Creating an Event Driven Computer Program

Unit Reference Number F/601/3179 – adapted for OO Java

This unit is completed in the workplace (or away from training) and will be assessed by a QA assessor. The unit has assessment criteria for which the assessors will check your understanding of that criterion. This is done using several methods – your product (i.e. program), your documentation (case study, test plans, test programs, support/maintenance document) and possibly through other methods (such as a professional discussion).

You will be required to create the following:

* Program to implement an event driven program, with on-screen help
* Case study detailing each criterion for this unit, short description of your understanding of the criteria and how you have met the criteria in your program
* Test plan and results for your program
* Unit test class/program
* Documentation for support and maintenance of your program

Your assessor once he/she has reviewed these may require changes to these (for errors or for further understanding) before looking at other ways to assess (such as professional discussion, presentation etc.)

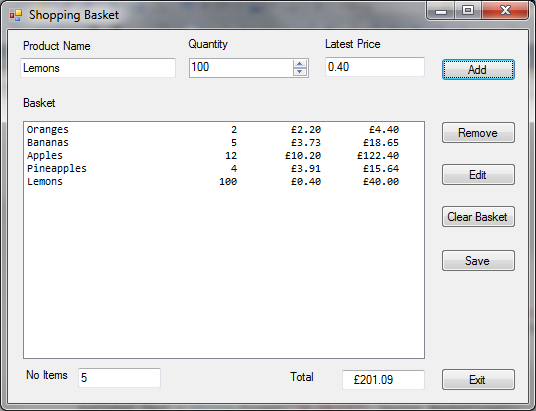
We are going to suggest that the program that you create is a Java GUI program to display a shopping basket and save the contents of the shopping basket to a file as a receipt.

For this you are going to have to create a Shopping Basket class that encapsulates the functionality of a basket. Within this we are expecting you to maintain a list of items within the basket. This class should not contain any UI (User Interface) elements at all. A specification is available for you to implement a Shopping Basket if you so wish.

Your program should provide the following functionality:

* Add an item to the basket, specifying the item name, latest price and quantity
* Remove an item completely from the basket
* Edit an item in the basket (i.e. change the quantity or latest price)
* Clear the whole basket
* Save a receipt for the basket to a file

A suggested interface is shown below:



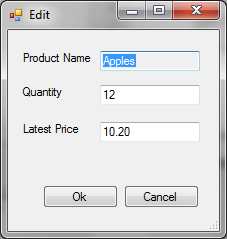
Details of the functionality required are listed below.

When your program starts it should display an empty basket and set the Number of Items and Total to zero respectively.

**Add Button**

When clicking the add button the program should validate the product name, quantity and latest price controls to ensure that the data in them is valid. It should then add the product to the basket which will then be displayed in the main area.

**Remove Button**

When clicking the remove button the program should check that an item(s) in the main area is selected and then remove them from the basket, which will then be re-displayed in the main area.

**Edit Button**

When clicking the edit button the program should check that an item in the main area has been selected. It should then be displayed in a new window allow the user to edit it. If they click on Ok the basket should be updated and then be re-displayed in the main area.

**Clear Basket Button**

When clicking the clear basket button the program should remove all items from the basket and the main area updated.

**Save Button**

When clicking the save button the program should save to disk a copy of the items in the basket which could then be used as a receipt.

**Exit Button**

When clicking on the exit button the program should exit.

The criterion for assessment is listed below. This project will be assessed by your assessor. The criterion can also be found at <http://register.ofqual.gov.uk/Unit/Details/F_601_317>.

Implement a software design using event driven programming

1. **Implement a software design using event driven programming**
   1. Identify the screen components and data and file structures required to implement a given design

*Provide an overview of your classes within your program and how they relate to your design (e.g. list box). For example your ShoppingBasket classes encapsulate the functionality of a shopping basket but do not have UI components (which are in your UI classes).*

* 1. Select, declare and initialise variable and data structure types and sizes to implement design requirements

*Show code examples of data structures being declared and initialised (e.g. ShoppingBasket within your GUI program, OrderItems within your ShoppingBasket).*

* 1. Select and assign properties to screen components to implement design requirements

*Show any methods you have implemented the design (e.g. Fonts on list boxes, Text on buttons) – make sure you show both attributes being set within the code and via the designer.*

* 1. Select and associate events (including parameter passing) to screen components to implement design requirements

*Show code examples of the button click events and explain. For parameter passing you should give information on how you used your JDialog instead of a JFrame (for editing an item).*

* 1. Implement event handling using control structures to meet the design algorithms

*Controls structures are statements such as if. Show code examples within event handlers dealing with if (e.g. only removing an item from the basket if an item has been selected).*

* 1. Select and declare file structures to meet design file storage requirements

*This is a slightly oddly worded criterion. You are going to save the file to disk, so write-up the format of the file that you save.*

* 1. Select and use standard input/output commands to implement design requirements

*Show code examples of any File-IO operations (such as PrintWriter or BufferWriter).*

* 1. Make effective use of operators and predefined functions

*Operators are things such as +, - and pre-defined method such as those to convert strings to numbers.*

* 1. Make effective use of an Integrated Development Environment (IDE) including code and screen templates

*Show how you have used the IDE (i.e. eclips)*

1. **Refine an event driven program to improve quality**
   1. Use an agreed standard for naming, comments and code layout

*Provide information on your naming convention (with examples), details of code commenting (examples, when appropriate, XML comments) and general code layout.*

* 1. Define user methods to replace repeating code sequences

*Provide examples of places where you have created methods to replace common/repeated code (e.g. displaying the basket).*

* 1. Implement data validation for inputs

*Provide code examples of any data validation for inputs (e.g. checking entries for prices are convertible to decimals).*

* 1. Identify and implement opportunities for error handling and reporting

*Provide code examples of places where you implement any error handling. Error handling is not validation of input fields – places are those such as raising/catching exceptions, checking a file exists and returning a result to then display an error message.*

1. **Test the operation of an event driven program**
   1. Make effective use of the debugging facilities available in the IDE

*Show via screenshots your use of the debugging facilities within IDE (eclips).*

* 1. Prepare a test strategy

*Your testing in this program needs to be in two ways. You already know how to write code and test the code – you should do this for your shopping basket/order items classes. You also need to create a test strategy for your program. This will be a list of tests you are going to perform.*

* 1. Select suitable test data and determine expected test results

*You should know this for testing code.*

*For your program, you need to create test data for each test (e.g. Check validation on adding a product, test data “Oranges”, “1”, “1.4). You then need to say what the expected result is (The product is added and displayed in the main area).*

* 1. Record actual test results to enable comparison with expected results

*Once you have created your “formal” test plan and test data you then need to run through it and create a test log of the expected results. Create an additional column on your test plan to say whether the test has passed/failed and put into your test log screenshots/details of tests that have failed.*

* 1. Analyse actual test results against expected results to identify discrepancies

*Write up some tests that have failed.*

* 1. Investigate test discrepancies to identify and rectify their causes

*Write up how you have fixed any issues you have found.*

1. **Document an event driven program**
   1. Create on-screen help to assist the users of a computer program

*Implement some on-screen help, e.g. Tooltips and show examples (both in Code and Screenshots).*

* 1. Create documentation for the support and maintenance of a computer program

*You need to create a further document so that another developer can come along to support/maintain your computer program. This is a large task where you should explain what each of your classes do along with the properties/fields/methods are responsible for.*

Shopping Basket

We suggest that you implement a class library with two classes (**ShoppingBasket** and **OrderItem**) which will be used to manage items in a shopping basket.

## Shopping Basket class

This class will manage the items in the shopping basket. It will contain a list of order items (i.e. List<OrderItem>) as well as other fields/properties as necessary.

The following constructors/methods/properties should be provided

* Constructor taking no arguments, creating an empty shopping basket and initialising any fields within the basket
* Method **AddProduct** taking arguments for the product name (string), latest product value (decimal) and quantity of products (int) purchased
* Method **AddProduct** taking arguments for the product name (string), and latest product value. This version of AddProduct will assume that the quantity will be one or more.
* Method **RemoveProduct** taking arguments of product name and quantity which will decrement the quantity. The quantity cannot go below zero, in which case the product will be removed from the shopping basket. If the product name does not exist in the basket then an exception should be thrown.
* Method **RemoveProduct** taking an argument of product name which will remove the product from the shopping basket
* Method **ClearBasket** which will remove all items from the shopping basket
* Method **OrderItems** which will give access to all the items in the basket
* Method **NumberOfProducts** which will give the total number of products in the shopping basket
* Method **BasketTotal** which will return the current total price of the items in the basket
* Method **NumberOfItems** which will return the total number of items on order in the shopping basket
* Method **CurrentPrice** which will take a product name as an argument and return the latest price. An exception should be thrown if the item does not exist in the basket
* Method **IsProductInBasket** which will take a product name as argument and return true/false if the product is currently in the basket
* Method **SaveBasket** will take as argument a filename and a receipt for all the items will be produced. The method will return true/false if the save operation has been successful

Decide whether any other constructors/methods/properties should be added to the class.

You will assume that the latest value of an individual product is the new current price.

The list of order items should be actively maintained within the class. No order items in the list should have a quantity of zero.

## OrderItem class

This class will be used to maintain a single order.

The following constructors/methods/properties should be created

* Constructor taking a product name and latest price. The quantity should default to one
* Constructor taking a product name, latest price and quantity.
* Method **getProductName** to return the product name
* Method **getLatestPrice** to return and set the latest price
* Method **getQuantity** to return the quantity
* Method **getTotalOrder** to return the total price of the order items (i.e. latest price \* quantity)
* Method **AddItems** to update the quantity by giving as arguments the latest price and the number of items to add. This method should return the updated quantity
* Method **AddItems** to update the quantity by giving as an argument the number of items to add. The method should leave the current price as it was and return the updated quantity
* Method **AddItem** to update the quantity by one and return the updated quantity
* Method **RemoveItems** to update the quantity by giving as arguments the number of items to remove. Quantities should not be able to be lower than 0. This method should return the updated quantity (or zero)
* Method **RemoveItem** to update the quantity by removing one item. Quantities should not be able to be lower than 0. This method should return the updated quantity (or zero)

It is suggested that you implement the OrderItem class initially and create a test project for it. Then implement the ShoppingBasket class and create tests for it.

You can add methods as needed to implement your code.