201241037, t.coupe@student.liverpool.ac.uk, comp 122

Assignment 04 – Comp 122

Part 1 – Sweet Shop GUI

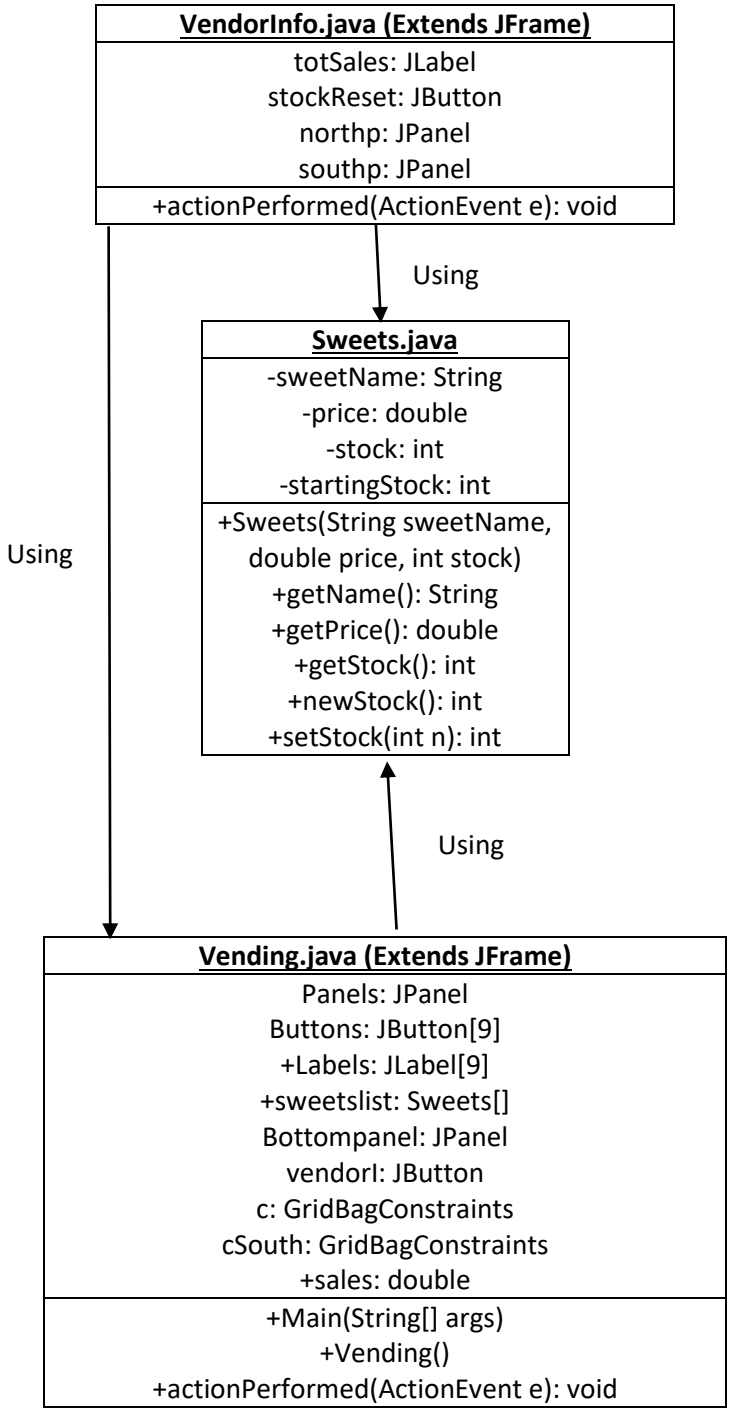
Summary of requirements

For the first part of this assignment we have been asked to create a shop GUI that allows users to select from 9 items. Each item has its own price and also its own stock value, each of the items is also represented by a button that if clicked means the user has bought an item. The GUI also had to have another additional button called "Vendor Information" which allows me to view the total sales of the items bought and also it gives me an option to reset all of the stock values back to full stock and also resetting the total sales count back to 0. Each time an item is bought I need update how much stock is left in the labels below each button to tell the user how many of each product they can buy.

Analysis of the problem

Overall the problem seemed to be fairly simple. We needed to use java swing to implement a GUI for 9 items in a sweet shop. To complete this problem I want to make use of the GridBagLayout in order to position my buttons and labels onto the frame. I will use a for loop to initialise all of the attributes that I will be needed in my JFrame, this means I don't have to repeat code as I can simply use a for loop and a counter to initialise each attribute into the frame. I will also create an action listener that loops through all of my 9 buttons as they will all have the same functionality. Once a button is pressed the stock level of that item needs to decrease and the total sales needs to increase. I will also use my action listener to tell the user the remaining stock. Once the user tries to buy an item that has no stock, they should be alerted by a pop up window tell them that there is no more stock available. Considering that I have another button that allows the user to reset the stock and also reset the total sales I will have to create another class that extends JFrame to create another window when the button is pressed. This class will of course need another action listener that will determine what happens when the stock reset button is pressed.

Class Diagrams



Class methods

In my **main** class I have most of the attributes that have been used on this task, this is because Vending.java is the main JFrame that holds all of the buttons and JLabels. Once the user interacts with these buttons I had to use an action listener method that I created which chooses what happens when each of the buttons are pressed. I created another button called “Vendor Information” which when clicked uses the VendingInfo class and opens up another JFrame.

In my **Sweets.java** I have a collection of methods that are pretty important when it comes to changing anything about the sweets being sold. These methods have been used in my action listeners mainly but also in the creation of various objects used in my JFrames.

The **VendingInfo.java** is another class that extends the JFrame which is used to create another window when a button is pressed. This VendingInfo class allows the user to see how much they have spent in total but also give the user the chance to reset the stocks back to their original value and also resetting total sales back to £0.00 so the user can purchase items again.

Pseudocode – Part 1

VendorInfo.java

```
Public class VendorInfo extends JFrame impliments ActionListener {
Vending f;
JLabel totSales = new JLabel("Total sales = "+f.sales, JLabel.CENTER)
JButton stockReset = new JButton("Reset Stock")

Public VendorInfo(Vending f) {
This.f = f
JPanel northp = new JPanel(gridbaglayout)
JPanel southp = new JPanel(gridbaglayout)
setSize(250, 150)
resizable(false)
northp.add(totSales)
southp.add(stockReset)
setVisible(true)
stockReset.addActionListener(this)
add(northp, BorderLayout.NORTH)
add(southp, borderLayout.SOUTH);
pack();
}

Public void actionPerformed(ActionEvent e) {
    If( e.getSource == stockReset) {
        For (int I = 0: I < f.sweetslist.length; i++) {
            f.sweetslist[i].setStock(4);
            f.labels[i].setText(f.sweetslist[i].getStock()+ " left")
            f.sales = 0.00
            totalsales.setText("Total sales = ", f.sales);
}
}
```

Sweets.java

```
Public class Sweets {
   Private String sweetName
   Private double price
   Private int stock
   Private int startingStock

   Public Sweets(sweetName, price, stock) {
       This.sweetName = sweetName
       This.price = price
       startingStock = stock
       this.stock = stock
   }
   Public String getName() {
```

```
        Return name
    }

    Public double getPrice()  {
        Return price
    }

    Public int getStock() {
        Return stock
    }

    Public int newStock() {
        Stock = stock -1
        Return stock
    }

    Public int setStock(int n) {
        Stock = n
        Return stock
    }
}
```

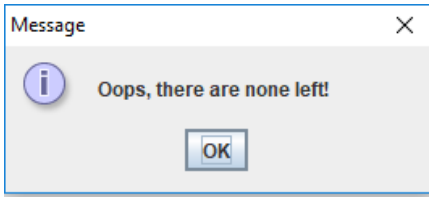
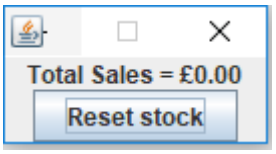
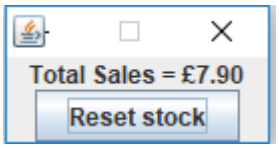
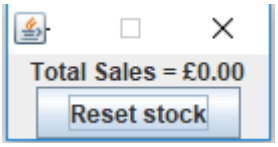
Vending.java

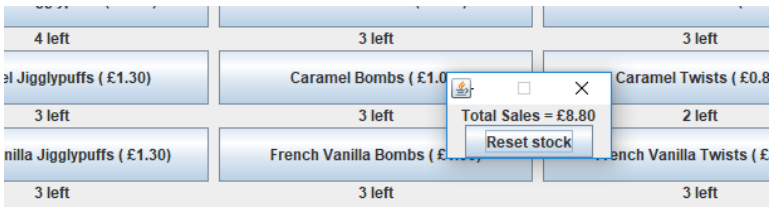
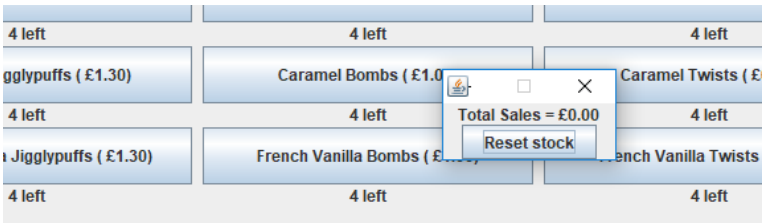
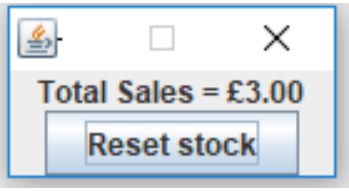
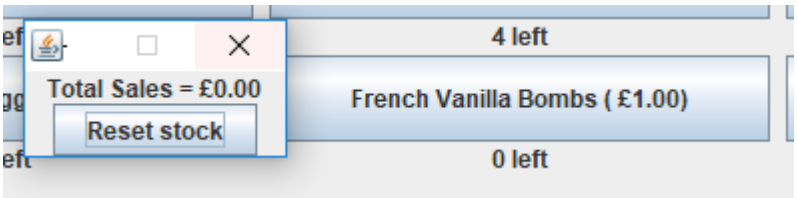
```
Public class Vending extends JFrame impliments ActionListener {
    Panels = new JPanel(GridBagLayout)
    Buttons = new JButton[9]
    Sweetslist = new Sweets[9]
    Bottompanel = new JPanel(GridBagLayout)
    vendorI = new JButton("vendor info")
    c = new GridBagConstraints
    cSouth = new GridBagConstraints
    double sales = 0.0
    VendorInfo vf;

Main(String[] args) {
    New Vending()
}

Public Vending() {
    Super("TwistyJigglyBomb suprises");
    setSize(800, 400)
    setResizable(false)
    sweetslist[0] = new Sweets(sweetname, price, stock)
    sweetslist[1] = new Sweets(sweetname, price, stock)
    sweetslist[2] = new Sweets(sweetname, price, stock)
    sweetslist[3] = new Sweets(sweetname, price, stock)
    sweetslist[4] = new Sweets(sweetname, price, stock)
    sweetslist[5] = new Sweets(sweetname, price, stock)
    sweetslist[6] = new Sweets(sweetname, price, stock)
    sweetslist[7] = new Sweets(sweetname, price, stock)
    sweetslist[8] = new Sweets(sweetname, price, stock)

int I = 0;

for (row = 0; row < 3; row++) {
    for (col = 0; col < 3; col++) {
        c.insets(0,4,0,4)
        buttons[i] = new JButton(sweetlist[i].getName)
        c.fill = GridBagConstraints.HORIZONTAL
        c.gridx = row
        c.gridy = col * 2
        buttons[i].setPreferredSize(250, 43)
        buttons[i].addActionListener(this)
```

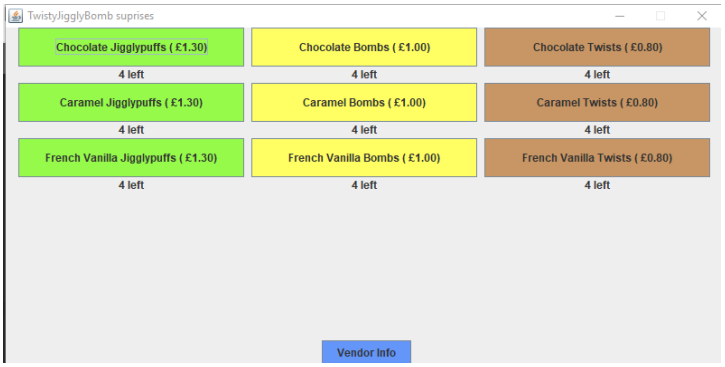
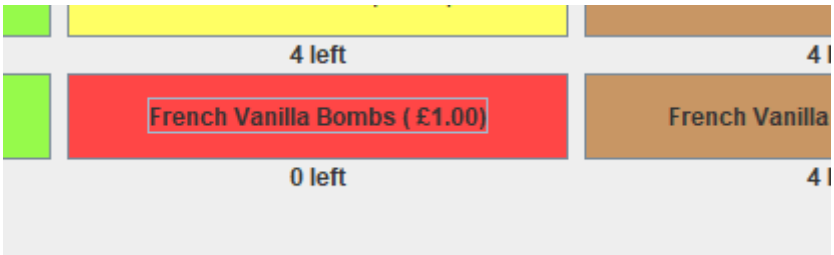
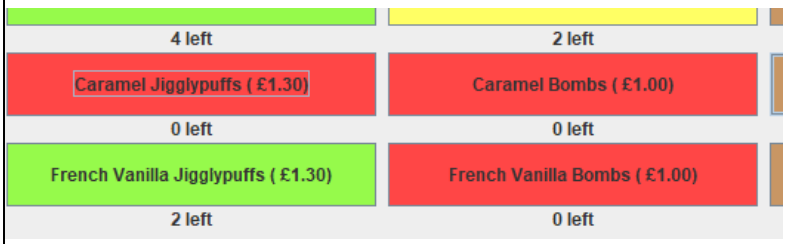
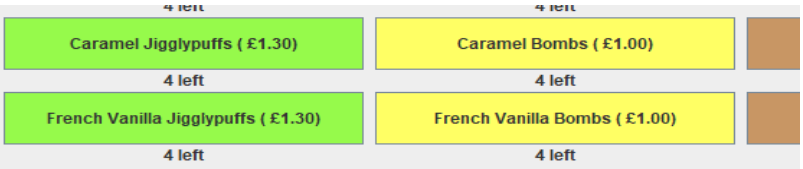
```
        panels.add(buttons[i], c)

        labels[i] = new JLabel(sweetslist[i].getStock+ “ left”)
        c.gridx = row
        c.gridy = col * 2 + 1
        c.insets(0,0,2,0)
        panels.add(labels[i], c)

        i + 1
    }
}
cSouth.insets(0,5,0,0)
bottompanel.add(vendorI, cSouth)
vendorI.addActionListener(this)
cSouth.fill = GridBagConstraints.HORIZONTAL
cSouth.gridx = 3;
cSouth.gridy = 1;
cSouth.anchor = EAST
add(panels, NORTH)
add(bottompanel, SOUTH)
CloseOperation(Exit_on_Close)
SetVisible(true)
}

Public void actionPerformed(ActionEvent e) {
    For (i = 0; I < sweetlist.length; i++) {
       If (e.getActionCommand == sweetlist[i].getname+(sweetlist[i].getprice)) {
           If (sweetlist[i].getStock <= 0) {
               JOptionPane(oops, there are none left!)
           }
           Else {
               Sweetslist[i].newStock()
               Labels[i].setText(sweetlist[i].getStock+ “ left”)
               Sales = sales + sweetlist[i].getPrice()
               Repaint();
           }
        }
        If (e.getSource() == vendorI) {
            VendorInfo VI = new VendorInfo(this)
            Break;
        }
     }
  }
}
```

Testing

Test number	Test Description	Expected result	Actual result (with proof)	Changes needed
1	Testing to make sure my main frame opens correctly and displays all of the 9 buttons and the	The program should work as expected and open up my program and then display the buttons	The program worked correctly and displayed the buttons to the user correctly. 	No changes required

	vendor information button			
2	Testing to ensure that the stock value of an item is decreased once a button is pressed	The program should update the JLabel below the button with the correct sotck value once the button is pressed	The program worked correctly and updated the JLabel below the button with the correct stock once the button had been pressed. 	No changes required
3	Testing to ensure the user is given a notice when the item has ran out of stock	The program should alert the user of the program that there is no stock remaining if the stock value is equal to 0.	The program worked correctly and gave the user a notice that there was no remaining stock. 	No changes required
4	Testing to ensure that my Vendor information button opens another frame for the user	The vendor information button should open another window for the user giving them the option to reset stock	The program worked correctly and opened up another window for the user. 	No changes required
5	Testing to ensure that once the stock reset button is pressed on the vendor information window, the total sales should reset to 0.	The program should reset the total sales back to “£0.00” once the stock reset button is pressed	The program worked correctly and updated the total sales value back to £0.00.  (before button press)  (after button press)	No changes required

6	Testing that all of the stock values reset back to the original value of 4 once the stock reset button is pressed	The program should reset all the values back to 4 once the stock reset button is pressed.	The program worked correctly and reset the values back to 4 once the button had been pressed.  (before reset stock button had been pressed)  (after reset stock button had been pressed)	No changes required
7	Testing that the total sales of the items is being correctly calculated.	I shall be buying 3 chocolate bombs for 1 pound and the total sales should be equal to £3.00	The total sales was calculated correctly and was equal to £3.00  	No changes required
8	Testing to see if the total sales on my vendor information window increases as I press buttons in my main frame.	I shall have the vendor information button open and I will try buying items to see if the total sales increases without me having to re-open vendor information.	This did not work correctly. My sales were being calculated correctly however in order for the value to update I needed to close and then re-open the vendor info window.  I bough 4 of the “French Vanilla bombs” while the vendor information window was open, however the total sales did not update until I closed and reopened the window.	I need to somehow change my vendor information frame so that when a button is pressed on the main frame it updates the total sales without having to close and then re-open the vendor information window.
9	Testing my additional	The program	The program worked correctly and the GUI looks much more user friendly due to the colour changes of the buttons.	No changes required

	coding of adding colours to the buttons. The button colours should be different based upon the type of sweet being sold	should display the different colours on the buttons correctly		
10	Testing that the buttons turn red when there is no remaining stock for that item.	If the stock is 0, the buttons should change red to alert the user there is no stock.	The program worked correctly and the buttons turned red when the stock value was 0. 	No changes required
11	Testing to ensure that when the stock reset button is pressed, the colours of the buttons should default back to the original colours from red.	The buttons should change back to the original colours.	The program worked correctly and set the buttons back from red to the orginal colours. (Before reset stock button was pressed)  (after reset stock button is pressed) 	No changes required

Part 2 – Doors

Summary of the requirements

In this task we were asked to consider 3 different players who all like to open doors. We were given a scenario where the user enters a number of doors via the args[0] which is then parsed into an integer. Each of the players however like to open the doors differently. For example, Sven likes to open doors for every perfect square number. For example, 1, 4, 9, 16, 25 ... etc. however Petra opens doors

based upon whether the door is a prime number or not, if the door is prime then Petra will toggle the door otherwise Petra will leave the door closed. In this part of the assignment we were asked to create an abstract class that all 3 of the players would be using. In this abstract class would be methods used by all of the players. The assessment also recommends creating two constructors for each player, one that will take an integer as a parameter but also another one which can take an array of Boolean values. We needed to tell the user of the program how many doors each player had opened based upon the amount of doors entered into the args[0]. Overall the structure of this part of the assignment should be similar to the structure we used in our assignment 3 part 1 for the cockroach scenario.

Analysis of the problem

Overall the structure of this task is very similar to how we structured our Cockroach task. We will be creating an abstract class that all of the players will be using, this will contain methods used by all of the players. I would then create a new file for each player where we specify the way each of them interact with the doors. I would then call all of these methods for each of the players in my main when outputting how many doors out of a given integer were opened by each player. Considering we will be taking input from args[0] we will need to use some try and catch statements to ensure that the user input into the args[0] is correct. Once we know the input is valid we can then use this to start interacting with each of the players. We can then output to the user how many doors doors have been toggled.