Group assignment

This is a group assignment to be performed in pairs. All students should inform me who their partner is, either in class or via email, no later than 11/27. After that time, unpartnered students will be paired by the instructor.

Objective

Create a user interface for entering orders from a catalog.

Activity

- Examine the class diagram below and the associated Java code. These implement a simple object-oriented catalog ordering scenario. There are four main classes: Order, OrderItem, Catalog and CatalogItem. A Catalog consists of a Vector of CatalogItem objects. Each CatalogItem has a name, description, price and sku #. An Order consists of a Vector of OrderItem objects. Each OrderItem has a CatalogItem and a quantity.

- Implement a user interface for order entry. The screen should look something like this:

![User Interface Diagram]

- When a quantity is entered into the “quantity” text box, the price should recalculate. When items are added to and removed from the order, the total price should update automatically. When the user clicks “OK”, the window should close and an invoice for the order should be printed to the console (System.out). Here is an example invoice:

```
Invoice for Witch's Supply House
Qty  Item      SKU  Item price  Item total
 1    eye of newt 123abc $2.99   $2.99
 5    tongue of dog 989dde $9.99  $49.95

Grand Total $52.94
```
• The user should be able to double-click on an item in the catalog and get a dialog box that displays the
details of the item.

![Dialog Box Example]

• You should implement this assignment in two classes: one for the order entry window and one for
the catalog item detail.

Submission

Students should

Turn in at the start of class a hardcopy of the code of your class file with a cover page clearly
indicating the number and name of the assignment and both students’ names and ID #s.

Before class time, submit a folder containing the complete JBuilder project for the Java classes to
the on-line course Drop Box for homework #7. This folder is accessible as a shared volume on lab
and classroom machines at \Doctor\Assignments\Burke\SDS 372\HW7. You must copy the
entire folder at once: files and folders placed on the server cannot be modified. Your folder should
be named with two student names and the assignment number. For example:
BurkeSylvester_HW7. If you make a mistake and have to submit the folder again, add a letter to
the end. The system will not permit you to overwrite your first submission. I will grade the most
recent folder (submitted up to classtime).

Assessment

This assignment will be assessed on the completeness of the solution to the problem. You should
concentrate on the functionality of the interface before concerning yourself with its appearance. Partial
solutions will be given partial credit.

Hints and Notes

1. Both of your classes should be sub-classes of JFrame and should use Swing components (JButton,
JTextField, etc.) exclusively.

2. Use anonymous listener classes as discussed in class.

3. You may wish to use the Formatter class defined in homework #5 as part of the toString method.

4. You will need to round all amounts to cents.

5. Feel free to put additional items into the catalog: adder's fork, blind-worm's sting, lizard's leg and
owlet's wing would be traditional. Find more in Macbeth Act 4, Scene 1.

6. To divide the interface by function as shown, you will need to place JPanel components on your
interface and add the interface widgets to them.

7. Once you start working on the interface’s appearance, the Box layout is probably the most useful
8. You will need to create the following event listeners:

- ListSelectionListener for the catalog list: when an item in the list is selected, its price is reflected in the price line of the catalog.
- MouseListener for the catalog list (to pop up the information dialog): when the user double clicks on an item in the catalog list, show the information dialog describing the item.
- ActionListener for the quantity text box: when the user enters a new quantity, the price of the order unit is updated to be the unit price x the quantity.
- ActionListener for the add button: when pressed, a new order item is added to the order with the selected item and current quantity.
- ListSelectionListener for the order list: allows selection of order items
- ActionListener for the remove button: deletes the selected item
- ActionListener for the done button: closes the window and prints the invoice.

9. This program will be long because each of the user interface widgets must be created and configured. Work first to get all of the widgets in place and then work on connecting them using event handling.

10. The user should not be able to change the order item price or the total price fields. You can control this by calling setEditable(false).on the appropriate JTextField or by making this widgets labels rather than text fields.

11. The JList object has a setListValues method that takes an array of Object. Because all Java objects are subtypes of Object, that means we can pass an array of CatalogItem objects to a JList using this method and get a list that displays our catalog items. The same is true for OrderItems. The toString method determines what text is displayed in the list box. A benefit of this approach is that the getSelectedValue() method returns a reference to the object that the user has selected – the individual CatalogItem or OrderItem.

12. The following class diagram illustrates the relationship between the Catalog, CatalogItem, Order and OrderItem classes.
Catalog java files

import java.util.Vector;

public class Catalog {

    private static String[][] s_catalogEntries =
    { {"eye of newt", "123abc",
       "Freshest newt eyes direct from the Louisiana swamps.", "2.99"},
     {"toe of frog", "456ffc",
       "Frog's toes clipped from our own stable of Texas bullfrogs.", "1.25"},
     {"wool of bat", "989dde",
       "Not common fruit bat hair, this is from genuine Chihuahua vampire bats.",
       "5.75"},
     {"tongue of dog", "989dde",
       "Whole dog's tongues packed in ice for freshness.", "9.99"} }; 

    Vector m_catalog;

    public Catalog()
    { 
        m_catalog = new Vector();
        for (int i = 0; i < s_catalogEntries.length; i++)
        { 
            CatalogItem item = new CatalogItem(s_catalogEntries[i]);
            m_catalog.add(item);
        }
    }

    public CatalogItem[] items()
    {
        CatalogItem[] items = new CatalogItem[m_catalog.size()];
        for (int i = 0; i < m_catalog.size(); i++)
        {
            items[i] = (CatalogItem) m_catalog.get(i);
        }
        return items;
    }

    public void add (CatalogItem item)
    {
        m_catalog.add(item);
    }
}
public class CatalogItem {

    String m_name;
    String m_description;
    String m_sku;
    double m_price;

    public CatalogItem(String name, String sku, String description, double price) {
        m_name = name;
        m_sku = sku;
        m_description = description;
        m_price = price;
    }

    public CatalogItem (String [] values) {
        m_name = values[0];
        m_sku = values[1];
        m_description = values[2];
        m_price = Double.parseDouble(values[3]);
    }

    public double getPrice () {
        return m_price;
    }

    public String getName () {
        return m_name;
    }

    public String getSku () {
        return m_sku;
    }

    public String getDescription() {
        return m_description;
    }

    public String toString () {
        return m_name;
    }
}
import java.util.Vector;

public class Order {
    Vector m_orderItems;

    public Order() {
        m_orderItems = new Vector();
    }

    public OrderItem [] items () {
        OrderItem [] items = new OrderItem [m_orderItems.size()];
        for (int i = 0; i < m_orderItems.size(); i++) {
            items[i] = (OrderItem) m_orderItems.get(i);
        }
        return items;
    }

    public void add (OrderItem item) {
        m_orderItems.add(item);
    }

    public void remove (OrderItem item) {
        m_orderItems.remove(item);
    }

    public double totalPrice () {
        double total = 0;
        OrderItem [] items = items();
        for (int i = 0; i < items.length; i++) {
            total += items[i].price();
        }
        return total;
    }
}
public class OrderItem {

    CatalogItem m_item;
    int m_quantity;

    public OrderItem(int quantity, CatalogItem item) {
        m_quantity = quantity;
        m_item = item;
    }

    public CatalogItem getItem () {
        return m_item;
    }

    public int getQuantity () {
        return m_quantity;
    }

    public double price () {
        return m_item.getPrice() * m_quantity;
    }

    public String toString () {
        return m_quantity + " " + m_item.getName() + " @ " + m_item.getPrice();
    }
}

Swing Example

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HelloSwing extends JFrame {

    private JLabel lblMessage = new JLabel();
    private JButton btnOK = new JButton("OK");

    public HelloSwing(String msg) {
        getContentPane().setLayout(new FlowLayout());
        lblMessage.setText(msg);
        getContentPane().add(lblMessage);
        btnOK.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }
            });
        getContentPane().add(btnOK);
    }

    public static void main(String[] args) {
        HelloSwing win = new HelloSwing("Hello World");
        win.setSize(300, 100);
        win.show();
    }
}
```