# *Programming II (420-B20-HR)*

# *Lab 9 –GUI Design*

Date assigned: Wednesday, March 23, 2016

Date due: **Wednesday, March 23, 2016**

**Learning Objectives**

At the end of this lab, the student will be able to:

1. use the Calendar and DateFormat classes to represent dates;
2. use a menu bar with menus and menu items to select actions.
3. use the **dispose()** method to close a frame
4. use a **JOptionPane** for data input
5. arrange the components of a container using the GridLayout layout manager
6. arrange the components of a container using the BoxLayout layout manager
7. add a titled border to a component
8. add a group of radio buttons to a component
9. reverse generate a class diagram from Java code in Visual Paradigm
10. create a class diagram using Visual Paradigm

**Commands, Methods and Structures Used:**

**Create and add a menu bar:**

**JMenuBar** *menuBarName* **= new JMenuBar();**

**this.setJMenuBar(***menuBarName* **);**

**Create menus and add to the menu bar:**

**JMenu** *menuName* **= new JMenu(***"menu label"*);

*menuBarName***.add(***menuName***);**

**Create menu items and add to a menu:**

**JMenuItem** *menuItemName* **= new JMenuItem("***menu item label***");**

*menuName***.add(***menuItemName***);**

**Close a frame:**

**dispose();**

**Add a component to a top-level Swing container:**

**getContentPane().add(***component***);**

**Add a component to a top-level Swing container using the BorderLayout manager:**

**getContentPane().add(***component***, "***area***");**

**Set the font for the text in a component:**

*component.***setFont(**"*name*",**Font.***style*,*fontsize*);

**Centre a label:**

*label.***setHorizontalAlignment(CENTER);**

**Right-align a label:**

*label.***setHorizontalAlignment(RIGHT);**

**Set the layout manager of a container to GridLayout:**

*container***.setLayout(new GridLayout(***numRows***,***numColumns***));**

**Set the layout manager of a container to BoxLayout:**

*container***.setLayout(**

**new BoxLayout(***container***,BoxLayout.Y\_AXIS));**

*container***.setLayout(new BoxLayout(***container***,BoxLayout.X\_AXIS));**

**Add a titled border to a component:**

*component***.setBorder(**

**BorderFactory.createTitledBorder("***title***"));**

**To Be Handed In:**

1. The ***username*\_B20\_L09\_Project** folder containing the Java source files should be uploaded to **Moodle**.

**To Start:**

1. Download and unzip the folder **B20\_L09\_Project** from **Moodle**. Rename it to ***username*\_B20\_L09\_Project**.
2. Start **Eclipse**.
3. Create a new Java Project called ***username*\_B20\_L09\_Project**.

# The Calendar Class

***Purpose:*** Learn to represent and manipulate dates and times using the Java **GregorianCalendar** class.

Learn to display the date or time in different formats using the **DateFormat** class.

***To Do:***

The **GregorianCalendar** Class

The **GregorianCalendar** class is a subclass of the **Calendar** class and represents a date and time using the Gregorian calendar (the calendar we use).

The **GregorianCalendar()** constructors

**GregorianCalendar():** instantiates an object representing the current date and time

**GregorianCalendar(int** *year***, int** *month***, int** *dayOfMonth***):** instantiates an object representing dayOfMonth/month/year. The months start numbering at 0. (e.g. GregorianCalendar(2009,2,3) instantiates an object representing Mar. 3, 2009.)

The **getTime()** method

* returns the complete date and time

The **add(int** *field***, int** *amount***)** method

* adds the specified amount to the specified field of the date. If amount is negative, it subtracts. The *field* parameter may be:

**Calendar.DATE** – add/subtract days

**Calendar.MONTH** – add/subtract months

**Calendar.YEAR** – add/subtract years

**Calendar.HOUR** – add/subtract hours

**Calendar.MINUTE** – add/subtract minutes

**Calendar.SECOND** – add/subtract seconds

## Create a class called **DateConverter** in the **b20\_l09\_project** package. It should have a **main()** method.

## We are going to use the **GregorianCalendar** class to represent a calendar date. It is a subclass of the **Calendar** class. Import **java.util.GregorianCalendar**.

## Instantiate a **GregorianCalendar** object in the **main()** method of the **DateConverter** class:

GregorianCalendar aDate = new GregorianCalendar();

## The **getTime()** method of the **Calendar** class returns a **Date** object. Use the **getTime()** method to output today's date and time from the **aDate** object. Run the **DateConverter** program.

The **DateFormat** Class

The **DateFormat** class can be used to format a **Date** object. Like the **NumberFormat** class it is an abstract class and cannot be instantiated using the **new** operator.

The **getDateInstance(int** *style* **)** method

The **getDateInstance()** method is used to create an instance of the **DateFormat** class. The **getDateInstance()** method has a single parameter indicating the output date style. There are 4 styles that can be used to format a date:

**DateFormat.SHORT**

**DateFormat.MEDIUM**

**DateFormat.LONG**

**DateFormat.FULL**

The **getTimeInstance(int** *style* **)** method

The **getTimeInstance()** method is used to create an instance of the **DateFormat** class. The **getTimeInstance()** method has a single parameter indicating the output time style. There are 4 styles that can be used to format a time:

**DateFormat.SHORT** ( h:mm [AM|PM] )

**DateFormat.MEDIUM** ( h:mm:ss [AM|PM] )

**DateFormat.LONG** ( h:mm:ss [AM|PM] *timezone*)

**DateFormat.FULL (** h:mm:ss o'clock[AM|PM] *timezone*)

The **format(Date** *date***)** method:

The **format()** method returns a String formatted in the style set in the **getDateInstance()** or **getTimeInstance()** methods.

## Import **java.text.DateFormat** and instantiate four instances of the **DateFormat** class – one using each of the styles.

## Display the value of **aDate.getTime()** using each of the **DateFormat** styles.

## Import **java.util.Calendar** and use the **Calendar add()** method to calculate the date 1 week from today (i.e. add 7 days to **aDate**). Use the **add()** method described on the previous page. Display the result using the **Medium DateFormat** style. Test your changes.

## Create a second **GregorianCalendar** object called **myBirthday**. Use the **GregorianCalendar(int** *year***, int** *month***, int** *dayOfMonth***)** constructor to instantiate it. Display your birthday using the **Full DateFormat** style to get the day of week you were born on. Test your changes. What do you notice about the month?

## Use the **add()** method to calculate the date of your 21st birthday. Display the result using the **Full DateFormat** style. Test your changes.

# Menus and MenuItems

***Purpose:*** Learn to implement menus and menu items in a JFrame.

***To Do:***

## We want to use a menu instead of buttons to select the reports to display in our Video Store Case Study. In the **InventoryReportsFrame** class, add the following components to create the components for a **menuBar** with a **File** menu which has an **Exit** menu item:

**private JMenuBar menuBar = new JMenuBar();**

**private JMenu fileMenu = new JMenu("File");**

**private JMenuItem exitMenuItem = new JMenuItem("Exit");**

## Now modify the **initialize()** method to add the menu bar to the frame, the menu to the menu bar and the menu item to the menu:

**this.setJMenuBar(menuBar);**

**menuBar.add(fileMenu);**

**fileMenu.add(exitMenuItem);**

## Add an actionListener for the exit menu item:

**exitMenuItem.addActionListener(this);**

## Handle the exit menu item in the **actionPerformed()** method. It should close the window, but not necessarily the application:

**if (e.getSource() == exitMenuItem)**

**this.dispose();**

## To see how this works, run the application. List a report and use the **File -> Exit** to exit.

## On your own:

### add a new menu called "Reports"

### add menu items to the Reports menu for "All Products", "Movies", "Games" and "Find Product".

### add actionListeners for each of the Reports menu items.

### modify the **actionPerformed()** method to call the appropriate **actionPerformed()** method for each of the menu items instead of for the buttons and text field.

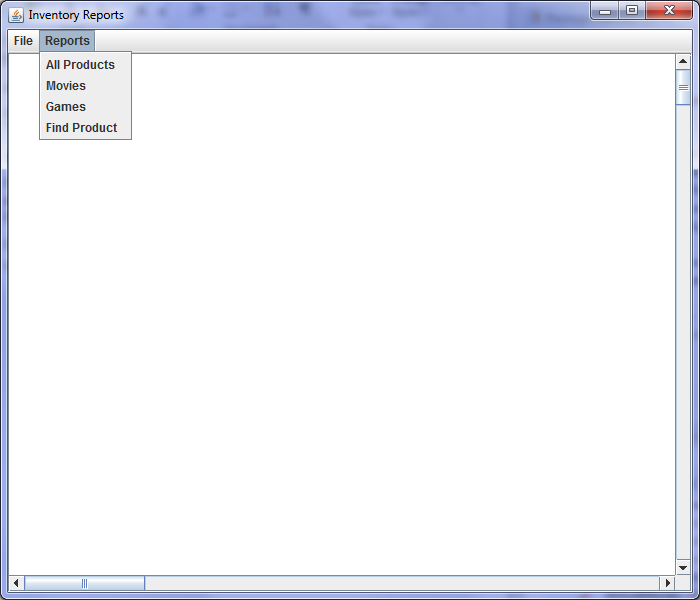
### comment out the code for the buttons and product number entry.

### replace the setBounds() calls for **areaDisplay** and **scrollPane** with the following:

**areaDisplay.setLocation(0, 0);**

**areaDisplay.setSize(getSize().width-15, getSize().height-60);**

**scrollPane.setLocation(0, 0);**

**scrollPane.setSize(getSize().width-15, getSize().height-60);**

Your completed frame should look like:

## Test the all products, movies and games reports. They should be the same as they were using the buttons. The find product menu isn't working yet, because there's no place to enter the product number we're looking for. Instead of using a text field, we're going to use an input dialog with a JOptionPane. Add the following at the beginning of the **fldProductNumber\_actionPerformed()** method:

String prodNumber = JOptionPane.*showInputDialog*(this,

"Enter product code:", "Find Product",

JOptionPane.*QUESTION\_MESSAGE*);

## Change fldProductNumber.getText() to prodNumber everywhere in the fldProductNumber\_actionPerformed() method.

## Test your changes.

# Create an About box

***Purpose:*** Code an About box to credit yourself for the program.

***To Do:***

## Create a new Swing **JPanel** using **WindowBuilder**. Call the panel **VideoStore\_AboutPanel**.

## Click the **Design** tab. Select the panel in the drawing pane. Click the **…** beside **border** in the **Properties** pane. Select **EtchedBorder** from the **Border type** drop-down box.

## Select **GridBagLayout** from the **Layouts** tool bar and click the panel in the drawing pane.

## Select **JLabel** from the Components toolbar. Move the mouse to the panel. A grid with red lines will appear. Position the mouse in column 0, row 0 and click. Type "Video Store System" as the text in the label. Change the name of the label to **lblTitle**.

Add the following labels in the same way:

|  |  |  |
| --- | --- | --- |
| **Label name** | **Position** | **Text** |
| lblAuthor | column 0, row 1 | *your name* |
| lblCopyright | column 0, row 2 | 2016 |
| lblCompany | column 0, row 3 | Heritage College |

## Click one of the labels. Click the 🡪🡨 key and select **Grow**. All the labels will expand to the full width of the panel.

## Resize the panel to an appropriate size for an about box.

## In the **VideoStoreFrame**, add a **Menu** object called **helpMenu** and a **MenuItem** object called **aboutMenuItem**. They should have appropriate labels. Add the menu and menu item to the menu bar after **rentalMenu**.

## Add an actionListener to **aboutMenuItem.**

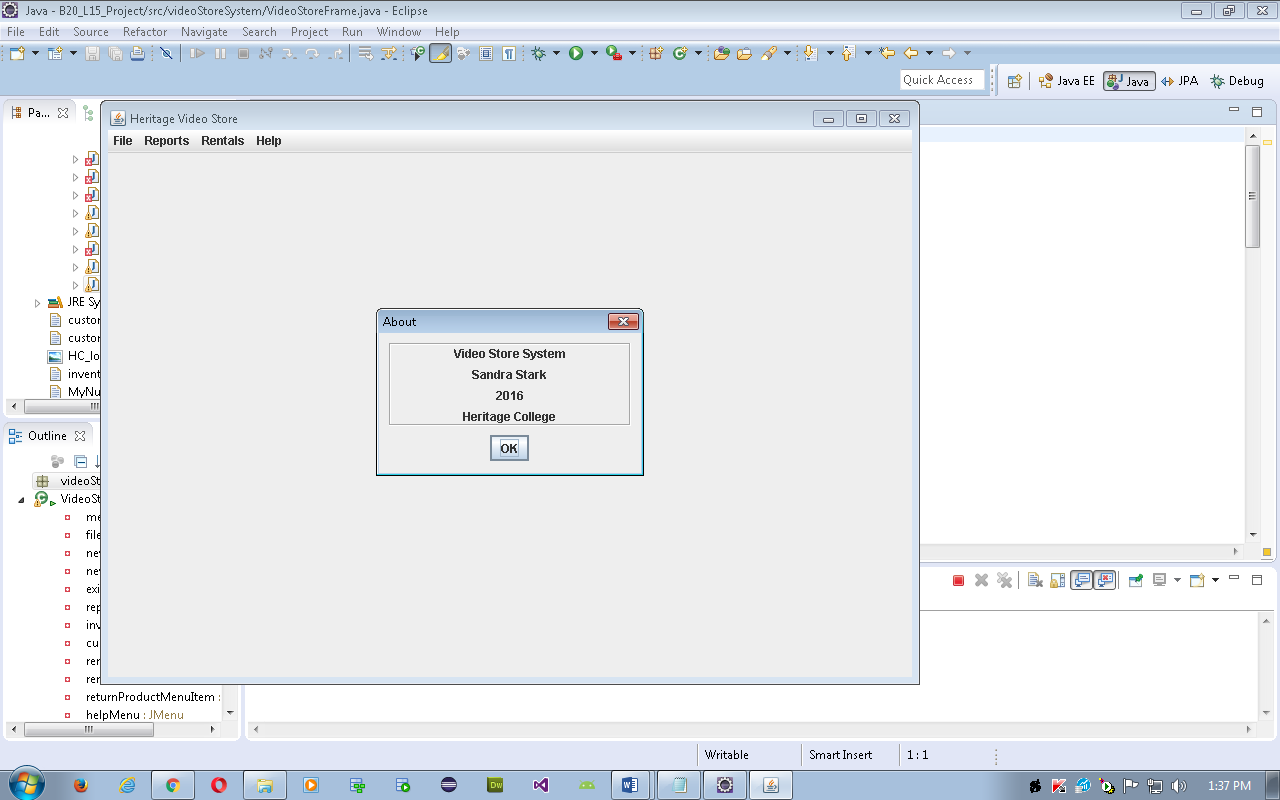
## Go to the **actionPerformed()** method for **aboutMenuItem** and add the following:

JOptionPane.showMessageDialog(**this**,

new VideoStore\_AboutPanel(), "About",

JOptionPane.PLAIN\_MESSAGE);

## Run **VideoStoreFrame** and test your **About** box. It should look similar to:



# Java Panels and Layout Managers

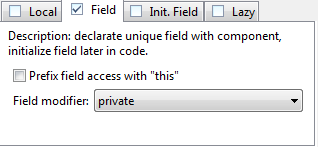
***Purpose:*** Learn how the **BorderLayout**, **FlowLayout** and **BoxLayout** managers arrange components on a frame.

***To Do:***

## Set WindowBuilder to add the components as instance variables and to implement the listeners in the parent class:

Window 🡪 Preferences 🡪 Swing

### 🡪 Code Generation 🡪 Click the **Field** tab:



### 🡪 Code Generation 🡪 Event Handlers 🡪 Check "Implement listener interface in parent class"

## Now, we are going to create the frame we designed in class:

**Heritage Movie and Game Rentals**

***Heritage College Video Store***

***Rental Receipt***

**Date**: dd-mon-yyyy

**Customer Number:** 999-9999

**Customer Name:** xxx--xxx

**Customer Phone:** 999-999-9999

**Media Product Title Return Fee**

**Type Number Date**

**-------------------------------------------**

xxxxx 9999 xxx—-xxx dd-mon-yyyy $99.99

xxxxx 9999 xxx—-xxx dd-mon-yyyy $99.99

**Subtotal: $999.99**

**HST: $999.99**

**Total owing: $999.99**

**Paid by xxxxxxxxxxx**

**Customer number:**

**TOTAL**

**PRINT**

**Product number:**

Cash

Credit Card

Debit Card

7 8 9

4 5 6

1 2 3

Movie 0 Game

Clear Enter

### use WindowBuilder to create the **RentalFrame** class in the **videoStoreSystem** package

### set the title to "Product Rental "

### click on the "Go to Definition" icon[[1]](#footnote-1) and set the bounds as follows:

setBounds(100, 100, 500, 450);

## Add the following components in an arrangement similar to the design on the previous page.

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Swing Component** | **Instance variable name** | **Properties** |
| form title | JLabel | lblTitle | **text**: "Heritage College Video Store"  **font**: Tahoma 18 Bold  **horizontalAlignment**: CENTER |
| customer number prompt | JLabel | lblCustomerNumber | **text**: "Customer Number:"  **horizontalAlignment**: RIGHT |
| customer number entry area | JTextField | fldCustomerNumber | **columns**: 10 |
| product number prompt | JLabel | lblProductNumber | **text**: "Product Number:"  **horizontalAlignment**: RIGHT |
| product number entry area | JTextField | fldProductNumber | **columns**: 10 |
| the keypad | for now we'll use a JTextArea | areaKeypad | **bounds**: (10, 106, 222, 122) |
| the total button | JButton | btnTotal | **text**: "Total" |
| the print button | JButton | btnPrint | **text**: "Print" |
| the rental receipt | JTextArea | areaReceipt | **editable**: false |
| payment options | for now we'll use a JTextArea | areaPaymentOptions | **bounds**: (49, 287, 124, 106) |

## Run **RentalFrame** to see your frame. Make any necessary adjustments to ensure that all words in the labels are visible.

# Using GridLayout

***Purpose:*** Learn to use the GridLayout layout manager and an array of buttons to create a calculator keypad.

***To Do:***

We want to add a calculator keypad to the frame to allow the user to enter the product code using a mouse. We will use an array of buttons for the keypad keys and use GridLayout to arrange the keys in a panel.

## Change **areaKeypad** to a **JPanel** called **panelKeypad**.

## Create an array of **String** objects called **keyLabel** initialized to the values on the keypad above. Refer to the diagram for the order of the array elements.

## Create an array of **JButtons** called **btnKey**. The size of the array should be the length of the **keyLabel** array.

## Add a private method called **initializeKeypad()**. It should contain a **for** loop that loops **keyLabel.length** times. In the loop you should:

### instantiate each element of the **btnKey** array to a **JButton** with the corresponding **keyLabel**;

### add each element of the **btnKey** array to **panelKeypad**.

## To make the blank key invisible, add the following after the end of the loop:

**btnKey[btnKey.length-2].setVisible(false);**

## Call the **initializeKeypad()** method from the constructor. Test your changes.

The keypad panel is displayed in **FlowLayout** not the tidy table we envisioned. We want to be organized in a 5x3 grid.

## At the beginning of the **initializeKeypad()** method, set the layout for the keypad:

**panelKeypad.setLayout(new GridLayout(5,3));**

## Test your changes.

# Radio Buttons

***Purpose:*** Learn to add radio buttons to a frame.

***To Do:***

We want to add radio buttons for the payment type to the frame.

## Change **areaPaymentOptions** to a **JPanel** called **panelPayment**.

## Create a **String** array called **paymentLabel** that contains: "*Cash*", "*Credit Card*", "*Debit Card*".

## Create an array of **JRadioButton**s called **rdbtnPayment** with a size equal to the length of the **paymentLabel** array.

## Add a private method called **initializePaymentButtons()**. For each **rdbtnPayment** radio button you should:

### instantiate it and set the label to the corresponding element of the **paymentLabel** array

### add it to **panelPayment**.

## Call the **initializePaymentButtons()** method from the constructor.

## Run **RentalFrame**. Try clicking the different radio buttons. What happens?

In order to make the radio buttons operate mutually exclusively, they must be grouped in a **ButtonGroup**. (i.e. When you click on one button, the one that was formerly clicked is unclicked.)

## Create and instantiate a private **ButtonGroup** object called **paymentGroup**.

## In the **initializePaymentButtons()** method, use the **add()** method to add each element of the **rdbtnPayment** array to **paymentGroup**.

## Test your changes.

## The panel uses FlowLayout. We want the labels to line up on the left hand margin. To do this, we are going to use BoxLayout. Add the following statement to your constructor:

**panelPayment.setLayout(new BoxLayout(panelPayment, BoxLayout.Y\_AXIS));**

# Labelled Borders

***Purpose:*** Learn to add labeled borders to a component.

***To Do:***

We want to add a border titled with "*Payment Options*" to the 3 radio buttons and a border titled "*Rental Receipt*" to the Rental Receipt.

## Use the **setBorder()** method to add a labelled border to **panelPayment** :

**panelPayment.setBorder(**

**BorderFactory.createTitledBorder("Payment Options"));**

## Test your changes.

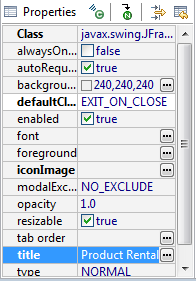
## Add a labelled border titled "*Rental Receipt*"to the **areaReceipt** text area. Test your changes.

## Open **VideoStoreFrame.java** and modify the **actionPerformed ()** method to instantiate and make visible the **RentalFrame** if the *rent products* option is chosen. Test your changes. You should now be able to add a new customer, add a new product, list all the products, all movies, all games, display a single product and rent products (not fully implemented) from the **VideoStoreFrame**.

# Homework

## Complete the **Week 9 Quiz** on Moodle by Mar. 27.

1. The Go to Definition icon is the second symbol with the circled N:

    [↑](#footnote-ref-1)