CS 12320: Mini Assignment #1
Mrs Miggins Pie Shop

Due on Wednesday, March 18th, 2015

Nathan Hughes (nah26@aber.ac.uk)

Contents

UML Diagrams

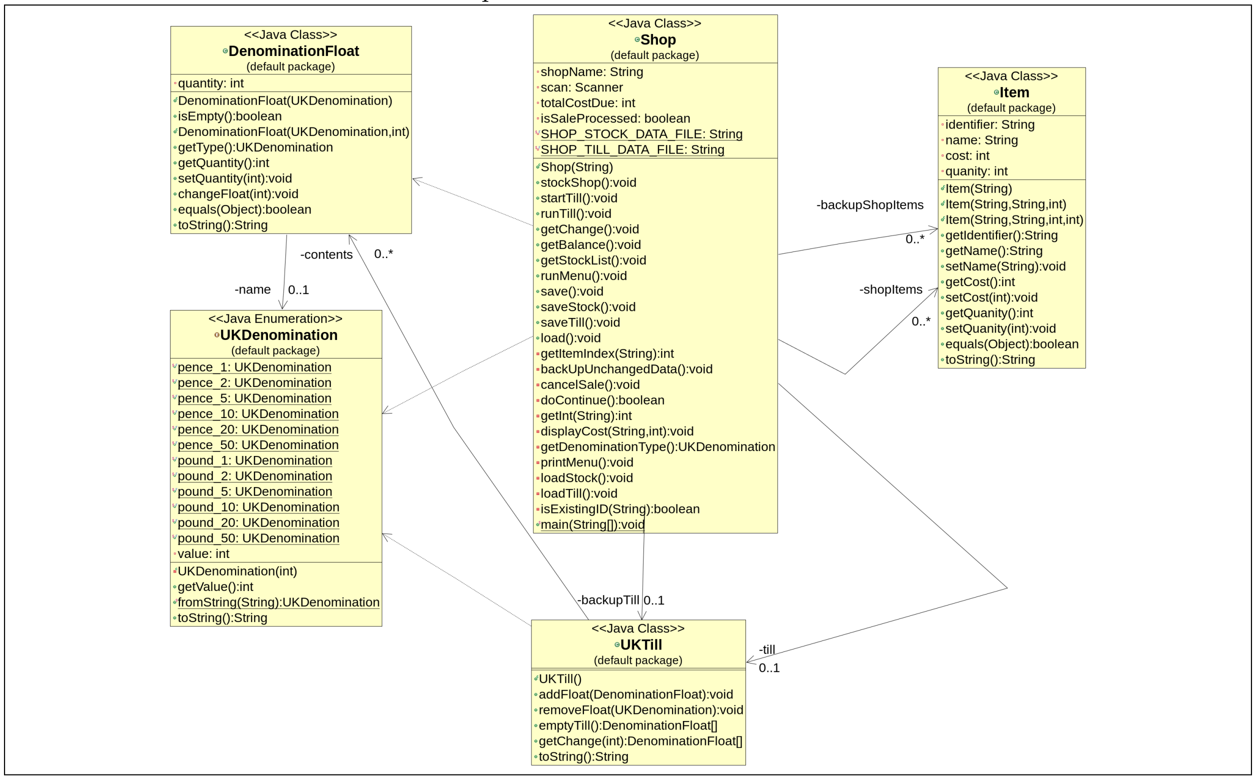
Relationships

Below are the class diagrams for Mrs Miggins shop application; the 5 classes are shown to have relationships between each other by 2 different types of lines. A solid line depicts an association relationship and a dashed/dotted line shows a dependency between classes

Beside every class object or function there is a coloured dot, green signifies a publicly accessible element whereas red shows that it is private and cannot be accessed outside of the class itself.

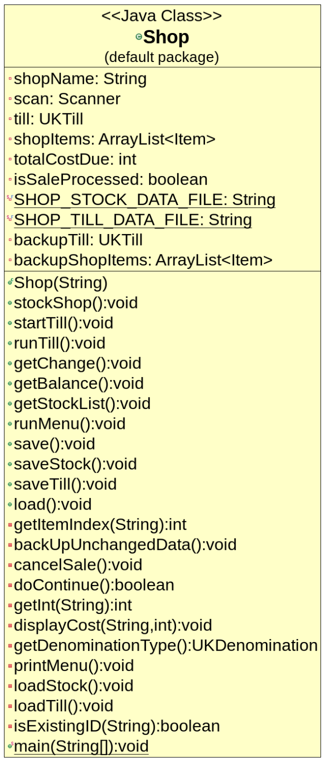
Functions can be identified as they have "()" following their name as well as their return type. Variables/Objects will also have their type noted following their names' also.

In the case of constant variables block capitals have been used to illustrate them.



UML Diagrams continued on next page...

Shop Class

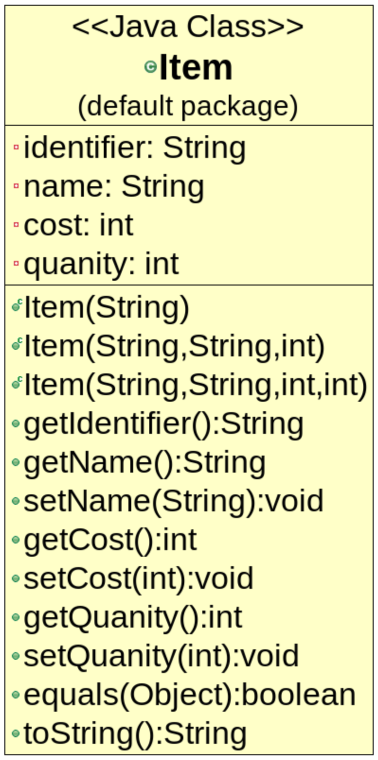


<<Java Class>>
Shop
(default package)

shopName: String
scan: Scanner
till: UKTill
shopItems: ArrayList<Item>
totalCostDue: int
isSaleProcessed: boolean
SHOP_STOCK_DATA_FILE: String
SHOP_TILL_DATA_FILE: String
backupTill: UKTill
backupShopItems: ArrayList<Item>

Shop(String)
stockShop():void
startTill():void
runTill():void
getChange():void
getBalance():void
getStockList():void
runMenu():void
save():void
saveStock():void
saveTill():void
load():void
getItemIndex(String):int
backUpUnchangedData():void
cancelSale():void
doContinue():boolean
getInt(String):int
displayCost(String,int):void
getDenominationType():UKDenomination
printMenu():void
loadStock():void
loadTill():void
isExistingID(String):boolean
main(String[]):void

Item Class



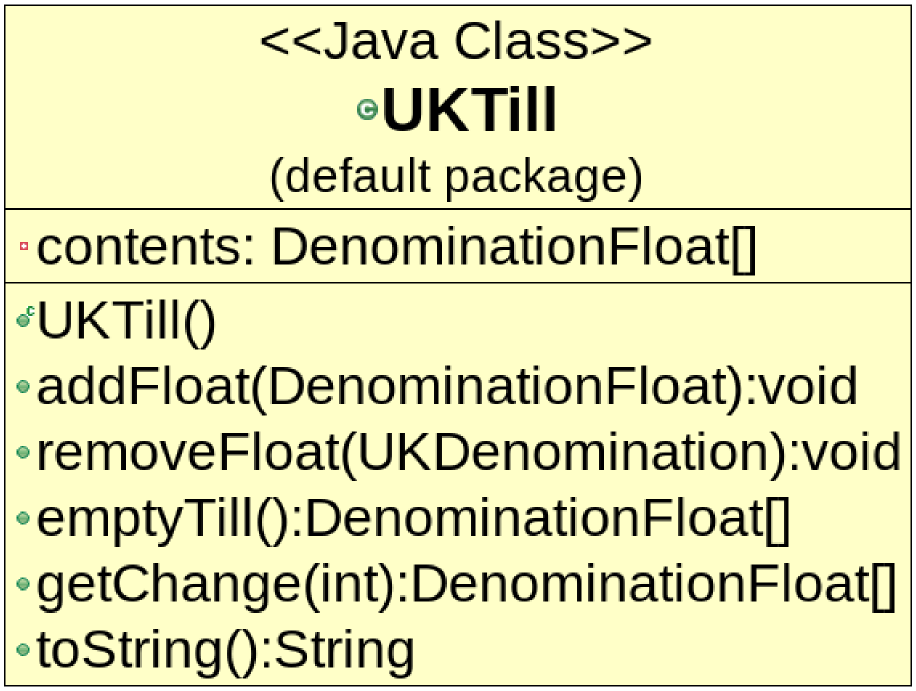
<<Java Class>>
Item
(default package)

identifier: String
name: String
cost: int
quanity: int

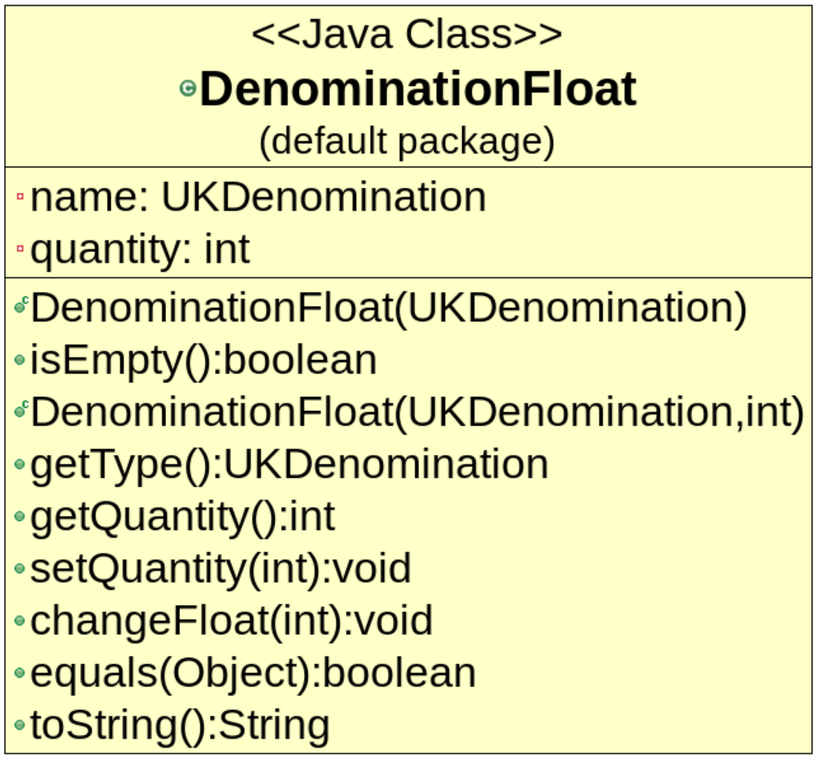
Item(String)
Item(String,String,int)
Item(String,String,int,int)
getIdentifier():String
getName():String
setName(String):void
getCost():int
setCost(int):void
getQuanity():int
setQuanity(int):void
equals(Object):boolean
toString():String

UML Diagrams continued on next page...

UKTill Class

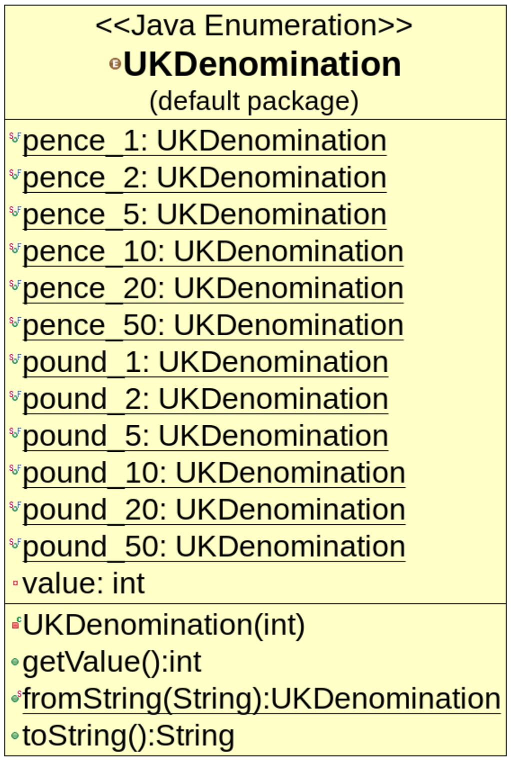

<<Java Class>>
UKTill
(default package)
contents: DenominationFloat[]
UKTill()
addFloat(DenominationFloat):void
removeFloat(UKDenomination):void
emptyTill():DenominationFloat[]
getChange(int):DenominationFloat[]
toString():String

DenominationFloat Class



UML Diagrams continued on next page. . .

UKDenomination Enumeration Class



Assignment Write Up

Tasks and Solutions

In this section I will outline my solutions to the problems given in this assignment, alongside each problem there will be a code snippet.

Load Till Function

The initial problem that I encountered for this function was having to separate the information contained in the ”till.txt” file. I got around this buy realizing that I could use an array of 24 items (12 x types of currency, 12 x quantities) to store all of the information. To separate the information I used delimiters of ”x” and newline characters. I copied some of the code already used for reading in input for currency and modified it to loop through the array of information that I had read in and add it to the till object.

```
private void loadTill(){

   //This array will hold all is read in from the scanner
   //from the text file
   String[] readTillResults;
   readTillResults = new String[24];

   //Reading in code
   Scanner readTillData = null;
   try {
      readTillData = new Scanner (new File(SHOP_TILL_DATA_FILE));
   } catch (FileNotFoundException e) {
      // Something went wrong loading the till
            System.err.println("Sorry but we were unable to load shop till data: "
                 + e.getMessage());
   }
   //assigning delimiters of both newline and a ':'
   readTillData.useDelimiter(" x |\\n");
   //index used to keep track of
   //position of applying info read from file
   int index = 0;
   //loop to read all of the text file
   while (readTillData.hasNext() && index < 24){
      readTillResults[index] = readTillData.next();
      //increment index
      index++;
   }
   //closing the reading file
   readTillData.close();

   //this uses a modified version of the already implemented
   //getDenominationType function as well as the startTill function
   for(index = 0; index < 24; index+=2){
      UKDenomination ct = UKDenomination.fromString(readTillResults[index]);
      int nc = Integer.valueOf(readTillResults[index+1]);
      DenominationFloat m = new DenominationFloat(ct, nc);
      till.addFloat(m);
   }
}
```

Save Till Function

This function uses the Java printWriter class and the FileWriter to completely overwrite the contents of the "till.txt" file whenever a save occurs. I chose to set the append parameter to false so that it was easier to keep writing the file's formatting simple

```java
public void saveTill(){
     //The first part of this function saves only the till and its
     //contents to a .txt file, it is set to overwrite all previous
     //data, I have done this to ensure the correct format within
     //the file is kept
     try(PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(SHOP_TILL_DATA_FILE
          , false)))) {
        out.println(till);
     }catch (IOException e) {
        System.out.println("There was a problem saving the till");
     }
   }
```

Load Stock Function

Loading the stock file was more (at least for me) difficult that loading in the till, as for stock there would be an unknown number of objects needing to be read in, this meant I didn't know where to stop reading or where each attribute of a stock item would be in the information read in.

I used a method similar to loading the till but this time used an ArrayList to store the data, used delimiters that would be found in the "stock.txt" file and used /4 to divide up information to match up the 4 variables of a stock item (ID, Name, Price and Quantity).

```java
   private void loadStock(){
      //going to store all of the files used in this
      ArrayList<String> strItems = new ArrayList<String>();
      //Reading in code
      Scanner readStockData = null;
      try {
         readStockData = new Scanner (new File(SHOP_STOCK_DATA_FILE));
      } catch (FileNotFoundException e) {
         // Something went wrong loading the till
                System.err.println("Sorry but we were unable to load shop stock data: "
                     + e.getMessage());
      }
      //assigning delimiters of both newline and a ':'
      readStockData.useDelimiter(":|\\n");
      //loop to read all of the text file
      while (readStockData.hasNext()){
         strItems.add(readStockData.next());
      }
      //finding the number of items by dividing by 4
      //as there are 4 properties per item
      //there will always be 4 properties
      //or the files will not be saved
      int numOfItems = (strItems.size() / 4);

      //closing the reading file
```

```
    readStockData.close();

    //For this loop, 0 is the ID, 1 is the Name,
    //2 is the price in pence and 3 is the quantity
    //
    //This loop adds the read in data to the programs stock list
    for(int index = 0; index < numOfItems; index++){

       Item itmTemp = new Item(strItems.get(0+(4*index)),
             strItems.get(1+(4*index)),
             Integer.valueOf(strItems.get(2+(4*index))),
             Integer.valueOf(strItems.get(3+(4*index))));

       shopItems.add(itmTemp);
    }
  }
```

Save Stock Function

I used the StringBuilder class here to quickly and easily build each entry into the stock file, this allowed me to be very specific with the output as it had to be compatible with the read method that would directly have to handle what this function does on next load.

Once the strings had been built then it was easy to use the FileWriter again to add these strings into the ”stock.txt” file

```
    public void saveStock(){
    //this function saves only the stock and its
    //info to a .txt file, it is set to overwrite all previous
    //data, I have done this to ensure the correct format within
    //the file is kept

    StringBuilder strStock = new StringBuilder();
    for(int index = 0; index < shopItems.size(); index++){

        strStock.append(shopItems.get(index).toString()).append('\n');

    }

    try(PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(SHOP_STOCK_DATA_FILE
        , false)))) {
       out.print(strStock);
    }catch (IOException e) {
       //exception handling left as an exercise for the reader

    }
  }
```

Assignment Write Up continued on next page. . .

Run Till Function

Run till will get all of the information about a sale from the customer,the barcode of the item they want and quantity of item. I used a lot of the Item class's functions such as getQuantity to do error checking within this function and the user's input

For my function I allowed the user to enter multiple items one after another to build a "shopping basket" of sorts, the total price would be saved as a global variable to be processed by the getChange() method.

```
public void runTill() {

   //set variable to false so we can track if sale is completed
   isSaleProcessed = false;

   //using nested loop to allow for multiple input
   //of multiple items up for sale at once
   do{

      //using this boolean for loop condition of input
      boolean isValidInput = false;
      //Variable to hold ID of sold item
      String id;

      //get the barcode of the item from the user
      do{
         System.out.println("Please enter in barcode of item");
         id = scan.nextLine();

         //check if is existing barcode
         if(!isExistingID(id)){
            System.out.println("Invalid barcode");
         }

      }while(!isExistingID(id));

      //get the index of the item that we would like to sell
      int index = getItemIndex(id);

      //variable to hold quantity
      int quantity = 0;

      //get the quantity from the user
      do{
         isValidInput = false;

         System.out.println("Please enter the quantity you'd like to purchase");
         System.out.println("There are " + shopItems.get(index).getQuanity() + " in stock
             currently");
         String temp = (scan.nextLine());

         //ensure that the quantity given is only an integer value
         if(!temp.matches("[0-9]+")){
            System.out.println("Please enter numbers only for the quantity");
            //try again
```

```
                continue;
            }
            if(Integer.parseInt(temp) > shopItems.get(index).getQuanity()){
                System.out.println("Please ensure that you have choosen a quantity that we can
                    provide");
                //try again
                continue;
            }
            //this input is valid therefore set it and break loop
            else{
                quantity = Integer.parseInt(temp);
                isValidInput = true;
            }

        //run loop while input is not valid
        }while(!isValidInput);

        //remove the quantity of items from stock and proceed
            //variable to hold old quantity temp
        int oldQuant = shopItems.get(index).getQuanity();
        shopItems.get(index).setQuanity(oldQuant - quantity);

        //
        totalCostDue += (shopItems.get(index).getCost()*quantity);

        //ask user if data is correct and output price
        System.out.println("Order is for: " + shopItems.get(index).getName() + " x " + quantity);
        System.out.println("The cost of this is: " + (shopItems.get(index).getCost()*quantity) );

        System.out.println("Would you like to continue adding items?");
    }while(doContinue());

    System.out.println("Your total cost is now: " + totalCostDue);
}
```

Get Change Function

Getting the change due to the customer was the biggest challenge I faced in this assignment. The struggle was trying to return readable output to the user in the form of how many of each coin/note was owed in change.

As I was so pedantic in returning tidy output I changed this function from returning and array and set it to return an ArrayList, the use of an ArrayList meant that I could easily delete values equal to 0 from it. The code for this is very messy and inefficient in it's memory usage. This cries out bad programming practice and shames me, had I more time perhaps I could do this better by improving my knowledge.

```
public ArrayList<DenominationFloat> getChange(int changeDue) {

    //temp value to hold change as an array before trimming
    DenominationFloat[] change = new DenominationFloat[UKDenomination.values().length];

    // Initialise the array with empty floats for each denomination. This is
    // a way of being able to subtract a 12 array from a 12 array and the indices
    // will match up !
    for (UKDenomination denom : UKDenomination.values()) {
       DenominationFloat denomFloat = new DenominationFloat(denom, 0);
       change[denom.ordinal()] = denomFloat;
    }

    //This variable dictates if the value has been edited within the loop
    //if it has then we know that there's reason to try again to make the change
    //more accurate
    boolean hasValueChanged;

    //Loop to create change
    do{

       //reset boolean
       hasValueChanged = false;

       //this is the same for every statement of this loop
       //we check if this type of currency can be used as change
       //if so we take away its value from what is due
       //take it out of the till
       //and add it to the change that will be returned
       if(changeDue >= UKDenomination.pound_50.getValue()
             && contents[11].getQuantity() > 0 ){
          changeDue -= 5000;
          contents[11].changeFloat(-1);
          change[11].changeFloat(1);
       }
       if(changeDue >= UKDenomination.pound_20.getValue()
             && contents[10].getQuantity() > 0 ){
          changeDue -= 2000;
          contents[10].changeFloat(-1);
          change[10].changeFloat(1);
          hasValueChanged = true;
       }
       if(changeDue >= UKDenomination.pound_10.getValue()
             && contents[9].getQuantity() > 0 ){
          changeDue -= 1000;
          contents[9].changeFloat(-1);
          change[9].changeFloat(1);
          hasValueChanged = true;
       }
       if(changeDue >= UKDenomination.pound_5.getValue()
             && contents[8].getQuantity() > 0 ){
          changeDue -= 500;
          contents[8].changeFloat(-1);
          change[8].changeFloat(1);
          hasValueChanged = true;
       }
       if(changeDue >= UKDenomination.pound_2.getValue()
             && contents[7].getQuantity() > 0 ){
          changeDue -= 200;
          contents[7].changeFloat(-1);
```

Assignment Write Up [Tasks and Solutions] continued on next page. . .

```
       change[7].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pound_1.getValue()
          && contents[6].getQuantity() > 0 ){
       changeDue -= 100;
       contents[6].changeFloat(-1);
       change[6].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_50.getValue()
          && contents[5].getQuantity() > 0 ){
       changeDue -= 50;
       contents[5].changeFloat(-1);
       change[5].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_20.getValue()
          && contents[4].getQuantity() > 0 ){
       changeDue -= 20;
       contents[4].changeFloat(-1);
       change[4].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_10.getValue()
          && contents[3].getQuantity() > 0 ){
       changeDue -= 10;
       contents[3].changeFloat(-1);
       change[3].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_5.getValue()
          && contents[2].getQuantity() > 0 ){
       changeDue -= 5;
       contents[2].changeFloat(-1);
       change[2].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_2.getValue()
          && contents[5].getQuantity() > 0 ){
       changeDue -= 2;
       contents[1].changeFloat(-1);
       change[1].changeFloat(1);
       hasValueChanged = true;
    }
    if(changeDue >= UKDenomination.pence_1.getValue()
          && contents[0].getQuantity() > 0 ){
       changeDue -= 1;
       contents[0].changeFloat(-1);
       change[0].changeFloat(1);
       hasValueChanged = true;
    }
    }while(changeDue > 0 || hasValueChanged);

    //used an array list here to trim down the output of the function
    ArrayList<DenominationFloat> trimmedChange = new ArrayList<DenominationFloat>();

    //looping to only add values > 0 to reduce output spam
    for(int index = 0; index < 12; index++){

      if(change[index].getQuantity() != 0){
        trimmedChange.add(change[index]);
      }

    }

    //return the array of the change required
    return trimmedChange;
  }
```

Assignment Write Up continued on next page...

Evaluation

Overall I feel as though I did well in handling this assignment, I succeeded in all of the requirements (they work, at least when I tested it).

The downfalls that I can see within my own code is that my functions may have ended up being too long and too complicated, again, if I had more time perhaps I could have better streamlined this and divided up my work into more functions and methods making the code easier to understand and more efficient. Additionally as I mentioned in the previous section there are certain functions which are not as memory efficient as they could be, I regretfully used quick and dirty fixes to some of my problems, it did work but if I was to release a patch for this program that would be a major point for me to go back and work on.

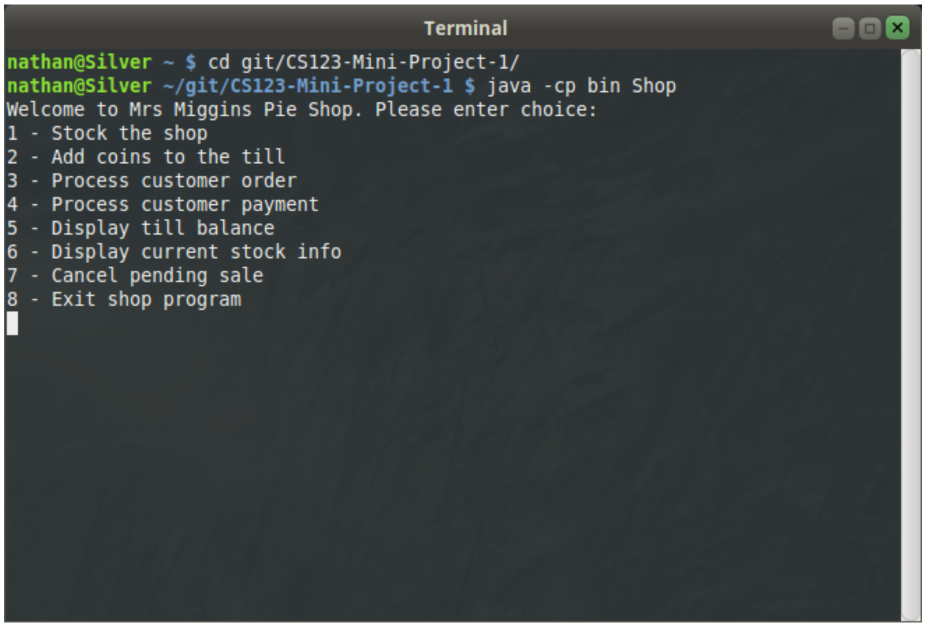
Moreover I feel that something I did well was error checking within my code, at every stage of user input I have provided strict input rules. The example of this shown below illustrates just one of the many places within my code that I added input error checking. This function doContinue() will only allow input of ”y” or ”n” anything else will request the user try again

```java
    private boolean doContinue() {

    //string to hold the input
    String answer;
    //boolean used to condition the loop until it gets
    //something that is valid
    boolean isValidInput;
    //do while loop to ensure correct input is given
    do{
       isValidInput = false;

       System.out.println("(Y/N)");
       answer = scan.nextLine().toUpperCase();
       if(answer.equalsIgnoreCase("N") ||
             answer.equalsIgnoreCase("Y")){
          isValidInput = true;
       }
       else{
          System.out.println("Error with input, try again");
       }
    }while(!isValidInput);

    return answer.equals("Y");
  }
```

If I had to, and perhaps I have missed something but I would award myself a 6.7/10 in marks for my code.

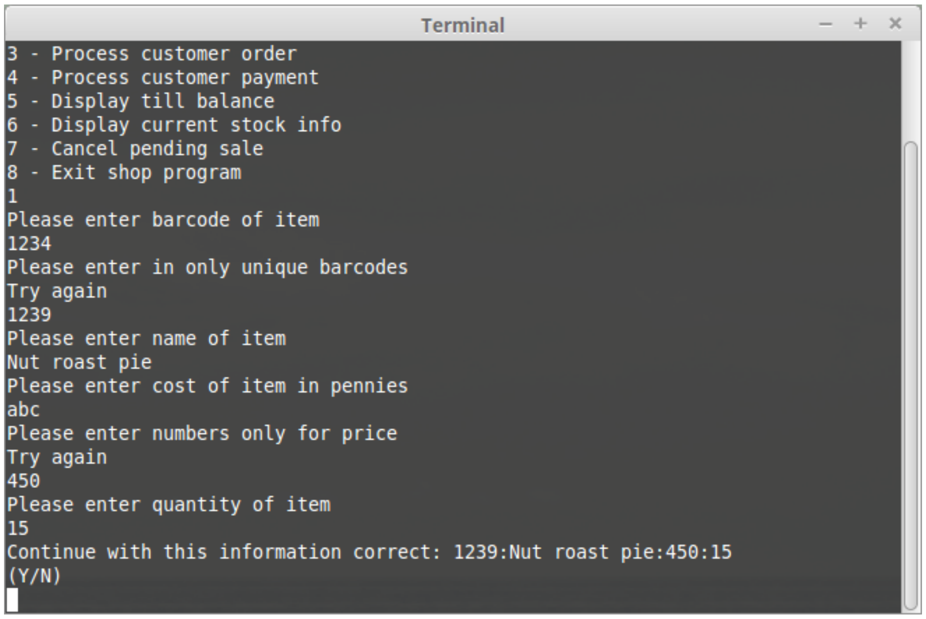
Code Running

Launching from Terminal



```
nathan@Silver ~ $ cd git/CS123-Mini-Project-1/
nathan@Silver ~/git/CS123-Mini-Project-1 $ java -cp bin Shop
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
```

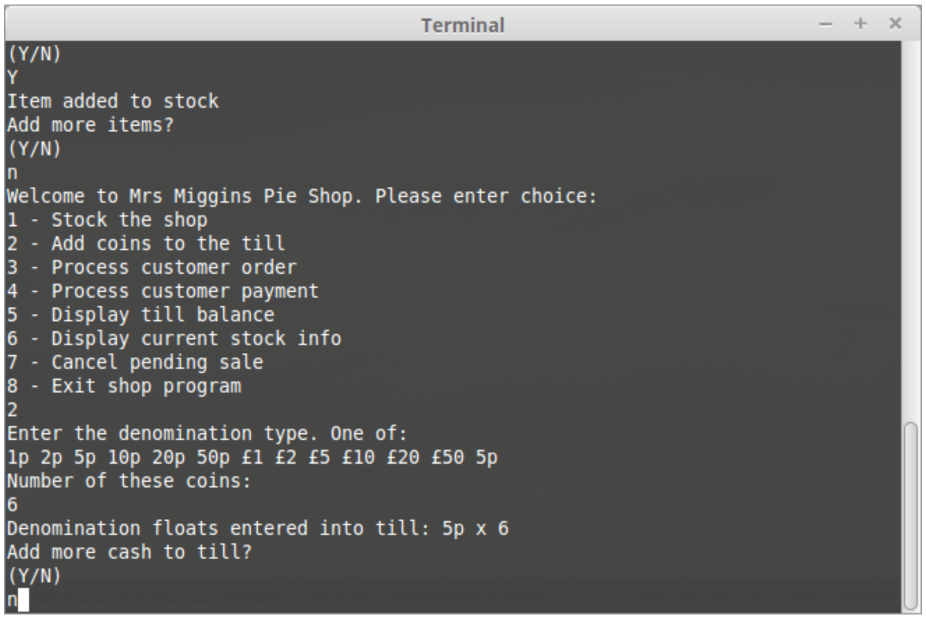
Adding Item to Stock



```
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
1
Please enter barcode of item
1234
Please enter in only unique barcodes
Try again
1239
Please enter name of item
Nut roast pie
Please enter cost of item in pennies
abc
Please enter numbers only for price
Try again
450
Please enter quantity of item
15
Continue with this information correct: 1239:Nut roast pie:450:15
(Y/N)
```

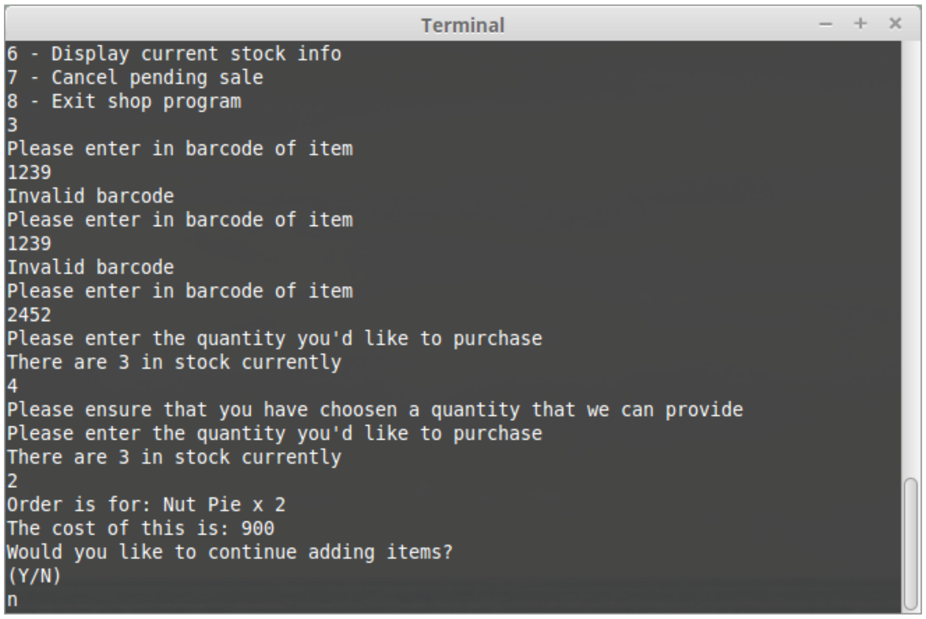
Code Running continued on next page. . .

Add Coins to the Till



```
(Y/N)
Y
Item added to stock
Add more items?
(Y/N)
n
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
2
Enter the denomination type. One of:
1p 2p 5p 10p 20p 50p £1 £2 £5 £10 £20 £50 5p
Number of these coins:
6
Denomination floats entered into till: 5p x 6
Add more cash to till?
(Y/N)
n
```

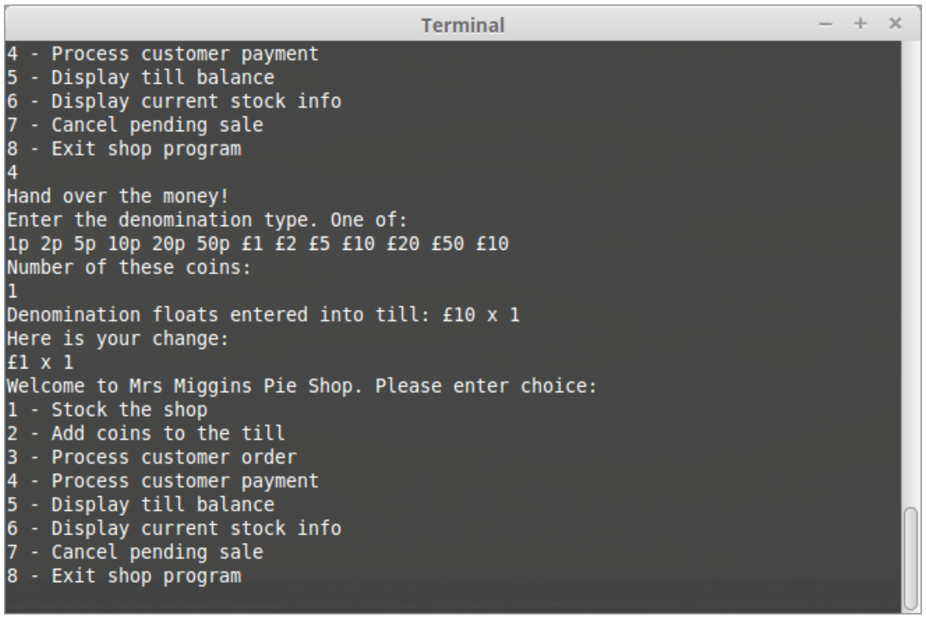
Processing Order



```
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
3
Please enter in barcode of item
1239
Invalid barcode
Please enter in barcode of item
1239
Invalid barcode
Please enter in barcode of item
2452
Please enter the quantity you'd like to purchase
There are 3 in stock currently
4
Please ensure that you have choosen a quantity that we can provide
Please enter the quantity you'd like to purchase
There are 3 in stock currently
2
Order is for: Nut Pie x 2
The cost of this is: 900
Would you like to continue adding items?
(Y/N)
n
```

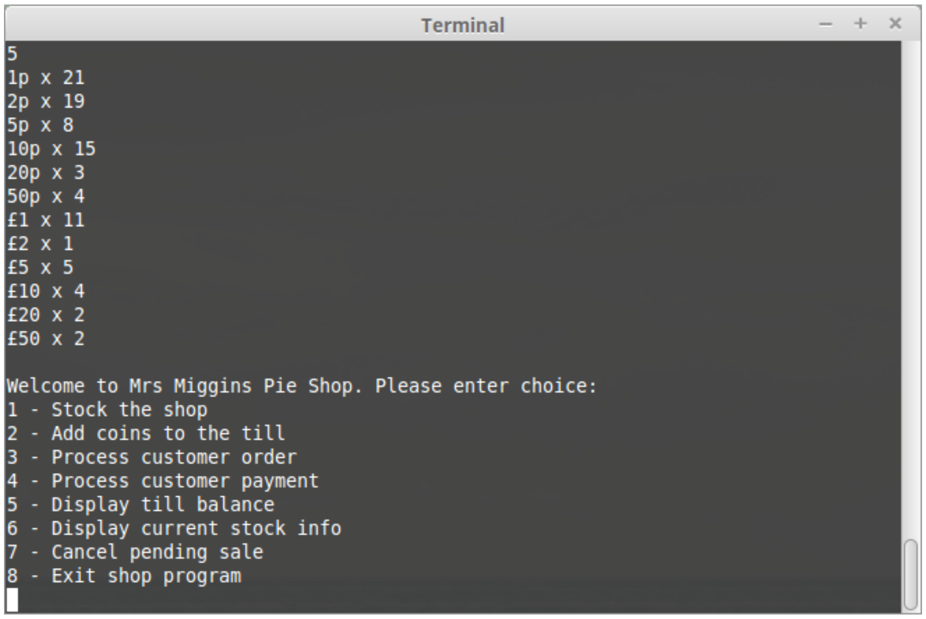
continued on next page. . .

Processing Payment



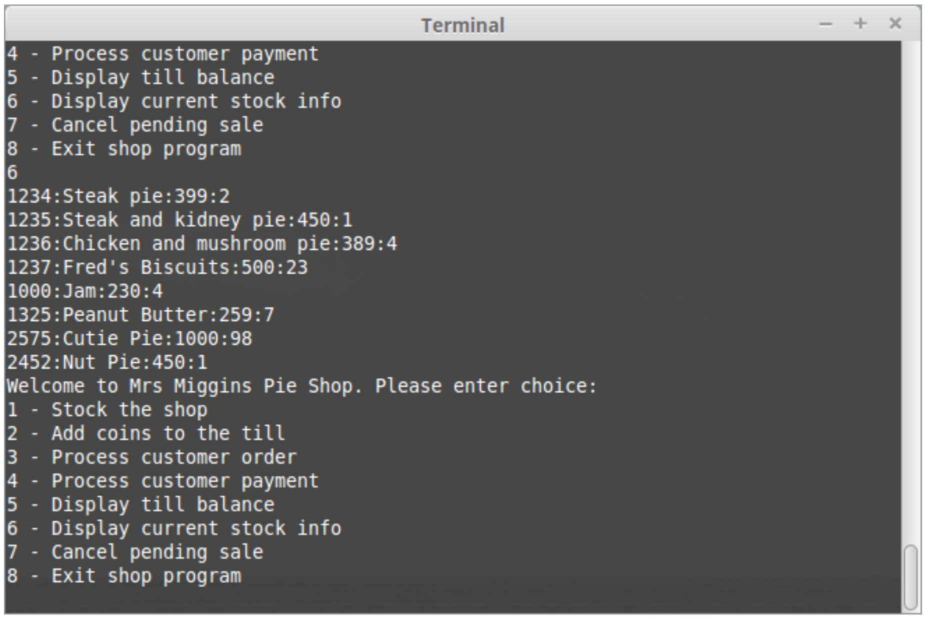
```
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
4
Hand over the money!
Enter the denomination type. One of:
1p 2p 5p 10p 20p 50p £1 £2 £5 £10 £20 £50 £10
Number of these coins:
1
Denomination floats entered into till: £10 x 1
Here is your change:
£1 x 1
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
```

Displaying Stock Info



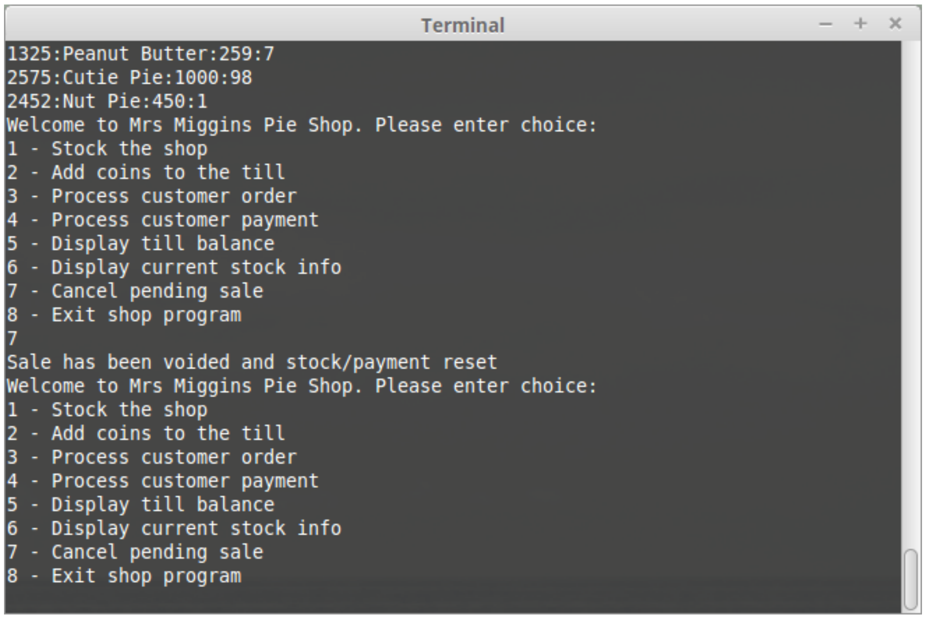
```
5
1p x 21
2p x 19
5p x 8
10p x 15
20p x 3
50p x 4
£1 x 11
£2 x 1
£5 x 5
£10 x 4
£20 x 2
£50 x 2

Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
```

Displaying Stock Info



```
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
6
1234:Steak pie:399:2
1235:Steak and kidney pie:450:1
1236:Chicken and mushroom pie:389:4
1237:Fred's Biscuits:500:23
1000:Jam:230:4
1325:Peanut Butter:259:7
2575:Cutie Pie:1000:98
2452:Nut Pie:450:1
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
```

Cancelling a Sale



```
1325:Peanut Butter:259:7
2575:Cutie Pie:1000:98
2452:Nut Pie:450:1
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
7
Sale has been voided and stock/payment reset
Welcome to Mrs Miggins Pie Shop. Please enter choice:
1 - Stock the shop
2 - Add coins to the till
3 - Process customer order
4 - Process customer payment
5 - Display till balance
6 - Display current stock info
7 - Cancel pending sale
8 - Exit shop program
```

continued on next page. . .