CS12420 Assignment 2

AberPizza Till

Craig Heptinstall crh13

Contents

Introduction

For this individual assignment, I have been the given the task of designing, implementing and evaluating a program based on an already partially completed deign, which concerns about a till system for a pizza company, AberPizza. By firstly analysing the problem that has been assigned to me, I can then look at ways in which I will create my program. Starting with the main aims and goals of the till system, I will need to be able to create a UML class diagram, and add to it in order to represent aspects of my program such as the till methods that calculate the total price, and the data classes that store the price for loading at other times.

The second goal I can see from this brief is the creation of JUnit testing, which will be involved once I have both completed my design and the program, and will require me to write separate classes within the project containing test code solely testing the data classes and calculation methods etc. of the till. Using an IDE such as NetBeans which I have chosen for this assignment will provide me with useful tools in testing with JUnit, allowing me to quickly create tests based on methods and classes and run these to produce results which I will address later in the assignment.

Looking further into the brief, it requires that a graphical user interface is created for the program, which will link to the tills system, allowing users to interact much easier using buttons, list components and menus. I will integrate a GUI into the classes that contain methods for creating the till, saving tills, paying for tills etc. Displaying the total amount into a JLabel rather than in command line will mean the program will be much more user friendly and attractive.

Alongside the testing of classes using JUnit, I will provide further test tables in this document, looking at the checking of the GUI side of the application, where buttons, labels and textboxes will be checked if they work as intended. For example, checking a label has outputted the correct amount for the till, and if the output of the currency is to the correct amount of decimals. Testing will firstly involve creating a test plan table, in which I will outline the functions of each test, how I will test each, alongside an expected result and a test number. Following this, I will then actually test the whole GUI using the test plan, and any unusual or failed tests will be documented and screenshots put in place explaining the issue, and how I could or did fix it.

Javadoc is the final goal of the implementation of the program, and will require me to comment all the classes, methods and variables that are used throughout outlining what the function of each is, where each links to one another and why I chose to use them in certain ways. I will also place my name, title of the program and date created in comments at the start of each class which will ensure the javadoc produced will be neat, easy to read and attractive to any user. Again using the in-built utilities of NetBeans, I will be easily able to create an html document containing all the variables etc. I specified in my comments. Creating this html document will mean any readers will be able to navigate easily through my program descriptors.

In the final section of this document I will complete with an evaluation of my program, planning and design, and the testing of the application. In this section I will say what I thought went well, what I thought didn't, how I could have improved my program, and a self-evaluation of my performance throughout the project. Giving me a total mark out of 100 based on how well I thought I achieved the outlined task and added any additional wow factors.

The Brief

To list the requirements quickly for what I will need to include in my application, below show the targeted features and options it will need.

- Start the till for the day- inputting the staff name, followed by the customer name.
- Pizza, side or drink selection- This will involve giving the user the ability to choose a type of food, and then add this to a list of ordered items.
- Provide the option to add more than one on an item, showing the total quantity of each in the order list.
- Calculate the rolling total cost of the till, and the total number of items.
- Provide a list of offers available to the user, meaning when a certain selection of items is ordered, the related offers get added to the order list, and the price deducted.
- The user should also be able to cancel and order, which will involve resetting the till, top remove all items from the list of orders.
- A remove function should be implemented allowing users to remove specific selected items from the list.
- The ability to change the tills staff or customer and then re-display it.
- To be able to place the order and pay, bringing up a frame that shows the total, the balance remaining and the cash tendered by the customer.
- After paying for an ordered item, the ability to create a receipt showing the ordered items, the customer name, staff name, tendered cash, change and any offered deductions.
- To allow staff to save the till when items are in an order, and then load this again at a different time if the till is re-opened.
- To allow the till to be exited and the till closed.
- To provide an easy to use GUI look and feel, to ensure that staff can navigate easily and perform functions quickly.

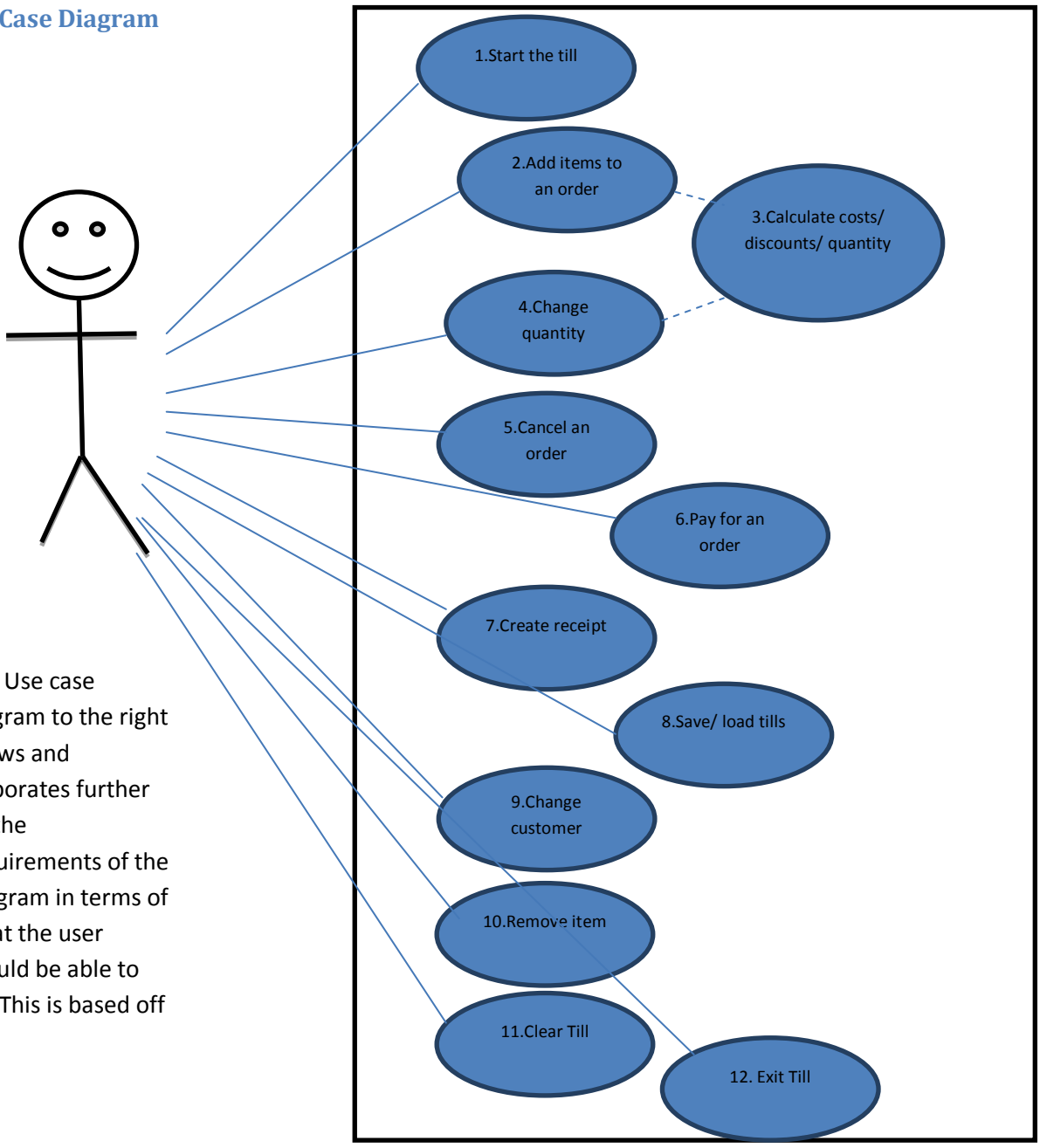
Planning and Design

In this planning and design section of my individual project, I will go into the details of my program plan, showing in the class diagrams how I will extend the design already given to me, and how the data, GUI and other important pieces of my program will work together to produce the final version. A set of other UML diagrams will back up my plan, showing the stages users will go through in order to perform simple tasks such as adding items, paying for tills or inserting the customer's name. These diagrams will provide me with a way of seeing what operations the user will need to perform to complete their task, and what the operations the program will need to perform itself. These step by step designs will provide me with detailed information on what methods, variables, classes and interfaces I will require in order to ensure the program meets the requirements set in the brief.

UML Diagrams

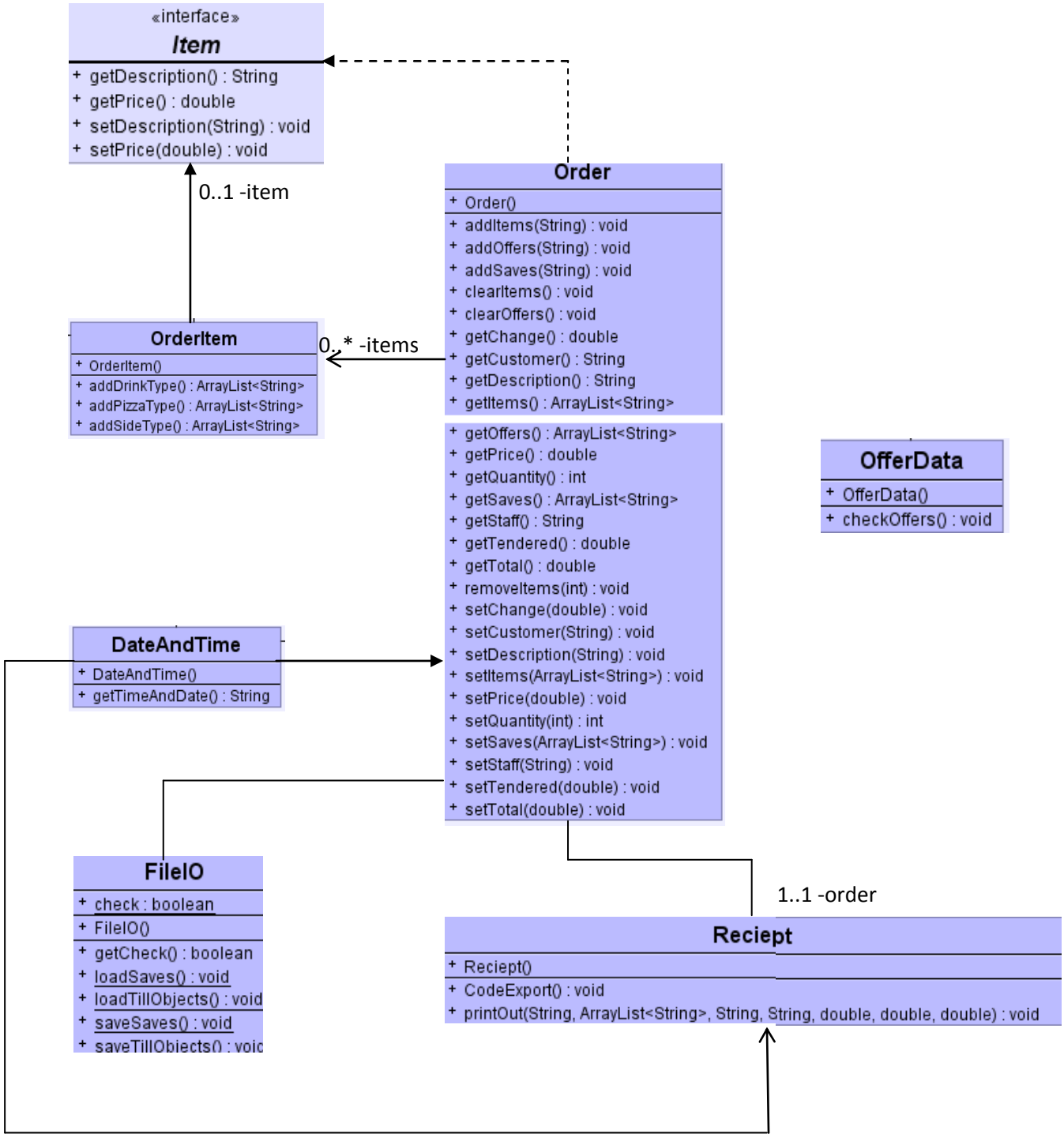
Unified modelling language is the best approach to the design of my program, and below I will show an array of different types of design, each covering different aspects of the program I will be creating. Starting with a use case diagram showing what operations the staff/ user should be able to perform using the program.

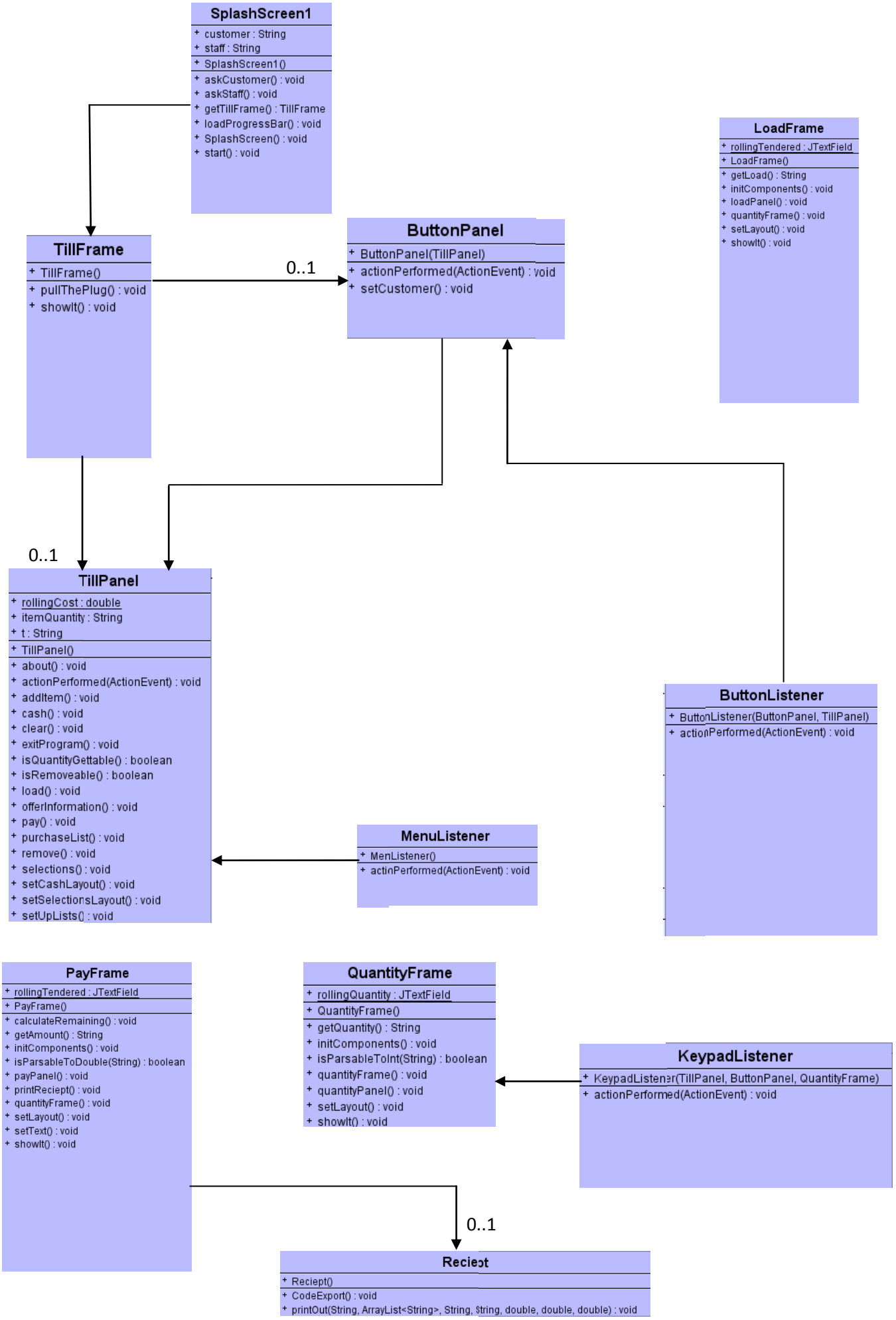
Use Case Diagram



The Use case diagram to the right shows and elaborates further on the requirements of the program in terms of what the user should be able to do. This is based off

Class Diagrams with Description




SplashScreen1
+ customer : String
+ staff : String
+ SplashScreen1()
+ askCustomer() : void
+ askStaff() : void
+ getTillFrame() : TillFrame
+ loadProgressBar() : void
+ SplashScreen() : void
+ start() : void
LoadFrame
+ rollingTendered : JTextField
+ LoadFrame()
+ getLoad() : String
+ initComponents() : void
+ loadPanel() : void
+ quantityFrame() : void
+ setLayout() : void
+ showIt() : void
TillFrame
+ TillFrame()
+ pullThePlug() : void
+ showIt() : void
0..1
ButtonPanel
+ ButtonPanel(TillPanel)
+ actionPerformed(ActionEvent) : void
+ setCustomer() : void
0..1
TillPanel
+ rollingCost : double
+ itemQuantity : String
+ t : String
+ TillPanel()
+ about() : void
+ actionPerformed(ActionEvent) : void
+ addItem() : void
+ cash() : void
+ clear() : void
+ exitProgram() : void
+ isQuantityGettable() : boolean
+ isRemoveable() : boolean
+ load() : void
+ offerInformation() : void
+ pay() : void
+ purchaseList() : void
+ remove() : void
+ selections() : void
+ setCashLayout() : void
+ setSelectionsLayout() : void
+ setUpLists() : void
ButtonListener
+ ButtonListener(ButtonPanel, TillPanel)
+ actionPerformed(ActionEvent) : void
MenuListener
+ MenListener()
+ actionPerformed(ActionEvent) : void
PayFrame
+ rollingTendered : JTextField
+ PayFrame()
+ calculateRemaining() : void
+ getAmount() : String
+ initComponents() : void
+ isParsableToDouble(String) : boolean
+ payPanel() : void
+ printReciept() : void
+ quantityFrame() : void
+ setLayout() : void
+ setText() : void
+ showIt() : void
QuantityFrame
+ rollingQuantity : JTextField
+ QuantityFrame()
+ getQuantity() : String
+ initComponents() : void
+ isParsableToInt(String) : boolean
+ quantityFrame() : void
+ quantityPanel() : void
+ setLayout() : void
+ showIt() : void
KeypadListener
+ KeypadListener(TillPanel, ButtonPanel, QuantityFrame)
+ actionPerformed(ActionEvent) : void
0..1
Reciept
+ Reciept()
+ CodeExport() : void
+ printOut(String, ArrayList<String>, String, String, double, double, double) : void

The two previous pages display my UML class diagram for the project. The first of these is for the data side of the program, the second for the GUI section. I decided to split these into two divisions to avoid creating too much of a complicated class diagram and also in order to be able to show where the separation lies between the program code. Below, I will give a brief description and explanations of the classes and methods themselves.

Item<<Interface>>
The interface of the data, this interface outlines a few methods used in the data class for the order including the get and set methods for the descriptions of the items and also the prices of the items. By following this interface, it will ensure that the order is kept consistent and information will not be missing.

Order
This class implements the item interface and contains the key methods of the program, such as adding items to the item list, setting the list of ordered items, getting the list of ordered items, and getting the details of each ordered item. It also stores the staff and customer name, sets the quantities of items and the price of the total order. This class interacts with the till panel class in the sense that the buttons that will be pressed such as adding an item will link up to this classes add item method and then update the array list of order items.

OrderItem
This class will be used rather than enumerated type classes to store the data for the pizzas, drinks, and sides available to select. It will store the description and prices of each item, and will be used to fill the JList within the till panel.

DateAndTime
The date and time class will be a small class that will primarily be used to get the current date and time based on the user's system, and will be sent to a string. This will be passed to the Order class to be used within each till order. It will also be sent to the receipt class in order to produce the receipt date and time of the transaction.

OfferData
The offer data class will store the data for the possible offers, and will check if any offers are available based on what items are within Order. The check offer methods are run every time the order is updated within the Order class, to keep the offers up to date and correct. It will be easy to add new offers by copying the existing offers and changing them.

FileIO
This will be responsible for the saving and loading of the orders. It will store the total cost of orders, the customer and staff names, along with the date and time of orders. This class will save to an XML file along with reading them. It also has a Boolean variable that will come from a check method which checks if it is possible to save and load. If this is not possible, an error will be thrown.

Receipt
This class exports receipt data from the order class and turns it into a text document by writing to a file selected by the user into the system. In the methods PrintOut, it takes in a set of strings and doubles to represent the items, names of staff and customer, and the data for the date and time. It then exports this into a printable receipt.

SplashScreen1
The Class SplashScreen1 will run after being called from the runnable class, and will load the splashscreen method, which creates a loading screen, followed by the start method. The start method creates a new instance of the till frame and till itself.

TillFrame
The Class TillFrame, holds the TillPanel, and also holds the menu bar, button panel and listener for the menu items. This is loaded from the SplashScreen1 class after startup. Also a new case of the Till Frame will be created and the old one closed when the new till button is pressed.

TillPanel
Contained within TillFrame. Holds the main section of the program, including the ability to add items, remove, clear the till, and create a new one. It links to all the data classes, getting and setting data for the items within the order lists.

LoadFrame
The Class LoadFrame opens a frame displaying a list of possible loads, then allows the user to select one, getting this and sending it to the FileIO class. This will get and set the data within the order class. Will be a simple frame, based on a load button, cancel button and a JList of possible loads based on the save files created.

PayFrame
The Class PayFrame. This class opens the pay frame, displaying the amount required, tendered, and remaining, and allows the staff to input the customer amount given. From that, the class will produce the change, and then create a receipt to a chosen location using the JFileChooser.

QuantityFrame
The Class QuantityFrame. Creates a frame displaying a keypad where the customer can input the amount of a certain item required, and then click 'ok' to send this to the Order class.

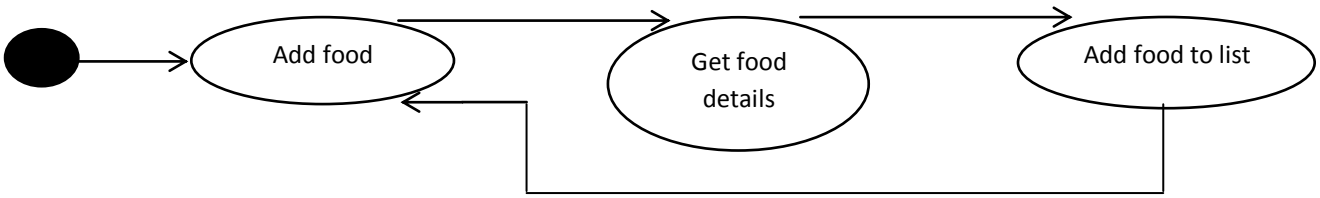
ButtonListener
Listener for ButtonPanel to allow program control buttons to perform functions when clicked.

KeypadListener
Listener for PayFrame and QuantityFrame to allow program control buttons to perform functions when clicked.

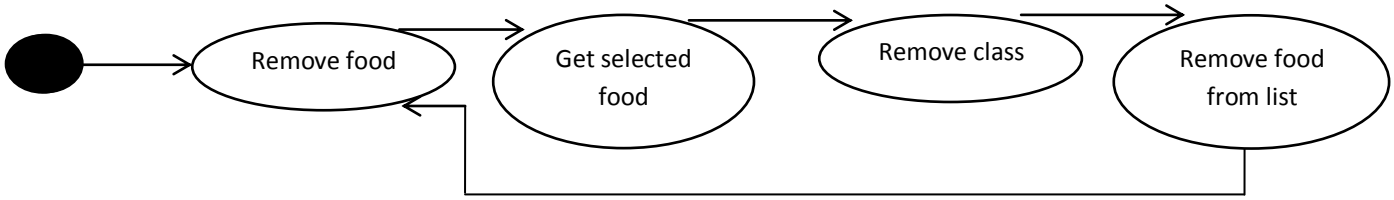
MenuListener
Listener for menu bar inTillFrame to functions to be performed when menu items are clicked.

State Diagrams

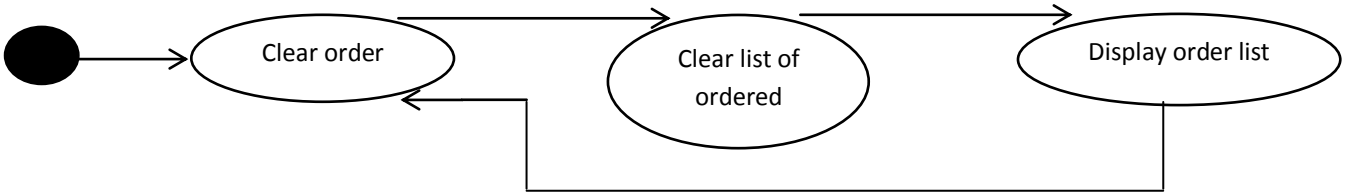
State Diagram 1: Adding food to the order



State Diagram 2: Removing food

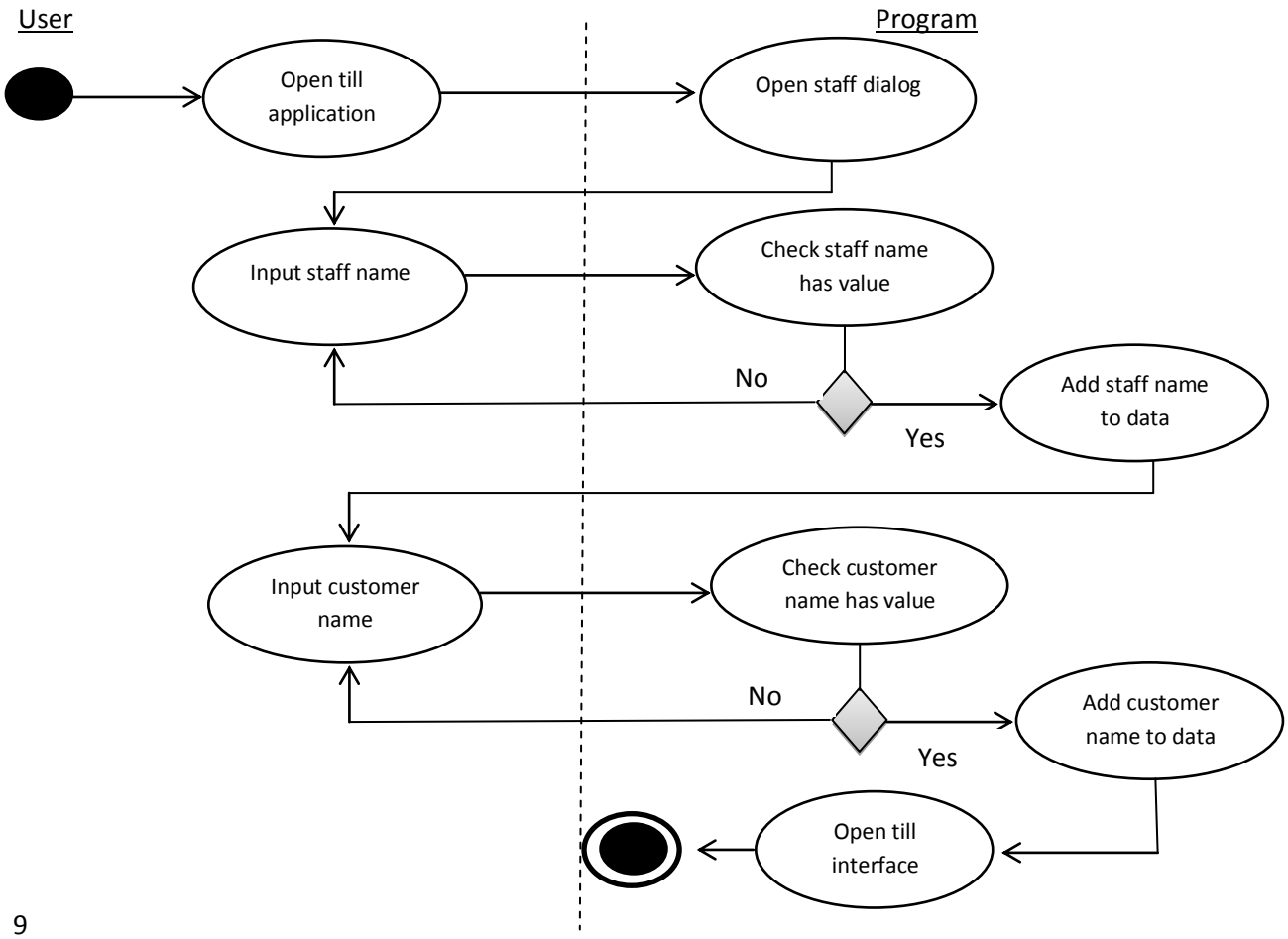


State Diagram 3: Clearing the order

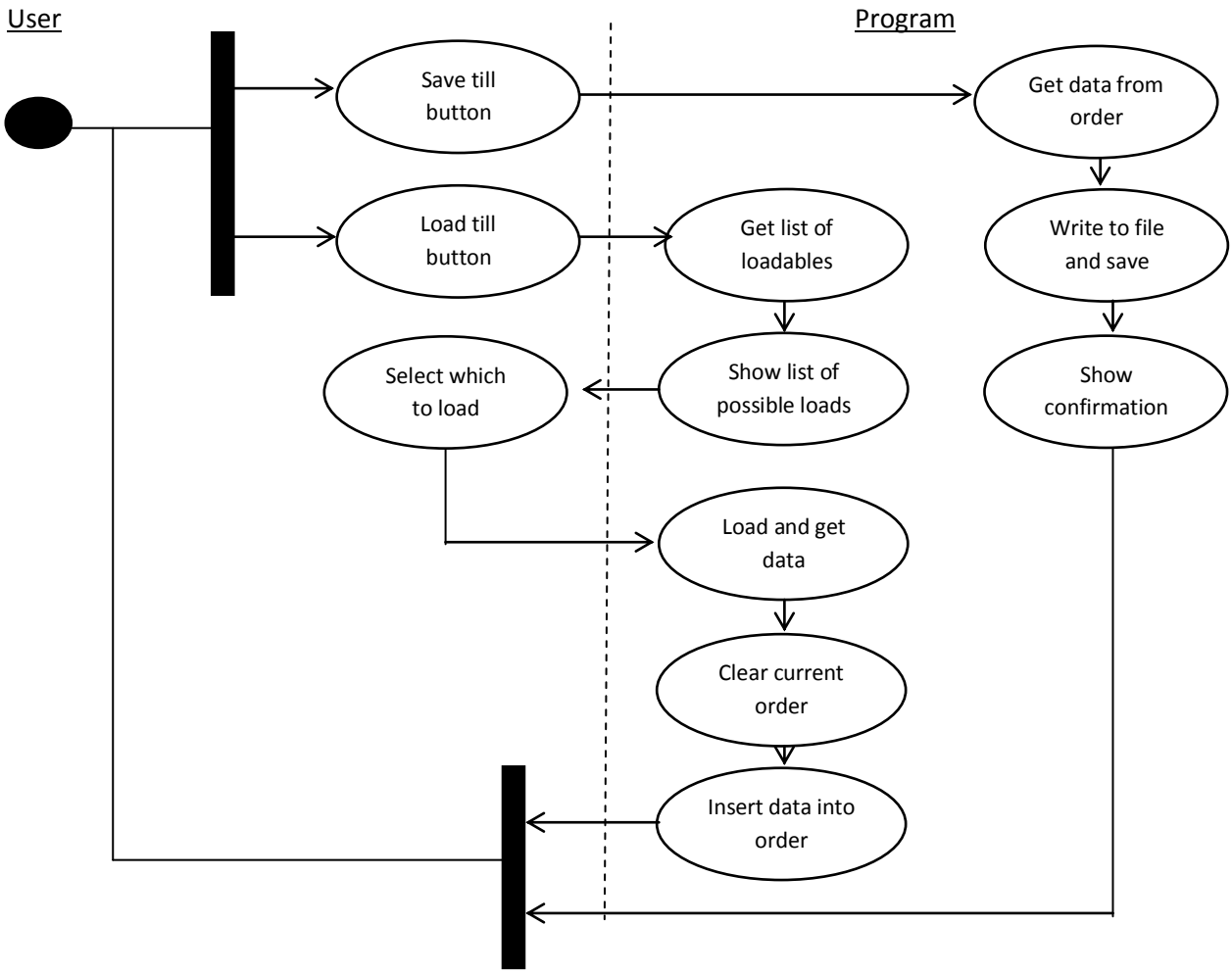


Activity Diagrams

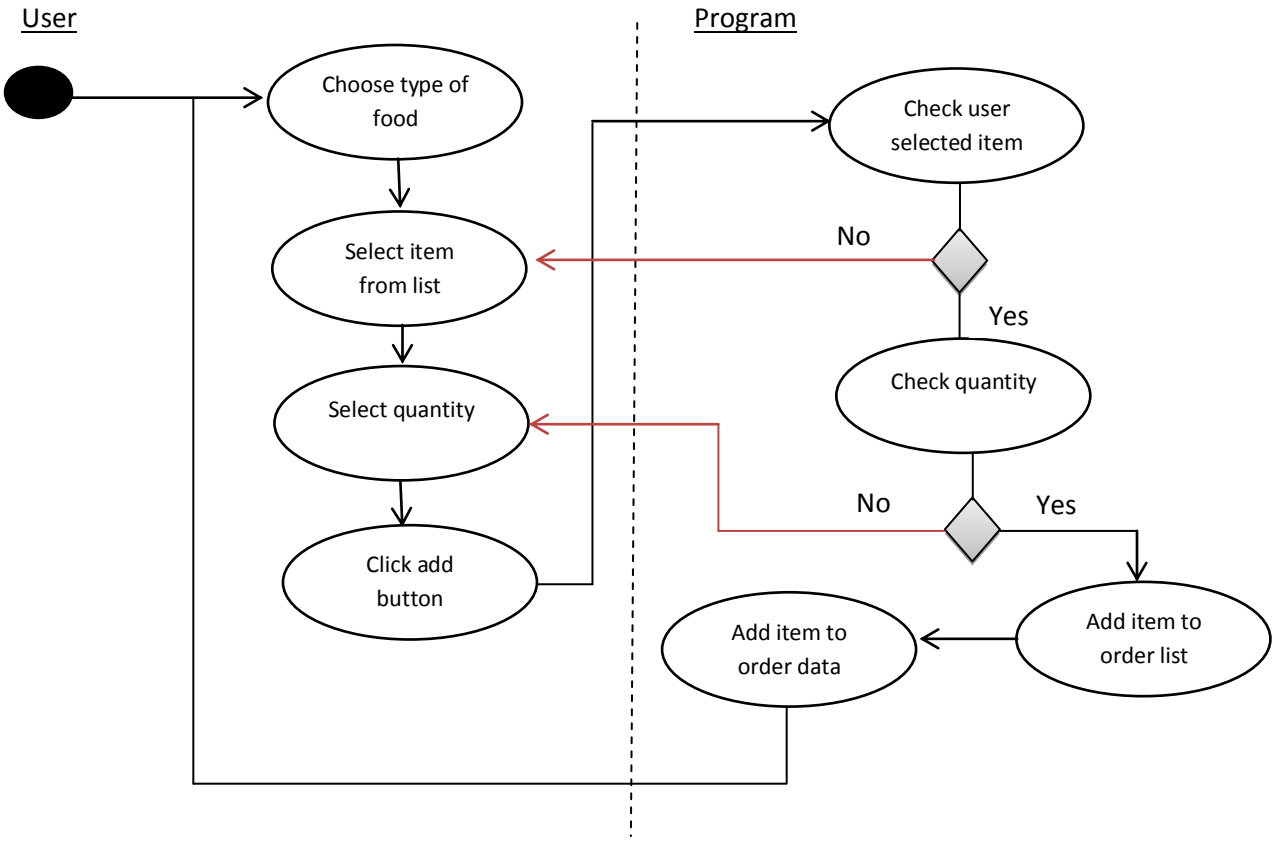
Activity Diagram1: Starting the till



Activity Diagram2: Loading and saving the till

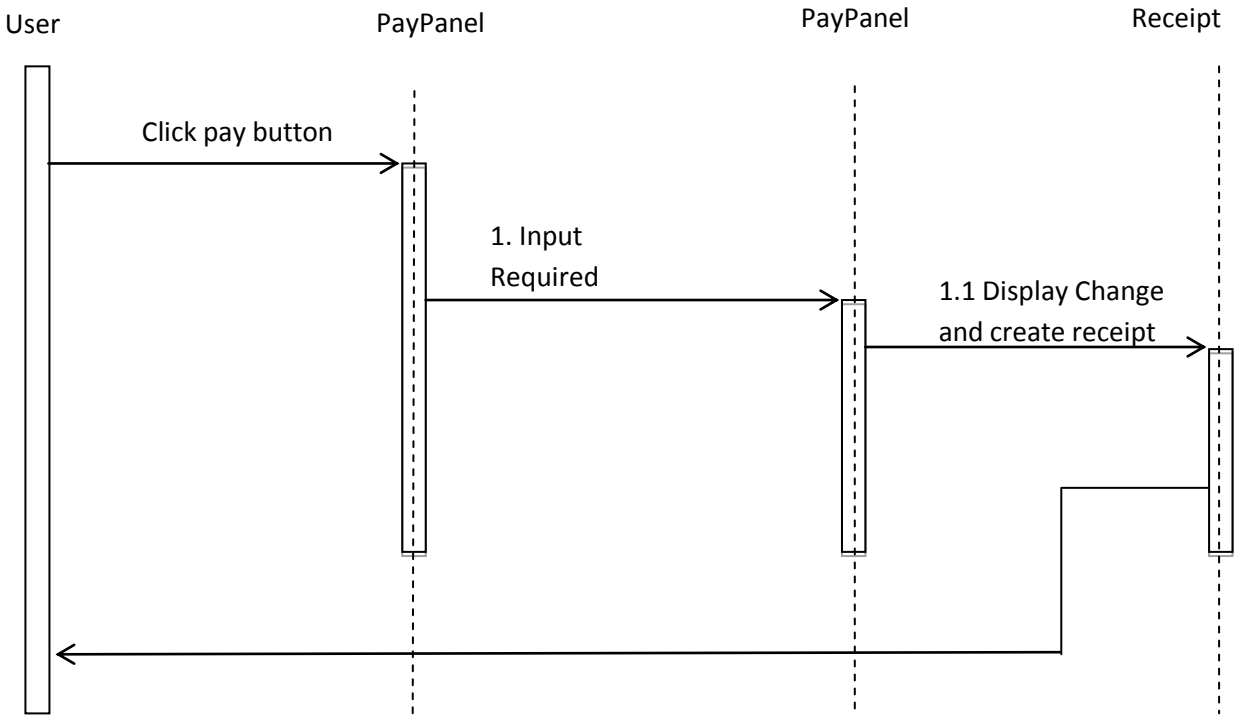


Activity Diagram 3: Selecting and adding items to the order

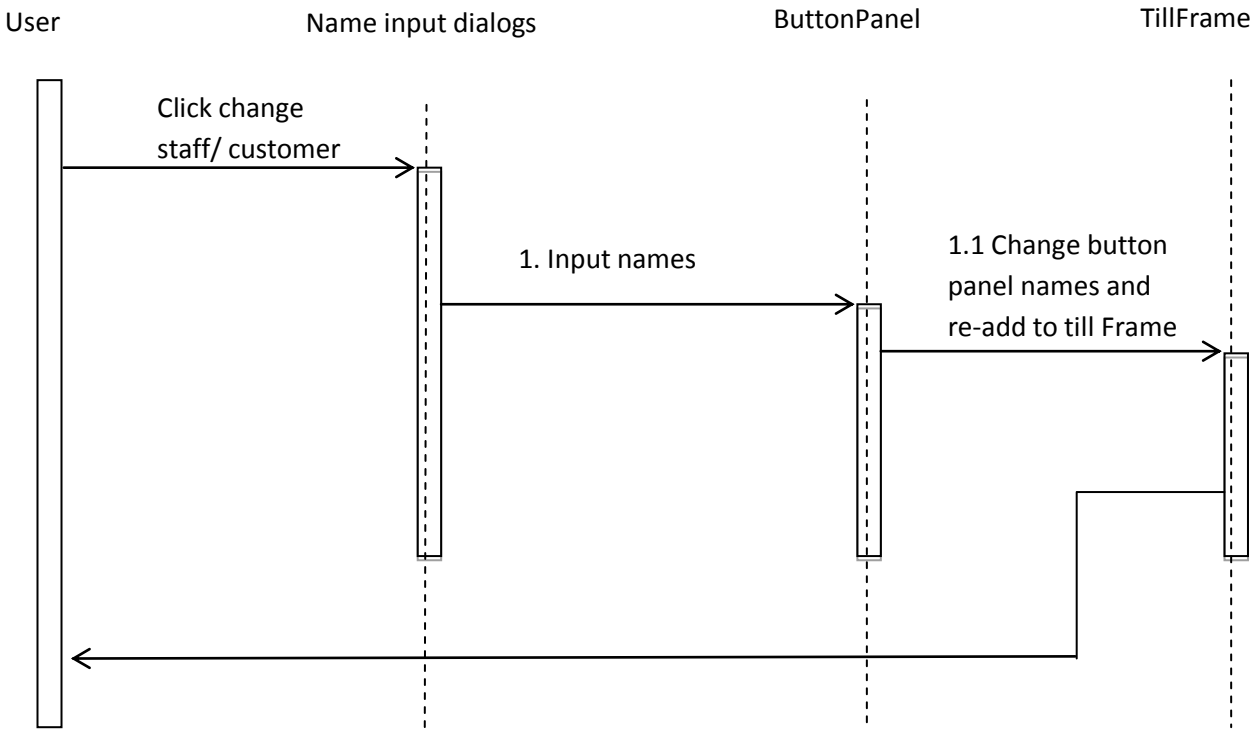


Sequence Diagrams

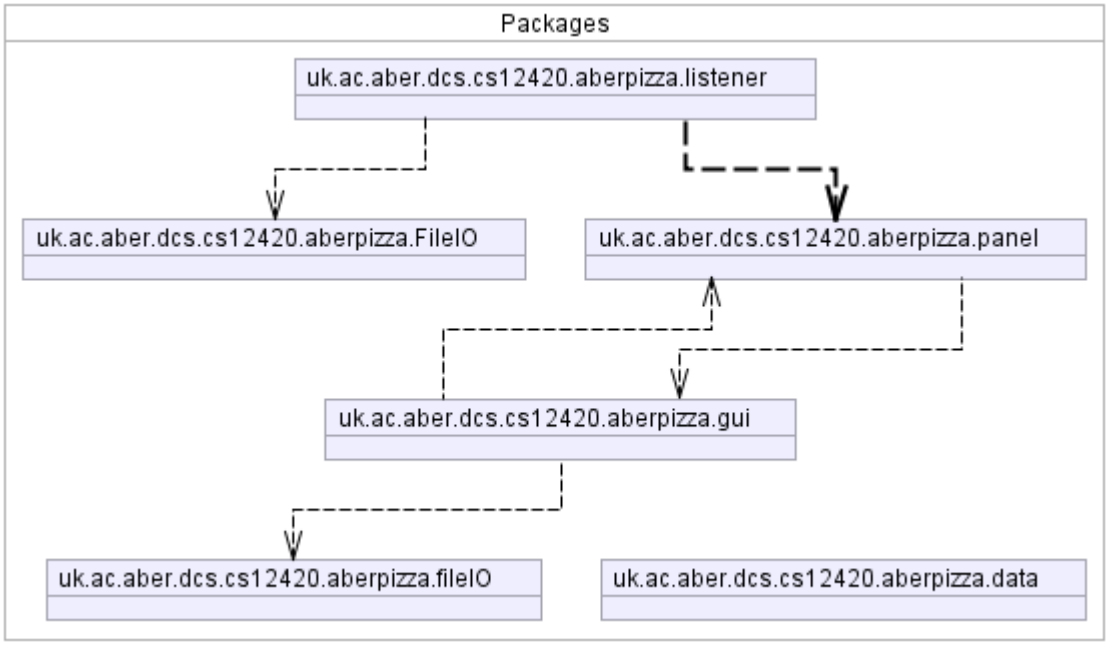
Sequence Diagram 1: Pay and create receipt



Sequence Diagram 2: Change staff and customer names



Package Diagram



UML Justification

Now I have completed the creation of my UML diagrams, I will now justify my design, and describe why I have designed the application this way. Starting with the Use case diagram I created based on the design presented to me, I simply listed here the actions the user/ staff would be able to perform using the till system. For instance, adding products to a list of order items, saving, loading, changing the customer name etc. I made this reflecting back on my description of the brief and what different requirements there were.

Next up in the UML design was the class diagram. The class diagram allowed me to organise the position of the methods, classes and relationships within the program,some of which were already given to me at the beginning of the brief. So I took the class diagram that had already been designed and added the rest of the classes, such as the graphical interface side of it, the creation of the receipts, the creation of save and load files, and also operations such as selecting quantities. Rather than performing backwards engineering and creating the class diagram based on some already written code, I wanted to fully plan the entire program out covering the whole brief before I started implementing it.

I decided to make state diagrams for my project also, which showed some simple loops and paths around my program. In the first two, I showed how the system would handle adding and removing items from an order. For instance, removing an item from the order meant starting with the user, who would begin by selecting and pressing the remove item button. The program would then go its part by getting the selected value from the list, and would then remove this from the order. It would finish the loop by displaying the list of ordered items again lopping back to the start of the operation if the user wanted to delete another item. State diagrams although not detailed, they show the loops and way the operations are gone through in large steps. Clearing the order was also shown in this way, from the user selecting the operation, up to re-displaying the list again.

After the state diagrams I created a set of activity UML diagrams. Activity diagrams help show a finer more detailed set of operational steps when performing actions. In activity diagrams I have the option to add loops, if statements, and a range of possible steps. I could use this type of UML for instance where the user started the till. Beginning with the user opening the application, then the program popping up with an insert staff name box where the user would type their name to move on to the next step. The diagram for this displays an 'If' statement which checks whether or not the input was empty. If the input was empty, the diagram then shows the program looping back to the same input box. If the input was not empty, then the program would repeat similarly for the customer name and then finishing by opening the till.

In another activity diagram I created, I displayed the way in which the application and user would interact when saving or loading their till. At the start of the diagram, I showed how the user was faced with two options, either to save or load, once they pick their option, two different paths would lead, one which would then make the application load the list of possible loads, for the user to select and load up, whilst the other would save the data of the current order and then add this to the save file. Both paths would meet up at the end, and then loop back to the beginning where the user could re-load or re- save the till. These diagrams are very useful in terms of the order operations must be done, where if statements take the user, and how different operations can link together.

My final piece of UML I used within this programming assignment was the use of sequence diagrams. Sequence diagrams are a little alike the state diagrams I first used, although are more detailed because of the way they show what classes the operations come from, and even the methods that perform the actions. The first of the two was the paying and creation of the receipt for the till. Again, using this type of UML helped me display the loops and relationships between operations.

To summarise on the use of my UML, and the overall planning aspect of my program, I can say that using the widest variety of UML as possible, and with descriptions will certainly help me towards implementing the program design, including the possibility to add any extra features I may wish or require. Designing an application like this first before implementation allows me to extend it if I need to easily, especially with the use of the interface which will allow future developer to simply add any extra methods to the till. It is important I now reflect this design in my completed program to the required standard and brief.

Implementation

Reasons for my Implementation

In this shorter section of the assignment, I will look at my approach to implementing the design, how I made any changes to it, and what additional items were implemented. I began writing my code based on the design given to me at the start, by creating a simple interface and the till class. I wanted to get the till operational in command line first, because from that I would be able to add the GUI, and then add more sophisticated actions such as the loading and saving, the payment frame, and the receipt creation.

Using the package folders given to me in the design brief at the start of this assignment, I was able to easily organise and place my classes and any new classes within the data, GUI, panels and file input/ outputs. This helped keep my program entirely organised, keeping bits of code that used the same interface within the same package folder.

Once I had gotten the majority of the program working up to the GUI implementation stage of the program, I was able to begin the stage of producing an interface which the user/ staff member could interact with when it came to selecting items of food, which I chose to be done through clicking on a selected list item. Making the interface of my program as graphical as possible meant that users of it would be able to use the order system with minimal use of a keyboard. I can show an example of this when it come to my quantity and pay panels. In both, I decided along with a text input box, that users would have the option to input numbers by clicking on a selection of buttons each carrying a digit. I also provided the option to delete the last character using a button, again minimising the need of a keyboard. This would act as an ideal solution for touch screen till systems.

Interlinking the graphical user interface with the data classes of my program proved most what simple, yet I had a few problems along the way in terms of getting the selected data from the list box containing the possible items to order. I decided to use a substring to solve this issue, which meant when a user selected and added an item, the program would get the description of the item by looking at text after a certain number of characters. By getting the substring of the items, I could easily compare this with my orders items, to check if any already existed, and if so, the program would then know to increase the quantity value rather than adding a repeat of the product.

Once I had got most of the visual GUI functioning properly, I then moved onto the save and loading of the orders customers would be placing. This meant following by following the design I would need to use XML language in order to save and load the objects including lists of items in the order, the total costs, item quantities, dates and times of the orders, and the staff and customer String names. Due to a lack of time in the project, I found that producing an XML save document would not be possible in the time frame I had left. As an alternative, I chose to use the output and input streams which allowed me to write the order information directly into a file type of my choice. This was a quick way to create the save and load class, and gave me greater flexibility in the save file type allowing users to open the saves in a text editor. I also decided that

This takes me to the receipt section of the implementation, in which I found very easy in terms of writing to txt file from my orders. I simply grabbed the data from the order data class and then passed this through the constructor of the receipt.

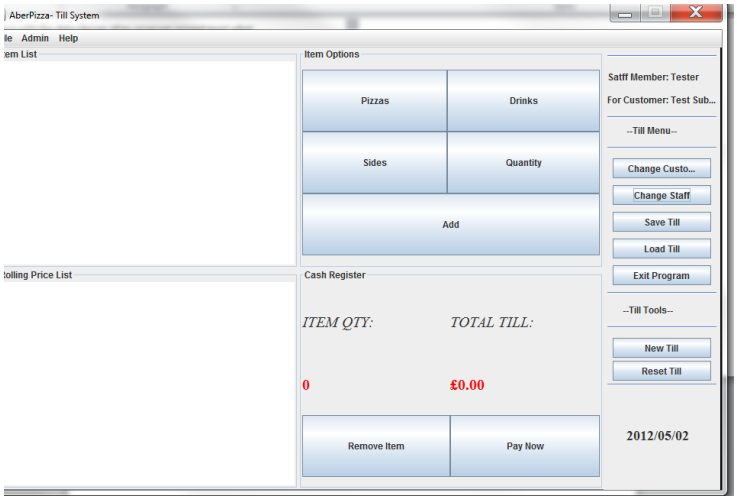
The pay panel was the final piece of implementation I added. This involved making yet another frame, with similar keypad buttons as the quantity panel, and then importing the data form the order, and ensuring it displayed for the user to see. The user would then input the amount tendered.

I Integrated some data in with the graphical user interface classes in order to use the data directly and then updated the data once an action had been performed using the buttons etc.

To summarise on the changes and additional features I added to the program I created based on the brief:

- Changed the way the program saves and loads, creating a load frame myself rather than using the JFileChooser.
- A scrollable JList box was implemented, meaning users could never lose the data if the list gets too long.
- Used a double type variable rather than the big decimal that was planned. I found this easier to use.
- Added a few colour changes to the text in the program, such as the total cost which came up as red. This was more visible and more attractive.

Based on the changes of the program I have made, and how I feel the implementation has gone, I think I have met the brief to a good deal with this program with a couple of exceptions. It does most of the methods and operations I had planned, and successfully stores and sets data across the classes when adding, removing and clearing the till items.



To the left is a screenshot of my graphical user interface. As you can see, the till item and order item lists are to the left, with the control buttons to the right. I have created a spring layout sidebar, along with a menu bar to maximise control in terms of creating a new till and resetting the till.

How to run my program

There are two ways of running my program:

- You can run by command line by simply locating to the java files, then compiling “javac *.java”. The program could then be run: “Java Runable.java”.
- The alternative way would be to simply run the executional jar file.

Testing

The testing of my program followed the implementation. Once I completed creating my final till application, I needed to test if not only the program met the requirements of the brief, but of my design also. I wanted to check if the program worked properly in extreme, boundary etc. situations where the program might be applicable to fail, through exceptions, or display incorrect or unusual data. In this section, I will display how I tested my program both in JUnit, and the same for the actual GUI of the program using test tables, including a test plan. Any fail tests will be corrected and fixed, where I will then report this below my tests saying what I did to fix the problems. I will start below by giving an explanation of the JUnit testing I performed on each class required, how the tests went, and how I fixed any problems.

JUnit Testing

Using JUnit testing for the data classes of my program proved the easiest and most efficient way to see the capabilities of my program, along with if it firstly performed the actions I required in terms of adding items to an order, removing them or clearing them. With JUnit testing I could try different pieces of data easily by using similar test code but just changing the input of the test. I tested each of my data classes, including the File Input/ Output class and receipt class.

I started by testing the smallest of the data classes which was the date and time class. This class had to be tested to check whether or not it was getting the correct time and date when it was being run. I performed this test quite easily by using the asserEquals method to check that the method result produced the same result if the method contents were in the test class.

I then moved onto the test class for the order class, this class which contained the majority of the get and set methods was easy enough to test, by simply testing if each method would set and then get the data again using the assertEquals method. I checked here by adding some data in, then checking the data had been set correctly. As for the remove and clear methods present, I simply used the remove method and clear methods, and checked the size of both to see if it had been reduced correctly/ reduced to the value of 0.

The offer data class was next in the Junit Tests. Here I simply added a combination of different pizzas and drinks I knew to make an offer. Then by running the method to check the offers, I tested if the offer I thought to be present actually was produced. I did the opposite of this by adding no items into the order, and then checking for a null value of offers.

As for the class which added the pizzas, sides and drinks to the product list, I create a set of tests which simply checked if an array of pizzas, drinks or sides matched the methods return value. If these two arrays matched, then the method was returning correctly the results.

Finally for the save, load and receipt methods of the program, I used JUnit testing here to check the save file was being created, loaded successfully and that the variables and data was not being lost. It also created a receipt using some dummy data to check if it was producing the receipts with the correct content. Throughout the testing of the data classes, any problems found with the tests could be fixed easily and quickly with the use of JUnit and I found this to be a very useful and satisfactory way of testing my data.

GUI Test Tables

In this section of my testing, I will now look at the testing side for the GUI of my application, which will firstly involve creating a test plan table, where I will outline the test element (what I will be testing), a description of the test I will be undertaking, and an expected result for each test. After this, I will make a similar table, extended from this one which will contain another two extra columns. These will be the actual result column, and the pass/fail column outlining if the selected test did as I presumed. In both test tables each test will hold a reference test number so I can talk about a test if it has failed and how/ if I solved any issues.

Test Plan

Test Number	Test Element	Test Description	Expected Result
1	SplashScreen1	Test if the splash screen load bar shows.	Splash Screen should load bar across.
2	SplashScreen1	Test null input repeats input box.	A null input should produce the input box to repeat appearing.
3	SplashScreen1	Insert correct text input.	Correct text should move the program to the next stage.
4	Till Panel item buttons	Test if clicking item buttons, possible item list changes.	When an item type is pressed, the item list should refresh showing new possible items.
5	Till Panel Select and add item	Test if user can select an item and then add this to order.	Should add the item selected to the order list.
6	Till Panel Select quantity open	Test if quantity frame opens when pressed	Quantity frame should open.
7	Quantity Panel input quantity	Test the input works within the quantity panel.	By pressing buttons, should be able to input values.
8	Till Panel Select order item and remove	Test if selecting then removing an item removes it form the list box	Items selected should remove, and be removed from order list box.
9	Till Panel Check if offers appear	Check if offer relevant is added automatically to the bottom of the order list.	Offer should be added to bottom of item order list.

10	Till panel pay button	Check if the pay button opens up the pay frame for checking out.	Pay frame should open.
11	Pay Panel display data	Checks if the data for the total order amount is carried through from till.	Pay frame total value text box should display relevant order total.
12	Pay Panel keypad buttons	Test the input works within the pay panel.	By pressing buttons, should be able to input values.
13	Pay panel note buttons	Test if the £ note buttons insert correct values into the money tendered text box.	By selecting a pre-set value, the tendered text should change.
14	Pay Panel cancel button	Test if the cancel button closes the pay frame.	The pay frame should close.
15	Pay panel checkout button	Test if the checkout allows user to finish the order when there is not enough money tendered.	A message box should appear asking the user to input a higher tendered value.
16	Pay panel checkout button	Test if the checkout allows user to finish the order when there is enough money tendered.	A message box should appear showing the change required, along with a receipt save dialog.
17	Pay panel receipt save	Test if receipt is created successfully after choosing save location.	The receipt should be written to the specified folder after payment.
18	Till button panel change customer button	Test if the change customer button brings up the customer input box.	The input box for customer name should appear.
19	Till button panel change customer button	Check if correct input changes value of customer name in button panel.	The button panel customer name should change.
20	Till button panel change staff button	Test if the change staff button brings up the customer input box.	The input box for staff name should appear.
21	Till button panel change staff button	Check if correct input changes value of staff name in button panel.	The button panel staff name should change.
22	Till button panel save button	Test if save button saves till correctly and shows confirm dialog.	The till should save and the confirmation dialog should appear.
23	Till button panel load button	Test if load button brings up load dialog box.	The load frame should appear.

24	Load panel	Test if load panel loads the list of possible load files.	The load panel load list should be populated.
25	Load panel load button	Test if selected load file loads up successfully.	The load file should update the till to the selected load.
26	Load panel cancel button	Test if cancel button disposes of load frame.	The load frame should close.
27	Till button panel exit button	Test if exit button opens confirmation dialog, and then closes the application.	The exit dialog should appear, and then close the program is selected.
28	Till button panel new till button	Test if the new till button disposes of the current till and opens a new one with fresh data.	The till should close and a new opened.
29	Till button reset till button	Test if the resets till button clear the lists and values of the order.	The till should be cleared, and values reset.
30	Till panel menu selection	Test if the menus on the top of the till panel work correctly.	Each menu bar should open when selected.
31	Till panel file menu load till	Test if load button brings up load dialog box.	The load frame should appear.
32	Till panel file menu save till	Test if save button saves till correctly and shows confirm dialog.	The till should save and the confirmation dialog should appear.
33	Till panel file menu exit till	Test if exit button opens confirmation dialog, and then closes the application.	The exit dialog should appear, and then close the program is selected.
34	Till panel admin menu new till	Test if the new till button disposes of the current till and opens a new one with fresh data.	The till should close and a new opened.
35	Till panel help menu about	Test if the help dialog box is brought up.	The help dialog box should appear.
36	Till panel help menu about offers	Test if the about offers dialog is brought up.	The offer dialog box should appear.

Test Table

Test Number	Test Element	Test Description	Expected Result	Actual Result	Pass/ Fail?
1	SplashScreen1	Test if the splash screen load bar shows.	Splash Screen should load bar across.	The load bar appears and moves across.	Pass
2	SplashScreen1	Test null input repeats input box.	A null input should produce the input box to repeat appearing.	The input dialog reappears until the input is not null.	Pass
3	SplashScreen1	Insert correct text input.	Correct text should move the program to the next stage.	The input dialog closes and moves on.	Pass
4	Till Panel item buttons	Test if clicking item buttons, possible item list changes.	When an item type is pressed, the item list should refresh showing new possible items.	Item list updates and list relevant type items.	Pass
5	Till Panel Select and add item	Test if user can select an item and then add this to order.	Should add the item selected to the order list.	Item is added to order list.	Pass
6	Till Panel Select quantity open	Test if quantity frame opens when pressed	Quantity frame should open.	Quantity frame opens.	Pass
7	Quantity Panel input quantity	Test the input works within the quantity panel.	By pressing buttons, should be able to input values.	Values can be inputted using buttons.	Pass
8	Till Panel Select order item and remove	Test if selecting then removing an item removes it form the list box	Items selected should remove, and be removed from order list box.	Items can be removed from order list by selecting and pressing remove button.	Pass
9	Till Panel Check if offers appear	Check if offer relevant is added automatically to the bottom of the order list.	Offer should be added to bottom of item order list.	Order is added when required items are in order list. Also removed when not.	Pass
10	Till panel pay button	Check if the pay button opens up the pay frame for checking out.	Pay frame should open.	Pay frame opens.	Pass
11	Pay Panel display data	Checks if the data for the total order amount is carried through from till.	Pay frame total value text box should display relevant order total.	The data is correctly shown in the frame.	Pass

12	Pay Panel keypad buttons	Test the input works within the pay panel.	By pressing buttons, should be able to input values.	Values can be changed by using buttons.	Pass
13	Pay panel note buttons	Test if the £ note buttons insert correct values into the money tendered text box.	By selecting a pre-set value, the tendered text should change.	Pressing a pre-set value button automatically changes the value to whatever the button says.	Pass
14	Pay Panel cancel button	Test if the cancel button closes the pay frame.	The pay frame should close.	The pay frame closes.	Pass
15	Pay panel checkout button	Test if the checkout allows user to finish the order when there is not enough money tendered.	A message box should appear asking the user to input a higher tendered value.	Message box appears saying not enough money has been tendered.	Pass
16	Pay panel checkout button	Test if the checkout allows user to finish the order when there is enough money tendered.	A message box should appear showing the change required, along with a receipt save dialog.	Message box opens telling user how much change is required, then opens the JFileChooser box for the receipt.	Pass
17	Pay panel receipt save	Test if receipt is created successfully after choosing save location.	The receipt should be written to the specified folder after payment.	Receipt is created to chosen file location and shows all relevant data.	Pass
18	Till button panel change customer button	Test if the change customer button brings up the customer input box.	The input box for customer name should appear.	The customer name input box appears.	Pass
19	Till button panel change customer button	Check if correct input changes value of customer name in button panel.	The button panel customer name should change.	The button panel customer name value changes respectively.	Pass
20	Till button panel change staff button	Test if the change staff button brings up the customer input box.	The input box for staff name should appear.	The staff name input box appears.	Pass
21	Till button panel change staff button	Check if correct input changes value of staff name in button panel.	The button panel staff name should change.	The button panel staff name value changes respectively.	Pass

22	Till button panel save button	Test if save button saves till correctly and shows confirm dialog.	The till should save and the confirmation dialog should appear.	The till is saved and confirmed by dialog box.	Pass
23	Till button panel load button	Test if load button brings up load dialog box.	The load frame should appear.	Load frame appears.	Pass
24	Load panel	Test if load panel loads the list of possible load files.	The load panel load list should be populated.	Load frame list of possible loads appear.	Pass
25	Load panel load button	Test if selected load file loads up successfully.	The load file should update the till to the selected load.	Loads file, but till does not update to loaded till.	Fail
26	Load panel cancel button	Test if cancel button disposes of load frame.	The load frame should close.	Load frame closes.	Pass
27	Till button panel exit button	Test if exit button opens confirmation dialog, and then closes the application.	The exit dialog should appear, and then close the program is selected.	Exit dialog appears, closing till if selected.	Pass
28	Till button panel new till button	Test if the new till button disposes of the current till and opens a new one with fresh data.	The till should close and a new opened.	The existing till does not close, yet it still opens a new one on top.	Fail
29	Till button reset till button	Test if the resets till button clear the lists and values of the order.	The till should be cleared, and values reset.	Resets till correctly.	Pass
30	Till panel menu selection	Test if the menus on the top of the till panel work correctly.	Each menu bar should open when selected.	All menu bars change colour over rollover and open showing buttons.	Pass
31	Till panel file menu load till	Test if load button brings up load dialog box.	The load frame should appear.	Load frame appears.	Pass
32	Till panel file menu save till	Test if save button saves till correctly and shows confirm dialog.	The till should save and the confirmation dialog should appear.	The till is saved and confirmed by dialog box.	Pass
33	Till panel file menu exit till	Test if exit button opens confirmation dialog, and	The exit dialog should appear, and then close the program is	Exit dialog appears, closing till if selected.	Pass

		then closes the application.	selected.		
34	Till panel admin menu new till	Test if the new till button disposes of the current till and opens a new one with fresh data.	The till should close and a new opened.	The existing till does not close, yet it still opens a new one on top.	Fail
35	Till panel help menu about	Test if the help dialog box is brought up.	The help dialog box should appear.	Help dialog opens, showing information.	Pass
36	Till panel help menu about offers	Test if the about offers dialog is brought up.	The offer dialog box should appear.	Offer dialog opens, showing information.	Pass

Revised tests

From looking at the actual test results on the GUI tests, I found that three tests failed and did not perform the actions that I had previously expected. Because of this, I will now look at revising these sections of my program where the code broke and did not do as expected. As a note, because test 28 and 34 which involve the creation of a new till perform the same action, I have decided to revise just one of these and then copy the solution to the other.

Test Fail- Test 25(Loading the till)

In this test, the requirement was that once the load button had been selected, and then a load opened, the till should clear the existing till and then load the item list/ total price, customer name, staff name, item quantity onto the till panel. After running this test, I found this did not function correctly. After following a simple debug, I found that the data loaded was being correctly carried to the panel, yet the data was not being added to the list of order items or even updating the labels containing the total cost etc.

Test Fail- Test 28/34(New till function)

In this fail, I have found that the problem was not huge in the fact that the new till did successfully start; the only issue was closing the old one. This happened in two test cases, because both were doing the same operation just from different buttons.

Evaluations

This is the final section of my programming assignment for aber pizza, which will contain most of the write up. I will look at all three stages of my project, from the design and planning, through implementation, to the testing of my completed program. This will help me to understand in the future of ways I could make a project like this run smoother and more efficient. I will also be able to explain in this section why I performed certain tasks the way I did, and how I could have made them better. This will be placed in the 'possible improvements' section of the assignment. Once this is completed, I will add a small self-evaluation to the end which will include a personal reflection of how I thought I performed in all aspects of the task. I will explain what I learned and what skills I developed throughout this project.

I will now begin by evaluating the first section of the assignment, which was the planning and design section of the assignment. one of the most important sections of an assignment like this, as it allowed me to see what classes, methods, operations I would need my program to perform in order to meet the brief requirements, along with my own. I began this section by explaining the brief that had been given to me, and then listed the requirements by bullet points so I could go back to these at other times to see what I had to do. Once the requirements were outlined, I created my use case diagram which I found very useful in terms of explaining the brief better, and showing what interaction would be required to be given to the user.

I then moved onto the class diagram section, which meant using the existing design given to me, then extending it/ changing it to make it a suitable plan for my application. In the diagram once created, I could easily add any methods I thought up, such as exporting to a receipt. The relationships within the class diagram was an important piece of the class diagram as it meant I could see where I would need to be making instances of certain classes and methods in one another. Along with the class diagram came a class description as standard for this type of UML, and proved useful again when it came back to looking at my design in the implementation stage.

Other diagrams within the design and planning section of my assignment included state, activity sequence and package UML's. Each going into more detail for sections of the program that would do main operations such as saving, loading, and calculations etc. Having these diagrams also show good practice in my planning and that I used a wide variety of tools to pan my program. Using more diagrams gave me a wider scope of design, and many different ways to run through it. Once I completed the entirety of the diagrams, I then justified my design in a page or so of writing to provide further detail to the diagrams and what they meant.

Once I had completed planning my program, I then had to implement it. I did this by pulling all my designs together, and then creating classes and methods based on what I had planned inters of operations and usability. In my reasons for implementation section in this document I explained what way/ stages I went through in order to implement my program, saying that I created the data classes first, and then added the GUI. To cover this point again, I think that integrating the GUI after the data helped make the system much easier to implement because I had the basic data structure set up and working before the interface was. At the end of this section, I placed a small section where I explained how to run the program, using the jar file, or through command line.

JUnit testing was the next stage in the project, and involved the creation of a set of test classes, each run from a test suite, all of which tested the key methods of the classes such as adding an item to the order list, removing and clearing items. I used the JUnit testing facility in my IDE because it helped me produce quick, easy to add tests, in which I could use the 'assertequals' , 'assertnull' and other test methods to check on values and data to ensure they data was changing, being added and removed as I had expected. In the JUnit testing, I found no real issues with the code and all of the tests did as I expected throughout the testing phase. Another good point I found from using the JUnit testing facility and the test suite I created is that JUnit shows exactly what errors occurred where, and why, making it possible for me to easily amend and problems where required.

Because my program was divided into data and GUI, I decided to create a separate range of tests for this section to check button operations, text box inputs and outputs of save files, receipts and the labels on the panels. I began the testing for the GUI with a test table, which involved displaying the test number, the test element and the expected result. I also described briefly how I was going to perform each test. This first table guided my tests as I ran through the program to test each selected element. With these results, I then recorded this into the actual tests table, along with a status if the test passed or failed. Any failed tests were revised below the table and I explained what I thought went wrong where, and how I might fix it or did fix it. This was a good standard of the way I did my project.

Once the testing was complete, I then created a conclusion, which is this section of the assignment. I have felt this has allowed me to look at how I did each section of the assignment and then how I thought it went in terms of creating a successful program built to the brief that was given to me, and more importantly the plan I created as it showed consistency in my project.

Possible Improvements

I will now look at any possible improvements I believe could be made to this project based on the program and documentation I have created. I will break this down into three small sections, improvements in the design, program, and in the testing of my program. Beginning with the design stage, although I managed to create a wide array of diagrams to represent my classes and operations, I think that more descriptive text could have been added to describe each diagram in more detailed way. For instance, in my simple use case diagram, it would have been possible to add branches off some of the user controls, showing what the program would do in order to complete a task asked by the user. Other than this small improvement to the design, I believe that I performed quite well in the designing of the system.

Moving straight to what I think possible improvements could be made within the completed program side of the project, I can see a number of small changes I would make to make it higher quality. For instance, rather than having the save till button in order to save the order when requested by the user, I would remove this button, and get the till to automatically save the till when an order is completed. This would make a much more organised system and only orders that were completed would show. It would also meet the brief more closely because I would be able to get the orders totals and produce a total for the day.

As another possible improvement within the completed product if possible and I had more time, the addition of an undo button would be implemented, which would add an extra feature, as a wow

factor. The more control offered to the user would make a better program. This brings me to the way users select the items. Rather than having a JList box within the till panel, I think if I had implemented a set of large buttons to represent products, this would have made it easier for users to select items with touch input, because of the small selection in the JList components.

I think rather than having a few different frames to contain the input for my payment, and quantity, I would have rather had the whole application contained within a changing window that updated with the next stage of an order. This would make transactions faster. Changing the customer and staff name could have been done easier also by just using a JTextBox rather than labels where the user would need to open a dialog box to change the values.

The final two improvements for the implementation of my program are the amount of wow factors used and the saving and loading of files. Because I decided not to save and write orders using XLM mark-up, this is an obvious improvement I would make next time, by finding more time in the project and then implementing this. It would be an easier to read way of saving orders in case the user wanted to open the save file directly and check the contents. As for the possibility of any extra wow features, colours, text and layout could have been made more attractive by using a better style for the program. This could have also meant the user would have been able to see buttons easier and follow the layout of the system.

In the testing section of the assignment, this is where the final section of the possible improvements lay. Where I created JUnit tests, there is always room for the addition of more methods tested, and tested different ways. By adding more tests, I would gain wider scope of possible outcomes and finding any hidden bugs.

Self-Evaluation

For the final section of this assignment as documentation, I will now give a short section on my own self-evaluation of the project, looking at how well I think I have performed, improved in certain aspects of java programming, and how I could have improved my performance overall. Starting with my performance as a whole in this project, I think that coming into it, I already had the basic knowledge and skills to create an application such as this one, but bringing all these skills together to actually build it was a challenge. I had experience in the Swing facility in java, but this program would require quite a bit of GUI, and this meant I needed to look up certain methods and ways of implementing some panels. For instance, in the pay frame, I used the grid bag layout, which I found made creating that panel considerably easier because I could pick the location of items in terms of cells.

As for the implementation of the underlying data classes, this was quite recognisable to me, creating a program and running it in command line first was useful. I could create print statements and debug the code to ensure that I was creating the correct methods etc. What I gained from creating the data and GUI classes separately was the knowledge that this was a much easier way to do it than create the whole application in one piece. It reduced time taken and let me get on with the rest of the project.

I felt I performed quite well too within the testing stage of the assignment, by successfully creating a range of JUnit test cases, along with a test suite that meant I could run all the tests in one go saving me yet more time when it came to check all the tests passed. I found this to be one of the easiest

sections of the task. I found that using JUnit testing made it easy for me to erase any issues with the code such as loading and saving. If I were to say I had any problems with this testing, I would say that this would be the way I wrote the test code itself. This was because I found some problems or failed tests that appeared were not because of my program code, but because of incorrect test code. I had to ensure that the code was precise and no errors occurred within it. This adds to what I have learnt in the project, which is consistency in my code, and ensuring everything is kept working as smoothly and easily as possible.

Revising the tests once complete made my project more flowing, and helped show how I had fixed or left the issues I encounted. I can now look back at these test revisions again in a future project if I ever encounted a problem like these again. I feel this section also added consistency to my project documentation and displayed that I did not just have failed tests and left the corresponding sections of code unfixed. It showed that I attempted to fix these problems, and that I tried to make the functions work as I expected in my test plan table. This increases the quality of my testing phase.

To summarise my own performance, I felt I have managed to meet the brief mostly, creating a fully functioning till system that does almost everything I planned, with the exception of showing total sales for the day and saving in the XML language. I found that I did not have enough time to perform this task, and because of this and my low amount of experience creating XML mark-up, I decided to concentrate on other section of the code. If I had a little more time, or spent less time on the small parts of code such as creating separate pay and quantity frames, I could have achieved these two small pieces of the brief given to me. I believe I have defiantly advanced my java skills further from this project, and if I was to do it again, I would use my know-how from now to produce a better product, that would include a wider scope of use.