SCHOOL OF COMPUTER SCIENCE

PRIFYSGOL
BANGOR
UNIVERSITY

Object Oriented Programming Project
30%

A Simple Stock Control System

Dave Perkins

Introduction

The XYZ Computer Company, an internet based hardware company, have hired you to help build a simple stock control system. The completed system will involve:

- Multiple classes;
- Use of files on disc to store stock data;
- A simple console based menu system;
- Exception handling.

In developing the system please make sure that you adhere *strictly* to the specification provided in these notes. If you are in doubt as to what you should supply please ask a member of the course team who will be happy to advise you.

Part 1: Developing the Class StockItem (15%)

The class **`StockItem`** has five instance variables:

```
String itemID;          // Five alpha-numeric characters
String itemDesc;        // Item description
double price;           // Item price in pounds sterling
int quantity;           // Quantity in stock
int reOrderLevel;       // Level at which to re-order
...
```

The system designer has asked you to supply:

- A constructor with five parameters to initialise each of the instance variables
- A **`toString`** method (see *Big Java: Late Objects* ,p.443)
- A **`format`** method (see *Big Java : Late Objects*, p.571)
- A **`get`** method for each instance variable;
- A set method for the instance variables **`price`** , **`quantity`** and **`reOrderLevel`**.

The source code for this class must be stored in a file called **`StockItem.java`**. Marks will be deducted if you depart from the above requirements. If in doubt about what you should provide ask a member of the course team.

Document your class in Javadoc style and generate the appropriate documentation.

For this part of the assignment you must submit

- Source code for **`StockItem`**
- Documentation generated using Javadoc

Part 2: Testing StockItem (10%)

Write a test driver called **`StockItemTester`** to perform unit testing on the class **`StockItem`**. The test driver must test every method in **`StockItem`** at least once and generate results together with a test narrative.

For this part of the assignment you must submit the following items:

- Source for **`StockItemTester`**
- A test plan
- A test log
- A screen shot of the test run

Part 3: Implementing the StockList Interface (25%)

Write a class to implement the interface **`StockList`** which is listed in Appendix 1. The class must be documented in Javadoc style. The implementation strategy must involve the use of a linked list and the class developed should be called **`StockLinkedList`**.

You should also revise the interface so that it is also documented in Javadoc style.

The two format methods listed in the interface should return formatted strings which can be used by *other* classes to display stock data in tabular form as illustrated below:

```
ItemID      Description            Price      Qnty       Re-Order Level
******      ***********            *****      ****       **************
...
P1234       Hard Drive 73 GB       75.00      267        50
…
```

Note that format methods are *not* themselves responsible for displaying data on the screen. If you are in doubt about what any of the methods should do ask a member of the course team.

For this part of the assignment you must submit the following items:

- Source for revised version of the interface **`StockList`**
- Source for **`StockLinkedList`**
- Documentation generated using Javadoc

Part 4: Testing the Implementation (10%)

Write a test driver called **`StockListTester`** to perform unit testing on the class **`StockLinkedList`**. The test driver must test every method in **`StockList`** at least once and generate results together with a test narrative.

For this part of the assignment you must submit the following items:

- Source for **`StockListTester`**
- A test plan
- A test log

Part 5: Saving Data to File (10%)

Add two methods to **`StockLinkedList`** to allow the stock list to be read from or written to a file on disc. The methods should have the following headers:

```
// Loads data from the stock file into the stock list
public void loadStockData(String filename)
{
    ...
}

// Saves data from stock list to file
public void saveStockData()
{
    ...
}
```

The methods above *must* be implemented using *object streams*. Note also that you will need to address the issue of exceptions (e.g. **`FileNotFoundException`**) and provide appropriate handlers.

You should also amend the interface.

One final point: to make the stock system as efficient as possible the stock file should only be updated by **`saveData`** if a write operation has occurred following the invocation of **`loadData`**.

Modify **`StockListTester`** so that the new methods can be tested.

For this part of the assignment you must submit the following items:

- Source for an amended **`StockListTester`**
- Source for an amended **`StockLinkedList`** and the interface **`StockList`**
- An amended test plan and test log

Part 6: Providing A User Interface (20%)

Create a class called **`StockListCLI`** to allow the user to interact with the stock list via the keyboard and console screen. For example, when the application starts the user should be greeted with a simple menu system illustrated below:

```
Stock List Main Menu
********************

1. Add an Item
2. Delete an Item
3. Update Item Price
4. Update Item Quantity
5. Update Re-Order Level
6. Print Stock List
7. Print ReOrder List
8. Exit

Select option [1-8] :> 1
```

Here the user has selected option 1 so they should be presented with an input screen.

```
Add New Item
***************

Enter ID                :>
Enter Description       :>
Enter Price             :>
Enter Quantity          :>
Enter Re-Order Level    :>

Enter another (Y/N)     :> n
```

Input for the main menu must be validated, if the user enters an invalid option they should be notified and then returned to the main menu. At this stage, however, you should not bother with any other form of data validation. (See Part 6)

Other options involving user input should operate in a similar fashion.

The menu system *must* be implemented using the **`switch`** construct and should *not* make use of any recursive programming techniques.

A code stub for the class providing the user interface is presented below:

```
public class StockListCLI
{
	private StockList stock = null;

	public StockListCLI(StockList stock){…}

	// Displays main menu and gets valid option from user
	public void doMenu() {}

	// Obtain input for stock list operation
	// and invoke operation
	private void doAddItem() {}
	private void doDeleteItem() {}
	private void doUpdateItemPrice() {}
	private void doUpdateItemQuantity() {}
	private void doUpdateReOrderLevel() {}

	// Display contents of stock list
	private void doPrintStockList() {}

	// Display contents of re-order list
	private void doPrintReorderLIst() {}

}
```

The private methods are 'helper' methods invoked from within the **doMenu()** method. Note also that **StockListCLI** requires a reference to a **StockList** object otherwise data received through the interface cannot be supplied to the stock system.

Having completed **StockListCLI** – a class which also lacks a **main** method - we need a third class to initiate the application. This is listed below:

```
public class StockListApp
{
	public static void main(String[] args)
	{
		// Create the stock list object
		StockList stockList = new StockLinkedList();
		stockList.loadStockData(args[0]);

		// Create an interface
		StockListCLI stockListCLI
			= new StockListCLI(stockList);

		// Display the menu
		stockListCLI.doMenu();
	}
```

```
}
```

For this part of the assignment you must submit the following items:

- Source for **`StockListCLI`**
- Documentation generated using Javadoc
- Source for **`StockListApp`**

To complete the project do *either* Part 7a or Part 7b.

Part 7a: Using Exceptions (10%)

Even in this relatively modest stock system all sorts of error situations might arise: for example, think of all the different ways in which invalid input may occur. Luckily, you are not required to build a system that is 100% error proof – assuming that such a system could ever be built and that is doubtful – but merely one that can deal with certain clearly defined errors. To handle these situations you must create your own exception classes and develop suitable exception handlers.

The errors that you are required to handle are listed below

- Attempting to add an item with a product identifier which already exists
- Attempting to delete a non-existent stock item

For this part of the assignment you are required to submit

- Screen shots demonstrating the effect of your error handlers

Part 7b: An Alternative Implementation (10%)

Use an array list to develop an alternative implementation of the **`StockList`** interface. The new class should be called **`StockArrayList`**. Make sure that you test this new implementation.

Challenge Problem (Optional)

Can you provide a 'Clear Screen' facility similar to the CLS command used in the command prompt?

Submission Notes

Use **Blackboard** to submit all of your source code files and requested documentation. Each source code file must:

- Contain a program header
- An appropriate level of comments
- Follow a consistent style of indentation
- Follow the usual Java conventions for class and variable names

Please submit your work through Blackboard by the stated deadline. Late submissions will be penalised in line with School policy.

Marks for these laboratory exercises are awarded with reference to the following:

- Program correctness
- Program structure
- Layout and use of naming conventions
- Comments
- Testing
- Conceptual understanding

When submitting work it is your responsibility to ensure that all work submitted is

- Consistent with stated requirements
- Entirely your own work
- On time

Warning. If you submit a program but demonstrate very poor understanding of how it works you will be heavily penalised. This may also trigger an investigation into possible plagiarism. Please note that there are **severe penalties** for submitting work which is not your own. *If you have used code which you have found on the Internet* or *from any other source* then you **must** signal that fact with appropriate program comments.

Note also that to obtain a mark **you must attend a lab session within two weeks of submission** and be prepared to demonstrate your program and answer questions about the coding. Non-attendance at labs will result in your work not being marked.

Appendix 1: Interface for StockList

```
public interface StockList
{
      // Adds item to stock list
      public void addItem(StockItem item);

      // Removes item identified by productID from stock list
      public void deleteItem(String itemID);

      // Updates price of existing item
      public void updateItemPrice(String itemID, double price);

      // Updates quantity of existing item
      public void updateItemQuantity(String itemID, int quantity);

      // Updates re-order level of existing item
      public void updateReOrderLevel(String itemID,
                                     int reOrderLevel);

      // Returns formatted representation of the stock list
      public String formatStockList();

      // Returns formatted representation of re-order list
      // Items are on this list if quantity < reOrderLevel
      public String formatReOrderList();

}
```