COMP 1006/1406
Assignment 2 – Carleton Foodorama (Part 2)

Due: Friday, July 21st 2006, before 11h55 pm

In this assignment, you will practice making GUI applications. You will use layout managers and learn how to use the Subject/Observer design pattern. You will also learn to print documents on a printer. This assignment is a continuation of your assignment 1. You have the choice between reusing your own code or taking the solution that will be posted on the course web site on July 14th between 0h05 am and 10h00 am. **You can not take someone else's assignment as a starting point.**

Part A (The Appropriate Layout):

[10 pts] In the previous assignment, you were not asked to provide a fancy GUI for the class OrderForm. Now, you have to do it.

- Modify the constructor of the OrderForm class in order to have components laid out nicely.

[10 pts] Create a Register class which extends JFrame. A register has a single private instance variable called orderForm (of type OrderForm). You have to create:

- A constructor which takes a menu as argument. The frame should contain, in the center part, an order form. The bottom part should be occupied by a "print" button.
- A main() method which first create a menu object from the file menu.txt. It then creates a Register using that menu. The application GUI should look as shown on the right.

Part B (Computing the Total):

[20 pts] Create Subject and Observer interfaces as shown in the course notes.

- Make the appropriate modifications in order to have Order implementing Subject and and OrderForm implementing Observer.
- In its update() method, OrderForm should get the sub-total, taxes and total from the order and disply it in the appropriate text fields.
- In the constructor of the OrderForm class, you have to make sure that the user can not modify the text fields associated to the sub-total, taxes and total. Those fields are only set by the program.

[35 pts] Create a private inner class within OrderForm called QuantityListener which implements the interface javax.swing.event.DocumentListener. Each instance of QuantityListener will listen on a given text field to update the order. A quantity listener has one private variable called **index** (of type int). This index indicates on which menu item it listens. You have to:

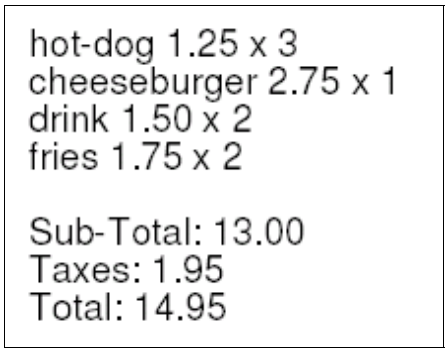
- Create a constructor to the class QuantityListener which takes the **index** as argument.
- Create a method setQuantity(MenuItem item, int quantity) in the Order class which allows to specify the quantity of a given item.
- Modify the constructor of OrderForm so that the text fields are put in an ArrayList. This ArrayList should be called **quantities** and be an instance variable of the OrderForm class.
- In the class OrderForm, as the text fields associated to menu items are created, add an instance of QuantityListener to their documents (Hint: you may have to use the getDocument() method on JTextField and the addDocumentListener() on Document!). The index you use as an argument to the listener constructor should be the one corresponding to the text field in the **quantities** array.
- Create a private update() method in the class QuantityListener. This method should do two things:
    1. It reads the quantity of items in the text field to which it is associated. In order to do this, it uses its local variable **index** to retrieve the appropriate text field in the **quantities** array. You can convert String into Integer using the appropriate Integer constructor.
    2. It then sets the quantity of the appropriate menu item in the order. Once again, the appropriate menu item can be retrieved in the **menu** using the **index**.
- In the QuantityListener class, each of the three method specified by the DocumentListener interface should make a call to the update() method.
- Once all of the above is done, the sub-total, taxes and total should update automatically as the user types new quantities.

Part C (Printing the Bill):

Printing a document in Java can be done the exact same way you would print a GUI component. If you do not have a printer at home, you can freely download CutePDF (www.cutepdf.com). This is a virtual printer driver which will allow you to print documents in a PDF file.

[15 pts] Modify the Order class so that it implements the java.awt.print.Printable interface. You only have to implement the print() method. If the page number is not zero, the method should return NO_SUCH_PAGE. Otherwise, you have to do the following:

- For each menu item for which the quantity is greater than zero, paint a line indicating the name of the item, its unit price, as well as the ordered quantity. As in the preceding assignment, prices should be displayed with exactly two decimal digits.
- Paint three other lines, one for the sub-total, one for the taxes and one for the total.
- When all is done, the method should return PAGE_EXISTS.
- The final result should look as shown on the right:



[10 pts] In the Register class constructor, just after the construction of the print button, add the following lines (you may have to rename some variables):

```
print.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
            PrinterJob printJob = PrinterJob.getPrinterJob();
            printJob.setPrintable(orderForm.getOrder());
            if (printJob.printDialog()) {
                  try {
                        printJob.print();
                  } catch (Exception PrintException) {
                        PrintException.printStackTrace();
                  }
            }
      }
});
```

Note: You will have to create a getOrder() method in the OrderForm class.

Do the appropriate testing to make sure the print button works correctly.

Look over the Internet to find what an anonymous class is. Explain in you own words what the above code does. Was it necessary to use an anonymous class? In your own words, what is the difference between an anonymous class and an inner class?

Submission

As usual, you will submit to the Raven system. You should submit an entire directory that contains all of your java files, your class files and any testing/output files that you created. Also, have a readme.txt file that clearly indicates which files are which.

Good Programming and Good Practice Requirements.

These requirements pertain regardless of what your application is supposed to do (i.e. regardless of the design requirements). These requirements are to ensure that your code is readable and maintainable by other programmers (or TA's in our case), and that your program is robust (It does not crash from bad object references). You will loose 5 marks from your total assignment mark for each of the following requirements that are not satisfied. If you do not satisfy requirement R0 you will get nothing for the assignment.

R0) IMPORTANT Uniqueness Requirement. The solution and code you submit MUST be unique. That is, it cannot be a copy, or be too similar, to someone else's assignment, or other code found elsewhere. A mark of 0 will be assigned to any assignment that is judged by us or the TA's not to be unique.

R1) All of your variables, methods and classes should have meaningful names that reflect their purpose. Do not follow the convention common in math courses where they say things like: "let x be the number of customers and let y be the number of products...". Instead call your variables numberOfCustomers or numberOfProducts. Your program should not have any variables called "x" unless there is a good reason for them to be called "x". (One exception: It's OK to call simple for-loop counters i, j and k etc. when the context is clear.)

R2) All variables in your classes should be private, unless a specific design requirements asks for them to be public (which is unlikely). We will design objects that provide services to others through their public methods. How they store their variables is their own private business.

R3) Robustness Requirements: Your program should never crash because of a "null pointer exception". This exception means that you are using a variable that does not actually refer to an object. We instruct the TA's to try and crash your program for this reason so guard against it.

R4) Comments in your code must coincide with what the code actually does. It is a very common bug in industry for people to modify code and forget to modify the comments and so you end up with comments that say one thing and code that actually does another. (By the way, try not to over-comment your code; instead choose good variable names and method names which make the code more "self commenting".)

R5) Java 1.5 Requirement: Your code should compile with the Java 1.5SDK without warnings.