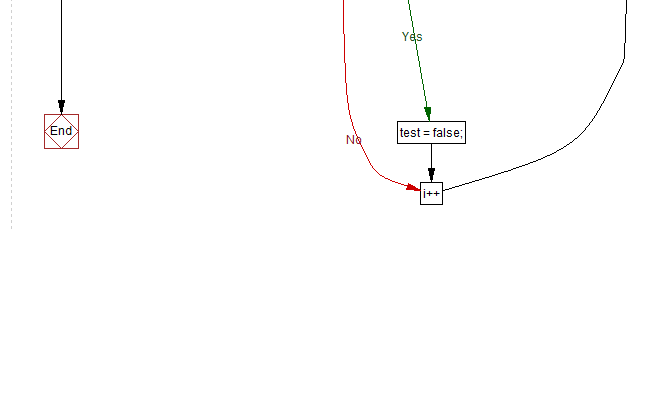
### 4.2.5 – Checkbox CheckChanged

****

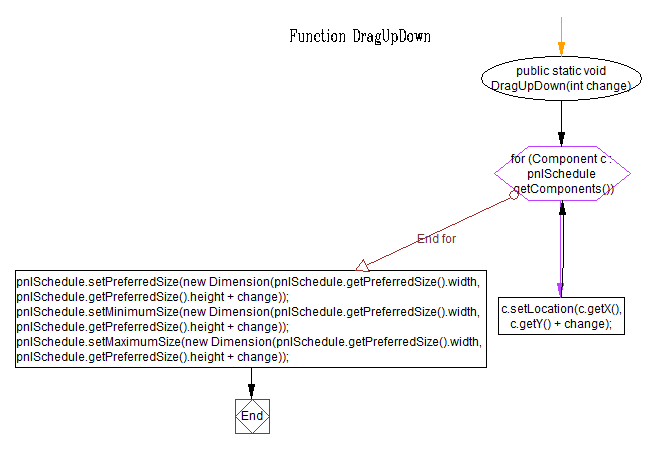
### 4.2.6 – Function CheckScheduleDataBase(String type, String name, String start, String end)

****

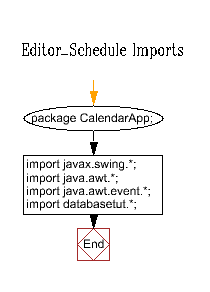
### Function CheckScheduleDataBase(String type, String name, String start, String end) (Bottom)

****

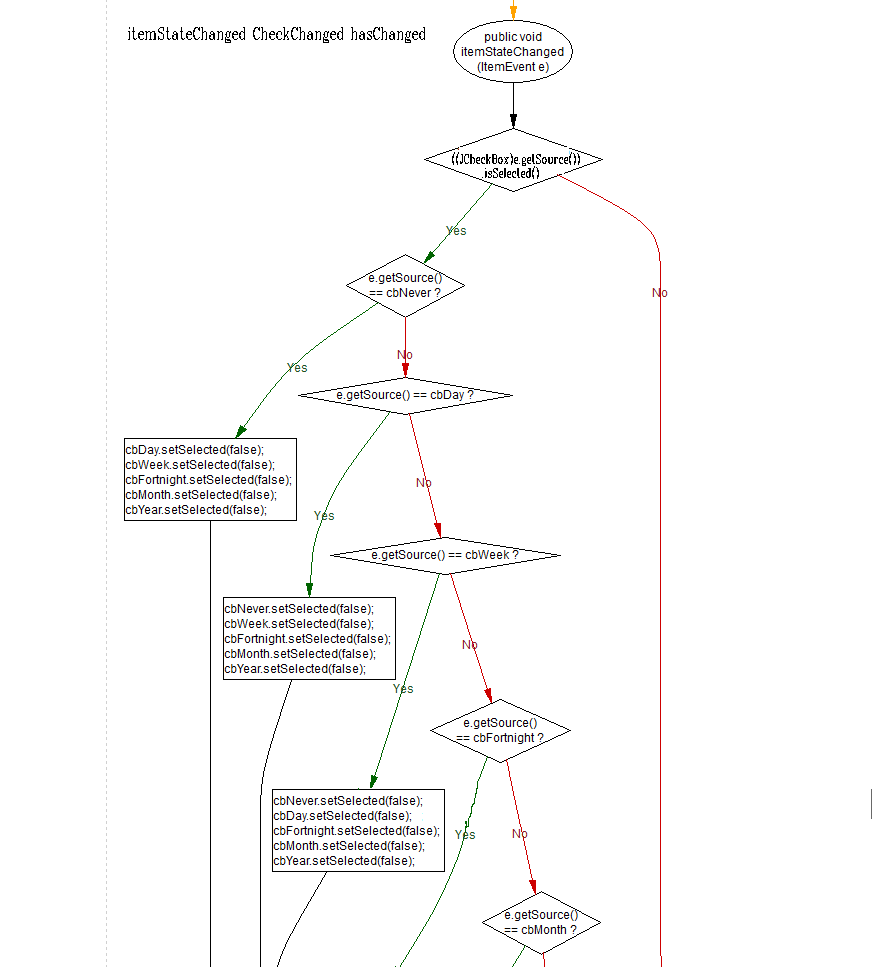
### 4.2.7 - Function DragUpDown(int change)

****

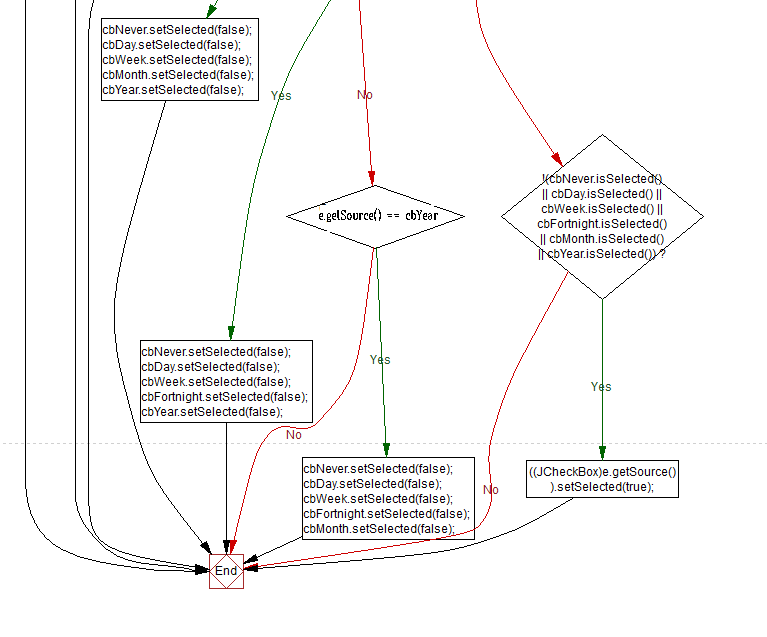
### 4.2.8 – Editor\_Schedule imports

****

### 4.2.9 – itemStateChanged CheckChanged hasChanged

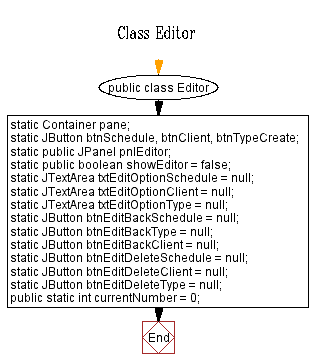
****

### itenStateChanged CheckChanged hasChanged (bottom)

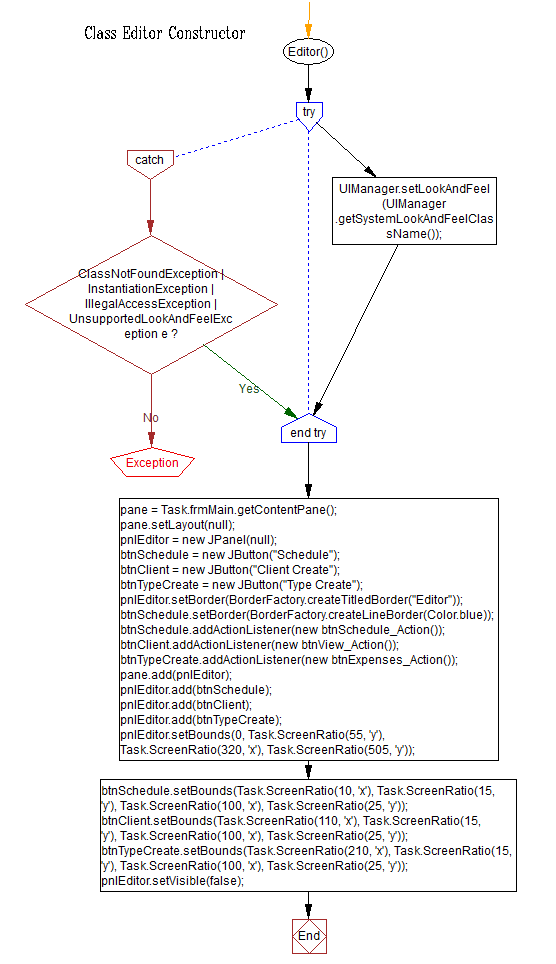
****

## 4.3 – Editor.java

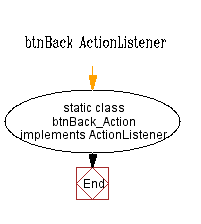
### 4.3.1 – Class Editor

****

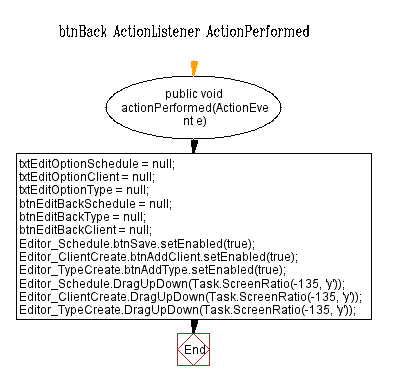
### 4.3.2 – Class Editor ConstructorF:\CalendarApp Final\Flow Charts\Editor\ClassEditorConstructor3.png

****

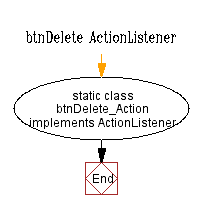
### 4.3.3 – btnBack ActionListener

****

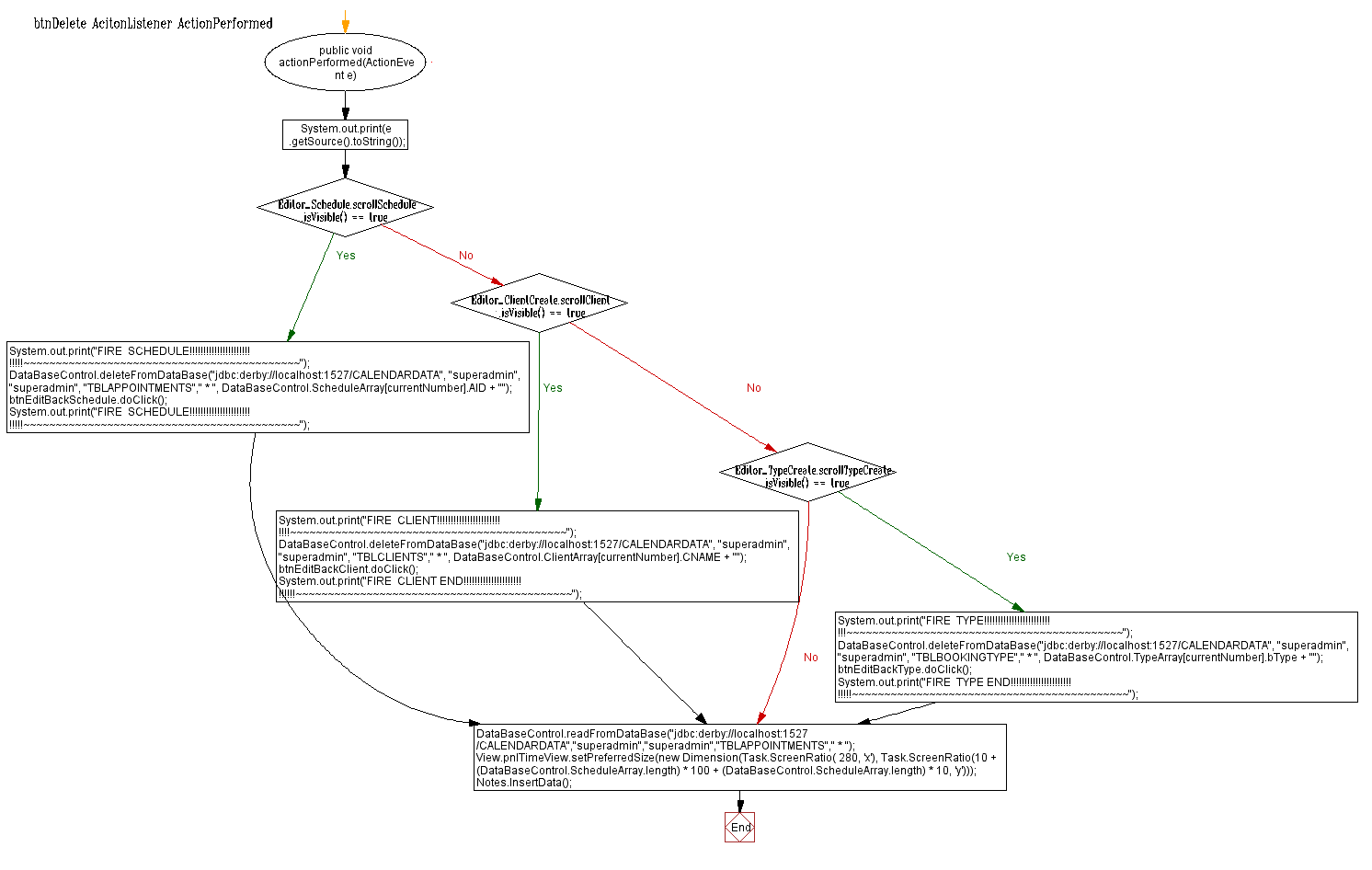
### 4.3.4 – btnBack ActionListener ActionPerformed

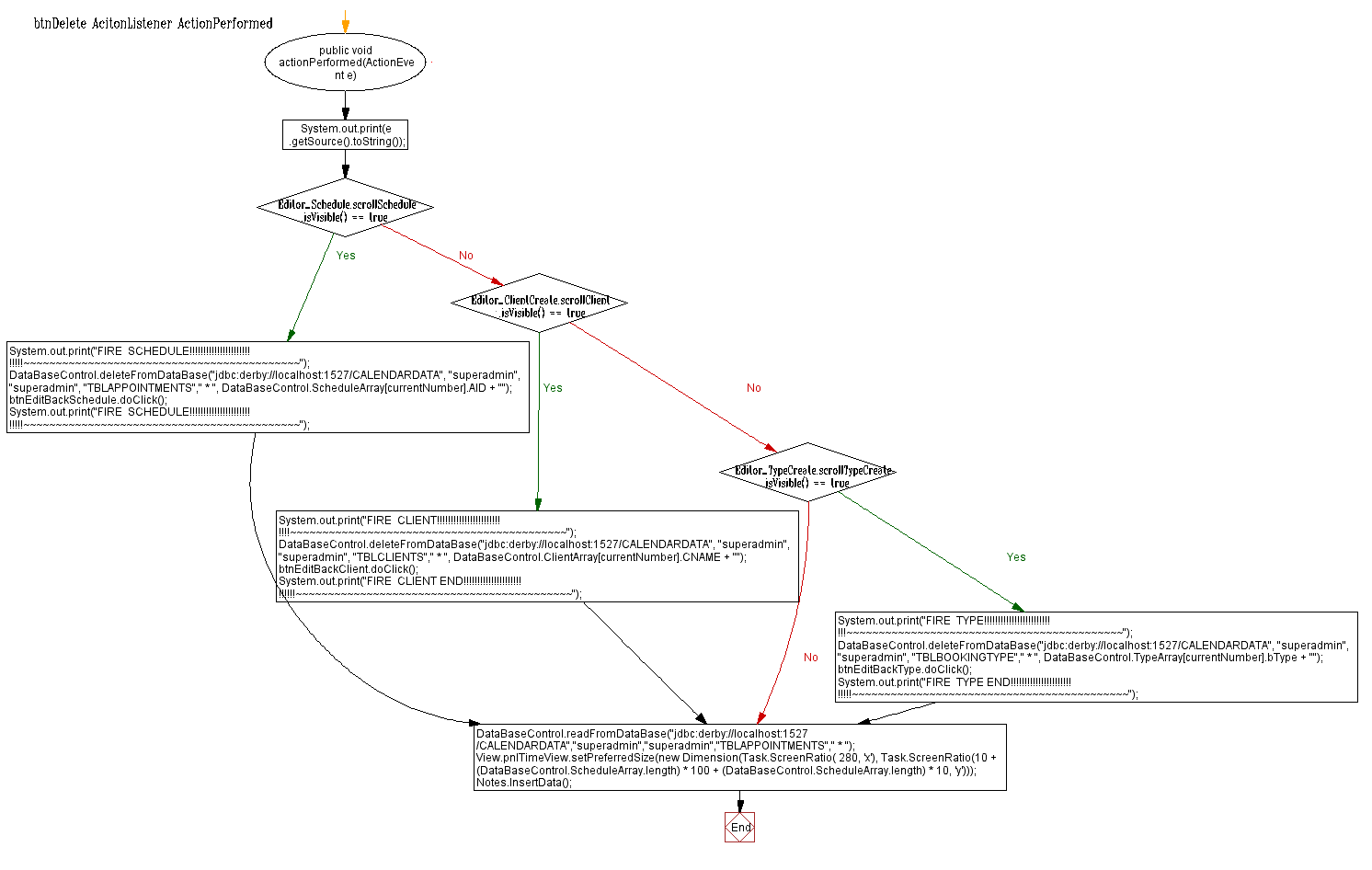
****

### 4.3.5 – btnDelete ActionListener

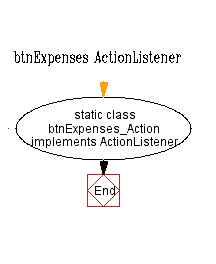
****

### 4.3.6 – btnDelete ActionListener ActionPerformed

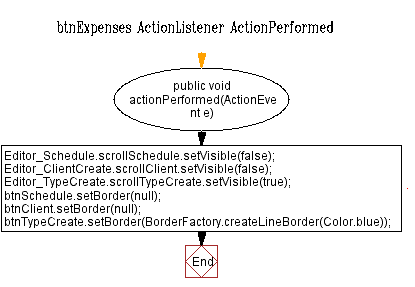
****

****

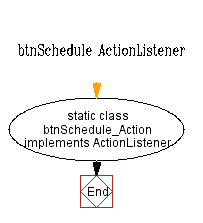
### 4.3.7 – btnExpenses ActionListener

****

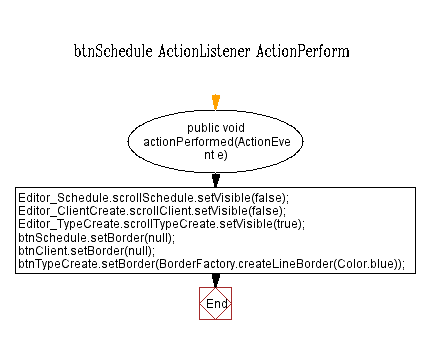
### 4.3.8 – btnExpenses ActionListener ActionPerformed

****

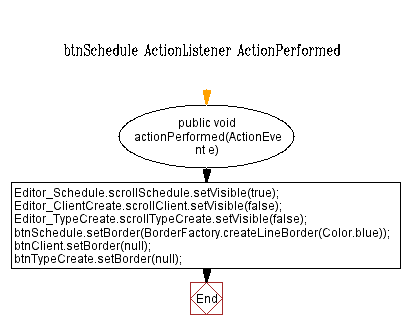
### 4.3.9 – btnSchedule ActionListener

****

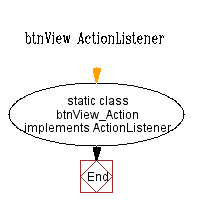
### 4.3.10 – btnSchedule ActionListener ActionPerform

****

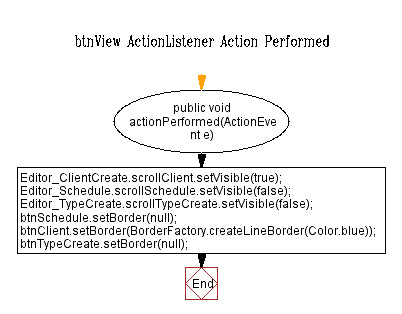
### 4.3.11 – btnSchedule ActionListener ActionPeformed

****

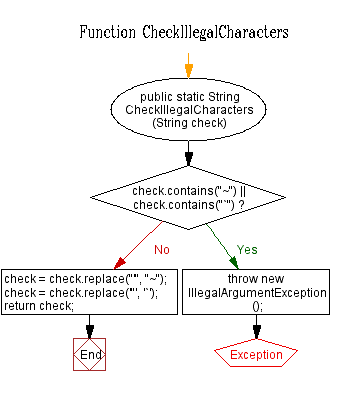
### 4.3.12 – btnView ActionListener

****

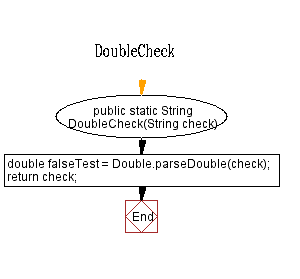
### 4.3.13 - btnView ActionListener ActionPeformed

****

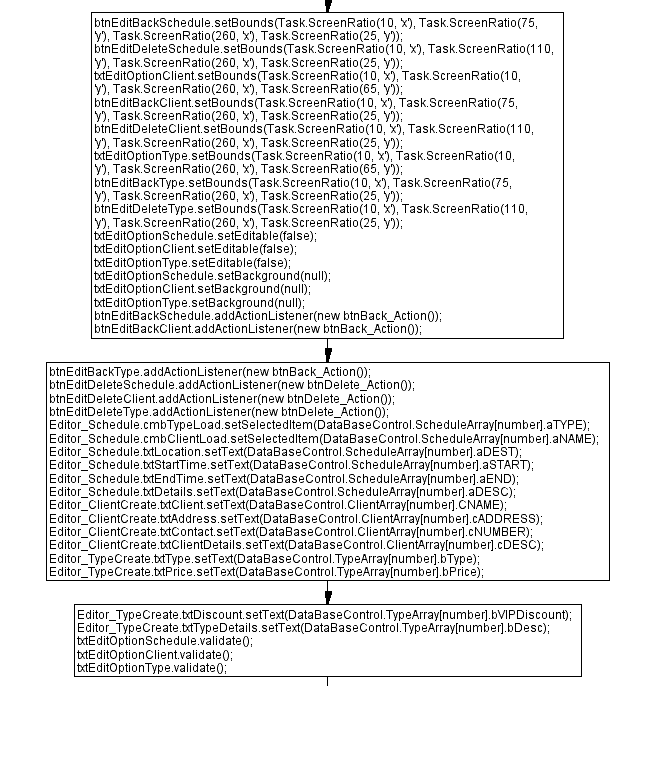
### 4.3.14 – Function CheckIllegalCharacters(String check)

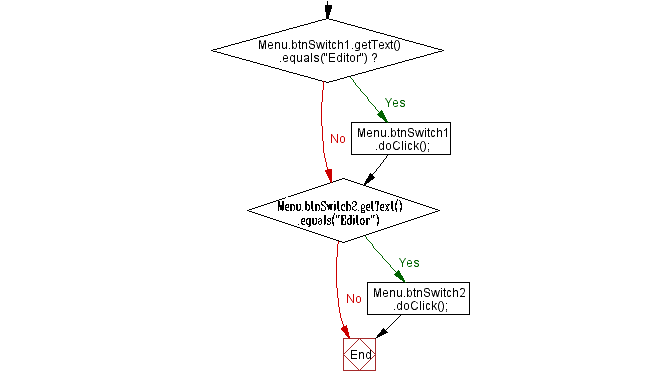
****

### 4.3.15 – DoubleCheck(String check)

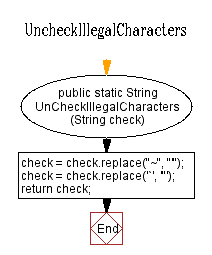
****

### 4.3.16 – Function ItemPressed(int number)F:\CalendarApp Final\Flow Charts\Editor\EditorFunctionItemPressed4part1.png

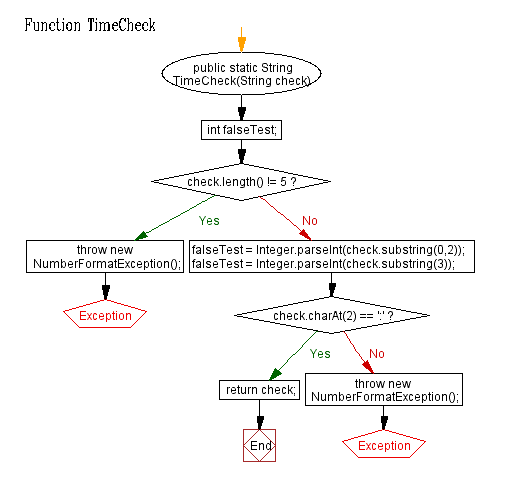
****

****

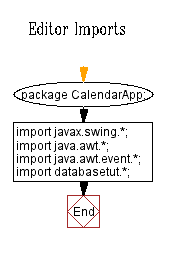
### 4.3.17 - UncheckIllegalCharacters

****

### 4.3.18 – TimeCheck(String check)

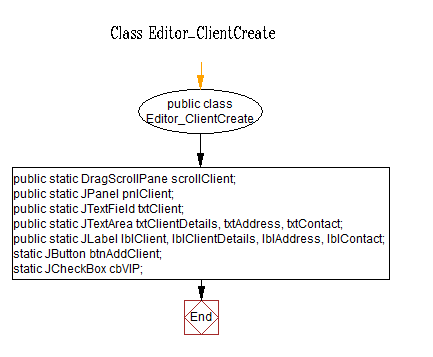
****

### 4.3.19 – Editor Imports

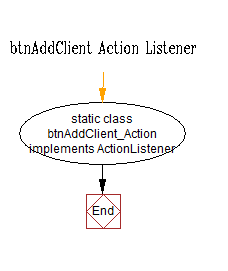
****

## 4.4 – Editor\_ClientCreate

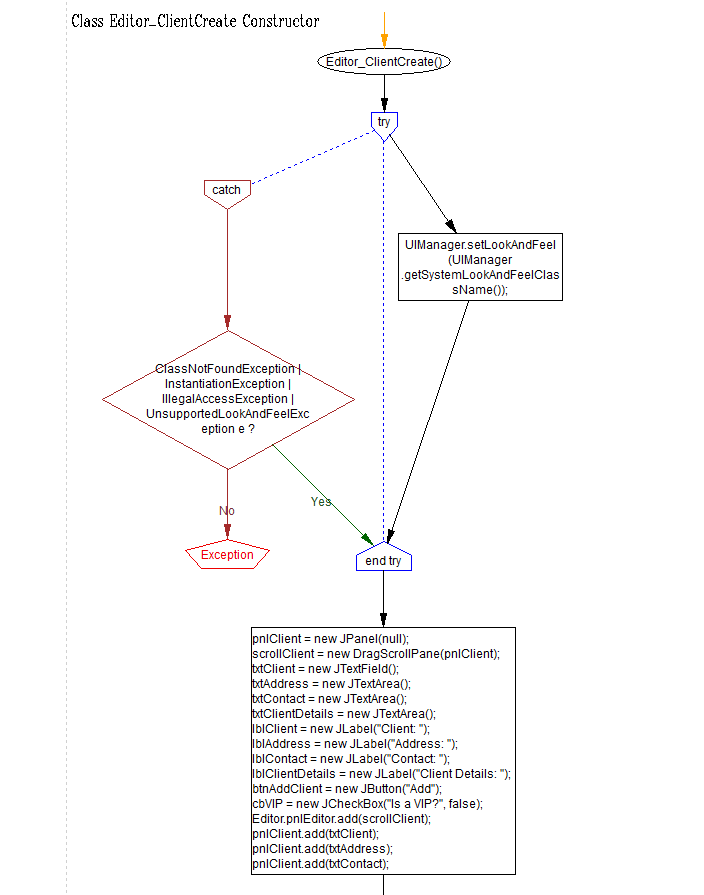
### 4.4.1 – Class Editor\_ClientCreate

****

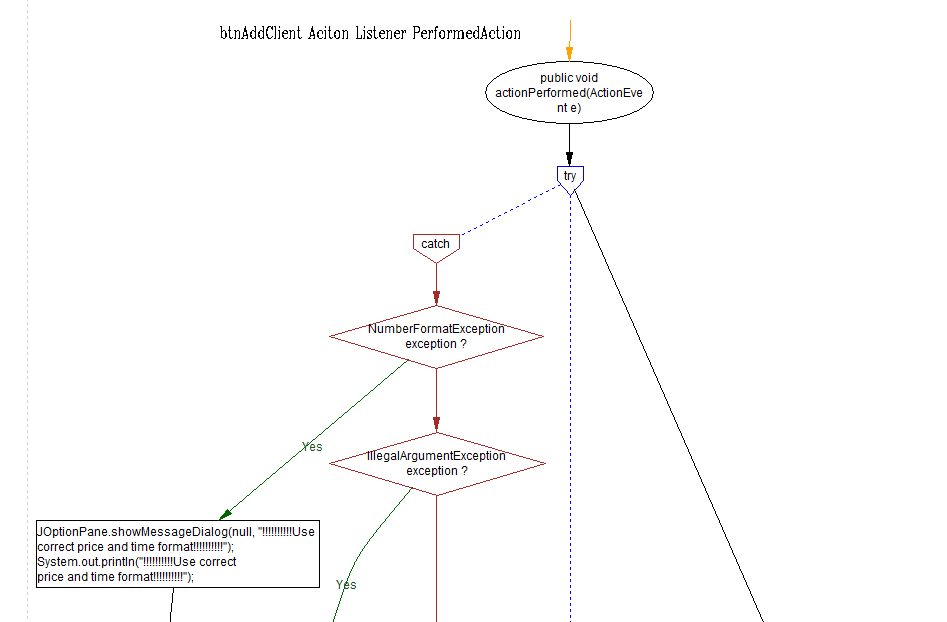
### 4.4.2 – btnAddClient ActionListener

****

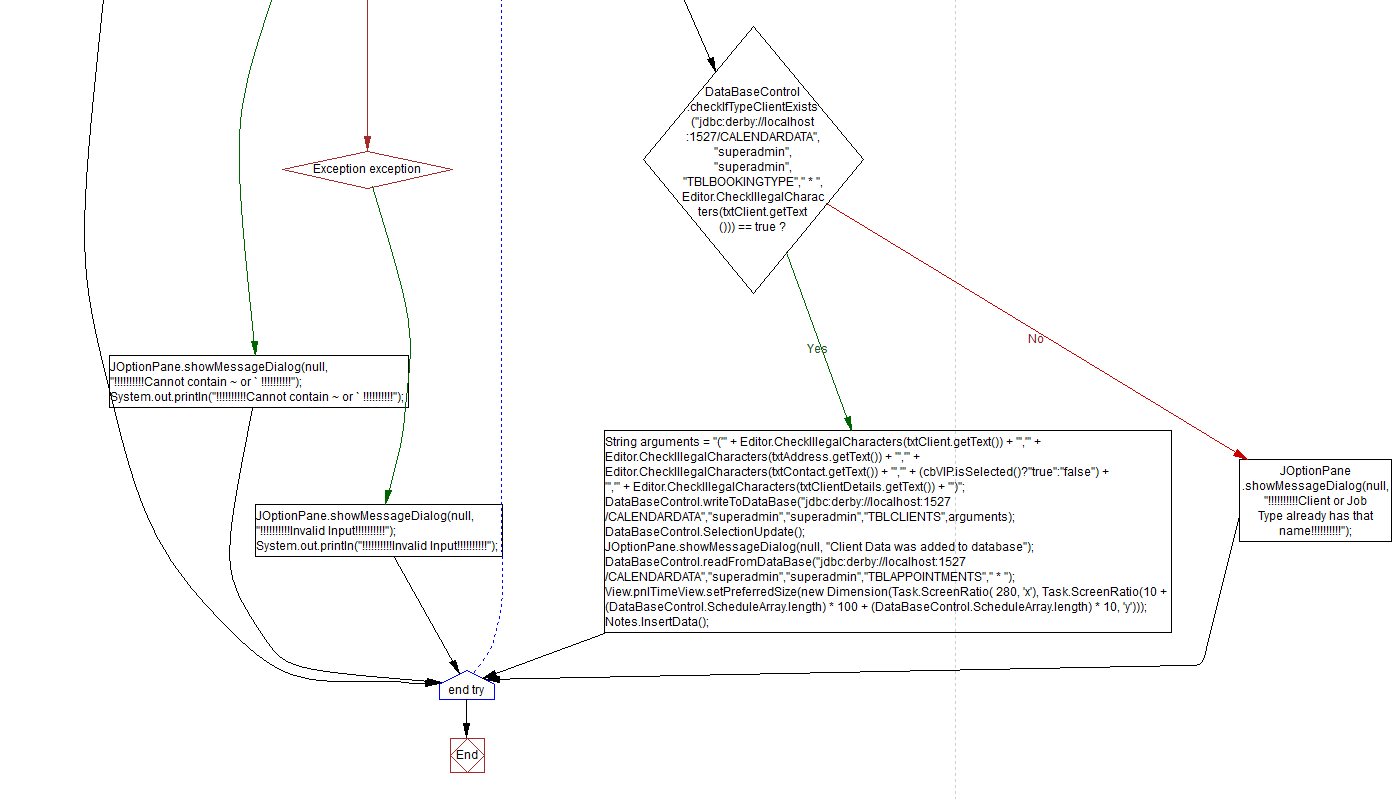
### 4.4.3 – Class Editor\_ClientCreate Constructor

****

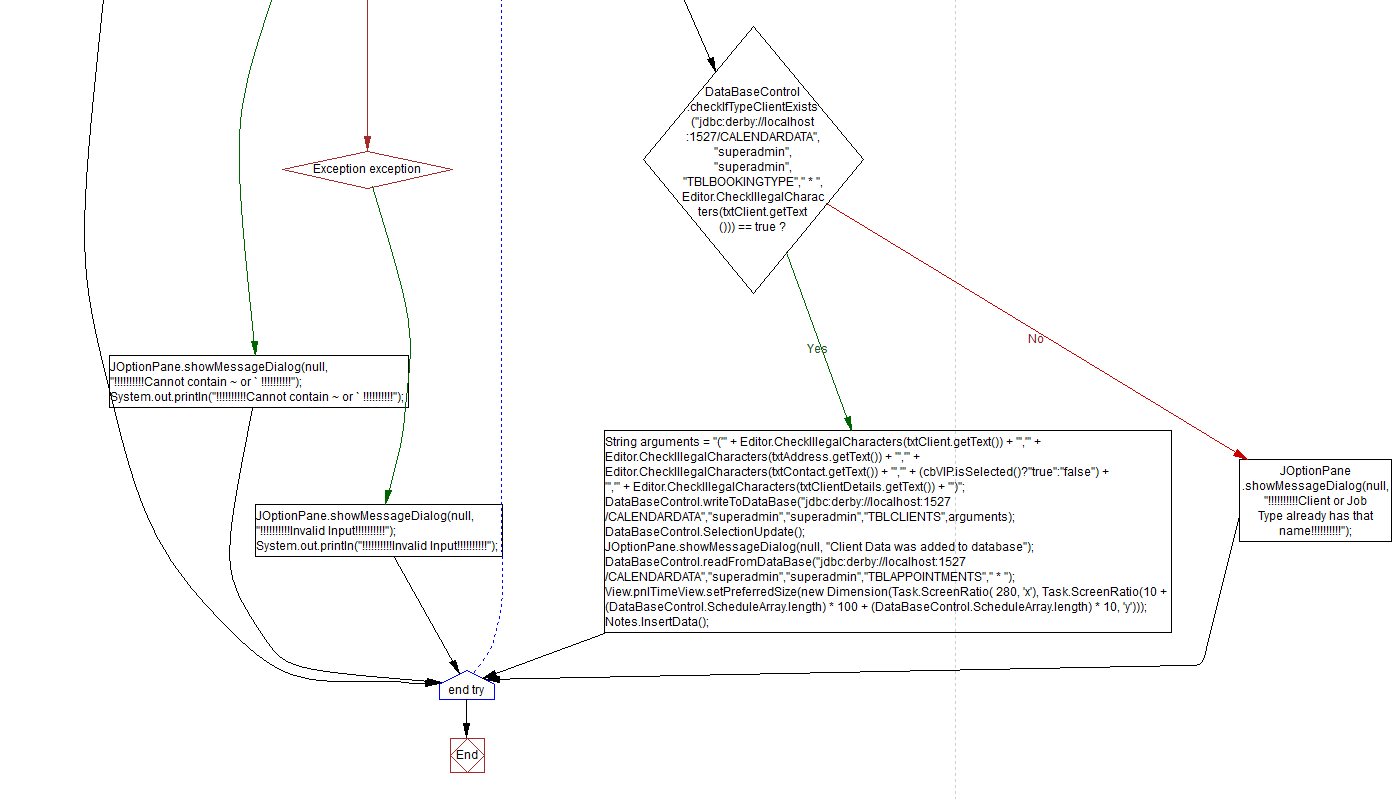
### 4.4.4 – btnAddClient ActionListener PerformedAction

****

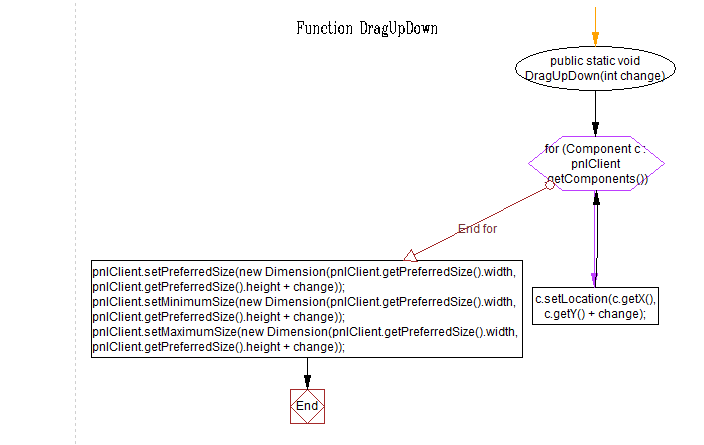
### btnAddClient ActionListener PerformedAction (right branch)

****

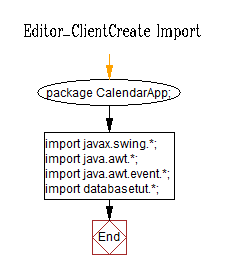
### btnAddClient ActionListener PerformedAction (left branches)

****

### 4.4.5 –Function DragUpDown

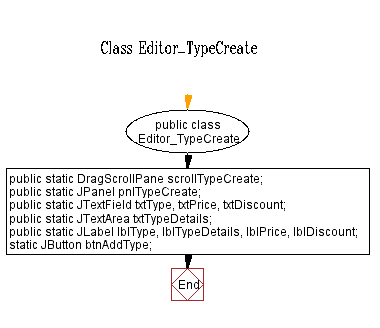
****

### 4.4.6 – Editor\_ClientCreate Imports

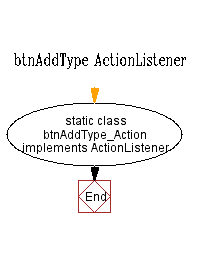
****

## 4.5 – Editor\_TypeCreate.java

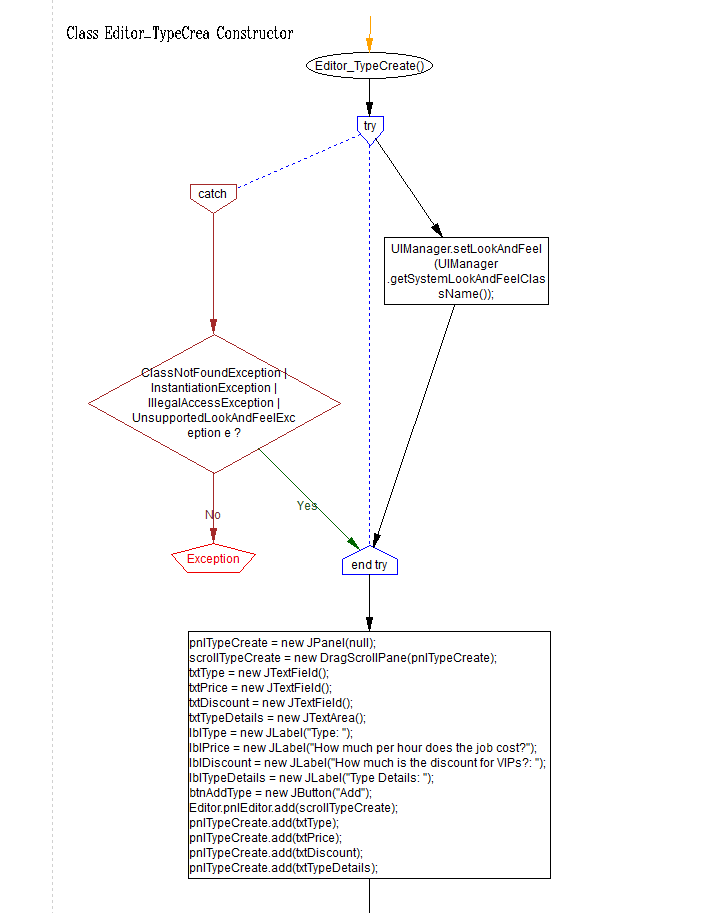
### 4.5.1 – Class Editor\_TypeCreate

****

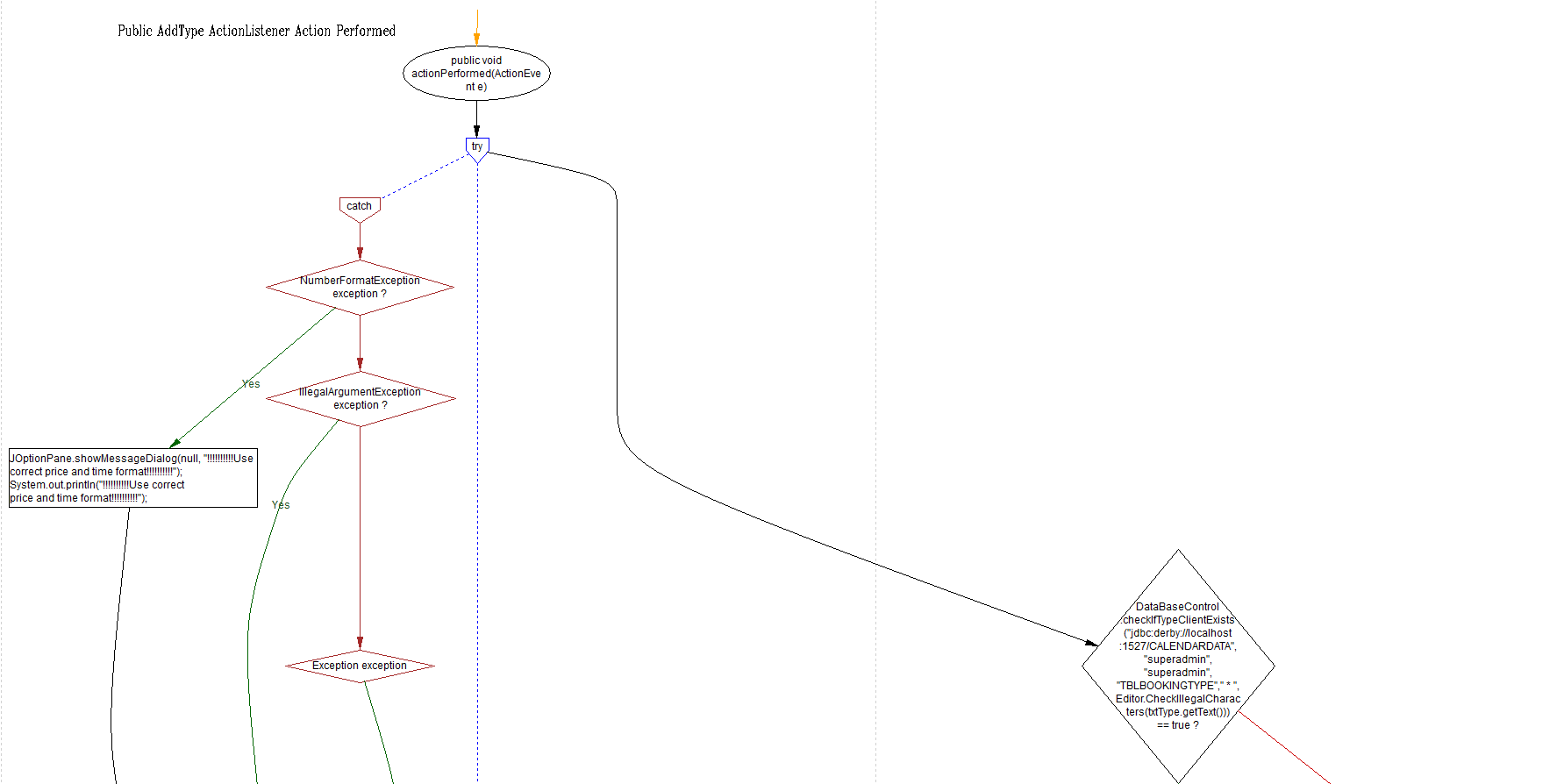
### 4.5.2 – btnAddType ActionListener

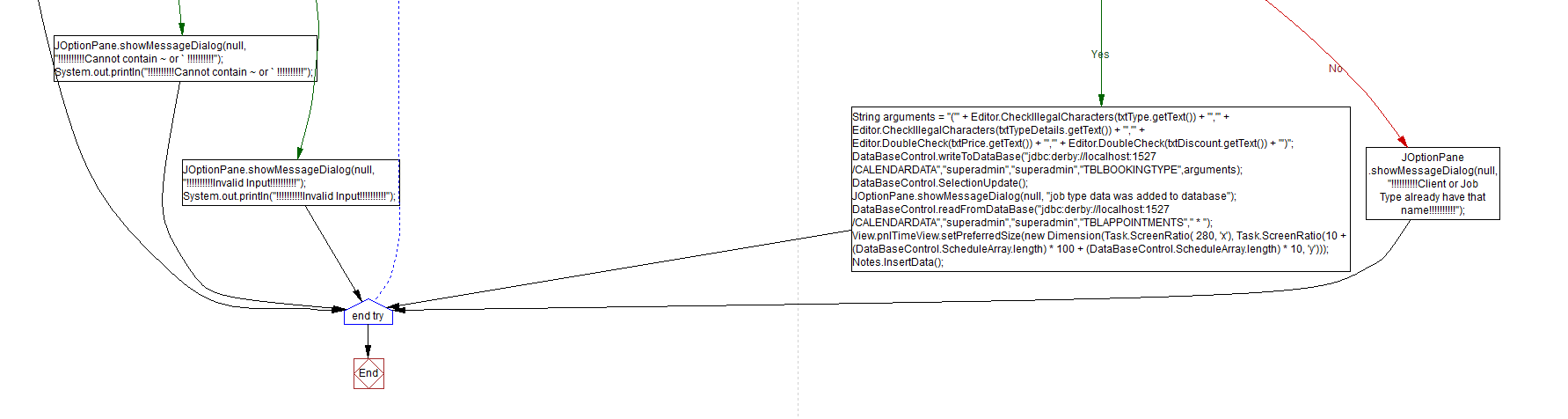
****

### 4.5.3 – Class Editor\_TypeCreate Constructor

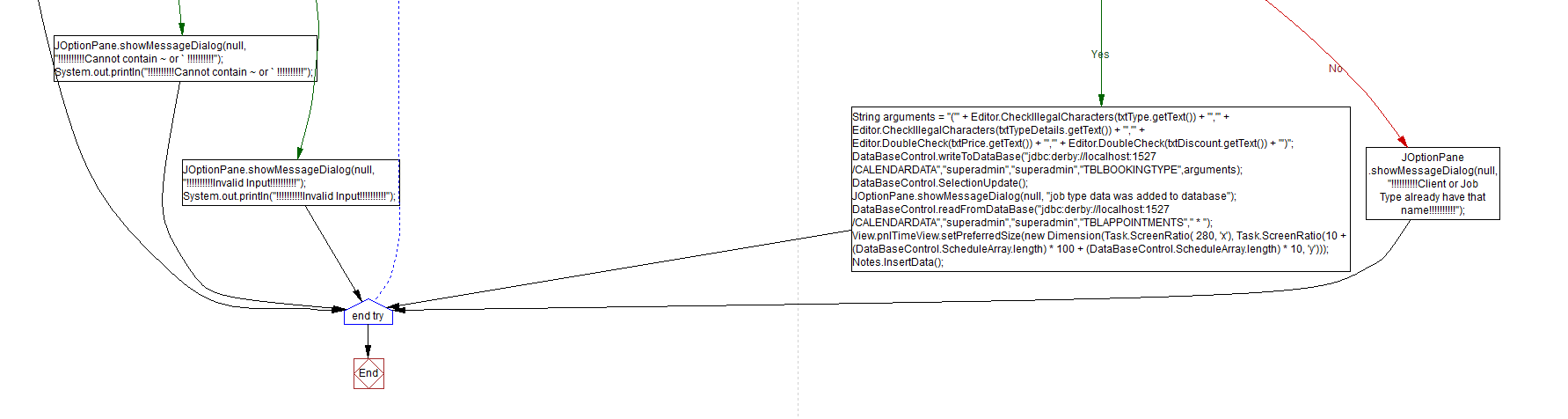
****

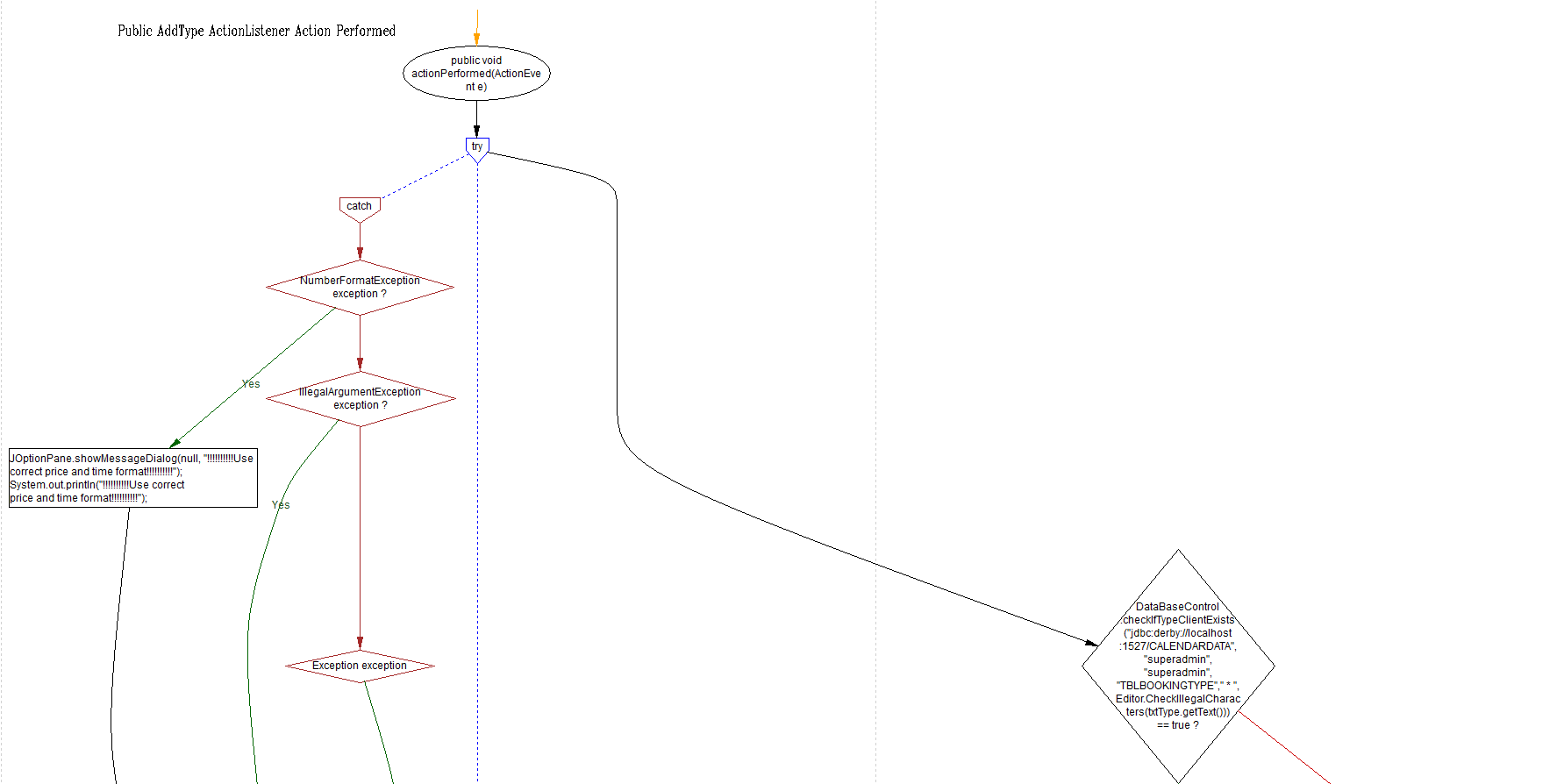
### 4.5.4 – Public AddType ActionListener ActionPerformed (left)

****

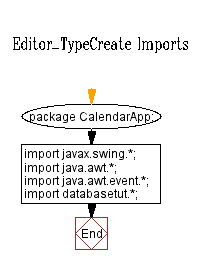
****

### Public AddType ActionListener ActionPerformed (right)

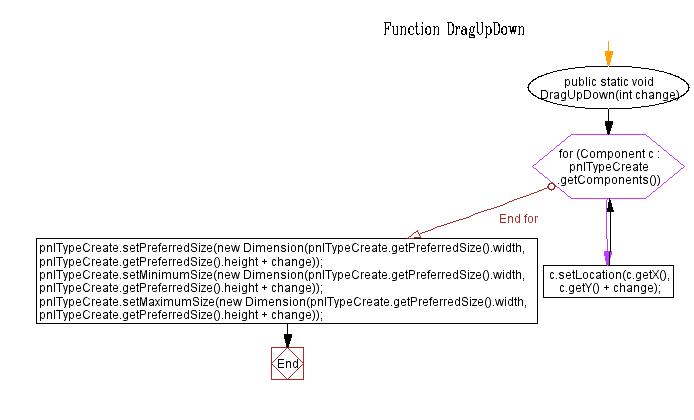
****

****

### 4.5.5 - Editor\_TypeCreate Imports

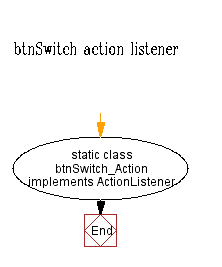
****

### 4.5.6 - Function DragUpDown(int change)

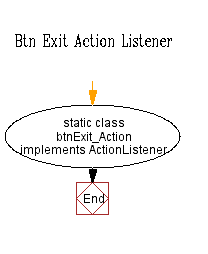
****

## 4.6 – Menu.java

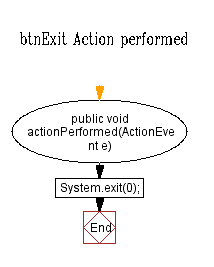
### 4.6.1 – btnSwitch ActionListener

****

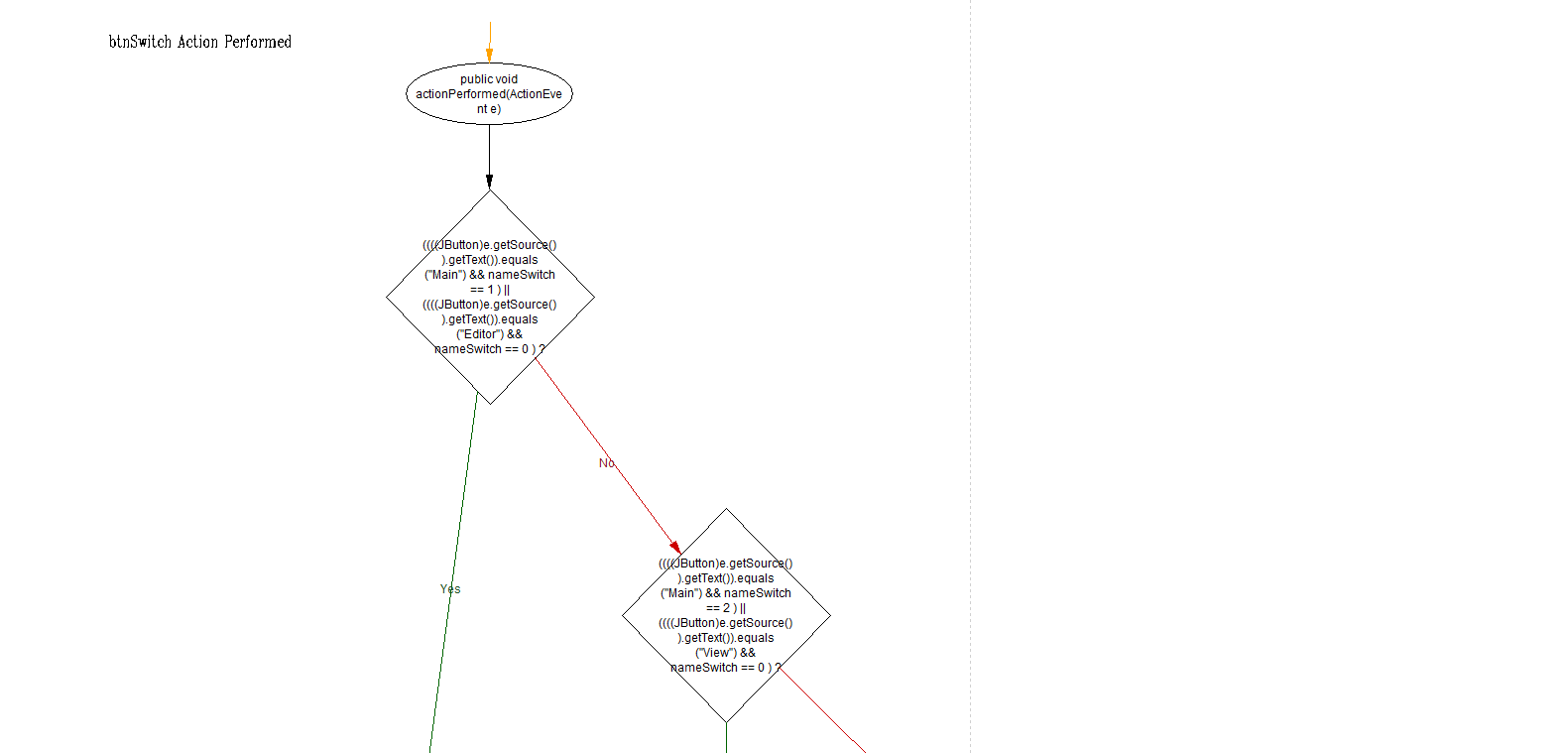
### 4.6.2 – btnExit ActionListener

****

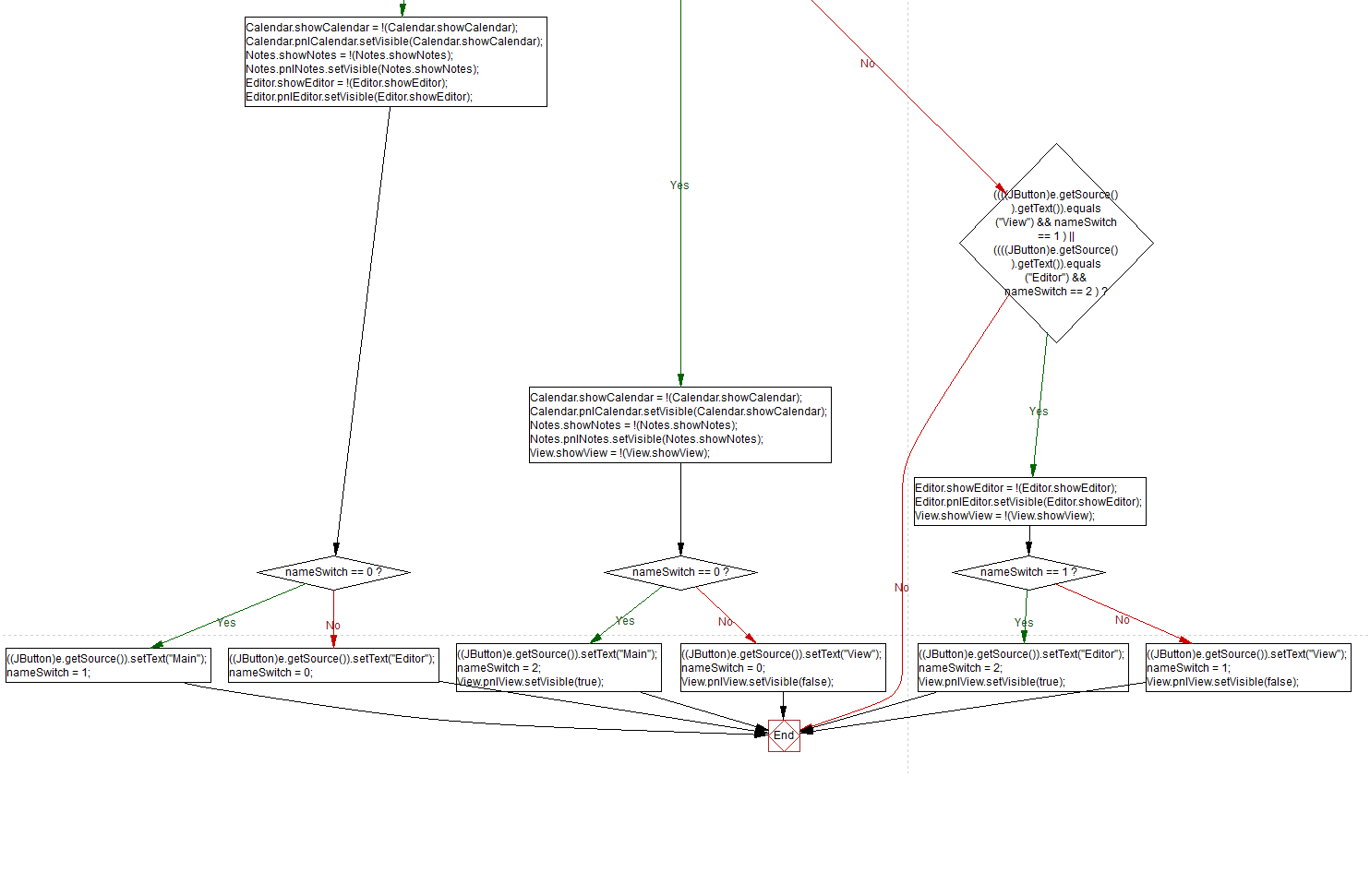
### 4.6.3 – btnExit ActionPeformed

****

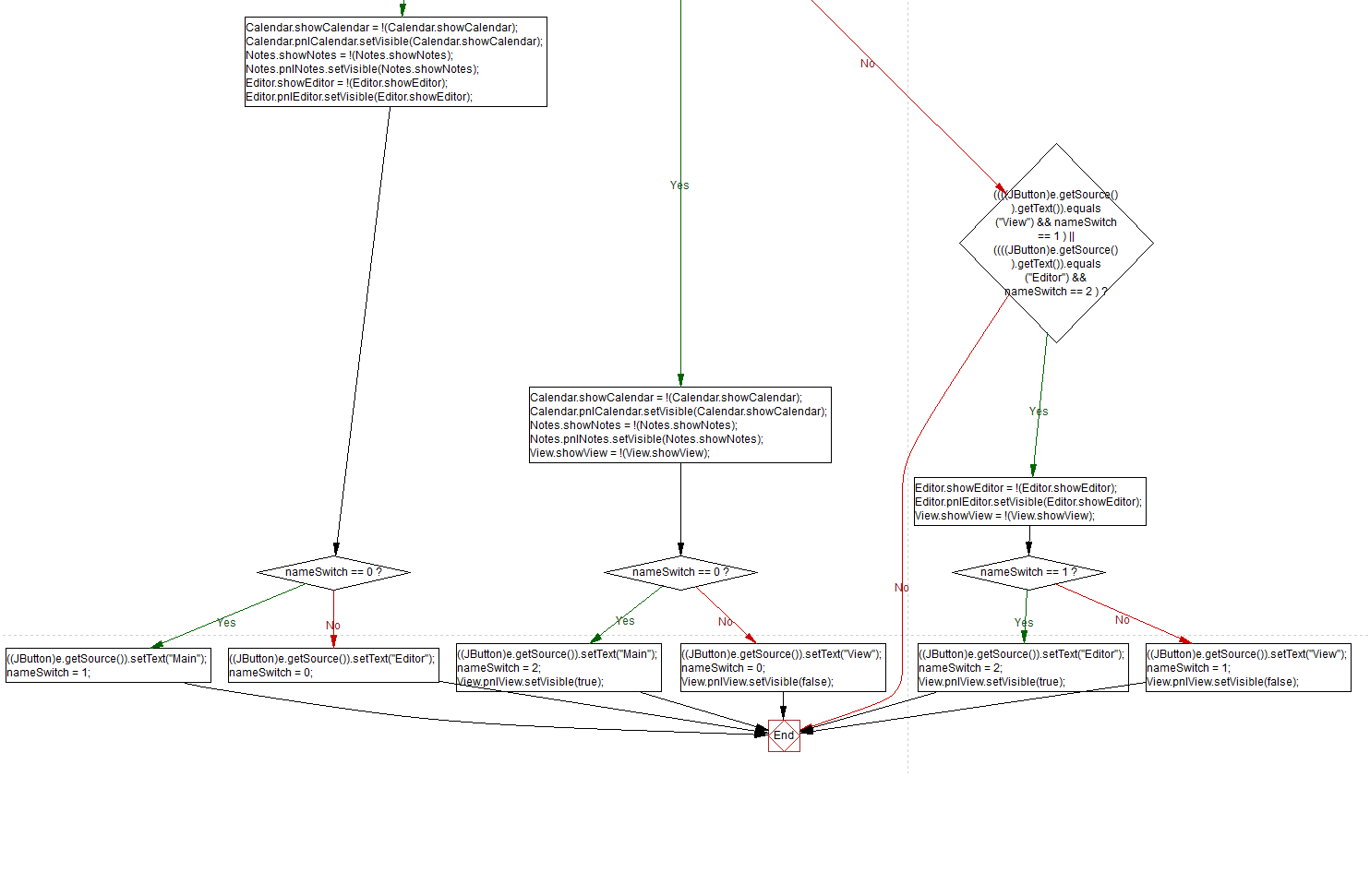
### 4.6.4 – btnSwitch ActionPeformed(top)

****

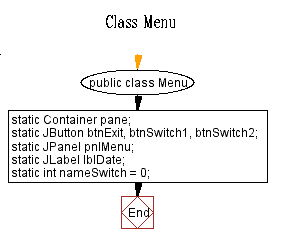
### btnSwitch ActionPerformed (bottom left)

****

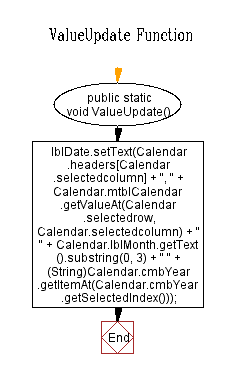
### btnSwitch ActionPerformed (bottom right)

****

### 4.6.5 – Class Menu

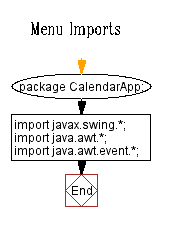
****

### 4.6.6 – Function ValueUpdate()

****

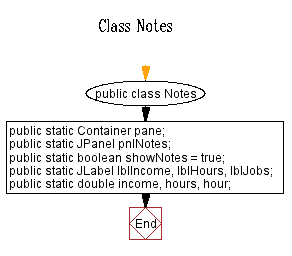
### 4.6.7 – Menu ConstructorF:\CalendarApp Final\Flow Charts\Menu\MenuConstructor3.png

### 4.6.8 – Menu Imports

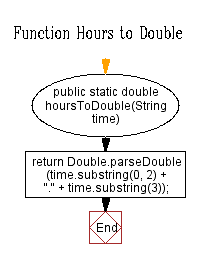
****

## 4.7 Notes.java

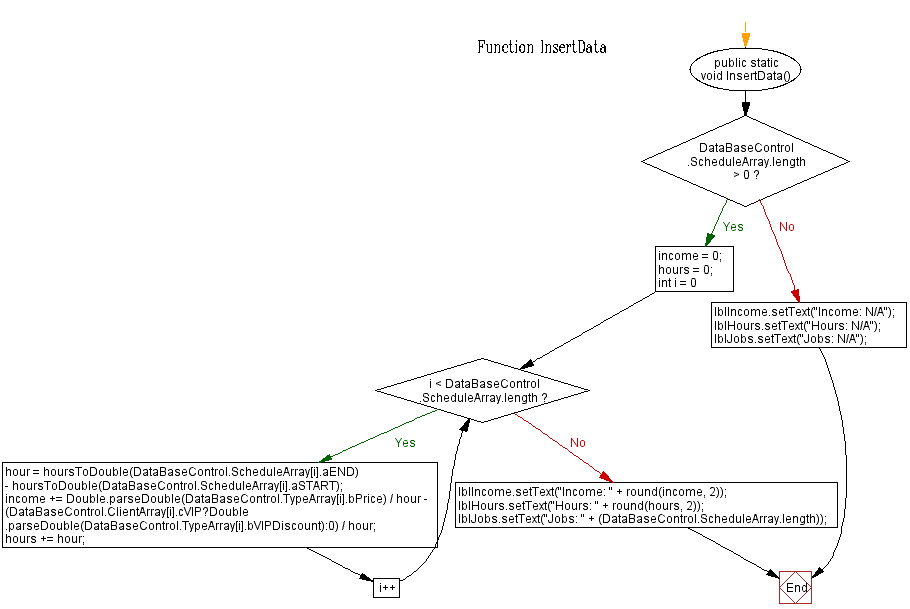
### 4.7.1 – Class Notes

****

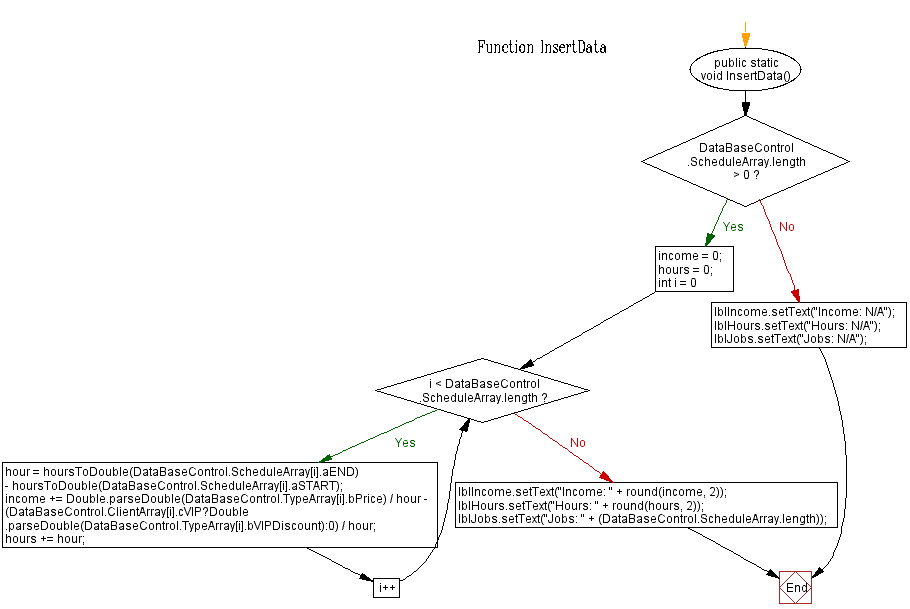
### 4.7.2 - Function hoursToDouble(String time)

****

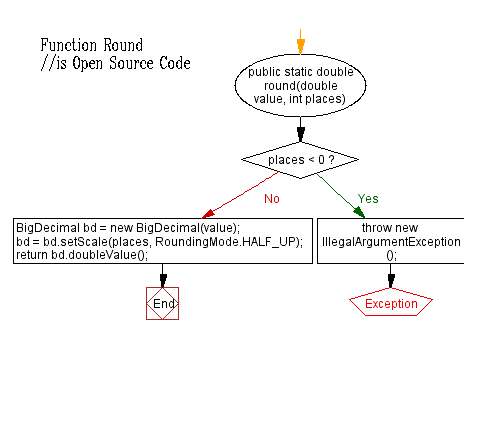
### 4.7.3 – Function InsertData() (right)

****

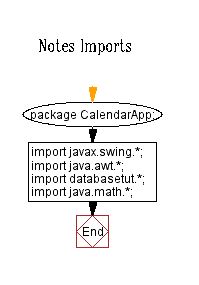
### Function insertData() (Left)

****

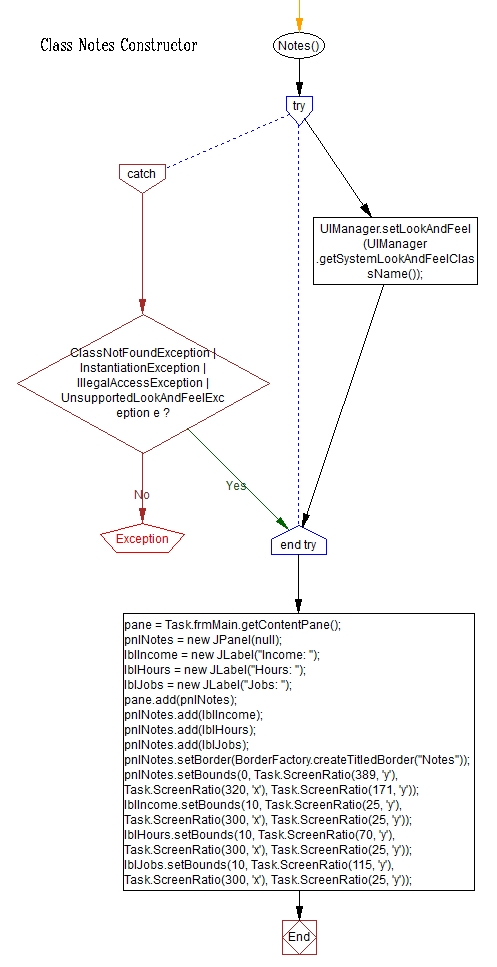
### 4.7.4 – Function Round(double value, int places)

****

### 4.7.5 – Notes Imports

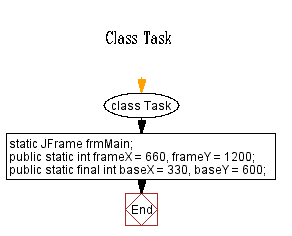
****

### 4.7.6 – Class Notes Constructor

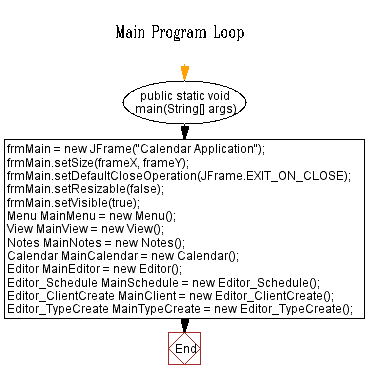
****

## 4.8 – Task.java

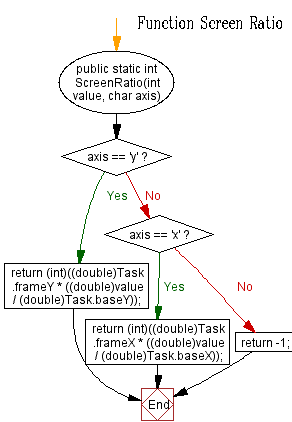
### 4.8.1 – Class Task

****

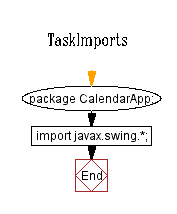
### 4.8.2 – Main Program Loop (String[] args)

****

### 4.8.3- Function ScreenRatio(int value, char axis)

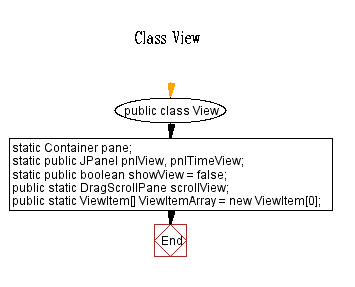
****

### 4.8.4 – Task Imports

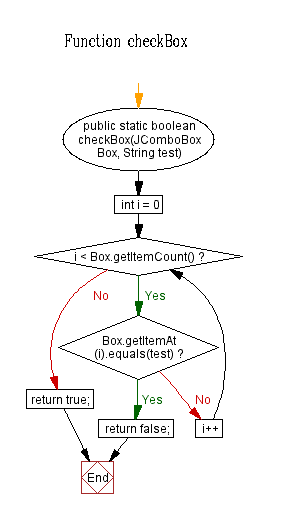
****

## 4.9 – View.java

### 4.9.1 – Class view

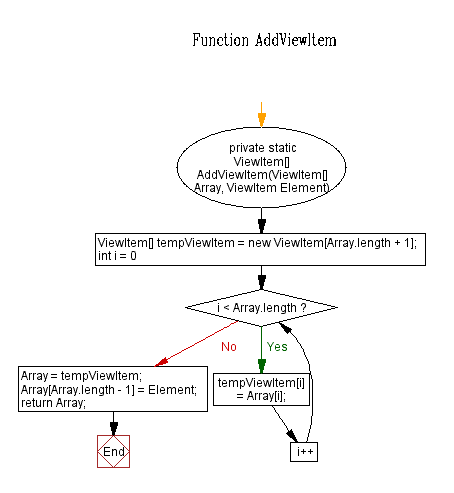
****

### 4.9.2 – Function checkbox(JComboBox Box, String test)

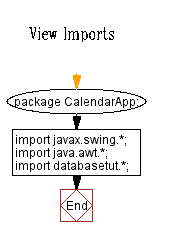
****

### 4.9.3 – Class View ConstructorF:\CalendarApp Final\Flow Charts\View\ClassViewConstructor3.png

### 4.9.4 - Function AddViewitem(ViewItem[] Array, ViewItem Element)

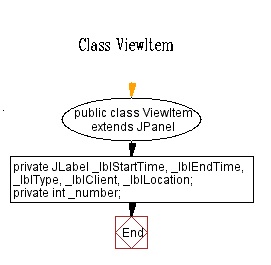
****

### 4.9.5 – View Imports

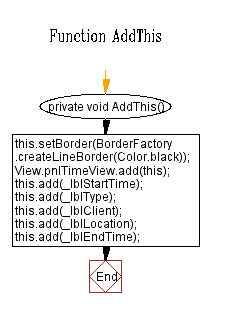
****

## 4.10 – ViewItem.java

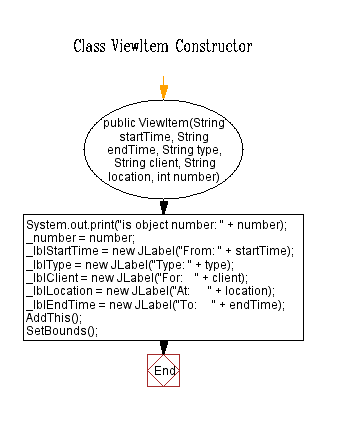
### 4.10.1 – Class ViewItem

****

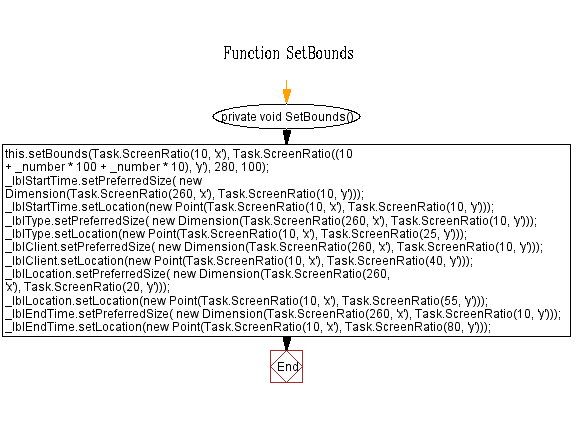
### 4.10.2 – Function AddThis()

****

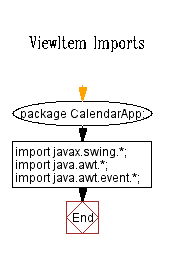
### 4.10.3 – Class ViewItem Constructor

****

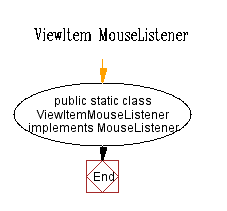
### 4.10.4 – Function setBounds()

****

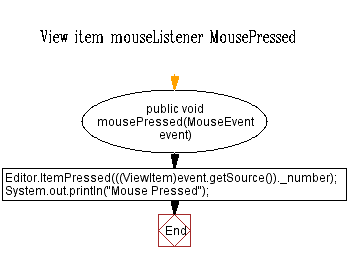
### 4.10.5 – ViewItem imports

****

### 4.10.6 – ViewItem MouseListener

****

### 4.10.7 – ViewItem MouseListener MousePressed

****

# 5.0 – Application Objects

## 5.1 – Calendar.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| lblMonth | Text Change | Changes text depending on the month currently shown in tblCalendar | Contains the name of the current month shown, for example if the currently selected month is January than it will contain the text “January”. |
| lblYear | n/a | n/a | Shows the text “Change year:” |
| btnPrev | Clicked | Changes the current month displayed on tblCalendar to the previous month. | Contains the text “<<” |
| btnNext | Clicked | Changes the current month displayed on tblCalendar to the next month. | Contains the text “>>” |
| tblCalendar | Date clicked | Upon clicking on a date, the application borders the selected date in deep blue and reads any entries for that specific date in the database, which is displayed in frmView. It also sets the date for which information will be edited in frmEdit. | Table containing all of the dates in the currently selected month. Has table headings Sun , Mon ,Tue ,Wed ,Thu ,Fri, and Sat. to signify the day of the week of the dates shown. |
| cmbYear | Clicked | Displays a list of years from 100 years before to 100 years after the current year, and changes the year displayed upon clicking on one of the years listed. | Displays the value of the current year selected while unclicked. |
| Pane | None | n/a | Contains everything located on in the frame. |
| mtblCalendar | Dragged | Moves up / down through tblCalendar | Scrolls up when the user clicks and drags down, and scrolls down when the user clicks and drags up. |
| stblCalendar (unused) | None | n/a | Was initially utilised to scroll up and down through the calendar; however given the size of the calendar this was no longer required. |
| pnlCalendar | None | n/a | Panel (group box) containing btnNext, btnPrev, tblCalendar, lblMonth, lblYear and cmbYear. |
| Cal | None | n/a | Gregorian calendar containing the date data that is referenced by tblCalendar. |

## 5.2 – Editor.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| pane | None | n/a | Contains everything located on in the frame. |
| btnSchedule | Clicked | Shows the appointment creation / editing tab | Contains the text “Schedule” |
| btnClient | Clicked | Shows the client creation / editing tab | Contains the text “Client create” |
| btnTypeCreate | Clicked | Shows the job type creation / editing tab | “Contains the text “Type create” |
| pnlEditor | None | n/a | Contains btnClient, btnSchedule and btnTypeCreate, as well as the objects displayed by user interaction with those buttons |
| txtEditOptionSchedule | None | n/a |  |
| txtEditOptionClient | None | n/a |  |
| txtEditOptionType | None | n/a |  |
| btnEditBackSchedule | Clicked | Ends schedule data editing |  |
| btnEditBackType | Clicked | Ends job type data editing |  |
| btnEditBackClient | Clicked | Ends client data editing |  |
| btnEditDeleteSchedule | Clicked | Deletes the currently open schedule data entry |  |
| btnEditDeleteClient | Clicked | Deletes the currently open client data entry |  |
| btnEditDeleteType | Clicked | Deletes the currently open job type data entry |  |

## 5.3 Editor\_ClientCreate.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| ScrollClient | Dragged | Scrolls the view of the form up when the scrollbar is moved up, and scrolls down when the scrollbar is moved down. | Located on the right side of the client editor menu. |
| pnlClient | None | n/a | Contains txtClient, txtAddress, txtContact and txtClientDetails, and lblClient, lblAddress, lblContact and lblClientDetails respectively. |
| txtClient | Text entry | While selected, reads keyboard input from the user. | Contains the name of the client, e.g. “Caleb Sazzwozzle” |
| txtClientDetails | Text entry | While selected, reads keyboard input from the user. | Contains any additional details about the client, e.g. “Client is colourblind”, or “Client lives on tall, hard to reach hill” |
| txtAddress | Text entry | While selected, reads keyboard input from the user. | Contains the client’s address, e.g. 3 Staffa Close Merrimac |
| txtContact | Text entry | While selected, reads keyboard input from the user. | Contains the client’s phone number, e.g. 55690951 |
| lblClient | None | n/a | Located directly above txtClient, contains the text “Client:” |
| lblClientDetails | None | n/a | Located directly above txtClientDetails, contains the text “Client Details:” |
| lblAddress | None | n/a | Located directly above txtAddress, contains the text “Address:” |
| lblContact | None | n/a | Located directly above txtContact, contains the text “Contact:” |
| btnAddClient | Click | Creates a new client entry in the database with the data located in txtClient, txtClientDetails, txtAddress, cbVip and txtContact. If any of these are empty , the value entered for that part of the database is empty. | Located at the top of the pnlClient, contains the text “Add” |
| cbVip | Check / Uncheck | Toggles the Boolean value of the client’s VIP status. If cbVip is unchecked, than the client is not logged as a VIP. If cbVip is checked, then the client is logged as a VIP. | Located between txtContact and txtClientDetails, contains the subheading “Is a VIP?” |

## 5.4 – Editor\_Schedule.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| scrollSchedule | Dragged | Scrolls the view of the form up when the scrollbar is moved up, and scrolls down when the scrollbar is moved down. | Located on the right side of the schedule editor menu. |
| pnlSchedule | None | n/a | Contains txtStartTime, txtEndTime, txtLocation, txtDetails, lblLocation, lblStartTime, lblEndTime, lblDetails, cmbTypeLoad, cmbClientLoad, btnSave, cbNever, cbDay, cbWeek, cbMonth, cbYear and pnlRepeat. |
| txtStartTime | Text entry | While selected, reads keyboard input from the user. | Contains the starting time of the appointment / event in 24 hour time  e.g. 0600 = 6 o’clock in the morning, and 1800 = 6 o’clock at night |
| txtEndTime | Text entry | While selected, reads keyboard input from the user. | Contains the ending time of the appointment / event in 24 hour time |
| txtLocation | Text entry | While selected, reads keyboard input from the user. | Contains where the appointment / event is located, e.g. “3 Staffa Close Merrimac” or “My office” |
| txtDetails | Text entry | While selected, reads keyboard input from the user. | Contains any additional salient details about the appointment / event. |
| lblLocation | None | n/a | Acts as a title for txtLocation, contains the text “Location:” |
| lblStartTime | None | n/a | Acts as a title for txtStartTime, contains the text “Start Time:” |
| lblEndTime | None | n/a | Acts as a title for txtEndTime, contains the text “End Time:” |
| lblDetails | None | n/a | Acts as a title for txtDetails, contains the text “Details:” |
| cmbTypeLoad | Click | When clicked, displays a list of all available job types present in the database. | When a job type is selected it displays the name of the job type, otherwise no text is displayed |
| cmbClientLoad | Click | When clicked, displays a list of all available clients present in the database. | When a client is selected it displays the name of the client, otherwise no text is displayed |
| btnSave | Click | Creates a new schedule entry in the database, using the information in cmbClientLoad, cmbTypeLoad, txtLocation, txtStartTime, txtEndTime, txtDetails and the repeat value set by cbNever, cbDay, cbWeek, cbMonth and cbYear. | Located at the top of pnlSave, contains the text “Save” |
| cbNever | Check/uncheck | Sets that the event never repeats after the current instance. When this is checked, all other checkboxes related to it (cbDay, cbWeek, cbMonth or cbYear) are unchecked. | Contained in pnlRepeat, has the subtitle “ |
| cbDay | Check/uncheck | Sets that the event repeats daily after the current instance. When this is checked, all other checkboxes related to it(cbNever, cbWeek, cbMonth or cbYear) are unchecked. | Contained in pnlRepeat, has the subtitle “Every Day”. |
| cbWeek | Check/uncheck | Sets that the event repeats weekly after the current instance. When this is checked, all other checkboxes related to it(cbNever, cbDay, cbMonth, cbYear) are unchecked. | Contained in pnlRepeat, has the subtitle “Every week”. |
| cbFortnight(removed) | (removed) | (removed) | Originally set that the event repeated one every two weeks; however coding difficulties prevented this feature from being implemented. |
| cbMonth | Check/uncheck | Sets that the event repeats monthly after the current instance. When this is checked, all other checkboxes related to it (cbNever, cbDay, cbWeek or cbYear)  are unchecked. | Contained in pnlRepeat, has the subtitle “Every month”. |
| cbYear | Check/uncheck | Sets that the event repeats yearly after the current instance. When this is checked, all other checkboxes related to it (cbNever, cbDay, cbWeek, cbMonth) are unchecked. | Contained in pnlRepeat, has the subtitle “Every year”. |
| pnlRepeat | None | n/a | Located between the txtEndTime and txtDetails, has the subheading “Repeat” and Contains cbNever, cbDay, cbWeek, cbMonth and cbYear. |

## 5.5 – Editor\_TypeCreate.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| scrollTypeCreate (not visible in-application) | Dragged | Scrolls the view of the form up when the scrollbar is moved up, and scrolls down when the scrollbar is moved down. | This object exists, however is invisible due to the fact that pnlTypeCreate is large enough to all fit on the screen at once, and does not require a scrollbar. |
| pnlTypeCreate | None | n/a | The main panel for the typecreate form, contains txtType, txtPrice, txtDiscount, txtTypeDetails, lblType, lblTypeDetails, lblPrice, and lblDiscount. |
| txtType | Text entry | While selected, reads keyboard input from the user. | Contains the name of the job type, e.g. “Haircut”, or “Lawnmowing” |
| txtPrice | Text entry | While selected, reads keyboard input from the user. | Contains the price / hourly rate for the job (excluding or including units), e.g. 50 |
| txtDiscount | Text entry | While selected, reads keyboard input from the user. | Contains the discount value as a percentage for VIP customers e.g. 15% |
| txtTypeDetails | Text entry | While selected, reads keyboard input from the user. | Contains a brief description of the job type, e.g. “Cut the customer’s hair for a set price.” |
| lblType | None | n/a | Acts as a heading for txtType, contains the text “Type:” |
| lblTypeDetails | None | n/a | Acts as a heading for txtTypeDetails, contains the text “Type Details:” |
| lblPrice | None | n/a | Acts as a heading for txtPrice, contains the text “How much per hour does the job cost?” |
| lblDiscount | None | n/a | Acts as a heading for txtDiscount, contains the text “How much is the discount for VIPs?:” |
| btnAddType | Click | Creates a new job type, using the information in txtType, txtPrice, txtDiscount and txtTypeDetails. | Located at the top of pnlTypeCreate, contains the text “Add” |

## 5.6 – menu.java

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| Pane | None | n/a | This is the main menu container for the application, containing btnExit, btnSwitch1(Editor), btnSwitch2(View), pnlMenu and lblDate. |
| btnExit | Click | Closes the application | Contained in pnlMenu and has the text “Exit” |
| btnSwitch1 |  | Switches to the edit form | Contained in pnlMenu and has the text “Editor” |
| btnSwitch2 |  | Switches to the view form | Contained in pnlMenu and has the text “View” |
| pnlMenu | None | n/a | Located at the top of the menu form and has the subtitle “Menu”, contains btnExit, btnSwitch1, btnSwitch2, and lblDate. |
| lblDate | Text changed | Changes the text contained in itself depending on the currently selected date on the calendar. | Located inside pnlMenu, changes text depending on the currently selected date. E.g. if the currently selected date is the 11th of March 2014, it will show the text “Tue, 11 Mar 2014” |

## 5.7 – notes.java (redundant)

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Object Events** | **Event Actions** | **Comments** |
| Object Name | Events | Event Actions | Comment |
| Pane | None | n/a | Located at the bottom of the main menu form, contains pnlNotes, showNotes, and txtNotes. |
| pnlNotes | None | n/a | Has the subtitle “Notes”, contains txtNotes. |
| showNotes | None | n/a | In earlier versions of the application, this toggled whether or not notes were displayed. |
| txtNotes | Text entry | Contains any additional notes about the selected date | Located inside pnlNotes, contains any additional information about the date. |

## 5.8 – task.java

|  |  |  |  |
| --- | --- | --- | --- |
| frmMain | None | n/a | JFrame containing all objects used by the application. |
| MainMenu | None | n/a | pnlMain object containing an exit button, as well as displaying the current date and time and buttons for switching to the view and the edit forms. |
| MainCalendar | Init | Initialises the calendar upon program startup | Only serves this purpose, is never referenced after creating the calendar |
| MainNotes | Init | Initialises the notes submenu upon program startup | Only serves this purpose, is never referenced after creating the notes submenu. |
| MainView | Init | Initialises the view menu upon program startup | Only serves this purpose, is never referenced after creating the view menu. |
| MainData | Init | Initialises the data menu upon program startup | Only serves this purpose, is never referenced after creating the data menu. |
| MainEditor | Init | Initialises the editor menu upon program startup | Only serves this purpose, is never referenced after creating the editor menu. |
| MainSchedule | Init | Initialises the schedule submenu upon program startup | Only serves this purpose, is never referenced after creating the schedule submenu. |
| MainClient | Init | Initialises the client submenu upon program startup | Only serves this purpose, is never referenced after creating the client submenu. |
| MainTypeCreate | Init | Initialises the job type creation submenu upon program startup | Only serves this purpose, is never referenced after creating the job type creation submenu. |

# 6.0 – Test Report

The testing process of the application involved a process of trial and error, where false information (or null values) was inserted into the input fields, and the results were recorded. The test cases are shown below:

## 6.1 – Schedule data input test report

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Name | Pay | Discount | Details | Expected  Value | Actual Value | Test Result |
| 1 | haircut | 55 | (null) | mow the client's lawn | Show dialogue box informing user that input fields are invalid | Same as expected | PASS |
| 2 | lawnmowing | null | 15 | help the client with their fitnes | Show dialogue box informing user that input fields are invalid | Same as expected | PASS |
| 3 | personal training | 80 | 7.5 | Help the client study | Show dialogue box informing user that input fields are invalid | data successfully inserted | FAIL |
| 4 | (null) | 75 | 13 | test' | Successful insertion | Same as expected | PASS |
| 5 | test1 | 32 | 14 | test~~ | Error message informing user not to use '~' or '`' characters | Same as expected | PASS |
| 6 | test2 | 34 | 12 | test`` | Error message informing user not to use '~' or '`' characters | Same as expected | PASS |
| 7 | test3 | ## | 12 | test | Error message informing user not to use '~' or '`' characters | Same as expected | PASS |
| 9 | test5~ | 12 | 1 | test | Error message informing user not to use '~' or '`' characters | Same as expected | PASS |
| 10 | test6 | hello | 12 | test | error message telling user to make sure the correct price format is used | Same as expected | PASS |
| 11 | test7 | 12 | hello | test | error message telling user to make sure the correct price format is used | Same as expected | PASS |
| 12 | test8 | 12 | 15 | test | error message telling user that the discount reduction is greater than the actual price | data successfully inserted | FAIL |
| 13 | test9 | 12 | 0 | test | Successful insertion | data successfully inserted | PASS |
| 14 | test10 | 12 | 12 | test | Successful insertion | data successfully inserted | PASS |
| 9 | test5~ | 12 | 1 | test | Error message informing user not to use '~' or '`' characters | Same as expected | PASS |
| 10 | test6 | hello | 12 | test | error message telling user to make sure the correct price format is used | Same as expected | PASS |
| 11 | test7 | 12 | hello | test | error message telling user to make sure the correct price format is used | Same as expected | PASS |

## 6.2 – Client data input test report

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Name | Address | Contact | VIP | details | Expected value | Actual value | Test results |
| 1 | Damon Murdoch | 3 Staffa Close Merrimac | 55690951 | FALSE | High school student | successful insertion | same as expected | PASS |
| 2 | Gareth Watson | (null) | 55563456 | TRUE | High school student | successful insertion | same as expected | PASS |
| 3 | Edward elric | Central command | (null) | TRUE | Amestrian State Alchemist | successful insertion | same as expected | PASS |
| 4 | Steven Stone | Mossdeep city | (null) | TRUE | Some guy | successful insertion | same as expected | PASS |
| 5 | Jarod Burger | [/b/](http://www.4chan.org/) | 127.0.0.1 | FALSE | test`` | Error message informing user not to use '~' or '`' characters | same as expected | PASS |
| 6 | Mr Ricardo | E3 | (null) | TRUE | Guy who's marking this | successful insertion | same as expected | PASS |
| 7 | Dave | Nopeville | 14424442 | FALSE | Test~~ | Error message informing user not to use '~' or '`' characters | same as expected | PASS |
| 8 | Bill Nye | Somewhere | 44432423 | FALSE | The science guy | successful insertion | same as expected | PASS |
| 9 | Rick Astley | Somewhere | 13333337 | TRUE | He's never gonna give you up | successful insertion | same as expected | PASS |

## 6.3 – Schedule data input test report

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Job | Client | Location | Start Time | End time | Repeats | Description | Expected value | Actual value | Result |
| 1 | Hair  cut | Bill Nye | Somewhere | 12:00 | 13:00 | Never | (null) | Successful insertion | Same as expected | PASS |
| 2 | Hair  cut | Damon Murdoch | 3 Staffa Close Merrimac | 14:30 | 15:30 | Monthly | “He needs a haircut badly” | Successful insertion | Same as expected | PASS |
| 3 | Hair  cut | Gareth Watson | Somewhere | 11:30 | 12:30 | weekly | “” | Successful insertion | Same as expected | PASS |
| 4 | Hair  cut | Edward Elric | (null) | 12:30 | 13:00 | never | “” | Error message informing user that there has been no location selected | Same as expected | PASS |
| 5 | Hair  cut | Rick Astley | The hairdressers | 13:00 | 14:00 | never | “~” | Error message informing user not to use ‘~’ or ‘`’ characters | Same as expected | PASS |
| 6 | Hair  cut | Bill Nye | Spaaace | 13:30 | 14:30 | never | “`” | Error message informing user not to use ‘~’ or ‘`’ characters | Same as expected | PASS |

## 6.4 – Third-Party Reviews

### 6.4.1 – Review 1: Steffi Tan

Colours used upon calendar are barely visible to the eye if under the wrong kind of lighting or graphics screen. Font types are inconsistent throughout the form.

User manual has yet been created and it quite hard to navigate and understand without it.

Placement of the Exit button is unconventional and may lead people astray when using the application. Logical navigation of buttons and labels should be implemented to help users interact with the interface. Starting from top left and working its way to bottom right. Users usually expect exit buttons to be located on the right hand side of the form. This is similar to the editor functionality of the application as the Save button is located at the top of the screen. This forces users to scroll back up to the top of the form if they want to save.

### 6.4.2 – Review 2: Jerry Lo

Colours on calendar are confusing and difficult to see at some points. Application is difficult to navigate without previous experience and / or access to a user manual. Button placement is irregular and confusing to users. Application cannot connect to databases successfully unless it is run through netbeans and / or the database has already been created.

# 7.0 – Evaluation (Damon Murdoch)

## 7.1 – Performance

This application operates efficiently without significant impacts upon system performance on most devices. The only time when the application shows performance issues is when the application has not successfully connected to a database and the user selects a date, causing the application to attempt to read data from a database that isn’t available. However, this isn’t a significant issue because it won’t occur unless a database isn’t present, and in that case the application won’t be functional anyway. However, if considered an issue this error could likely be rectified by inserting an ‘if’ statement in the code to check if a database is present before running code which assumes one to exist, which would likely reduce the system-intensiveness of the application in that case greatly.

Steps were taken to ensure that code used in the application was as optimised as possible, however it is likely that code could be further optimised after this stage of development. This would likely improve the performance of the application even further.

The overall size of the application is around 1.3 megabytes, however over a third of the application’s size is due to derbyclient.jar, which takes up 524 kilobytes. Derbyclient is important because it’s a java SQL driver, and without it the application would be unable to connect to a database even if one was available.

## 7.2 – Reliability

The application fulfils the majority of the functions outlined in the problem description; however some features have been unable to be implemented.

### Features Implemented:

* Utilisation of a calendar feature for selecting dates
* Highlighting the current date and weekends on the displayed calendar
* Multiple forms – Main form, view form and edit form.
* Ability for the user to store additional information on selected dates as notes
* Utilisation of an SQL database to store data, in which the entries can be later accessed for editing or deletion by the user of the application.
* Ability for the user to create client, schedule and job type data.

### Missing Features:

* Migrating the application to android devices
* Ability for android versions to utilise the camera on the user’s device to take service receipt snapshots
* Ability for the user to view and edit client, schedule and job type data after creation
* Ability for the user to log expenses data, including a snapshot of the service receipt to be used as proof of purchase.
* Ability for this application to access a database without having it initialised by netbeans or another application beforehand

Most key features of the application were able to be implemented, and most of the additional features which have been omitted are insignificant or unnecessary and the application is able to function effectively without them. However, there is one crippling issue which prevents the application from being reasonably effective. This is that without having the database pre-created via netbeans or another process, and the application cannot connect to a database. This is due to the application’s inability to instantiate databases, and relies on the database already being created. Therefore unless the database already exists it is impossible fort the application to function in the intended fashion.

## 7.3 – User-Friendliness

The application is user-friendly to a certain extent; however there are significant issues which impact user interaction with the program. The UI design was not the primary concern throughout development, as the most important feature of the application is its database connectivity, which took a great amount of time to configure. The following will evaluate the application’s UI in relation to user interface design principles, and this is reflected by the third-party reviews.

*Structure Principle – Design should organise the user purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another.*

The application attempts to purposefully organise related objects, however, as stated in user review 1, it is unlikely the user will immediately understand how the application works. For example, on the main form where the user selects the date, it is unlikely that the user will immediately realise that you have to select a date on the calendar and then press the ‘editor’ button in order to create schedule data for that date, instead of pressing the editor button to switch to the edit form and then selecting a date to edit. This issue could be rectified by setting that the application automatically opens the edit schedule form upon clicking on a date instead of selecting it, or showing a textbox inside the form which tells the user to open the ‘editor’ tab to view the data. Likewise, it is not explicitly stated that the ‘view’ form only shows schedule data for the selected date, and this may therefore cause considerable confusion to users who are unaware of this. Reading the user manual would aid the user in some of these aspects; however generally speaking the user of the application should be able to understand its functionalities without having to read the user manual beforehand. The application itself also never mentions editing previously created appointments / events, which may lead the user to believe that data cannot be edited after it has been created. This is a significant issue because if the user is unable to edit schedule data, it is impossible for them to update / delete events that have changed or been cancelled, and this would clutter the interface, along with the database with entries that are unnecessary.

*Simplicity Principle – The design should make simple, common tasks easy, communicating clearly and simply in the user’s own language, and providing good shortcuts that are meaningfully related to longer procedures.*

The application simply organises some elements, however there are significant optimisations which could be made to the User Interface. Common tasks are relatively easy, as to create new data the user only needs to switch to the editor form, select a data entry type and then fill out the fields to create a new entry, and to view schedule items the user only has to click the ‘view’ button which is located at the top of all forms except for itself. However, there are significant issues with ease-of-access and shortcuts with other parts of the application. As stated in user review 1, the exit button is located in an unconventional place, which could confuse end users. For example, to create an event for a date other than the date currently selected, the user has to switch back to the main form and then select the new date, and then return to the editor form and enter the data. This is more complicated and time consuming than it needs to be, for example having a small date selection tab at the top of the form (three listboxes, one containing the year, one containing the month, one containing the day) to change the currently selected date would be a much easier and more user-friendly way to do this. Another issue is that there are no shortcuts such as tab indexes or hotkeys to streamline the data creation / menu navigational processes. This would be especially useful in the data creation forms, where pressing tab sometimes shifts to the next text field, but sometimes the user has to manually select it with the text box, which is an unnecessary waste of time for the user. Another issue with the data creation forms, as mentioned in user review 1, is that the save button is inconveniently located at the top of the form.. This requires the user to scroll back up to the top of the form once they are finished entering data, and this is an unnecessary inconvenience and a waste of time for the end user. It would be better if the save button was located at the bottom of the menu so that the user can easily save their entry without having to scroll back to the top.

*Visibility Principle – The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information.*

Some information is visible to the end user; however there are significant issues with simple data viewing for the user. As stated in user review 1, there is no visual indication on the calendar itself to suggest which dates do or do not contain any jobs. This forces the user to manually select each date and then switch to the ‘view’ form to see if there are any jobs logged for that date, which is extremely unwieldy and an unnecessary waste of time for the user. Another design feature which requires improvement is the usage of checkboxes to select how often the event repeats in the schedule editor menu. Checkboxes are most effective for Boolean values (yes/no, male/female, etc), and are inefficient when 3 or more are used in conjunction. This is significant as the repeat checkboxes take up far more room than would be required if another method would be used, for example a list box could be used to select how often the event repeats and it would take up far less space on the form than the checkboxes.

*Feedback Principle – The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.*

There are very few user prompts which inform the user when an action, change in condition, or error / exception occurs. These are shown in the console for the application compiler as a development debug, however there are very few popup displays which actively inform the end user of any action which the application makes. For example, there is no confirmation that an entry has been added to the database after clicking the ‘add’ button on any of the data creation menus. An error message is shown when a user attempts to use a ‘or a ~ in the application, however there are no other instances where the user is informed of any errors that occur within the application. This is a significant issue because if the user encounters an issue, they will often not be informed of what is occurring and will be unable to attempt to fix the issue. For example, if schedule data is created, yet when the user presses the ‘add’ button it is not inserted into the database, the user will have no information on what is causing the issue and will have no assistance in troubleshooting / attempting to fix the issue.

*Tolerance Principle – The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.*

This is an area where significant improvements are required for the application to be fully functional. At this point in development, job types and client data is unable to be edited using the application front end, and the only job type which can be edited using the application is schedule data. This is an extremely important problem, as if the user makes a mistake when creating data for a client or job type, the application offers no way of fixing the broken entry. Likewise if the user wishes to delete a job type or client, they are unable to do so. This means that if the user makes a mistake, they must create a completely new entry, and the old entry is still left behind. This is likely to lead to the database becoming oversaturated with irrelevant and erroneous entries, which will make program navigation and usage cumbersome.

*Reuse Principle – The design should reuse internal and external components and behaviours, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.*

The design reuses internal and external components and behaviours occasionally, as the editor menus function similarly regardless of the data type you wish to create, and this makes it easier for the user to operate these parts of the application without having to remember the specifics for each data type.

## 7.4 – Problems

There were numerous significant issues encountered during the design and development stages of the application. The teamwork between both developers was inefficient at some points, as both of us were working hard on the application but there was often little communication between parties. This resulted in both members winding up with different versions of the application. This was extremely evident when Damon was working on commenting the application, as frequently Damon would be commenting code, and then Gareth would inform him afterwards that he had changed that code and made the commenting irrelevant. This often resulted in both parties having to repeat changes that had already been made before, which was an unnecessary waste of development time. This was improved later in the development cycle, as both parties worked on other parts of the application (Gareth on the Java UI frontend, and Damon on the SQL database operation), and after the SQL database access was completed Damon became primarily involved in the development of the documentation. Other than the above, one of the most significant issues was that neither of the developers were experienced in SQL before the application went into development, and this meant that significant amounts of research had to go into properly utilising SQL databases and understanding the concepts and language elements of SQL. Another issue was that the SQL / database referencing parts of the program were written by Damon, and the majority of the java front-end / UI elements were coded by Gareth. This lead to difficulty understanding each other’s code, for example it was especially difficult for Gareth to analyse the database coding when he had no previous experience with SQL. Another issue that arose when evaluating each other’s code was that Gareth and Damon both utilise different naming principles when it comes to creating variables, objects, and functions. For example, at the beginning of the application’s development, Gareth named variables using full capitals, e.g. VariableName; whereas Damon named variables by capitalising all words after the first one, e.g. variableName. Later on in the project Gareth began using the second naming convention to maintain consistency; however some variables in the application still follow the previous convention.

## 7.5 – Improvements

There are numerous improvements that could be made to improve the application. The most important improvement at this point, which is currently an application – breaking issue, is that when the application is built and not run through netbeans it is unable to create a new database. This means that unless the application is run through debug compiler mode, it is impossible for the user to do anything that involves a database. This is extremely important and completely limits the functionality of the application, and needs to be fixed before the application can be considered useable. Other than this, the most important improvement to the application would be UI optimisation. At this point, the UI (User Interface) of the application isn’t as user-friendly as it could be, and new users of the application might find it more difficult to use the application than they should. If this is rectified by improving the UI to fit with human design principles, users will find it easier to operate the application, and data entry / access will become a more streamlined and optimised process. Another improvement would be the creation of an installer executable for the application, which creates the database for the application to reference and installs the application on the user’s system simply and effectively. Another improvement which could be made, which was originally intended, was to port the application to android devices. As previously stated, the application was originally intended for android devices; however time constraints and technology issues prevented this from eventuating. It is suggested that in the future, the application could be ported to android, and for it to make full use of touch – screen and camera features. Another significant future of improvement would be database security. At the moment, the application makes no attempts to verify the user is the owner of the computer, as the database is not protected by a user-created password. This causes concerns about the safety of data in the application, as clients could have their details stolen. This could be improved by allowing the user to set their own username and password for the application.

# 8.0 – Evaluation (Gareth Watson)

## 8.1 Documentation

The creation of the actual program has been a long process and after about 5 weeks of continuously works has only within the last 2 weeks of the project come to a point where it is stable enough to even be considered adequate for submission. It was agreed that the project should be functional enough to achieve most of its main goals without common breaking, before most documentation begins. The length of the process of making the program however has caused the need to start creating some documentation before the completion of the code. Small discrepancies have thus occurred within the documentation, particularly the class diagrams; either in the situation that the purpose of certain elements have evolved or more functions has been added. There also appears to be a problem in that some of the aspects of the screenshots may be out dated. Small problems in class diagrams can however be not particularly important since the information that was lost can be seen in a technical aspect through the javadoc file located on the USB storage device supplied with this paper.

## 8.2 General Principles in Code and Algorithm Design

Although the coded side of the element may seem to be at least slightly more and more chaotic as the program progresses has actually planned to appeal to high end principles of the field.

Top down Design ~ codding in Java, especially through net beans, had given an opportunity to properly explorer this principle that would not have been available in any linear base structure. The original design of the program called for a structure of descending element ranked by how far they are attached from the frame. The first section is the top menu, which remains constant, and the wider screen that changes depending on buttons in menu. From this model it was intended that there would be three sections that can switch between each other in the main. It was determined that these would also be individual objects and thus facilitate the top down design principle where elements related to each segment would be found in that object. It has mainly held to be true, but there are instances with this obsession of classification where it might have been better to put a function in another section since most of its references reference that section even though its subject is related to the segment it is in.

Correct variable Naming **~** the use of correctly names variables had for the first part of the program in the earliest versions had been highly sporadic. Around the second week of development this problem was assessed and all variables, that did not have correct variable naming, were fixed up. In the final version of the program nearly all variables have been corrected, however as the program expanded small mistakes such as accidently naming a function with a lowercase started letter has appeared. Correct variable naming was in that second week considered to be a high priority since of its importance in group work. Correctly named variables can give some insight in itself to the purpose of that variable.

Commenting ~ Basic commenting had been applied in the development. It was deemed Important that all functions and classes were commented to inform readers of their basic function that they were trying to achieve. Highly integrated commenting detailing near every line and loop has been considered too complex and time consuming for the allocated time and not strictly necessary.

## 8.3 Flowcharts

The mechanical level flowcharts are allocated on the Digital USB storage device. Because of the length of the code the hand creation of mechanical mechanism flowcharts were deemed not primary. Those being considered not primary can also be contributed to the lesser need of flowcharts in current projects were most people having the skills to properly read the codded aspect of a program. The mechanical flowcharts are purely included to show the flow of certain algorithms code readers might find too complex or vague to understand.

## 8.4 Connected program to the database

The most complex part and least understood of the program is the DataBaseControl class and the actual connection to the data base. The name of the database when connected must be calenderdata and the username and password is both superadmin. The use of a username and password of a data base in this length is obscure and has little meaning but was implemented simply because it was included in a tutorial done. It might have been a bad idea where that the original implementation of the database was done by Damon but however I was tasked with the integration of it into the program. It was also unfortunate the neither of us have had classes on data basing and keeping efficiency. The connection to the data base may be vague but can be summed up as mostly linear system where in some cases the database can call itself with a different table to be able to gather all data from one function call. The database is definitely the strangest part of the program and the part that needs work. Its implication is most likely the reason the rest of the program might seem less orderly than originally intended due to its connections to the database.

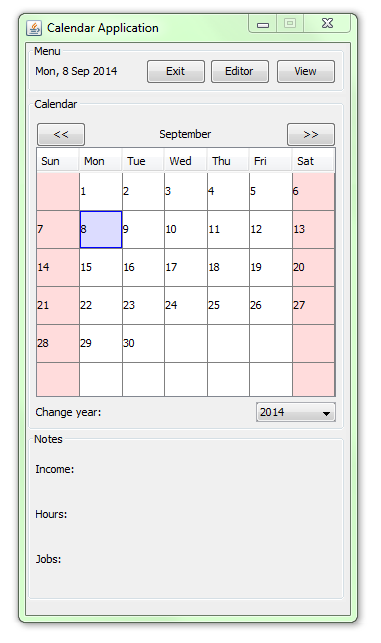
## 8.5 Distribution of Work

It should be noted that the since this work is a dual work that different aspects were done by Damon and I depending on who we deemed to be best at each field. Damon was deemed to be best at data basing and documentation, and I was deemed best at coding and flowcharting. Design was inputted by both of us and all aspects have been somewhat affected by each other except for the evaluations which were done mostly on our observations on what each other and ourselves have done and our respective experience.

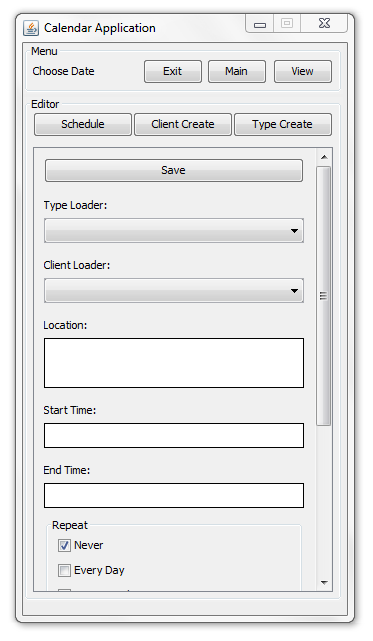
**Appendixes**

# 9.0 – Screenshots

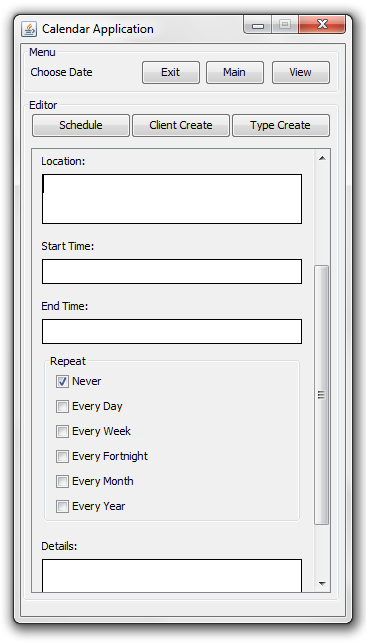
## 9.1 – Main Menu

****

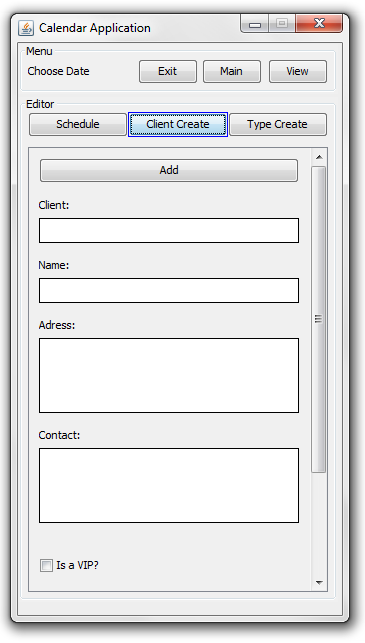
## 9.2 – Schedule Editor Window

****

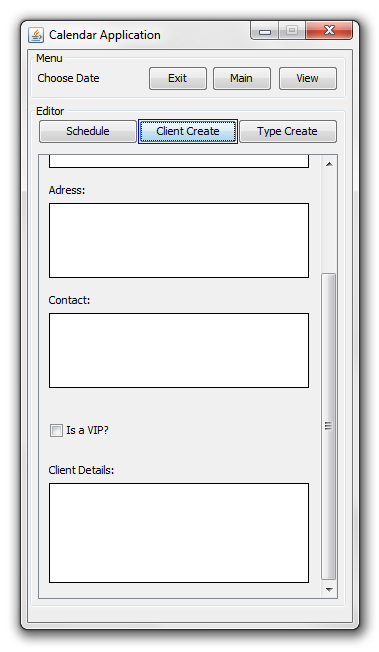
## 9.3 - Schedule Editor Window (cont’d)



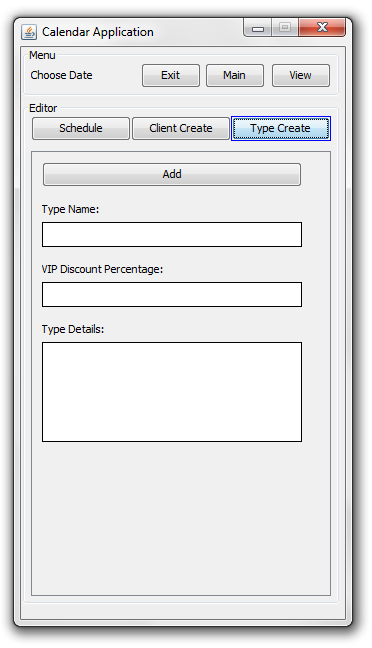
## 9.4 - Client Edit Window



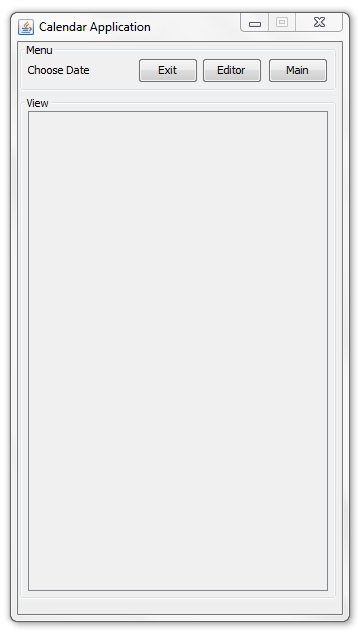
## Client Edit Window (cont’d)



## 9.5 – Job Type Edit Window



## 9.6 – View Menu:



# 10.0 – Source Code

## 10.1 – CalendarApp.package

### 10.1.1 – Calendar.java

// package declaration

package CalendarApp;

import javax.swing.\*;

import javax.swing.table.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

import databasetut.\*;

// Public class Task

public class Calendar {

// ~~ Object Initialisation ~~

// JLabel object - able to display text / images

static JLabel lblMonth, lblYear;

// Java interactable button

static JButton btnPrev, btnNext;

// Java information table

static JTable tblCalendar;

// Java drop-down box

static JComboBox cmbYear;

// Java object container

static Container pane;

//Table model, whatever that is

static DefaultTableModel mtblCalendar;

// Scrollbar

static JScrollPane stblCalendar;

// Generic, lightweight container

static public JPanel pnlCalendar;

public static boolean showCalendar = true;

public static final String[] headers = {"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};

// Initialisation of integer values relating to date / time

static int realDay, realMonth, realYear, currentMonth, currentYear, realRow, realColumn;

//public static int Framex = 330, Framey = 600;

//public static int Basex = 330, Basey = 600;

public static int selectedcolumn = -1, selectedrow = -1;

public static String selectedmonth = "", selectedyear = "", selectedYear = "";

Calendar() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

// ~~ Preparing frame ~~

// Instantiates the frame

//Gets the content pane

pane = Task.frmMain.getContentPane();

// Applies null layout

pane.setLayout(null);

// Creates controls

lblMonth = new JLabel ("January");

lblYear = new JLabel ("Change year:");

cmbYear = new JComboBox();

btnPrev = new JButton("<<");

btnNext = new JButton(">>");

mtblCalendar = new DefaultTableModel(){public boolean isCellEditable(int rowIndex, int mCollIndex){return false;}};

tblCalendar = new JTable(mtblCalendar);

stblCalendar = new JScrollPane(tblCalendar);

pnlCalendar = new JPanel(null);

// Sets the border

pnlCalendar.setBorder(BorderFactory.createTitledBorder("Calendar"));

// Registers action listeners

btnPrev.addActionListener(new btnPrev\_Action());

btnNext.addActionListener(new btnNext\_Action());

cmbYear.addActionListener(new cmbYear\_Action());

tblCalendar.addMouseListener(new mouseEventDemo());

// Adds controls to the pane

pane.add(pnlCalendar);

pnlCalendar.add(lblMonth);

pnlCalendar.add(lblYear);

pnlCalendar.add(cmbYear);

pnlCalendar.add(btnPrev);

pnlCalendar.add(btnNext);

pnlCalendar.add(stblCalendar);

//sets that the calendar will not show a scrollbar vertically

stblCalendar.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_NEVER);

//sets the bounds for pnlCalendar and the items contained in it.

pnlCalendar.setBounds(0, Task.ScreenRatio(55, 'y'), Task.ScreenRatio(320, 'x'), Task.ScreenRatio(335, 'y'));

lblMonth.setBounds( (Task.ScreenRatio(160, 'x') - lblMonth.getPreferredSize().width/2), Task.ScreenRatio(25, 'y'), Task.ScreenRatio(100, 'x'), Task.ScreenRatio(25, 'y'));

lblYear.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(305, 'y'), Task.ScreenRatio(80, 'x'), Task.ScreenRatio(20, 'y'));

cmbYear.setBounds(Task.ScreenRatio(230, 'x'), Task.ScreenRatio(305, 'y'), Task.ScreenRatio(80, 'x'), Task.ScreenRatio(20, 'y'));

btnPrev.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(25, 'y'), Task.ScreenRatio(50, 'x'), Task.ScreenRatio(25, 'y'));

btnNext.setBounds(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'), Task.ScreenRatio(50, 'x'), Task.ScreenRatio(25, 'y'));

stblCalendar.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(50, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(250, 'y'));

// Gets the actual date, month, and year

// Creates Gregorian Calendar

GregorianCalendar cal = new GregorianCalendar();

// Gets the current day

realDay = cal.get(GregorianCalendar.DAY\_OF\_MONTH);

// Gets the current month

realMonth = cal.get(GregorianCalendar.MONTH);

// Gets the current year

realYear = cal.get(GregorianCalendar.YEAR);

// Sets the currentmonth variable to the actual month

currentMonth = realMonth;

// Sets the currentyear variable to the actual year

currentYear = realYear;

for(int i = 0; i<7; i++){

mtblCalendar.addColumn(headers[i]);

}

// Sets the background colour of tbCalendar's parent to tbCalendar's colour

tblCalendar.getParent().setBackground(tblCalendar.getBackground());

// No resize / reorder

tblCalendar.getTableHeader().setResizingAllowed(false);

tblCalendar.getTableHeader().setReorderingAllowed(false);

// Single cell selection

tblCalendar.setColumnSelectionAllowed(true);

tblCalendar.setRowSelectionAllowed(true);

tblCalendar.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

// Set row / column count

tblCalendar.setRowHeight(Task.ScreenRatio(38, 'y'));

mtblCalendar.setColumnCount(7);

mtblCalendar.setRowCount(6);

// Populate table

for(int i=realYear - 100; i<= realYear+100; i++){

cmbYear.addItem(String.valueOf(i));

}

// Refresh calendar

System.out.print("Refresh! : ");

refreshCalendar(realMonth,realYear);

}

private static final String[] months = {"January","February","March","April","May","June","July","August","September","October","November","December"};

private static String selectedMonth;

// Method for reloading the calendar at the updated month and yea

public static void refreshCalendar(int month, int year){

selectedMonth = "" + ((month < 9)?"0" + (month+1):(month+1));

// ~~ Variables ~~

// Array of Strings containing the names of every month.

int nod, som; // number of days, start of month

// Previous month and next month button are enabled.

btnPrev.setEnabled(true);

btnNext.setEnabled(true);

// If currently viewing month is 10 years before actual year, disable the previous month button.

if (month == 0 && year <= realYear - 10){btnPrev.setEnabled(false);}

// If currently viewing month is 100 years after the actual year, disable the next month button.

if (month == 11 && year >= realYear + 100){btnNext.setEnabled(false);}

// label for month is set to the current month

lblMonth.setText(months[month]);

// lblMonth is resized

lblMonth.setBounds((Task.ScreenRatio(160, 'x') - lblMonth.getPreferredSize().width/2),25,180,25);

// Sets the selected item of combo box year to the current year

cmbYear.setSelectedItem(String.valueOf(year));

// Clears the table

for(int i = 0; i<6; i++){

for(int j = 0; j<7; j++){

// table calendar's value is set to null at position i,j

mtblCalendar.setValueAt(null,i,j);

}

}

// Get first month and number of days

GregorianCalendar cal = new GregorianCalendar(year,month, 1);

nod = cal.getActualMaximum(GregorianCalendar.DAY\_OF\_MONTH);

som = cal.get(GregorianCalendar.DAY\_OF\_WEEK);

// Draw calendar

for(int i = 1; i<=nod; i++){

int row = (Integer)((i+som-2)/7);

int column = (i+som-2)%7;

mtblCalendar.setValueAt(i,row,column);

if ((int)mtblCalendar.getValueAt(row, column) == realDay && selectedYear == "") {

selectedrow = row;

selectedcolumn = column;

}

}

// Apply renders

if (selectedYear == "") {

whenClickedActions(selectedrow, selectedcolumn);

} else {

tblCalendar.setDefaultRenderer(tblCalendar.getColumnClass(0), new CellRenderer());

}

}

//handles going back a month in the calendar panel

static class btnPrev\_Action implements ActionListener {

public void actionPerformed (ActionEvent e){

if (currentMonth == 0){

currentMonth = 11;

currentYear -=1;

}

else {

currentMonth -= 1;

}

refreshCalendar(currentMonth,currentYear);

}

}

//handles going forward a month in the calendar panel.

static class btnNext\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

if(currentMonth == 11){

currentMonth = 0;

currentYear += 1;

}

else {

currentMonth += 1;

}

refreshCalendar(currentMonth, currentYear);

}

}

/\* static class cell\_Action extends mouseEventDemo {

public void mousePressed (ActionEvent e){

System.out.print("PIE");

}

} \*/

//handles changing year via the year combobox

static class cmbYear\_Action implements ActionListener {

public void actionPerformed (ActionEvent e){

if (cmbYear.getSelectedItem() != null){

String b = cmbYear.getSelectedItem().toString();

currentYear = Integer.parseInt(b);

if (selectedYear != "") {

refreshCalendar(currentMonth, currentYear);

}

}

}

}

//general function for displaying message boxes

public static void infoBox(String infoMessage, String title){

{

JOptionPane.showMessageDialog(null,infoMessage, title, JOptionPane.INFORMATION\_MESSAGE);

}

}

//handles when a calendar cell is clicked on

public static class mouseEventDemo implements MouseListener {

public void mousePressed(MouseEvent e){

//for Mouse Pressed Statements

if (Editor.txtEditOptionSchedule != null) {

Editor.btnEditBackSchedule.doClick();

}

isCellEditable(e);

}

public void mouseReleased(MouseEvent e){}

public void mouseEntered(MouseEvent e){}

public void mouseExited(MouseEvent e){}

public void mouseClicked(MouseEvent e){}

}

/\*

// Get first month and number of days

GregorianCalendar cal = new GregorianCalendar(year,month, 1);

nod = cal.getActualMaximum(GregorianCalendar.DAY\_OF\_MONTH);

som = cal.get(GregorianCalendar.DAY\_OF\_WEEK);

// Draw calendar

for(int i = 1; i<=nod; i++){

int row = (Integer)((i+som-2)/7);

int column = (i+som-2)%7;

}

\*/

//gets month as an integer value representative of a month string.

public static int getIntMonth(String month) {

for (int i = 0; i < 12; i++) {

if (month.equals(months[i])) {

return i+1;

}

}

return -1;

}

//function for handling what happens when a selection changes on the calendar grid manually or automatically.

static void whenClickedActions(int row, int column) {

if (tblCalendar.getValueAt(row, column) != null) {

selectedrow = row;

selectedcolumn = column;

selectedmonth = lblMonth.getText();

selectedyear = (String)cmbYear.getItemAt(cmbYear.getSelectedIndex());

//System.out.print("column: " + selectedcolumn + " row: " + selectedrow + " ; ");

tblCalendar.setDefaultRenderer(tblCalendar.getColumnClass(0), new CellRenderer());

Menu.ValueUpdate();

selectedYear = ((int)tblCalendar.getValueAt(selectedrow, selectedcolumn) < 10?"0" + mtblCalendar.getValueAt(selectedrow, selectedcolumn):mtblCalendar.getValueAt(selectedrow, selectedcolumn)) + selectedMonth + (String)cmbYear.getItemAt(cmbYear.getSelectedIndex());

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

View.pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio( 280, 'x'), Task.ScreenRatio(10 + (DataBaseControl.ScheduleArray.length) \* 100 + (DataBaseControl.ScheduleArray.length) \* 10, 'y')));

Notes.InsertData();

}

}

//function for if a new cell is selected manually passing the data onto whenClickedActions.

public static void isCellEditable(EventObject getEvent) {

MouseEvent me = (MouseEvent) getEvent;

JTable table = (JTable) (me.getSource());

Point point = me.getPoint();

whenClickedActions(table.rowAtPoint(point), table.columnAtPoint(point));

}

//cell renderer for creating the colours on the grid.

static class CellRenderer extends DefaultTableCellRenderer {

public Component getTableCellRendererComponent(JTable table,Object value, boolean selected, boolean focused, int row , int column){

super.getTableCellRendererComponent(table, value, selected, focused, row, column);

// the following piece of of code does repeat from the main cell renderer, however,

// it was found that when one piece of rendering is done all must be done to avoid errors.

// If it's a weekend

if (column == 0 || column == 6){

setBackground(new Color(255,220,220));

}

// If it's a weekday

else {

setBackground(new Color(255,255,255));

}

if (value != null){

if (Integer.parseInt(value.toString()) == realDay && currentMonth == realMonth && currentYear == realYear){

setBackground(new Color(220,220,255));

}

}

if (selectedrow == row && selectedcolumn == column && selectedmonth == lblMonth.getText() && selectedyear == (String)cmbYear.getItemAt(cmbYear.getSelectedIndex()) && value != null) {

setBorder(BorderFactory.createLineBorder(Color.blue));

}

else {

setBorder(null);

}

setForeground(Color.black);

return this;

}

}

}

**10.1.2 – DragScrollPane.java**

//Open source taken from

//http://stackoverflow.com/questions/22873405/java-jscrollpane-move-by-dragging

package CalendarApp;

import java.awt.Cursor;

import java.awt.Point;

import java.awt.Rectangle;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.HierarchyEvent;

import java.awt.event.HierarchyListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JComponent;

import javax.swing.JScrollPane;

import javax.swing.JViewport;

public class DragScrollPane extends JScrollPane {

private static final long serialVersionUID = 1L;

DragScrollPane(JComponent objectToMove) {

super(objectToMove);

ViewportDragScrollListener l = new ViewportDragScrollListener(

objectToMove, false);

JViewport gridScrollPaneViewport = getViewport();

gridScrollPaneViewport.addMouseMotionListener(l);

gridScrollPaneViewport.addMouseListener(l);

gridScrollPaneViewport.addHierarchyListener(l);

}

class ViewportDragScrollListener extends MouseAdapter implements

HierarchyListener {

private static final int SPEED = 4;

private static final int DELAY = 10;

private final Cursor dc;

private final Cursor hc = Cursor

.getPredefinedCursor(Cursor.HAND\_CURSOR);

private final javax.swing.Timer scroller;

private final JComponent label;

private final Point startPt = new Point();

private final Point move = new Point();

private boolean autoScroll = false;

public ViewportDragScrollListener(JComponent comp, boolean autoScroll) {

this.label = comp;

this.autoScroll = autoScroll;

this.dc = comp.getCursor();

this.scroller = new javax.swing.Timer(DELAY, new ActionListener() {

public void actionPerformed(ActionEvent e) {

JViewport vport = (JViewport) label.getParent();

Point vp = vport.getViewPosition();

vp.translate(move.x, move.y);

label.scrollRectToVisible(new Rectangle(vp, vport.getSize()));

}

});

}

public void hierarchyChanged(HierarchyEvent e) {

JComponent c = (JComponent) e.getSource();

if ((e.getChangeFlags() & HierarchyEvent.DISPLAYABILITY\_CHANGED) != 0

&& !c.isDisplayable() && autoScroll) {

scroller.stop();

}

}

@Override

public void mouseDragged(MouseEvent e) {

JViewport vport = (JViewport) e.getSource();

Point pt = e.getPoint();

int dx = startPt.x - pt.x;

int dy = startPt.y - pt.y;

Point vp = vport.getViewPosition();

vp.translate(dx, dy);

label.scrollRectToVisible(new Rectangle(vp, vport.getSize()));

move.setLocation(SPEED \* dx, SPEED \* dy);

startPt.setLocation(pt);

}

@Override

public void mousePressed(MouseEvent e) {

((JComponent) e.getSource()).setCursor(hc);

startPt.setLocation(e.getPoint());

move.setLocation(0, 0);

if (autoScroll) {

scroller.stop();

}

}

@Override

public void mouseReleased(MouseEvent e) {

((JComponent) e.getSource()).setCursor(dc);

if (autoScroll) {

scroller.start();

}

}

@Override

public void mouseExited(MouseEvent e) {

((JComponent) e.getSource()).setCursor(dc);

move.setLocation(0, 0);

if (autoScroll) {

scroller.stop();

}

}

}

}

### 10.1.3 – Editor.java

//package declaration

package CalendarApp;

// Imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import databasetut.\*;

public class Editor {

// Java object container

static Container pane;

static JButton btnSchedule, btnClient, btnTypeCreate;

// Tabbel panel to place objects.

static public JPanel pnlEditor;

static public boolean showEditor = false;

Editor() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

// ~~ Preparing frame ~~

// Instantiates the frame

//Gets the content pane

pane = Task.frmMain.getContentPane();

// Applies null layout

pane.setLayout(null);

// Creates controls

pnlEditor = new JPanel(null);

btnSchedule = new JButton("Schedule");

btnClient = new JButton("Client Create");

btnTypeCreate = new JButton("Type Create");

// Sets the border

pnlEditor.setBorder(BorderFactory.createTitledBorder("Editor"));

btnSchedule.setBorder(BorderFactory.createLineBorder(Color.blue));

// Registers action listeners

btnSchedule.addActionListener(new btnSchedule\_Action());

btnClient.addActionListener(new btnView\_Action());

btnTypeCreate.addActionListener(new btnExpenses\_Action());

//adds components to their relative containers.

pane.add(pnlEditor);

pnlEditor.add(btnSchedule);

pnlEditor.add(btnClient);

pnlEditor.add(btnTypeCreate);

//sets the bounds of panel Editor and the compenents wihin it.

pnlEditor.setBounds(0, Task.ScreenRatio(55, 'y'), Task.ScreenRatio(320, 'x'), Task.ScreenRatio(505, 'y'));

btnSchedule.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(100, 'x'), Task.ScreenRatio(25, 'y'));

btnClient.setBounds(Task.ScreenRatio(110, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(100, 'x'), Task.ScreenRatio(25, 'y'));

btnTypeCreate.setBounds(Task.ScreenRatio(210, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(100, 'x'), Task.ScreenRatio(25, 'y'));

//sets panel Editors default view status to invisible.

pnlEditor.setVisible(false);

}

//Object instantiation for Dynanamic editing area.

static JTextArea txtEditOptionSchedule = null;

static JTextArea txtEditOptionClient = null;

static JTextArea txtEditOptionType = null;

static JButton btnEditBackSchedule = null;

static JButton btnEditBackType = null;

static JButton btnEditBackClient = null;

static JButton btnEditDeleteSchedule = null;

static JButton btnEditDeleteClient = null;

static JButton btnEditDeleteType = null;

public static int currentNumber = 0;

//Function for handling what happens if a schedule item is pressed in view screen.

public static void ItemPressed(int number) {

currentNumber = number;

//if the view screen is not currently active, activate it.

if (txtEditOptionSchedule == null) {

txtEditOptionSchedule = new JTextArea();

txtEditOptionClient = new JTextArea();

txtEditOptionType = new JTextArea();

btnEditBackSchedule = new JButton("Stop Editing");

btnEditBackType = new JButton("Stop Editing");

btnEditBackClient = new JButton("Stop Editing");

btnEditDeleteSchedule = new JButton("Delete");

btnEditDeleteClient = new JButton("Delete");

btnEditDeleteType = new JButton("Delete");

//pushes down all other components in the 3 editor panels to make room for the editing options.

Editor\_Schedule.DragUpDown(Task.ScreenRatio(135, 'y'));

Editor\_ClientCreate.DragUpDown(Task.ScreenRatio(135, 'y'));

Editor\_TypeCreate.DragUpDown(Task.ScreenRatio(135, 'y'));

}

//sets the informative text of the editing options.

txtEditOptionSchedule.setText("Reading From: " + "\n" + "Client: " + DataBaseControl.ScheduleArray[number].aNAME + "\n" + "Type: " + DataBaseControl.ScheduleArray[number].aTYPE);

txtEditOptionClient.setText("Reading From: " + "\n" + "Client: " + DataBaseControl.ScheduleArray[number].aNAME + "\n" + "Type: " + DataBaseControl.ScheduleArray[number].aTYPE);

txtEditOptionType.setText("Reading From: " + "\n" + "Client: " + DataBaseControl.ScheduleArray[number].aNAME + "\n" + "Type: " + DataBaseControl.ScheduleArray[number].aTYPE);

//disables the relative save/add button.

Editor\_Schedule.btnSave.setEnabled(false);

Editor\_ClientCreate.btnAddClient.setEnabled(false);

Editor\_TypeCreate.btnAddType.setEnabled(false);

//adds the editor options to their relative panels.

Editor\_Schedule.pnlSchedule.add(txtEditOptionSchedule);

Editor\_Schedule.pnlSchedule.add(btnEditBackSchedule);

Editor\_Schedule.pnlSchedule.add(btnEditDeleteSchedule);

Editor\_ClientCreate.pnlClient.add(txtEditOptionClient);

Editor\_ClientCreate.pnlClient.add(btnEditBackClient);

Editor\_ClientCreate.pnlClient.add(btnEditDeleteClient);

Editor\_TypeCreate.pnlTypeCreate.add(txtEditOptionType);

Editor\_TypeCreate.pnlTypeCreate.add(btnEditBackType);

Editor\_TypeCreate.pnlTypeCreate.add(btnEditDeleteType);

//sets the bounds of the editing options.

txtEditOptionSchedule.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(65, 'y'));

btnEditBackSchedule.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(75, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnEditDeleteSchedule.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(110, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtEditOptionClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(65, 'y'));

btnEditBackClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(75, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnEditDeleteClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(110, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtEditOptionType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(65, 'y'));

btnEditBackType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(75, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnEditDeleteType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(110, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

//set that the informative text area for editing options cannot itself be editted.

txtEditOptionSchedule.setEditable(false);

txtEditOptionClient.setEditable(false);

txtEditOptionType.setEditable(false);

//sets that the informative text has the same default background as other components.

txtEditOptionSchedule.setBackground(null);

txtEditOptionClient.setBackground(null);

txtEditOptionType.setBackground(null);

// Registers action listeners

btnEditBackSchedule.addActionListener(new btnBack\_Action());

btnEditBackClient.addActionListener(new btnBack\_Action());

btnEditBackType.addActionListener(new btnBack\_Action());

btnEditDeleteSchedule.addActionListener(new btnDelete\_Action());

btnEditDeleteClient.addActionListener(new btnDelete\_Action());

btnEditDeleteType.addActionListener(new btnDelete\_Action());

//Loads the data from the viewItem object clicked on to the 3 seperate panels

//Loads Schedule

Editor\_Schedule.cmbTypeLoad.setSelectedItem(DataBaseControl.ScheduleArray[number].aTYPE);

Editor\_Schedule.cmbClientLoad.setSelectedItem(DataBaseControl.ScheduleArray[number].aNAME);

Editor\_Schedule.txtLocation.setText(DataBaseControl.ScheduleArray[number].aDEST);

Editor\_Schedule.txtStartTime.setText(DataBaseControl.ScheduleArray[number].aSTART);

Editor\_Schedule.txtEndTime.setText(DataBaseControl.ScheduleArray[number].aEND);

Editor\_Schedule.txtDetails.setText(DataBaseControl.ScheduleArray[number].aDESC);

//Loads Client

Editor\_ClientCreate.txtClient.setText(DataBaseControl.ClientArray[number].CNAME);

Editor\_ClientCreate.txtAddress.setText(DataBaseControl.ClientArray[number].cADDRESS);

Editor\_ClientCreate.txtContact.setText(DataBaseControl.ClientArray[number].cNUMBER);

Editor\_ClientCreate.txtClientDetails.setText(DataBaseControl.ClientArray[number].cDESC);

//Loads Type

Editor\_TypeCreate.txtType.setText(DataBaseControl.TypeArray[number].bType);

Editor\_TypeCreate.txtPrice.setText(DataBaseControl.TypeArray[number].bPrice);

Editor\_TypeCreate.txtDiscount.setText(DataBaseControl.TypeArray[number].bVIPDiscount);

Editor\_TypeCreate.txtTypeDetails.setText(DataBaseControl.TypeArray[number].bDesc);

//validates the text to make sure the text changed after each new creation

//this was found to be needed after problem checking.

txtEditOptionSchedule.validate();

txtEditOptionClient.validate();

txtEditOptionType.validate();

//Switches the panel view from the view screen to the editor screen.

//this works since this function should only be fired from the view screen.

if (Menu.btnSwitch1.getText().equals("Editor")) {

Menu.btnSwitch1.doClick();

}

if (Menu.btnSwitch2.getText().equals("Editor")) {

Menu.btnSwitch2.doClick();

}

}

//Function used to convert ' and " into ~ and ` so that they can go into a data base without error.

public static String CheckIllegalCharacters(String check) {

if(check.contains("~") || check.contains("`")) {

throw new IllegalArgumentException();

}

check = check.replace("'", "~");

check = check.replace('"', '`');

return check;

}

//Function used when reading from a database to return ~ and ` to ' and ".

public static String UnCheckIllegalCharacters(String check) {

check = check.replace("~", "'");

check = check.replace('`', '"');

return check;

}

//Function used to make sure time is in the format hh:mm otherwise it throws an error.

//should only be used in a try catch loop

public static String TimeCheck(String check) {

int falseTest;

if (check.length() != 5) {

throw new NumberFormatException();

} else {

falseTest = Integer.parseInt(check.substring(0,2));

falseTest = Integer.parseInt(check.substring(3));

if (check.charAt(2) == ':') {

return check;

} else {

throw new NumberFormatException();

}

}

}

//simple function to ensure a value is a double, otherwise throws an error/

//should only be used in a try catch loop.

public static String DoubleCheck(String check) {

double falseTest = Double.parseDouble(check);

return check;

}

//Handles quiting editting options mode.

static class btnBack\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

//sets the components of the options mode to null.

txtEditOptionSchedule = null;

txtEditOptionClient = null;

txtEditOptionType = null;

btnEditBackSchedule = null;

btnEditBackType = null;

btnEditBackClient = null;

//enables the save and edit options again.

Editor\_Schedule.btnSave.setEnabled(true);

Editor\_ClientCreate.btnAddClient.setEnabled(true);

Editor\_TypeCreate.btnAddType.setEnabled(true);

//Returns the position of components to their default positions.

Editor\_Schedule.DragUpDown(Task.ScreenRatio(-135, 'y'));

Editor\_ClientCreate.DragUpDown(Task.ScreenRatio(-135, 'y'));

Editor\_TypeCreate.DragUpDown(Task.ScreenRatio(-135, 'y'));

}

}

//handles deleting parts from the database.

static class btnDelete\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

// String arguments = "(" + DataBaseControl.ScheduleArray[currentNumber].AID + ",'" + DataBaseControl.ScheduleArray[currentNumber].aNAME + "','" + DataBaseControl.ScheduleArray[currentNumber].aSTART + "','" + DataBaseControl.ScheduleArray[currentNumber].aEND + "','" + DataBaseControl.ScheduleArray[currentNumber].aDESC + "','" + DataBaseControl.ScheduleArray[currentNumber].aTYPE + "','" + Calendar.selectedYear + "','" + txtLocation.getText() + "')";

System.out.print(e.getSource().toString());

//if scrollSchedule is visible and therefore if th schedule screen is open delete the scheduled and then leave editting mode.

if (Editor\_Schedule.scrollSchedule.isVisible() == true) {

System.out.print("FIRE SCHEDULE!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLAPPOINTMENTS"," \* ", DataBaseControl.ScheduleArray[currentNumber].AID + "");

btnEditBackSchedule.doClick();

System.out.print("FIRE SCHEDULE!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

}

//if scrollClient is visible and therefore if th Client screen is open delete the client data

else if (Editor\_ClientCreate.scrollClient.isVisible() == true) {

System.out.print("FIRE CLIENT!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLCLIENTS"," \* ", DataBaseControl.ClientArray[currentNumber].CNAME + "");

btnEditBackClient.doClick();

System.out.print("FIRE CLIENT END!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

}

//if scrollTypeCreate is visible and therefore if the TypeCreate screen is open delete delete the type data.

else if (Editor\_TypeCreate.scrollTypeCreate.isVisible() == true) {

System.out.print("FIRE TYPE!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLBOOKINGTYPE"," \* ", DataBaseControl.TypeArray[currentNumber].bType + "");

btnEditBackType.doClick();

System.out.print("FIRE TYPE END!!!!!!!!!!!!!!!!!!!!!!!!!!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

}

//read in the new data in the data base and opdate the panel size and the note data section.

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

View.pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio( 280, 'x'), Task.ScreenRatio(10 + (DataBaseControl.ScheduleArray.length) \* 100 + (DataBaseControl.ScheduleArray.length) \* 10, 'y')));

Notes.InsertData();

}

}

//handles switich to the view subpanel

static class btnView\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

Editor\_ClientCreate.scrollClient.setVisible(true);

Editor\_Schedule.scrollSchedule.setVisible(false);

Editor\_TypeCreate.scrollTypeCreate.setVisible(false);

btnSchedule.setBorder(null);

btnClient.setBorder(BorderFactory.createLineBorder(Color.blue));

btnTypeCreate.setBorder(null);

}

}

//handles switching to the schedule subpanel.

static class btnSchedule\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

Editor\_Schedule.scrollSchedule.setVisible(true);

Editor\_ClientCreate.scrollClient.setVisible(false);

Editor\_TypeCreate.scrollTypeCreate.setVisible(false);

btnSchedule.setBorder(BorderFactory.createLineBorder(Color.blue));

btnClient.setBorder(null);

btnTypeCreate.setBorder(null);

}

}

//handles switching to the type subpanel.

static class btnExpenses\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

Editor\_Schedule.scrollSchedule.setVisible(false);

Editor\_ClientCreate.scrollClient.setVisible(false);

Editor\_TypeCreate.scrollTypeCreate.setVisible(true);

btnSchedule.setBorder(null);

btnClient.setBorder(null);

btnTypeCreate.setBorder(BorderFactory.createLineBorder(Color.blue));

}

}

}

### 10.1.4 – Editor\_ClientCreate.java

//package declaration

package CalendarApp;

// Imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import databasetut.\*;

public class Editor\_ClientCreate {

//Object Instantiation

public static DragScrollPane scrollClient;

public static JPanel pnlClient;

public static JTextField txtClient;

public static JTextArea txtClientDetails, txtAddress, txtContact;

public static JLabel lblClient, lblClientDetails, lblAddress, lblContact;

static JButton btnAddClient;

static JCheckBox cbVIP;

Editor\_ClientCreate() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

// Creates controls

pnlClient = new JPanel(null);

scrollClient = new DragScrollPane(pnlClient);

txtClient = new JTextField();

txtAddress = new JTextArea();

txtContact = new JTextArea();

txtClientDetails = new JTextArea();

lblClient = new JLabel("Client: ");

lblAddress = new JLabel("Address: ");

lblContact = new JLabel("Contact: ");

lblClientDetails = new JLabel("Client Details: ");

btnAddClient = new JButton("Add");

cbVIP = new JCheckBox("Is a VIP?", false);

//adds components to to be attached to their relative containers.

Editor.pnlEditor.add(scrollClient);

pnlClient.add(txtClient);

pnlClient.add(txtAddress);

pnlClient.add(txtContact);

pnlClient.add(txtClientDetails);

pnlClient.add(cbVIP);

pnlClient.add(lblClient);

pnlClient.add(lblAddress);

pnlClient.add(lblContact);

pnlClient.add(lblClientDetails);

pnlClient.add(btnAddClient);

//sets the borders of text areas for consistency.

txtClient.setBorder(BorderFactory.createLineBorder(Color.black));

txtAddress.setBorder(BorderFactory.createLineBorder(Color.black));

txtContact.setBorder(BorderFactory.createLineBorder(Color.black));

txtClientDetails.setBorder(BorderFactory.createLineBorder(Color.black));

cbVIP.setBorder(BorderFactory.createLineBorder(Color.black));

//scrolling options for scrollClient

scrollClient.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollClient.setAutoscrolls(true);

// Registers action listeners

btnAddClient.addActionListener(new btnAddClient\_Action());

//sets bounds for scrollclient and all components within it.

scrollClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(50, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(445, 'y'));

pnlClient.setPreferredSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(450, 'y')));

pnlClient.setMinimumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(450, 'y')));

pnlClient.setMaximumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(450, 'y')));

txtClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(70, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtAddress.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(130, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(50, 'y'));

txtContact.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(215, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtClientDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(335, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(100, 'y'));

cbVIP.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(270, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(45, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblAddress.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(105, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblContact.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(190, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblClientDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(310, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnAddClient.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

//Set the starting view state to invisible.

scrollClient.setVisible(false);

}

//function for moving all components of pnl calendar either up or down depending on the value of change.

//where - is up and + is down.

public static void DragUpDown(int change) {

for (Component c : pnlClient.getComponents()) {

c.setLocation(c.getX(), c.getY() + change);

}

pnlClient.setPreferredSize(new Dimension(pnlClient.getPreferredSize().width, pnlClient.getPreferredSize().height + change));

pnlClient.setMinimumSize(new Dimension(pnlClient.getPreferredSize().width, pnlClient.getPreferredSize().height + change));

pnlClient.setMaximumSize(new Dimension(pnlClient.getPreferredSize().width, pnlClient.getPreferredSize().height + change));

}

//handles adding client data to the database.

static class btnAddClient\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

//try loop in place to catch expected errors from functions that throw them if the right data has not been added.

try {

//if the client name does not currerently exist in the data base, add the client data to the database and update affected components.

if (DataBaseControl.checkIfTypeClientExists("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLBOOKINGTYPE"," \* ", Editor.CheckIllegalCharacters(txtClient.getText())) == true) {

String arguments = "('" + Editor.CheckIllegalCharacters(txtClient.getText()) + "','" + Editor.CheckIllegalCharacters(txtAddress.getText()) + "','" + Editor.CheckIllegalCharacters(txtContact.getText()) + "','" + (cbVIP.isSelected()?"true":"false") + "','" + Editor.CheckIllegalCharacters(txtClientDetails.getText()) + "')";

DataBaseControl.writeToDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLCLIENTS",arguments);

DataBaseControl.SelectionUpdate();

JOptionPane.showMessageDialog(null, "Client Data was added to database");

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

View.pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio( 280, 'x'), Task.ScreenRatio(10 + (DataBaseControl.ScheduleArray.length) \* 100 + (DataBaseControl.ScheduleArray.length) \* 10, 'y')));

Notes.InsertData();

} else {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Client or Job Type already has that name!!!!!!!!!!");

}

} catch (NumberFormatException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

System.out.println("!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

} catch (IllegalArgumentException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

System.out.println("!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

} catch (Exception exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Invalid Input!!!!!!!!!!");

System.out.println("!!!!!!!!!!Invalid Input!!!!!!!!!!");

}

}

}

}

### 10.1. 5 – Editor\_Schedule.java

// Package declaration

package CalendarApp;

// Imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import databasetut.\*;

public class Editor\_Schedule {

//Object Instantiation

public static DragScrollPane scrollSchedule;

public static JPanel pnlSchedule;

static JTextField txtStartTime, txtEndTime;

static JTextArea txtLocation, txtDetails;

static JLabel lblLocation, lblStartTime, lblEndTime, lblDetails;

public static JComboBox cmbTypeLoad, cmbClientLoad;

static JLabel lblTypeLoad, lblClientLoad;

static JButton btnSave;

static JCheckBox cbNever, cbDay, cbWeek, cbFortnight, cbMonth, cbYear;

static JPanel pnlRepeat;

Editor\_Schedule() {

// Places comstructor variables into global Calendar equivilents.

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

//Creates new components

pnlSchedule = new JPanel(null);

scrollSchedule = new DragScrollPane(pnlSchedule);

txtLocation = new JTextArea("");

txtStartTime = new JTextField("hh:mm - 24 hour time");

txtEndTime = new JTextField("hh:mm - 24 hour time");

txtDetails = new JTextArea();

lblLocation = new JLabel("Location: ");

lblStartTime = new JLabel("Start Time: ");

lblEndTime = new JLabel("End Time: ");

lblDetails = new JLabel("Details: ");

lblTypeLoad = new JLabel("Type Loader: ");

lblClientLoad = new JLabel("Client Loader: ");

cmbTypeLoad = new JComboBox();

cmbClientLoad = new JComboBox();

btnSave = new JButton("Save");

cbNever = new JCheckBox("Never", true);

cbDay = new JCheckBox("Every Day", false);

cbWeek = new JCheckBox("Every Week", false);

cbFortnight = new JCheckBox("Every Fortnight", false);

cbMonth = new JCheckBox("Every Month", false);

cbYear = new JCheckBox("Every Year", false);

pnlRepeat = new JPanel(null);

// Registers action listeners

btnSave.addActionListener(new btnSave\_Action());

cbNever.addItemListener(new CheckBox\_CheckChanged());

cbDay.addItemListener(new CheckBox\_CheckChanged());

cbWeek.addItemListener(new CheckBox\_CheckChanged());

cbFortnight.addItemListener(new CheckBox\_CheckChanged());

cbMonth.addItemListener(new CheckBox\_CheckChanged());

cbYear.addItemListener(new CheckBox\_CheckChanged());

//adds components to relative containers

Editor.pnlEditor.add(scrollSchedule);

pnlSchedule.add(txtLocation);

pnlSchedule.add(txtStartTime);

pnlSchedule.add(txtEndTime);

pnlSchedule.add(txtDetails);

pnlSchedule.add(lblLocation);

pnlSchedule.add(lblStartTime);

pnlSchedule.add(lblEndTime);

pnlSchedule.add(lblDetails);

pnlSchedule.add(lblTypeLoad);

pnlSchedule.add(lblClientLoad);

pnlSchedule.add(cmbTypeLoad);

pnlSchedule.add(cmbClientLoad);

pnlSchedule.add(btnSave);

pnlSchedule.add(pnlRepeat);

pnlRepeat.add(cbNever);

pnlRepeat.add(cbDay);

pnlRepeat.add(cbWeek);

pnlRepeat.add(cbFortnight);

pnlRepeat.add(cbMonth);

pnlRepeat.add(cbYear);

//sets the border of text for consistency

txtLocation.setBorder(BorderFactory.createLineBorder(Color.black));

txtStartTime.setBorder(BorderFactory.createLineBorder(Color.black));

txtEndTime.setBorder(BorderFactory.createLineBorder(Color.black));

txtDetails.setBorder(BorderFactory.createLineBorder(Color.black));

pnlRepeat.setBorder(BorderFactory.createTitledBorder("Repeat"));

//sets scrolling options for scrollSchedule

scrollSchedule.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollSchedule.setAutoscrolls(true);

//sets the bounds of ScrollSchedule and the components within it.

scrollSchedule.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(50, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(445, 'y'));

pnlSchedule.setPreferredSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(690, 'y')));

pnlSchedule.setMinimumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(690, 'y')));

pnlSchedule.setMaximumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(690, 'y')));

txtLocation.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(190, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(50, 'y'));

txtStartTime.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(275, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtEndTime.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(335, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(575, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(100, 'y'));

lblLocation.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(165, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblStartTime.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(250, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblEndTime.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(310, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(550, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblTypeLoad.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(45, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblClientLoad.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(105, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

cmbTypeLoad.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(70, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

cmbClientLoad.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(130, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnSave.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

pnlRepeat.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(370, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(170, 'y'));

cbNever.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

cbDay.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(40, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

cbWeek.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(65, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

cbFortnight.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(90, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

cbMonth.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(115, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

cbYear.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(140, 'y'), Task.ScreenRatio(240, 'x'), Task.ScreenRatio(25, 'y'));

//Updates the selecting in the combo box

DataBaseControl.SelectionUpdate();

}

//function that moves all components within pnlSchedule either up or down depending on the value of change.

public static void DragUpDown(int change) {

for (Component c : pnlSchedule.getComponents()) {

c.setLocation(c.getX(), c.getY() + change);

}

pnlSchedule.setPreferredSize(new Dimension(pnlSchedule.getPreferredSize().width, pnlSchedule.getPreferredSize().height + change));

pnlSchedule.setMinimumSize(new Dimension(pnlSchedule.getPreferredSize().width, pnlSchedule.getPreferredSize().height + change));

pnlSchedule.setMaximumSize(new Dimension(pnlSchedule.getPreferredSize().width, pnlSchedule.getPreferredSize().height + change));

}

//handles the save option for Schedule.

static class btnSave\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

//if the selected year is valid

if (Calendar.selectedYear != null) {

//\_actualDate.equals(tempCheck)|| \_actualDate.substring(0,2).equals(tempCheck) || "ALL".equals(tempCheck) || Calendar.headers[Calendar.selectedcolumn].equals(tempCheck) || (\_actualDate.substring(2, 4).equals(tempCheck) && \_actualDate.substring(0,2).equals(tempCheck) )

String inputDate = "";

//gets the data keyword to be used for repeating event options.

if (Editor\_Schedule.cbNever.isSelected()) {

inputDate = Calendar.selectedYear;

}

else if (Editor\_Schedule.cbDay.isSelected()) {

inputDate = "ALL";

}

else if (Editor\_Schedule.cbWeek.isSelected()) {

inputDate = Calendar.headers[Calendar.selectedcolumn];

}

else if (Editor\_Schedule.cbMonth.isSelected()) {

inputDate = Calendar.selectedYear.substring(0,2);

}

else if (Editor\_Schedule.cbYear.isSelected()) {

inputDate = Calendar.selectedYear.substring(0,2)+Calendar.selectedYear.substring(2,4);

}

//try loop purposely placed to catch exceptions sent from functions checking to make sure they have been given valid input.

try {

//if the schedule item does not overlap in time or have start time after end time... create the new item and update affected components.

if (checkScheduleDataBase((String)cmbTypeLoad.getSelectedItem(), (String)cmbClientLoad.getSelectedItem(), txtStartTime.getText(), txtEndTime.getText())) {

String arguments = "(" + Editor.CheckIllegalCharacters(DataBaseControl.Amount+"") + ",'" + Editor.CheckIllegalCharacters((String)cmbClientLoad.getSelectedItem()) + "','" + Editor.TimeCheck(txtStartTime.getText()) + "','" + Editor.TimeCheck(txtEndTime.getText()) + "','" + Editor.CheckIllegalCharacters(txtDetails.getText()) + "','" + Editor.CheckIllegalCharacters((String)cmbTypeLoad.getSelectedItem()) + "','" + Editor.CheckIllegalCharacters(inputDate) + "','" + Editor.CheckIllegalCharacters(txtLocation.getText()) + "')";

System.out.println(arguments);

DataBaseControl.writeToDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS",arguments);

JOptionPane.showMessageDialog(null, "The appointment was added to database");

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

View.pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio( 280, 'x'), Task.ScreenRatio(10 + (DataBaseControl.ScheduleArray.length) \* 100 + (DataBaseControl.ScheduleArray.length) \* 10, 'y')));

Notes.InsertData();

} else {

JOptionPane.showMessageDialog(null, "Item already exists or has overlapping times," + "\n" + "Please check the view screen");

}

} catch (NumberFormatException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

System.out.println("!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

} catch (IllegalArgumentException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

System.out.println("!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

} catch (Exception exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Invalid Input!!!!!!!!!!");

System.out.println("!!!!!!!!!!Invalid Input!!!!!!!!!!");

}

}

}

}

//test parameters for if a job overlaps or already exists.

private static boolean checkScheduleDataBase(String type, String name, String start, String end) {

boolean test = true;

for (int i = 0; i < DataBaseControl.ScheduleArray.length; i++) {

if (//If the schedule data base contains a job on the same day that has conflicting time.

//If It fully encompasses that of another job,

(Notes.hoursToDouble(start) < Notes.hoursToDouble( DataBaseControl.ScheduleArray[i].aSTART) && Notes.hoursToDouble(end) > Notes.hoursToDouble(DataBaseControl.ScheduleArray[i].aEND))

|| // or

//If It starts at the same time or anywhere up to but not including the end time of another job,

(Notes.hoursToDouble(start) >= Notes.hoursToDouble(DataBaseControl.ScheduleArray[i].aSTART) && Notes.hoursToDouble(start) < Notes.hoursToDouble(DataBaseControl.ScheduleArray[i].aEND))

|| // or

//If It ends at the same time or anywhere before up to but not including the start time of another job.

(Notes.hoursToDouble(end) > Notes.hoursToDouble(DataBaseControl.ScheduleArray[i].aSTART) && Notes.hoursToDouble(end) <= Notes.hoursToDouble(DataBaseControl.ScheduleArray[i].aEND))

|| //or

//the end time is before the start time or they are equal.

(Notes.hoursToDouble(end) <= Notes.hoursToDouble(start))

)

{

test = false;

}

}

return test;

}

//function to make sure that only one checkbox can be selected at a time.

static class CheckBox\_CheckChanged implements ItemListener {

public void itemStateChanged(ItemEvent e) {

if ( ((JCheckBox)e.getSource()).isSelected()) {

if (e.getSource() == cbNever) {

cbDay.setSelected(false);

cbWeek.setSelected(false);

cbFortnight.setSelected(false);

cbMonth.setSelected(false);

cbYear.setSelected(false);

} else if (e.getSource() == cbDay) {

cbNever.setSelected(false);

cbWeek.setSelected(false);

cbFortnight.setSelected(false);

cbMonth.setSelected(false);

cbYear.setSelected(false);

} else if (e.getSource() == cbWeek) {

cbNever.setSelected(false);

cbDay.setSelected(false);

cbFortnight.setSelected(false);

cbMonth.setSelected(false);

cbYear.setSelected(false);

//Fortnight has been deemed impracticaly and a small fix has been applied here to back it unselectable.

} else if (e.getSource() == cbFortnight) {

cbNever.setSelected(true);

cbDay.setSelected(false);

cbWeek.setSelected(false);

cbFortnight.setSelected(false);

cbMonth.setSelected(false);

cbYear.setSelected(false);

} else if (e.getSource() == cbMonth) {

cbNever.setSelected(false);

cbDay.setSelected(false);

cbWeek.setSelected(false);

cbFortnight.setSelected(false);

cbYear.setSelected(false);

} else if (e.getSource() == cbYear) {

cbNever.setSelected(false);

cbDay.setSelected(false);

cbWeek.setSelected(false);

cbFortnight.setSelected(false);

cbMonth.setSelected(false);

}

} else if (!(cbNever.isSelected() || cbDay.isSelected() || cbWeek.isSelected() || cbFortnight.isSelected() || cbMonth.isSelected() || cbYear.isSelected())){

((JCheckBox)e.getSource()).setSelected(true);

}

}

}

}

### 10.1.6 - Editor\_TypeCreate.java

//package declaration

package CalendarApp;

// Imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import databasetut.\*;

public class Editor\_TypeCreate {

//object Instantiation

public static DragScrollPane scrollTypeCreate;

public static JPanel pnlTypeCreate;

public static JTextField txtType, txtPrice, txtDiscount;

public static JTextArea txtTypeDetails;

public static JLabel lblType, lblTypeDetails, lblPrice, lblDiscount;

static JButton btnAddType;

Editor\_TypeCreate() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

// Creates controls

pnlTypeCreate = new JPanel(null);

scrollTypeCreate = new DragScrollPane(pnlTypeCreate);

txtType = new JTextField();

txtPrice = new JTextField();

txtDiscount = new JTextField();

txtTypeDetails = new JTextArea();

lblType = new JLabel("Type: ");

lblPrice = new JLabel("How much per hour does the job cost?");

lblDiscount = new JLabel("How much is the discount for VIPs?: ");

lblTypeDetails = new JLabel("Type Details: ");

btnAddType = new JButton("Add");

//adds components to their relative containers.

Editor.pnlEditor.add(scrollTypeCreate);

pnlTypeCreate.add(txtType);

pnlTypeCreate.add(txtPrice);

pnlTypeCreate.add(txtDiscount);

pnlTypeCreate.add(txtTypeDetails);

pnlTypeCreate.add(lblType);

pnlTypeCreate.add(lblPrice);

pnlTypeCreate.add(lblDiscount);

pnlTypeCreate.add(lblTypeDetails);

pnlTypeCreate.add(btnAddType);

//sets the border of some components for consistency.

txtType.setBorder(BorderFactory.createLineBorder(Color.black));

txtPrice.setBorder(BorderFactory.createLineBorder(Color.black));

txtDiscount.setBorder(BorderFactory.createLineBorder(Color.black));

txtTypeDetails.setBorder(BorderFactory.createLineBorder(Color.black));

//sets options for scrollTypeCreates scrollbar.

scrollTypeCreate.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollTypeCreate.setAutoscrolls(true);

// Registers action listeners

btnAddType.addActionListener(new btnAddType\_Action());

//sets the bounds of scrollTypeCreate and components within it.

scrollTypeCreate.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(50, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(445, 'y'));

pnlTypeCreate.setPreferredSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(435, 'y')));

pnlTypeCreate.setMinimumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(435, 'y')));

pnlTypeCreate.setMaximumSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(435, 'y')));

txtType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(70, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtPrice.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(130, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtDiscount.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(190, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

txtTypeDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(250, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(100, 'y'));

lblType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(45, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblPrice.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(105, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblDiscount.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(165, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

lblTypeDetails.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(225, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

btnAddType.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y'), Task.ScreenRatio(260, 'x'), Task.ScreenRatio(25, 'y'));

//Visible is set to false so it won't appear when Editor is first opened

scrollTypeCreate.setVisible(false);

}

//moves all components within panel typecreate up or down depinding on the value of change.

public static void DragUpDown(int change) {

for (Component c : pnlTypeCreate.getComponents()) {

c.setLocation(c.getX(), c.getY() + change);

}

pnlTypeCreate.setPreferredSize(new Dimension(pnlTypeCreate.getPreferredSize().width, pnlTypeCreate.getPreferredSize().height + change));

pnlTypeCreate.setMinimumSize(new Dimension(pnlTypeCreate.getPreferredSize().width, pnlTypeCreate.getPreferredSize().height + change));

pnlTypeCreate.setMaximumSize(new Dimension(pnlTypeCreate.getPreferredSize().width, pnlTypeCreate.getPreferredSize().height + change));

}

//handles TypeCreate data being added to the data base.

static class btnAddType\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

//try loop purposely placed to catch expected errors sent from functions determining if the right input has been put into the boxes.

try {

//if the Type name does not already exist in the database write the type data to the datbase and update affectedc omponents.

if (DataBaseControl.checkIfTypeClientExists("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLBOOKINGTYPE"," \* ", Editor.CheckIllegalCharacters(txtType.getText())) == true) {

String arguments = "('" + Editor.CheckIllegalCharacters(txtType.getText()) + "','" + Editor.CheckIllegalCharacters(txtTypeDetails.getText()) + "','" + Editor.DoubleCheck(txtPrice.getText()) + "','" + Editor.DoubleCheck(txtDiscount.getText()) + "')";

DataBaseControl.writeToDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLBOOKINGTYPE",arguments);

DataBaseControl.SelectionUpdate();

JOptionPane.showMessageDialog(null, "job type data was added to database");

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

View.pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio( 280, 'x'), Task.ScreenRatio(10 + (DataBaseControl.ScheduleArray.length) \* 100 + (DataBaseControl.ScheduleArray.length) \* 10, 'y')));

Notes.InsertData();

} else {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Client or Job Type already have that name!!!!!!!!!!");

}

} catch (NumberFormatException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

System.out.println("!!!!!!!!!!Use correct price and time format!!!!!!!!!!");

} catch (IllegalArgumentException exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

System.out.println("!!!!!!!!!!Cannot contain ~ or ` !!!!!!!!!!");

} catch (Exception exception) {

JOptionPane.showMessageDialog(null, "!!!!!!!!!!Invalid Input!!!!!!!!!!");

System.out.println("!!!!!!!!!!Invalid Input!!!!!!!!!!");

}

}

}

}

### 10.1.7 - Menu.java

//package declaration

package CalendarApp;

//imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Menu {

//Object Instantiation

// Java object container

static Container pane;

static JButton btnExit, btnSwitch1, btnSwitch2;

// Generic, lightweight container

static JPanel pnlMenu;

// JLabel object - able to display text / images

static JLabel lblDate;

Menu() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

// ~~ Preparing frame ~~

// Instantiates the frame

//Gets the content pane

pane = Task.frmMain.getContentPane();

// Applies null layout

pane.setLayout(null);

//creates the menu objects

btnExit = new JButton("Exit");

btnSwitch1 = new JButton("Editor");

btnSwitch2 = new JButton("View");

pnlMenu = new JPanel(null);

lblDate = new JLabel("Choose Date");

// Sets the border

pnlMenu.setBorder(BorderFactory.createTitledBorder("Menu"));

// Registers action listeners

btnExit.addActionListener(new btnExit\_Action());

btnSwitch1.addActionListener(new btnSwitch\_Action());

btnSwitch2.addActionListener(new btnSwitch\_Action());

// Adds controls to the pane

pane.add(pnlMenu);

pnlMenu.add(btnExit);

pnlMenu.add(btnSwitch1);

pnlMenu.add(btnSwitch2);

pnlMenu.add(lblDate);

//sets the bounds for the menu objects.

pnlMenu.setBounds(0, Task.ScreenRatio(1, 'y'), Task.ScreenRatio(320, 'x'), Task.ScreenRatio(50, 'y'));

btnExit.setBounds(Task.ScreenRatio(120, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(60, 'x'), Task.ScreenRatio(25, 'y'));

btnSwitch1.setBounds(Task.ScreenRatio(185, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(60, 'x'), Task.ScreenRatio(25, 'y'));

btnSwitch2.setBounds(Task.ScreenRatio(250, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(60, 'x'), Task.ScreenRatio(25, 'y'));

lblDate.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(200, 'x'), Task.ScreenRatio(25, 'y'));

}

//function to set the Datalabel to the current date using data from Calendar.

public static void ValueUpdate() {

lblDate.setText(Calendar.headers[Calendar.selectedcolumn] + ", " + Calendar.mtblCalendar.getValueAt(Calendar.selectedrow, Calendar.selectedcolumn) + " " + Calendar.lblMonth.getText().substring(0, 3) + " " + (String)Calendar.cmbYear.getItemAt(Calendar.cmbYear.getSelectedIndex()));

}

//handles exiting the program.

static class btnExit\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

System.exit(0);

}

}

static int nameSwitch = 0;

//handles switiching beteen the view, edit and main panels using two buttons.

static class btnSwitch\_Action implements ActionListener {

public void actionPerformed(ActionEvent e){

//switching to main or edditor

if ( ((((JButton)e.getSource()).getText()).equals("Main") && nameSwitch == 1 ) || ((((JButton)e.getSource()).getText()).equals("Editor") && nameSwitch == 0 ) ) {

Calendar.showCalendar = !(Calendar.showCalendar);

Calendar.pnlCalendar.setVisible(Calendar.showCalendar);

Notes.showNotes = !(Notes.showNotes);

Notes.pnlNotes.setVisible(Notes.showNotes);

Editor.showEditor = !(Editor.showEditor);

Editor.pnlEditor.setVisible(Editor.showEditor);

if (nameSwitch == 0) {

((JButton)e.getSource()).setText("Main");

nameSwitch = 1;

} else {

((JButton)e.getSource()).setText("Editor");

nameSwitch = 0;

}

}

//switching to main or view

else if ( ((((JButton)e.getSource()).getText()).equals("Main") && nameSwitch == 2 ) || ((((JButton)e.getSource()).getText()).equals("View") && nameSwitch == 0 ) ) {

Calendar.showCalendar = !(Calendar.showCalendar);

Calendar.pnlCalendar.setVisible(Calendar.showCalendar);

Notes.showNotes = !(Notes.showNotes);

Notes.pnlNotes.setVisible(Notes.showNotes);

View.showView = !(View.showView);

if (nameSwitch == 0) {

((JButton)e.getSource()).setText("Main");

nameSwitch = 2;

View.pnlView.setVisible(true);

} else {

((JButton)e.getSource()).setText("View");

nameSwitch = 0;

View.pnlView.setVisible(false);

}

}

//switching to view or edditor.

else if ( ((((JButton)e.getSource()).getText()).equals("View") && nameSwitch == 1 ) || ((((JButton)e.getSource()).getText()).equals("Editor") && nameSwitch == 2 ) ) {

Editor.showEditor = !(Editor.showEditor);

Editor.pnlEditor.setVisible(Editor.showEditor);

View.showView = !(View.showView);

if (nameSwitch == 1) {

((JButton)e.getSource()).setText("Editor");

nameSwitch = 2;

View.pnlView.setVisible(true);

} else {

((JButton)e.getSource()).setText("View");

nameSwitch = 1;

View.pnlView.setVisible(false);

}

}

}

}

}

### 10.1.8 - Notes.java

package declaration

package CalendarApp;

Imports

import javax.swing.;

import java.awt.;

import databasetut.;

import java.math.;

public class Notes {

object Instantiation

public static Container pane;

public static JPanel pnlNotes;

public static boolean showNotes = true;

public static JLabel lblIncome, lblHours, lblJobs;

variable initialisation

public static double income, hours, hour;

Notes() {

Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException InstantiationException IllegalAccessException UnsupportedLookAndFeelException e){}

sets up the pane

pane = Task.frmMain.getContentPane();

creates notes objects

pnlNotes = new JPanel(null);

lblIncome = new JLabel(Income );

lblHours = new JLabel(Hours );

lblJobs = new JLabel(Jobs );

adds components of notes to their relative containers.

pane.add(pnlNotes);

pnlNotes.add(lblIncome);

pnlNotes.add(lblHours);

pnlNotes.add(lblJobs);

Sets the border

pnlNotes.setBorder(BorderFactory.createTitledBorder(Notes));

pnlNotes.setBounds(0, Task.ScreenRatio(389, 'y'), Task.ScreenRatio(320, 'x'), Task.ScreenRatio(171, 'y'));

lblIncome.setBounds(10, Task.ScreenRatio(25, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(25, 'y'));

lblHours.setBounds(10, Task.ScreenRatio(70, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(25, 'y'));

lblJobs.setBounds(10, Task.ScreenRatio(115, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(25, 'y'));

}

function to be called to update the data inside notes.

public static void InsertData() {

if (DataBaseControl.ScheduleArray.length 0) {

income = 0;

hours = 0;

for (int i = 0; i DataBaseControl.ScheduleArray.length; i++) {

hour = hoursToDouble(DataBaseControl.ScheduleArray[i].aEND) - hoursToDouble(DataBaseControl.ScheduleArray[i].aSTART);

income += Double.parseDouble(DataBaseControl.TypeArray[i].bPrice) hour - (DataBaseControl.ClientArray[i].cVIPDouble.parseDouble(DataBaseControl.TypeArray[i].bVIPDiscount)0) hour;

hours += hour;

}

lblIncome.setText(Income + round(income, 2));

lblHours.setText(Hours + round(hours, 2));

lblJobs.setText(Jobs + (DataBaseControl.ScheduleArray.length));

} else {

lblIncome.setText(Income NA);

lblHours.setText(Hours NA);

lblJobs.setText(Jobs NA);

}

}

a function to check to convert time formate to double format.

public static double hoursToDouble(String time) {

return Double.parseDouble(time.substring(0, 2) + . + time.substring(3));

}

Method of rounded decimals taken from

httpstackoverflow.comquestions2808535round-a-double-to-2-decimal-places

public static double round(double value, int places) {

if (places 0) throw new IllegalArgumentException();

BigDecimal bd = new BigDecimal(value);

bd = bd.setScale(places, RoundingMode.HALF\_UP);

return bd.doubleValue();

}

}

### 10.1.9 View.java

//package declaration

package CalendarApp;

// Imports

import javax.swing.\*;

import java.awt.\*;

import databasetut.\*;

public class View {

//Object Instantiation and Variable Initialisation

static Container pane;

// Tabbel panel to place objects.

static public JPanel pnlView, pnlTimeView;

static public boolean showView = false;

public static DragScrollPane scrollView;

public static ViewItem[] ViewItemArray = new ViewItem[0];

//public ResultSet currentSet;

View() {

// Attempts to set up UI Manager

try {UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());}

// Catches ClassNotFound, Instantiation, IllegalAccess, and UnsupportedLookAndFeel Exceptions.

catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e){}

//Gets the content pane

pane = Task.frmMain.getContentPane();

// Applies null layout

pane.setLayout(null);

// Creates controls

pnlView = new JPanel(null);

pnlTimeView = new JPanel(null);

scrollView = new DragScrollPane(pnlTimeView);

// Sets the border

pnlView.setBorder(BorderFactory.createTitledBorder("View"));

// Registers action listeners

pane.add(pnlView);

pnlView.add(scrollView);

scrollView.setAutoscrolls(true);

//sets the bounds of the panels in view. The objects in views are created in the editor section and are done Dynamically.

scrollView.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(15, 'y'), Task.ScreenRatio(300, 'x'), Task.ScreenRatio(480, 'y'));

pnlTimeView.setPreferredSize(new Dimension(Task.ScreenRatio(280, 'x'), Task.ScreenRatio(470, 'y')));

pnlView.setBounds(0, Task.ScreenRatio(55, 'y'), Task.ScreenRatio(320, 'x'), Task.ScreenRatio(505, 'y'));

//sets the default visibility of the panel to be invisible.

pnlView.setVisible(false);

//ViewItem seenView = new ViewItem("10am","12am","Mow the lawn", "Ricordial", "E3", 0);

}

//function that checks a combobox for if a string is in it.

public static boolean checkBox(JComboBox Box, String test) {

for (int i = 0; i < Box.getItemCount(); i++) {

if (Box.getItemAt(i).equals(test)) {

return false;

}

}

return true;

}

//function that Creates the viewitems for View

public static void CreateViewItems() {

System.out.println("Creating Item!!!!!!");

for (int i = 0; i < DataBaseControl.ScheduleArray.length; i++) {

ViewItemArray = AddViewItem( ViewItemArray, new ViewItem(DataBaseControl.ScheduleArray[i].aSTART, DataBaseControl.ScheduleArray[i].aEND, DataBaseControl.ScheduleArray[i].aTYPE, DataBaseControl.ScheduleArray[i].aNAME, DataBaseControl.ScheduleArray[i].aDEST, i) );

ViewItemArray[i].addMouseListener(new ViewItem.ViewItemMouseListener());

}

}

//Function that adds each new vewItem to its local array.

private static ViewItem[] AddViewItem(ViewItem[] Array, ViewItem Element) {

ViewItem[] tempViewItem = new ViewItem[Array.length + 1];

for (int i = 0; i < Array.length; i++) {

tempViewItem[i] = Array[i];

}

Array = tempViewItem;

Array[Array.length - 1] = Element;

return Array;

}

}

### 10.1.10 - ViewItem.java

package declaration

package CalendarApp;

// Imports

import javax.swing.;

import java.awt.;

import java.awt.event.;

public class ViewItem extends JPanel {

Objext instantiation and variable initialisaiton.

private JLabel \_lblStartTime, \_lblEndTime, \_lblType, \_lblClient, \_lblLocation;

public int \_number;

public ViewItem(String startTime, String endTime, String type, String client, String location, int number) {

System.out.print(is object number + number);

uses the constructor variables to create components for the view object.

\_number = number;

\_lblStartTime = new JLabel(From + startTime);

\_lblType = new JLabel(Type + type);

\_lblClient = new JLabel(For + client);

\_lblLocation = new JLabel(At + location);

\_lblEndTime = new JLabel(To + endTime);

AddThis();

SetBounds();

}

funciton that sets the size of the ViewItem and of all its components.

private void SetBounds() {

this.setBounds(Task.ScreenRatio(10, 'x'), Task.ScreenRatio((10 + \_number 100 + \_number 10), 'y'), 280, 100);

\_lblStartTime.setPreferredSize( new Dimension(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(10, 'y')));

\_lblStartTime.setLocation(new Point(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(10, 'y')));

\_lblType.setPreferredSize( new Dimension(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(10, 'y')));

\_lblType.setLocation(new Point(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(25, 'y')));

\_lblClient.setPreferredSize( new Dimension(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(10, 'y')));

\_lblClient.setLocation(new Point(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(40, 'y')));

\_lblLocation.setPreferredSize( new Dimension(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(20, 'y')));

\_lblLocation.setLocation(new Point(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(55, 'y')));

\_lblEndTime.setPreferredSize( new Dimension(Task.ScreenRatio(260, 'x'), Task.ScreenRatio(10, 'y')));

\_lblEndTime.setLocation(new Point(Task.ScreenRatio(10, 'x'), Task.ScreenRatio(80, 'y')));

}

function that adds itself to the panel timeview and all relative components to itself

private void AddThis() {

this.setBorder(BorderFactory.createLineBorder(Color.black));

View.pnlTimeView.add(this);

this.add(\_lblStartTime);

this.add(\_lblType);

this.add(\_lblClient);

this.add(\_lblLocation);

this.add(\_lblEndTime);

}

handles the click events of the viewitems.

public static class ViewItemMouseListener implements MouseListener {

public void mouseClicked(MouseEvent event) {}

public void mouseEntered(MouseEvent event) {}

public void mouseExited(MouseEvent event) {}

public void mousePressed(MouseEvent event) {

Editor.ItemPressed(((ViewItem)event.getSource()).\_number);

System.out.println(Mouse Pressed);}

public void mouseReleased(MouseEvent event) {}

}

}

## 10.2 – databasetut.package

### 10.2.1 – DataBaseControl.java

// package declaration

package databasetut;

//imports

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import javax.swing.\*;

import CalendarApp.\*;

// Reads information from a database

public class DataBaseControl{

public static String[] stringSchedule = new String[2];

public static int[] intSchedule = new int[4];

public static String[] stringClient;

public static int[] intClient;

public static String[] stringType;

public static int[] intType;

public static int Amount = 0;

public static LocalClientItem[] ClientArray;

public static LocalScheduleItem[] ScheduleArray;

public static LocalTypeItem[] TypeArray;

// Reads from a database, taking the host, username, password, items to search for and table to find it in as inputs.

public static void readFromDataBase(String host, String username, String password,String table, String arguments) {

try {

// Connects to the database at address host, using the provided username and password

System.out.println("Connecting to " + host);

Connection con = DriverManager.getConnection(host,username,password);

System.out.println("Connected database successfully...");

// Creates a statement for the connected database

System.out.println("Creating statement...");

Statement stmt = con.createStatement();

// Creates a string of SQL code, using the data to be selected and the table for it to be selected from as parameters

String SQL = "SELECT " + arguments + " FROM " + table;

// Creates a resultset from reading the data in the table

System.out.println("Reading data...");

ResultSet rs = stmt.executeQuery(SQL);

// If connected table is TBLAPPOINTMENTS

if ("TBLAPPOINTMENTS".equals(table.toUpperCase())){

System.out.println("Opening table appointments");

for (int i = 0; i < View.ViewItemArray.length; i++) {

View.pnlTimeView.remove(View.ViewItemArray[i]);

}

View.ViewItemArray = new ViewItem[0];

ScheduleArray = new LocalScheduleItem[0];

Amount = 0;

// While there is unread information in rs

while(rs.next()){

Amount++;

String tempCheck = rs.getString("ADATE");

if (Calendar.selectedYear.equals(tempCheck)|| Calendar.selectedYear.substring(0,2).equals(tempCheck) || "ALL".equals(tempCheck) || Calendar.headers[Calendar.selectedcolumn].equals(tempCheck) || ((Calendar.selectedYear.substring(0, 2)+Calendar.selectedYear.substring(2, 4)).equals(tempCheck) ) ) {

ScheduleArray = AddScheduleData( ScheduleArray, new LocalScheduleItem(rs.getInt("AID"), rs.getString("ANAME"), rs.getString("ATYPE"), rs.getString("ASTART"), rs.getString("AEND"), rs.getString("ADESC"), rs.getString("ADEST") ) );

}

}

// System.out.print(ScheduleArray[0].aNAME + " " + ScheduleArray[0].aTYPE + " " + ScheduleArray[0].aSTART + " " + ScheduleArray[0].aEND + " " + ScheduleArray[0].aDESC + " " + ScheduleArray[0].aDEST + " ");

if (ScheduleArray.length != 0) {

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLBOOKINGTYPE", " \* ");

}

}

// If connected table is TBLBOOKINGTYPE

if ("TBLBOOKINGTYPE".equals(table.toUpperCase())){

System.out.println("Table: type");

TypeArray = new LocalTypeItem[ScheduleArray.length];

// While there is unread information in rs

while(rs.next()){

String tempCheck;

for (int i = 0; i < ScheduleArray.length; i++) {

if (Editor.CheckIllegalCharacters(ScheduleArray[i].aTYPE).equals(tempCheck = rs.getString("BTYPE") ) ) {

TypeArray[i] = new LocalTypeItem(tempCheck, rs.getString("BPRICE"), rs.getString("BVIPDISCOUNT"), rs.getString("BDESC"));

}

}

}

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLCLIENTS"," \* ");

}

// If connected table is TBLCLIENTS

if ("TBLCLIENTS".equals(table.toUpperCase())){

ClientArray = new LocalClientItem[ScheduleArray.length];

System.out.println("Table: clients");

// While there is unread information in rs

while(rs.next()){

String tempCheck;

for (int i = 0; i < ScheduleArray.length; i++) {

if (Editor.CheckIllegalCharacters(ScheduleArray[i].aNAME).equals( tempCheck = rs.getString("CNAME") ) ) {

ClientArray[i] = new LocalClientItem(tempCheck, rs.getString("CADDRESS"), rs.getString("CNUMBER"), rs.getString("CVIP"), rs.getString("CDESC") );

}

}

}

// If the username is Ricardo

if (username.equals("Ricardo")){

// String Ricardo = "love"; Ricardo = "life...";

System.out.println("Ricardo is love, Ricardo is life...");

}

DeleteCheck();

View.CreateViewItems();

}

}

// Catches exceptions thrown by SQL Errors

catch (SQLException se) {

// Prints the error message of the error thrown

System.out.print("Task failed: ");

System.out.println(se.getMessage());

}

// Always do this after the try... catch loop

finally {

// Print 'Goodbye!' to the console to signify the closing of the database

System.out.println("Ending Table Connection...");

}

}

public static void PrintArrays() {

for (int i = 0; i < ScheduleArray.length; i++) {

System.out.println("AID: " + ScheduleArray[i].AID + "; TYPE: " + ScheduleArray[i].aTYPE + "; NAME: " + ScheduleArray[i].aNAME + "; START: " + ScheduleArray[i].aSTART + "; END: " + ScheduleArray[i].aEND + "; DEST: " + ScheduleArray[i].aDEST + "; DESC: " + ScheduleArray[i].aDESC);

}

}

public static void DeleteCheck() {

LocalScheduleItem[] tempSchedule;

for (int i = 0; i < ScheduleArray.length; i++) {

if (TypeArray[i] == null || ClientArray[i] == null) {

//DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLAPPOINTMENTS"," \* ", ScheduleArray[i].AID + "");

tempSchedule = new LocalScheduleItem[ScheduleArray.length - 1];

for (int j = 0; j < i; j++) {

tempSchedule[j] = ScheduleArray[j];

}

for (int j = i; j < tempSchedule.length; j++) {

tempSchedule[j] = ScheduleArray[j+1];

tempSchedule[j].AID = j;

}

i--;

ScheduleArray = tempSchedule;

}

}

System.out.print(ScheduleArray.length);

}

public static void SelectionUpdate() {

DataBaseControl.SelectionUpdateBack("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLCLIENTS"," \* ");

}

// Reads from a database, taking the host, username, password, items to search for and table to find it in as inputs.

private static void SelectionUpdateBack(String host, String username, String password,String table, String arguments) {

try {

// Connects to the database at address host, using the provided username and password

System.out.println("Connecting to " + host);

Connection con = DriverManager.getConnection(host,username,password);

System.out.println("Connected database successfully...");

// Creates a statement for the connected database

System.out.println("Creating statement...");

Statement stmt = con.createStatement();

// Creates a string of SQL code, using the data to be selected and the table for it to be selected from as parameters

String SQL = "SELECT " + arguments + " FROM " + table;

// Creates a resultset from reading the data in the table

System.out.println("Reading data...");

ResultSet rs = stmt.executeQuery(SQL);

// If connected table is TBLCLIENTS

if ("TBLCLIENTS".equals(table.toUpperCase())){

System.out.println("Table: clients");

Editor\_Schedule.cmbClientLoad.removeAllItems();

// While there is unread information in rs

String tempCheck;

while(rs.next()){

if (View.checkBox(Editor\_Schedule.cmbClientLoad, (tempCheck = rs.getString("CNAME")))) {

Editor\_Schedule.cmbClientLoad.addItem(Editor.UnCheckIllegalCharacters(tempCheck));

}

}

DataBaseControl.SelectionUpdateBack("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLBOOKINGTYPE"," \* ");

}

// If connected table is TBLCLIENTS

if ("TBLBOOKINGTYPE".equals(table.toUpperCase())){

System.out.println("Table: Type");

Editor\_Schedule.cmbTypeLoad.removeAllItems();

// While there is unread information in rs

String tempCheck;

while(rs.next()){

if (View.checkBox(Editor\_Schedule.cmbTypeLoad, (tempCheck = rs.getString("BTYPE")))) {

Editor\_Schedule.cmbTypeLoad.addItem(Editor.UnCheckIllegalCharacters(tempCheck));

}

}

}

}

// Catches exceptions thrown by SQL Errors

catch (SQLException se) {

// Prints the error message of the error thrown

System.out.print("Task failed: ");

System.out.println(se.getMessage());

}

// Always do this after the try... catch loop

finally {

// Print 'Goodbye!' to the console to signify the closing of the database

System.out.println("Ending Table Connection...");

}

}

// Deletes data from the selected table

public static void deleteFromDataBase(String host, String username, String password, String table, String arguments, String check){

// Attempt this

try {

// Connects to the database

System.out.println("Connecting to " + host );

Connection conn = DriverManager.getConnection(host,username,password);

System.out.println("Connected to database successfully.");

// creates the java jdbc statement

System.out.println("Creating statement...");

//Statement statement = conn.createStatement();

Statement statement = conn.createStatement (ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_UPDATABLE);

String SQL = "SELECT " + arguments + " FROM " + table;

// Creates a resultset from reading the data in the table

System.out.println("Reading data...");

ResultSet rs = statement.executeQuery(SQL);

System.out.println("1");

if ("TBLAPPOINTMENTS".equals(table.toUpperCase())){

System.out.println("2");

System.out.println(rs.getRow());

System.out.println("3");

boolean deleted = false;

boolean messageHasFired = false;

// While there is unread information in rs

while(rs.next()){

System.out.println("3");

if (check.equals(rs.getInt("AID") + "")) {

deleted = true;

rs.deleteRow();

rs.next();

}

if (deleted == false) {

rs.updateInt("AID", rs.getRow() - 1);

rs.updateRow();

} else {

rs.updateInt("AID", rs.getRow() - 2);

rs.updateRow();

}

if ((check.equals(rs.getString("ATYPE") + "") || check.equals(rs.getString("ANAME") + "")) && messageHasFired == false) {

messageHasFired = true;

JOptionPane.showMessageDialog(null, "A job exists with that type of job, please recreate this type of job with the exact name or those jobs will be deleted");

}

if (check.equals(rs.getString("ANAME") + "")) {

JOptionPane.showMessageDialog(null, "A job exists with that client, please recreate this client with the exact name or those jobs will be deleted");

}

System.out.print("FUDGE IT ALL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

}

DataBaseControl.readFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLAPPOINTMENTS"," \* ");

}

if ("TBLBOOKINGTYPE".equals(table.toUpperCase())){

// While there is unread information in rs

while(rs.next()){

if (check.equals(rs.getString("BTYPE"))) {

rs.deleteRow();

}

}

DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLAPPOINTMENTS"," \* ", check);

}

if ("TBLCLIENTS".equals(table.toUpperCase())){

// While there is unread information in rs

while(rs.next()){

if (check.equals(rs.getString("CNAME"))) {

rs.deleteRow();

}

}

DataBaseControl.deleteFromDataBase("jdbc:derby://localhost:1527/CALENDARDATA", "superadmin", "superadmin", "TBLAPPOINTMENTS"," \* ", check);

}

}

// Catches any SQL Exceptions thrown

catch (SQLException e) {

// Prints the errormessage of e

System.out.print("Task failed: ");

System.out.println(e.getMessage());

}

// Always runs after the try/catch

finally {

// Prints 'goodbye' to the console to signify the ending of the function

System.out.println("Goodbye!");

}

}

// Class for writing a new object to the database

public static void writeToDataBase(String host, String username, String password, String table, String arguments){

// Attempt this

try{

// Connects to the database specified by host, using the username and password combination provided as arguments

System.out.println("Connecting to " + host );

try (Connection conn = DriverManager.getConnection(host,username,password)) {

System.out.println("Connected to database successfully.");

// create our java jdbc statement

System.out.println("Creating statement...");

Statement statement = conn.createStatement();

/\* String arguments is structured in the following patterns -

tblAppointments - (aID, cID, 'aBookingType', aDate, aStart, aEnd, 'aDesc')

tblBookingType - ('bType', 'bDesc', bPrice)

tblClients - (cID, 'cName', 'cAddress', cNumber, cVip)

\*/

// Adds the new data to the selected table

System.out.println("Adding data to table...");

statement.executeUpdate("INSERT INTO " + table + " VALUES" + arguments);

System.out.println("Successfully added data to table.");

// Close the connection to the database

System.out.println("Closing database connection...");

}

}

// if an SQL Exception occurs

catch (SQLException se) {

// Print the error message to the console

System.out.print("Task failed: ");

System.out.println(se.getMessage());

}

// Runs after the try/catch loop

finally {

// Prints 'goodbye' to the console to signify the end of the function

System.out.println("Goodbye!");

}

}

// Reads from a database, taking the host, username, password, items to search for and table to find it in as inputs.

static private boolean exists = true;

public static boolean checkIfTypeClientExists(String host, String username, String password,String table, String arguments, String check) {

try {

// Connects to the database at address host, using the provided username and password

System.out.println("Connecting to " + host);

Connection con = DriverManager.getConnection(host,username,password);

System.out.println("Connected database successfully...");

// Creates a statement for the connected database

System.out.println("Creating statement...");

Statement stmt = con.createStatement();

// Creates a string of SQL code, using the data to be selected and the table for it to be selected from as parameters

String SQL = "SELECT " + arguments + " FROM " + table;

// Creates a resultset from reading the data in the table

System.out.println("Reading data...");

ResultSet rs = stmt.executeQuery(SQL);

if ("TBLBOOKINGTYPE".equals(table.toUpperCase())){

// While there is unread information in rs

while(rs.next()){

if (check.equals(rs.getString("BTYPE"))) {

exists = false;

}

}

return DataBaseControl.checkIfTypeClientExists("jdbc:derby://localhost:1527/CALENDARDATA","superadmin","superadmin","TBLCLIENTS"," \* ", check);

}

if ("TBLCLIENTS".equals(table.toUpperCase())){

// While there is unread information in rs

while(rs.next()){

if (check.equals(rs.getString("CNAME"))) {

exists = false;

}

}

boolean swap = exists;

System.out.println("BOOLEAN SWAP: "+ swap + "errrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr");

exists = true;

return swap;

}

return false;

}

// if an SQL Exception occurs

catch (SQLException se) {

// Print the error message to the console

System.out.print("Task failed: ");

System.out.println(se.getMessage());

return false;

}

// Runs after the try/catch loop

finally {

// Prints 'goodbye' to the console to signify the end of the function

System.out.println("Goodbye!");

}

}

private static LocalScheduleItem[] AddScheduleData(LocalScheduleItem[] Array, LocalScheduleItem Element) {

LocalScheduleItem[] tempSchedule = new LocalScheduleItem[Array.length + 1];

for (int i = 0; i < Array.length; i++) {

tempSchedule[i] = Array[i];

}

Array = tempSchedule;

Array[Array.length - 1] = Element;

return Array;

}

}

### 10.2.2- LocalClientitem.java

//package declaration

package databasetut;

//Imports

import CalendarApp.\*;

//an object to represent a row of database data.

public class LocalClientItem {

//Variable Initialization.

public String CNAME, cADDRESS, cDESC, cNUMBER;

public boolean cVIP;

public LocalClientItem(String \_cNAME, String \_cADDRESS, String \_cNUMBER, String \_cVIP, String \_cDESC) {

//sets the constructor values to public and unreads them from database format.

CNAME = Editor.UnCheckIllegalCharacters(\_cNAME);

cADDRESS = Editor.UnCheckIllegalCharacters(\_cADDRESS);

cNUMBER = Editor.UnCheckIllegalCharacters(\_cNUMBER);

cVIP = \_cVIP.equals("true");

cDESC = Editor.UnCheckIllegalCharacters(\_cDESC);

}

}

### 10.2.3 – LocalScheduleItem.java

//package declaration

package databasetut;

//Imports

import CalendarApp.Editor;

//an object to represent a row of database data.

public class LocalScheduleItem {

//variable instantiation

public int AID;

public String aNAME, aTYPE, aDESC, aDEST, aSTART, aEND;

public LocalScheduleItem(int \_AID, String \_aNAME, String \_aTYPE, String \_aSTART, String \_aEND, String \_aDESC, String \_aDEST) {

//sets the constructor values to public and unreads them from database format.

AID = \_AID;

aNAME = Editor.UnCheckIllegalCharacters(\_aNAME);

aSTART = \_aSTART;

aEND = \_aEND;

aTYPE = Editor.UnCheckIllegalCharacters(\_aTYPE);

aDESC = Editor.UnCheckIllegalCharacters(\_aDESC);

aDEST = Editor.UnCheckIllegalCharacters(\_aDEST);

}

}

### 10.2.4 – LocalTypeItem.java

//package declaration

package databasetut;

//imports

import CalendarApp.\*;

//an object to represent a row of database data.

public class LocalTypeItem {

//variabel instantiation

public String bType, bDesc, bPrice, bVIPDiscount;

public LocalTypeItem(String \_bType, String \_bPrice, String \_bVIPDiscount, String \_bDesc) {

//sets the constructor values to public and unreads them from database format.

bType = Editor.UnCheckIllegalCharacters(\_bType);

bPrice = \_bPrice;

bVIPDiscount = \_bVIPDiscount;

bDesc = Editor.UnCheckIllegalCharacters(\_bDesc);

}

}