AberPizza

CS124 Project documentation

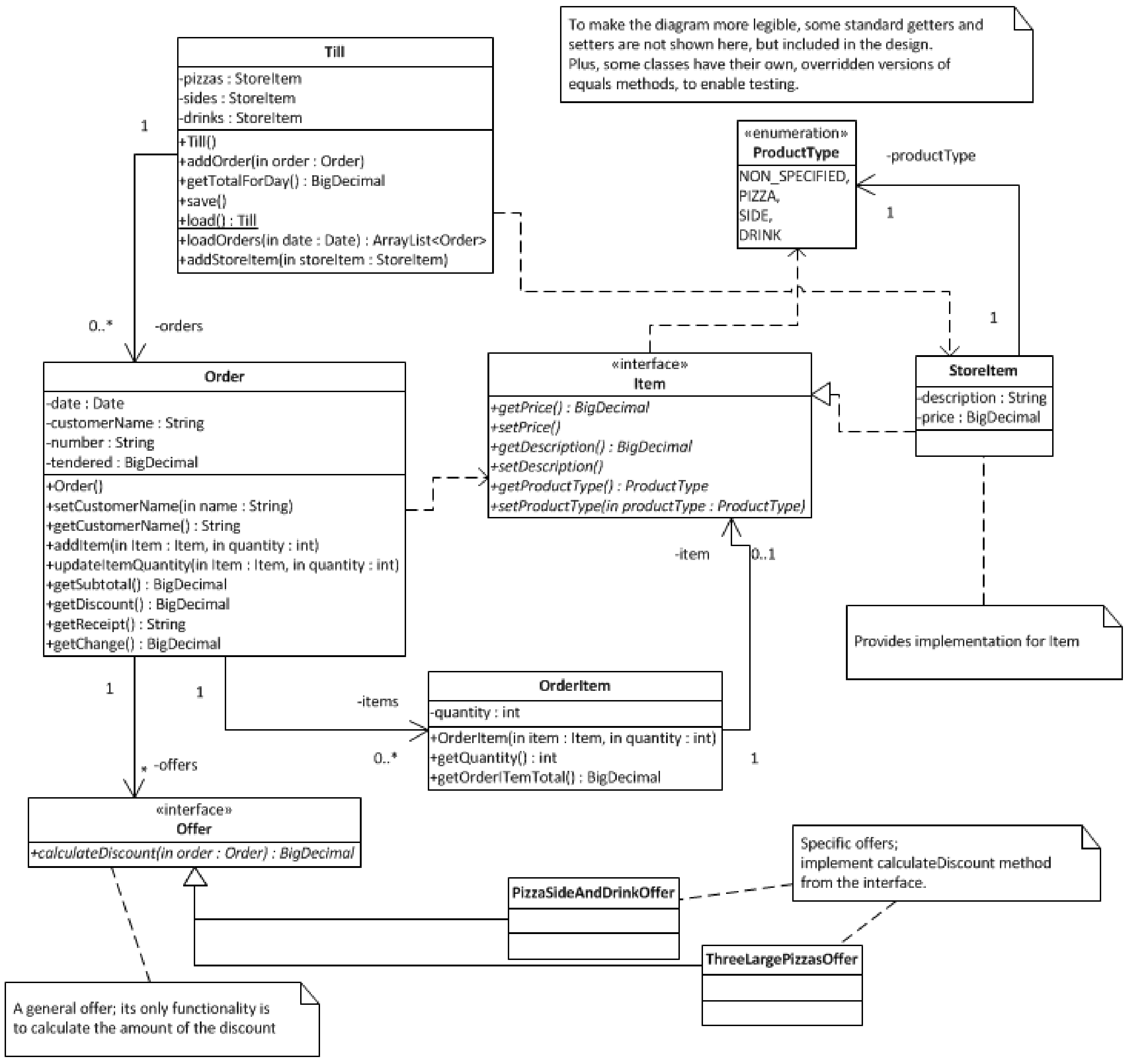
Author:
Kamil Mrowiec
kam20@aber.ac.uk

5/4/2012

Content

Data Structure Design

UML diagram



To make the diagram more legible, some standard getters and setters are not shown here, but included in the design. Plus, some classes have their own, overridden versions of equals methods, to enable testing.
Till
-pizzas : StoreItem
-sides : StoreItem
-drinks : StoreItem
+Till()
+addOrder(in order : Order)
+getTotalForDay() : BigDecimal
+save()
+load() : Till
+loadOrders(in date : Date) : ArrayList<Order>
+addStoreItem(in storeItem : StoreItem)
«enumeration» ProductType
NON_SPECIFIED, PIZZA, SIDE, DRINK
-productType
1
1
1
0..*
-orders
Order
-date : Date
-customerName : String
-number : String
-tendered : BigDecimal
+Order()
+setCustomerName(in name : String)
+getCustomerName() : String
+addItem(in Item : Item, in quantity : int)
+updateItemQuantity(in Item : Item, in quantity : int)
+getSubtotal() : BigDecimal
+getDiscount() : BigDecimal
+getReceipt() : String
+getChange() : BigDecimal
«interface» Item
+getPrice() : BigDecimal
+setPrice()
+getDescription() : BigDecimal
+setDescription()
+getProductType() : ProductType
+setProductType(in productType : ProductType)
StoreItem
-description : String
-price : BigDecimal
Provides implementation for Item
-item
0..1
1
1
-items
0..*
1
* -offers
OrderItem
-quantity : int
+OrderItem(in item : Item, in quantity : int)
+getQuantity() : int
+getOrderITemTotal() : BigDecimal
«interface» Offer
+calculateDiscount(in order : Order) : BigDecimal
Specific offers; implement calculateDiscount method from the interface.
PizzaSideAndDrinkOffer
ThreeLargePizzasOffer
A general offer; its only functionality is to calculate the amount of the discount

Description

Item

Since pizzas, sides and drinks are not different from “data” point of view – all those consist of description and price, I decided not to create separate classes for them, but use one general class instead. However, sometimes in the program it was necessary to recognise with what kind of product it was working. To make that possible, I added two simple methods to the interface – `getProductType()` and `setProductType(ProductType productType)`, based on enum type called `ProductType`.

StoreItem

Class implementing `Item` interface. It represents any product available for sale.

OrderItem

My implementation of this class is rather unchanged, in comparison to the initial design. I only added getters and setters for item and overridden version of `equals(Object other)` method (what was necessary when testing, to test whether `Till` has been loaded correctly).

Order

In this class I implemented the design given, and added few functionalities. I assumed that every order should have its own “number”, indicating the date as well, and decided to add such field as String (e. g. 3/20120501). Secondly, I think that the receipt should contain the amount of money tendered, and it is a good idea to keep that data in the order as well. Another vital issue is to represent the change, but for that a field is not necessary, since change can be simply calculated by subtracting subtotal from tendered, and that is what `getChange()` method does. Both number and tendered values have their getters and setters; I also overrode equals method there.
I used HTML in getReceipt() method, because it seemed to be the best way to get professionally-looking receipt. HTML also makes it independent of the platform, so a receipt may easily be exported to a file and viewed wherever necessary, keeping the formatting.
An order also has a final, static array of `Offers`, to calculate discount (more about that in a few paragraphs).

Till

When it comes to Till class, there I made probably most changes in comparison to the initial design. I implemented everything as was given, and added a few things. I decided to store here all StoreItems that are available for sale, in three ArrayLists, each for one group of items. That allowed items to be saved to .xml file (called settings.xml) and loaded when the program starts, and thanks to that, it was possible to introduce another functionality – adding new items to the Till from Admin menu, in the program.

My approach to saving and loading Till is slightly unconventional – I decided to save orders in separate .xml files, so that orders from one day are kept in one file. That gives flexibility in loading and storing data – when program starts, only orders from current day are loaded, and user may delete some files, leaving rest untouched. Following that thought, I separated loading orders from `load()` method, and added new one, called `loadOrders(Date date)`. That method returns an ArrayList of Orders, that has been taken on the day given as parameter; it is called in `load()`, with current date. The method is also helpful when displaying previous orders (more about that in GUI design section).

I found out that default `XMLEncoder` does not support BigDecimals, and created `MyXMLEncoder` class, that enables this functionality by setting custom PersistenceDelegate. My solution is based on an example I found on Oracle forum (https://forums.oracle.com/forums/message.jspa?messageID=4664166#4664166).

Offers

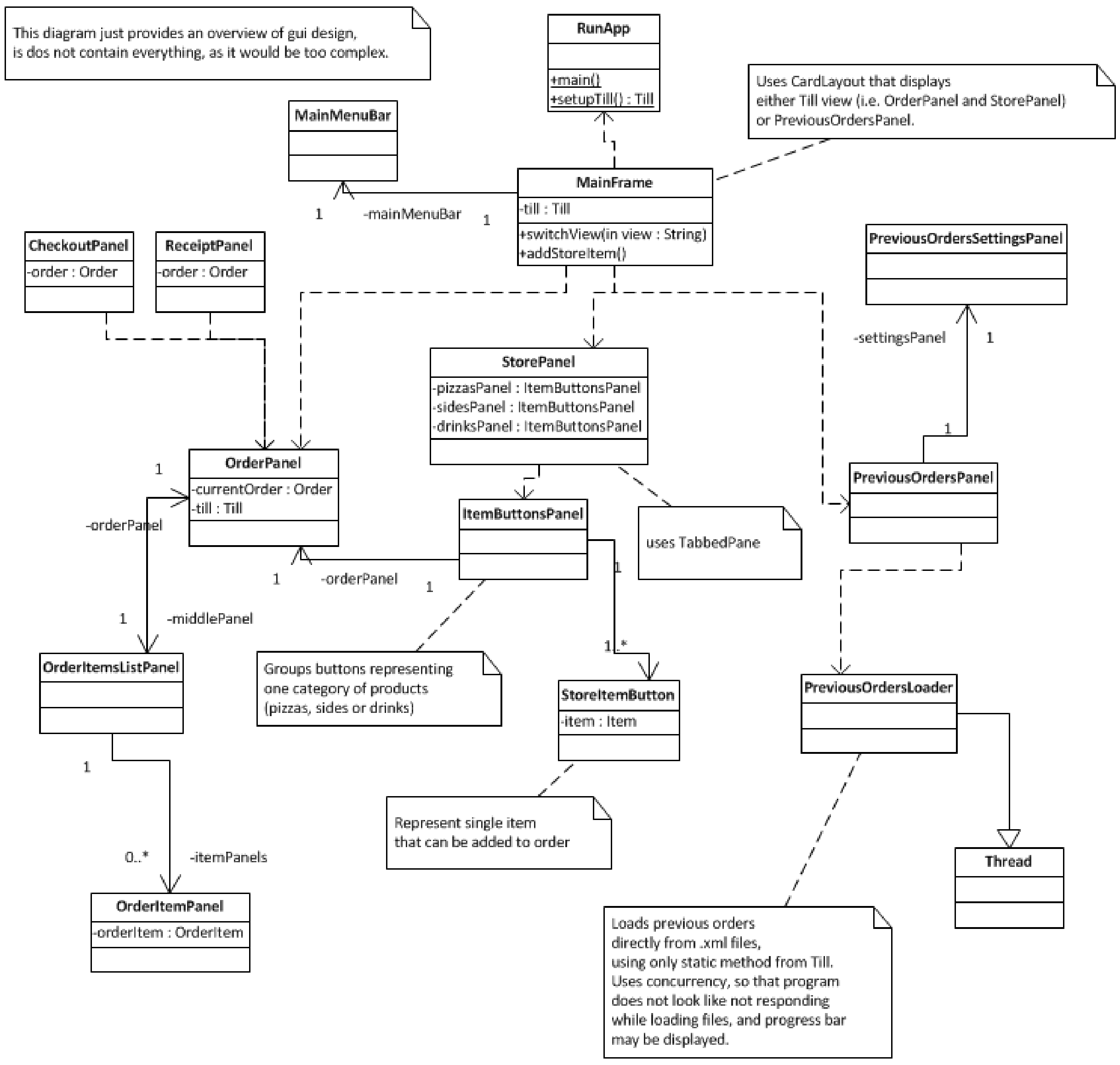
To implement different type of offers, I assumed that best solution is to create an interface with only one method – `calculateDiscount(Order order)`, that returns an amount of the discount, calculated for the order given as parameter. Every specific offer is simply a class implementing the interface. `Order` class contains a final, static array of type `Offer`, that contains instances of all existing specific offers. To calculate discounts, it simply uses foreach loop that runs through the array, calling `calculateDiscount(Order order)` on each element, and then picks the one with highest profit to the customer.

I implemented two different offers, like described in project specification. `ThreeLargePizzasOffer` applies when a customer takes at least three large pizzas, the cheapest one is then free. Other offer, `PizzaSideAndDrinkOffer`, takes place when a customer orders a large pizza, a side and a drink, and the discount is 20% of value of those three items.

The offers may easily be extended, all that needs to be done is to create another class and put an object of the class to the array in `Order`.

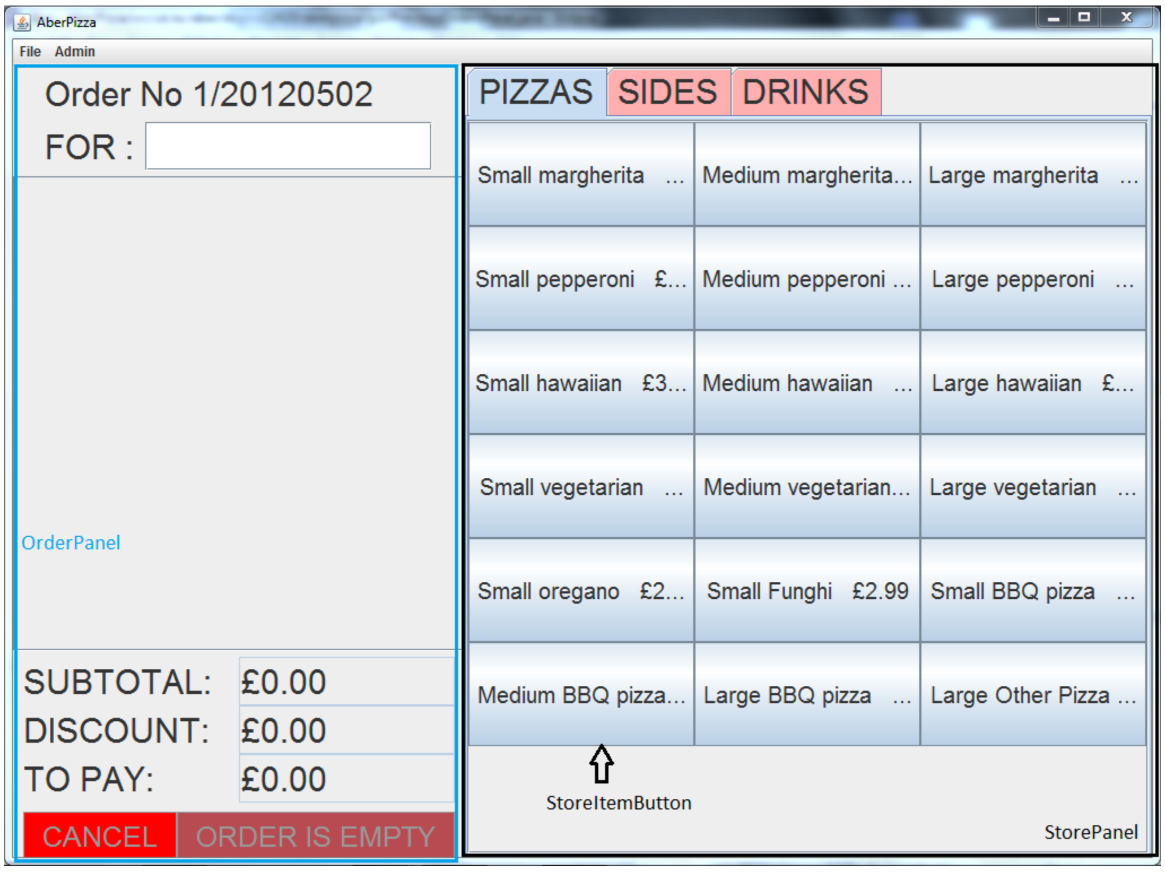
Graphical User Interface Design

UML Diagram

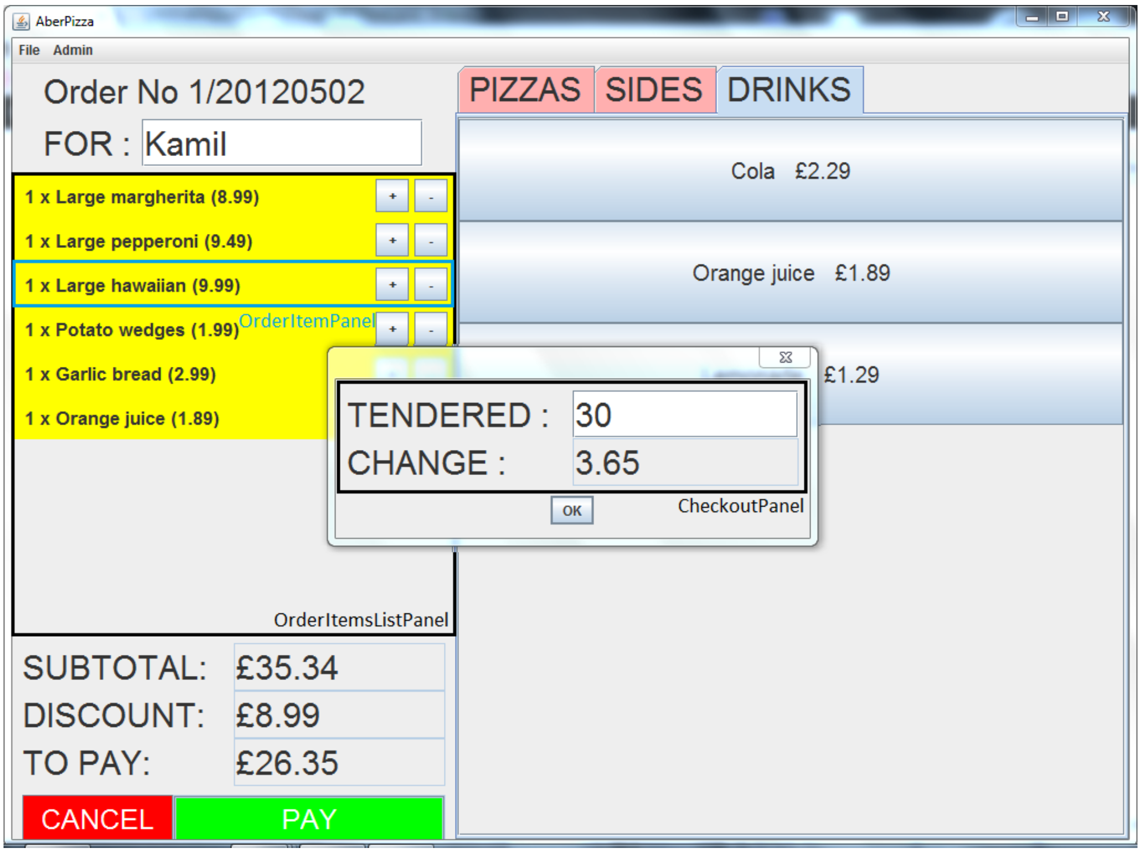

This diagram just provides an overview of gui design, is dos not contain everything, as it would be too complex.
RunApp
+main()
+setupTill() : Till
Uses CardLayout that displays either Till view (i.e. OrderPanel and StorePanel) or PreviousOrdersPanel.
MainMenuBar
MainFrame
-till : Till
+switchView(in view : String)
+addStoreItem()
1
-mainMenuBar
1
CheckoutPanel
-order : Order
ReceiptPanel
-order : Order
PreviousOrdersSettingsPanel
-settingsPanel
1
StorePanel
-pizzasPanel : ItemButtonsPanel
-sidesPanel : ItemButtonsPanel
-drinksPanel : ItemButtonsPanel
1
OrderPanel
-currentOrder : Order
-till : Till
1
-orderPanel
PreviousOrdersPanel
ItemButtonsPanel
uses TabbedPane
1
-orderPanel
1
1
1
-middlePanel
OrderItemsListPanel
Groups buttons representing one category of products (pizzas, sides or drinks)
1..*
StoreItemButton
-item : Item
PreviousOrdersLoader
1
Represent single item that can be added to order
0..*
-itemPanels
Thread
OrderItemPanel
-orderItem : OrderItem
Loads previous orders directly from .xml files, using only static method from Till. Uses concurrency, so that program does not look like not responding while loading files, and progress bar may be displayed.

Screenshots

Few screenshots depicting different parts of the GUI.



Screenshot 1 : Till view, default screen after launching the program



Order No 1/20120502
FOR : Kamil

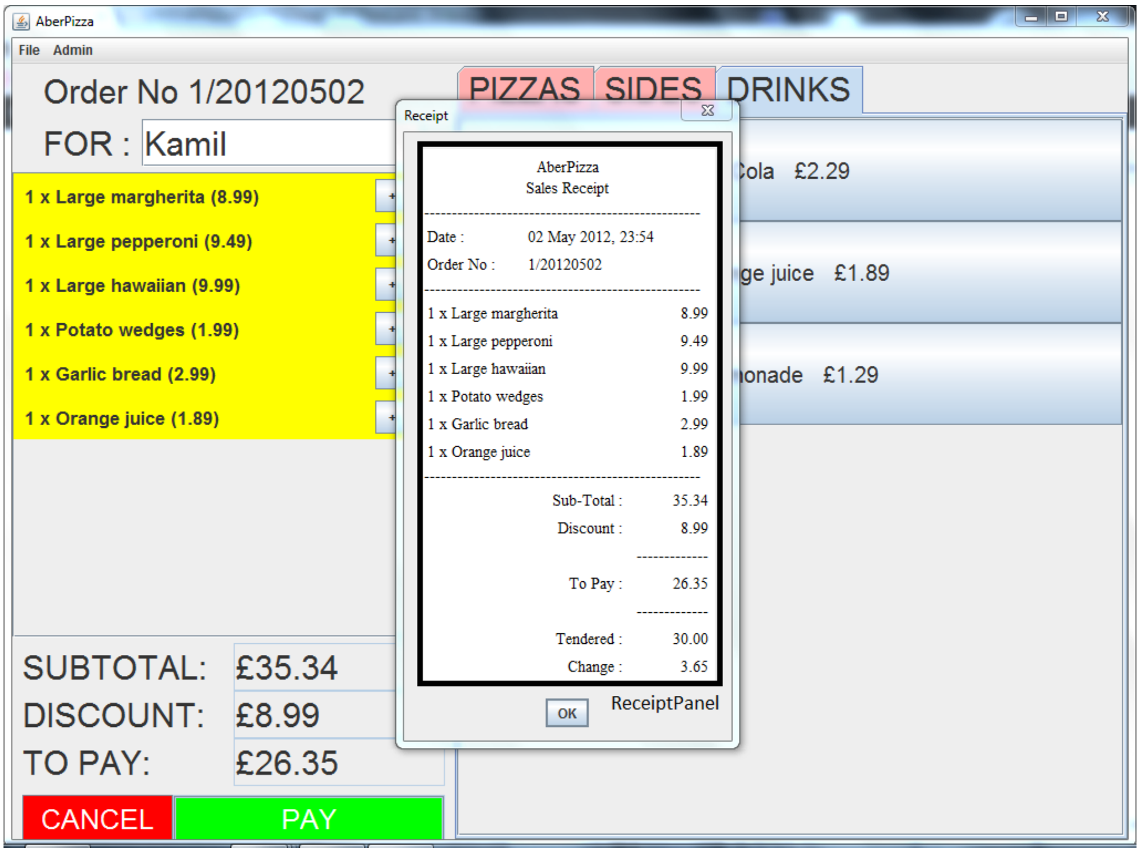
1 x Large margherita (8.99)
1 x Large pepperoni (9.49)
1 x Large hawaiian (9.99)
1 x Potato wedges (1.99)
1 x Garlic bread (2.99)
1 x Orange juice (1.89)

Cola £2.29
Orange juice £1.89
£1.29

TENDERED : 30
CHANGE : 3.65

SUBTOTAL: £35.34
DISCOUNT: £8.99
TO PAY: £26.35

Screenshot 2 : After adding few items to order and clicking PAY button



Order No 1/20120502
FOR : Kamil

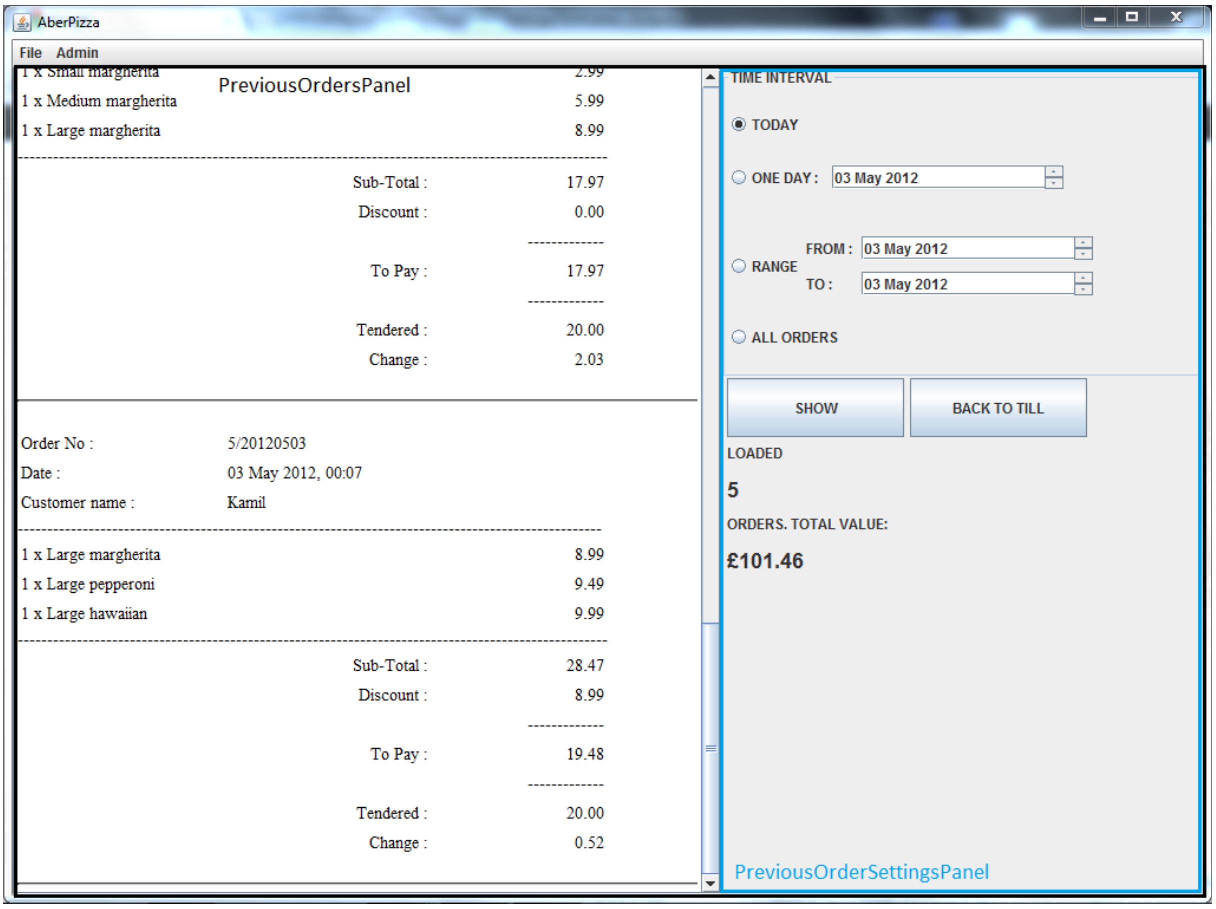
1 x Large margherita (8.99)
1 x Large pepperoni (9.49)
1 x Large hawaiian (9.99)
1 x Potato wedges (1.99)
1 x Garlic bread (2.99)
1 x Orange juice (1.89)

Cola £2.29
ge juice £1.89
onade £1.29

```
              AberPizza
            Sales Receipt
----------------------------------------
Date :       02 May 2012, 23:54
Order No :   1/20120502
----------------------------------------
1 x Large margherita                8.99
1 x Large pepperoni                 9.49
1 x Large hawaiian                  9.99
1 x Potato wedges                   1.99
1 x Garlic bread                    2.99
1 x Orange juice                    1.89
----------------------------------------
                    Sub-Total :    35.34
                     Discount :     8.99
                              ----------
                      To Pay :     26.35
                              ----------
                    Tendered :     30.00
                      Change :      3.65
```

SUBTOTAL: £35.34
DISCOUNT: £8.99
TO PAY: £26.35

Screenshot 3 : After confirming tendered amount, receipt displayed



AberPizza

File Admin

PreviousOrdersPanel

```
1 x Small margherita                                   2.99
1 x Medium margherita                                  5.99
1 x Large margherita                                   8.99
-----------------------------------------------------------
                               Sub-Total :            17.97
                               Discount :              0.00
                                                -----------
                               To Pay :               17.97
                                                -----------
                               Tendered :             20.00
                               Change :                2.03

Order No :              5/20120503
Date :                  03 May 2012, 00:07
Customer name :         Kamil
-----------------------------------------------------------
1 x Large margherita                                   8.99
1 x Large pepperoni                                    9.49
1 x Large hawaiian                                     9.99
-----------------------------------------------------------
                               Sub-Total :            28.47
                               Discount :              8.99
                                                -----------
                               To Pay :               19.48
                                                -----------
                               Tendered :             20.00
                               Change :                0.52
```

TIME INTERVAL

[✓ TODAY]

ONE DAY : 03 May 2012

RANGE FROM : 03 May 2012 TO : 03 May 2012

ALL ORDERS

SHOW BACK TO TILL

LOADED

5

ORDERS. TOTAL VALUE:

£101.46

PreviousOrderSettingsPanel

Screenshot 4 : Previous orders view

Testing

Data

To test the behaviour of all classes from data package I used JUnit framework. I tried to test every class in a detailed way, paying attention to every method. Those tests have been explicitly described in Javadoc comments. All of them succeeded.

GUI test table

TEST TABLE FOR GRAPHICAL USER INTERFACE				
Requirement	**Action**	**Expected result**	**Pass /Fail**	**Comments**
1. Start Till for day	Starting the program.	Main frame should load, till view should be visible and order taken that day and items for sale should be loaded.	P	
Till view:				
2. Add Items to the Order	Clicking one of big buttons representing store items.	Proper item should be added to the “list” of order items, total prices and discounts should be refreshed.	P	
	Clicking a button representing item that has previously been added to order.	Quantity of the order item should be increased and prices and discounts updated.	P	
3. Change quantity for an Item in Order	Clicking plus button in the panel representing one of order items.	Quantity of the order item should be increased and prices and discounts updated.	P	
	Clicking minus button in the panel representing one of order items.	Quantity of the order item should be decreased and prices and discounts updated; if previous quality was 1, the item is deleted from order.	P	
4. Cancel order	Clicking Cancel button when order and customer name are empty	Cancel button should be inactive, therefore no change is expected.	P	
	Clicking Cancel button when only customer name is filled in.	Customer name should be reset.	P	
	Clicking Cancel button	A dialog should be displayed,	P	

	when there are items in order.	asking whether user wants to cancel the order, and proper action should be done, dependant of what user chose.		
5. Pay for order	Clicking Pay button when either customer name is missing, or order is empty.	Button should be inactive, its background red, and proper message displayed on it, instead of “PAY” text.	P	Pay button interacts with users other input.
	Filling in the customer name and/or adding first item to order, so that both prerequisites are fulfilled.	Pay buttons background should turn green and button should get active. Should turn inactive again when required.	P	
	Clicking Pay button when order is complete.	New window should appear (checkout frame), that allows user to input tendered amount.	P	
	Filling tendered money field in checkout frame.	Change should be calculated and displayed in the frame. If tendered money is not enough to pay for the order, that fact should be indicated in change field.	P	
	Clicking OK in the checkout frame when tendered money is not sufficient.	Warning message should be shown and checkout frame should appear again.	P	
6. Show receipt	Clicking OK in the checkout frame when tendered money is sufficient.	Checkout frame should disappear; receipt frame should be displayed instead.	P	Receipt is created using HTML tags, so that it looks amazing.
	Clicking OK in the receipt frame;	Receipt panel should disappear, order panel should be clear for next order and number of order increased.	P	
	Choosing “Show previous orders” from menu bar.	View should be switched to previous order.	P	
Previous orders view:				
7. View sales History for the day.	Clicking Show button.	Progress bar should appear in lower right part of the screen. It should keep moving until all orders are loaded. When loading is finished: Previous orders from time interval chosen by the user should be displayed in text area. Under the buttons a text label should appear, displaying total number of orders and total price.	P	Loading orders takes place in separate thread, so that the program does not look like it is not responding. That allows to display a loading bar.

	Clicking back to till button.	View should change.	P	
8. Close Till for the day	Closing the program by either clicking on X button in upper right corner, or by choosing Quit option from file menu.	Before the frame is closed, the till should be saved to settings.xml, and list of order to proper xml file.	P	Orders from every day are saved in separate xml file, called: yyyy_mm_dd.xml

Self-evaluation

Summing up my work on this assignment, I am satisfied with the result. Not only did I not omit anything, I also made few additional things. For me, difficulty of the assignment was rather moderate; it was definitely time consuming, but I did not struggle with anything. Using numerous Swing components was sometimes complicated, and required some research, but Java documentation is very clear and accessible, and I found there answers to all my questions.

I implemented a few additional things to earn WOW points:

1. An option to add items for sale from the program.
2. In previous orders preview, user can choose time period from which orders will be displayed; it can be current day, other, specified day, few days or all existing orders.
3. Loading the orders runs in separate thread, so that program is constantly responding and progress bar is displayed.
4. Interactive PAY button – it will automatically turn active when customer name is given and the order is not empty, otherwise it cannot be clicked, and its label indicates what is missing.