Project OOP – Guidance on the Program/Methods Implementation

Input Method

This method is used quite a lot within my program, this method is quite simple. As it helped me format my program as I went along, as I could easily insert “input(“Test”);” as this would allow me to instantly structure this text to suit the appearance with the rest of my program. This method is implemented within the Vending Machine class. However, this method has been set towards public as I would prefer to access this method throughout the package. Allowing me to call it within ‘VendingMachineMenu.java’ etc. This method takes a parameter ‘input’ which is just simply the text I want to print out. Which is then printed through the System class. In a structured format.

```
~~ Balance added: £0.00
```

```
 */
public static void input(String input) {
    System.out.print("\n~~ " + input);
}
```

AddCoinsToHash Method

This method is quite important within the Program, as the purpose of this method is to set the HashMap, “trackCoinsVal”. I use a HashMap within my Program as this allows me to store a Coin type, along with the amount available within the Program. Therefore, if the Vending machine owes changes towards a user, for example “0.50”. Once this type of change is given. Depending on the amount. The key that represents the coin given, reduces the value of that key by 1 (Or Depending how much change is given). You can see this information within the Engineer level of the program. This method is called quite early within the program, as this must be created before the program actually loads. This method is called within the Vending Machine constructor, depending on default settings or not. This method is created within the Vending Machine class as well.

```
Coin: £0.05, x2
Coin: £0.10, x2
Coin: £0.20, x2
Coin: £0.50, x2
Coin: £1.00, x1
Coin: £2.00, x4
```

```
private void addCoinsToHash() {
    for (int i = 0; i < coins.size(); i++) {
        trackCoinsVal.put(coinType[i], 2);
    }
}
```

DetermineCoin Method

The determineCoin method is created within the Vending machine class. This method is used whenever a user is inserting a coin within the Program. As the problem is, within the specification of the Project. We use a method called “InsertCoin”. However, this method takes an int parameter. This is alright, but this can cause a problem for whenever the user is typing which coin, he wants, such as 50p (0.50). Which causes a problem to determine which coin the user wants as he can’t enter this type of input. Therefore, by creating this method it allows the user a String input instead. When it comes to this area of the Program.

As they are able to enter any String available, such as “0.50 or £1”. AS once this input goes into the method, “DetermineCoin”. It will find either the value contains a ‘£’ or not. If it does, it grabs the substring of the amount after the pound sign is found. Therefore, if the user enters ‘£1’. It will grab (1) and parse it as an Integer and value within the String. If it doesn’t contain this sign, it will multiply the value of the String, for example “0.50” by 100. From there, it can be parsed and sent towards the InsertCoin, where it will be transferred back towards the double input the user wanted. By dividing by 100. Which will be tested within the HashSet if it contains this value. Depending on what is inserted, will be added towards the User Money or not.

```
public void determineCoin(String coin) {
    int coinInput = 0;
    if(coin.contains("£"))
        insertCoin(Integer.parseInt(coin.substring(1)));
    else {
        double newDoubleTmp = Double.parseDouble(String.valueOf(coin));
        coinInput = (int) (newDoubleTmp*100);
        insertCoin(coinInput);
    }

}
```

CountRemainingItems

This method is just used within the System information Menu of the program. As this will generate an Integer that will represent all of the items that have a greater Quantity than 0. Therefore, finding out how many items are still valid to buy within the machine. This can be useful to determine, when the Engineer should restock. As this would give a good insight of this. This method is created within the Vending Machine class, therefore being privately accessed. Which returns an int (amount of items available).

```
private int countRemainingItems() {
    int res = 0;
    for (int i = 0; i < itemCount; i++)
        if (stock[i].getQty() > 0)
            res++;
    return res;
}
```

purchaseRequest Method

This method is used to determine whether a user is able to buy an item within the machine or not. This method is created within the Vending machine class, which has a private access. This method takes the parameter of an int (index). This index is used to determine which item the user is looking to buy within the machine. The main purpose of this method is too comparing the User Money towards the price of the item they are looking to buy. I check this using an IF statement. The vending machine also has to be in Vending mode. If the user has equal or more money than the item and the state of the vending machine is in this state. The user is able to buy this item. This method does not check the qty of the item. The method returns a Boolean variable. If the purchase is possible, a true variable, returns.

```
public boolean purchaseRequest(int index) {
    boolean requestValid = false;
    try {
        if (getUserMoney() >= stock[index].getPrice()) {
            if(vmStatus != Status.VENDING_MODE) {
                input("Vending Machine is in Service mode, Contact Engineer!");
                return false;
            }
            requestValid = true;
        } else {
            input("Money entered is not enough for required item, please insert more coins");
            requestValid = false;
        }
    } catch (ArrayIndexOutOfBoundsException e) {
    }
    return requestValid;
}
```

addSetItems Method

This method is used for the default settings within the vending machine, if the 'structure.csv' file is not found. Therefore, this method is called within the constructor of the vending machine class. This is just a simple method that adds a set of default VendItem Objects that add towards the stock array at the start of the program. This method is privately accessed as it is

```
private void addSetItems() {
    String[] items = {"Coke", "Fanta", "Crisps", "Sweets", "Chicken", "Red Bull", "Hot Dog"};

    stock[itemCount++] = new VendItem(items[0],  unitPrice: 1.5,  qtyAvailable: 4);
    stock[itemCount++] = new VendItem(items[1],  unitPrice: 1.5,  qtyAvailable: 2);
    stock[itemCount++] = new VendItem(items[2],  unitPrice: 0.5,  qtyAvailable: 3);
    stock[itemCount++] = new VendItem(items[3],  unitPrice: 1.2,  qtyAvailable: 3);
    stock[itemCount++] = new VendItem(items[4],  unitPrice: 2.0,  qtyAvailable: 5);
    stock[itemCount++] = new VendItem(items[5],  unitPrice: 1.5,  qtyAvailable: 2);
    stock[itemCount++] = new VendItem(items[6],  unitPrice: 1.5,  qtyAvailable: 2);

}
```

engineerChange Method

This method is held within the Vending machine class of the program. It is called within the Part2 of the project. When the engineer wants to change the status of the vending machine. Therefore, when this method is called, the user will get an option of what State does he want the vending machine in.

```
~~ [1] Vending Mode [2] Service mode
~~:
```

Therefore, this method will determine what the user selects. For example, if the user enters the value '1'. The method will read this input and determine what this means. If the user selects 1, the vmStatus variable will be changed to 'VENDING_MODE', if the user selects the option '2'. The vmStatus will be changed towards 'SERVICE_MODE'. This method also validates input as well, if they enter anything except these options, the method will ignore and print text, saying the vending mode has not been changed.

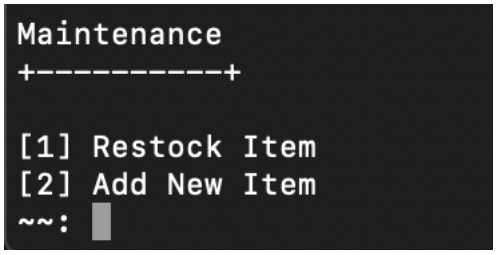
```
public void engineerChange() {
    int choice = 0;
    input("[1] Vending Mode" + " [2] Service mode\n~~: ");
    try {
        choice = sc.nextInt();
    } catch (Exception e) {
        input("Please enter one of the following options!");
    }

    if (choice > 2)
        System.out.print("\n~~ Please enter one of the following options, Mode has not been changed!");

    switch (choice) {
        case 1:
            this.vmStatus = Status.VENDING_MODE;
            input("Vending-Machine mode has be changed to: " + vmStatus.getStatus() + "\n");
            break;
        case 2:
            this.vmStatus = Status.SERVICE_MODE;
            input("Vending-Machine mode has be changed to: " + vmStatus.getStatus() + "\n");
            break;
    }
}
```

maintenance Method

This method is used to give the engineer access to either restock an item or to add a new item Object towards the vending machine stock array. This method is publicly accessed as it used within the 'Main.java'. Therefore, if the user gets within this area of the program where this is called (Engineer Level). Where the user will get this option. The purpose of this method is to allow the items of the vending machine to be altered without entering and changing the source code.



This method contains a lot of validation to make sure that the correct input is entered. For example, an ID within the range. If the user selects the option 1, he will be able update the stock for an item. If the user enters the value 2, the 'newItemMaintenance' method is called.

```
public void maintenance() {
    int choice = 0;
    int id = 0;
    int qty = 0;
    System.out.println("\nMaintenance\n" +
            "+----------+");
    System.out.print("\n[1] Restock Item\n[2] Add New Item\n~~: ");

    try {
        choice = sc.nextInt();
    } catch (Exception e) {
        input("Please enter one of the following options");
    }

    //calling methods, depending on input
    switch (choice) {
        case 1:
            try {
                input("Item ID: ");
                id = sc.nextInt();
                input("Quantity: ");
                qty = sc.nextInt();
            } catch (Exception e) {
                input("Wrong input expected");
            }
            if (stock[id].restock(qty)) {
                stock[id].setQtyAvailable(qty);
                input("Item: '" + stock[id].getName() + "', Qty is now updated to: " + stock[id].getQty());
            } else
                input("Item restock amount is cannot be > 10 & >1, Restock Qty needs to be greater than current amount\nCurrent Qty of '"+stock[id].getName()+"' is: "+stock[id].getQty());
        break;
        case 2: newItemMaintenance();
    }
}
```

newItemMaintenance Method

The method is used within the engineer level of the program. As this will display the name, price and qty. The purpose of this method is to allow the user to add an extra item towards the vending machine. As the user is allowed to enter data about the VendItem object being created. If the data entered is validated correctly and no errors. The item will be created and added towards the stock array. Therefore, being able to buy and view this item for the user. This method is privately accessed, as it doesn't need to be called outside of this class. Which also returns nothing.

```java
private void newItemMaintenance() throws IllegalArgumentException {
    boolean validBreak = false;
    double price = 0.0;
    int qty = 0;
    String item = "";

    //Entering name
    System.out.print("Item Name: ");
    sc.nextLine();
    try {
        item = sc.nextLine();
    } catch (Exception e) {
        input("Invalid input expected");
        validBreak = true;
    }
    //Entering price
    System.out.print("Item Price: ");
    try {
        price = sc.nextDouble();
    } catch (Exception e) {
        input("Invalid input expected");
        validBreak = true;
    }

    if (price < 0) {
        input("Price can't be negative");
        validBreak = true;
    }
    if (price % 0.5 != 0) {
        input("Price isn't divisible by the Integer: 5. Price must be nomination of '0.05, '0.01");
        validBreak = true;
    }
    //Entering the quantity of the item.
    System.out.print("~~ Quantity: ");
    try {
        qty = sc.nextInt();
```

saveTextFileData Method

The purpose of this method is to write the data that has been changed or affected recently (Up to date Data). Therefore, this will find each piece of item along with their quantity sold. Therefore, depending on what items are sold today, once this method has been called. It will grab the information currently stored, from totally money. Along with items that have been sold. This data will be stored to the file, 'data.txt', which will look like this.

```
Sun Mar 08 13:53:43 GMT 2020
+----------------------------------------+
|            Vending Machine Data        |
+----------------------------------------+

               ITEMS SOLD
+-----------------------------------------+
   ID    |   QUANTITY SOLD     |    ITEM
   0     |         0           |    Coke
   1     |         0           |    Fanta
   2     |         0           |    Crisps
   3     |         3           |    Sweets
   4     |         1           |    Chicken
   5     |         0           |    Red Bull
   6     |         0           |    Hot Dog
+-----------------------------------------+
TOTAL AMOUNT MADE: £12.1
```

This method is privately accessed as it is not needed within other classes. This method is called within the program whenever the user tries to reset the data. The program asks the user whether they want to save the data or not. The method just formats data in a

structured way to look presentable. I use the Print Writer object, as it works well with writing data.

```
private void saveTextFileData() throws IOException {
    String print = "";
    Date date = new Date();
    date.getTime();
    PrintWriter writer = new PrintWriter( fileName: "data.txt");
    String title = date + "\n";
    title += "+-----------------------------------------------+" + "\n|            Vending Machine Data          |"
            + "\n+-----------------------------------------------+";

    print = title;
    print += "\n" + "\n               ITEMS SOLD" + "\n+-----------------------------------------------+\n";
    print += "   ID    |   QUANTITY SOLD     |    ITEM    \n";
    for (int i = 0; i < itemCount; i++) {
        print += "   " + i + "     |         " + stock[i].getAmountSold() + "           |    " + stock[i].getName() + "\n";
    }
    print += "+-----------------------------------------------+\nTOTAL AMOUNT MADE: £"
            + this.totalMoney;
    writer.println(print + "\n");
    writer.close();
}
```

Change Method

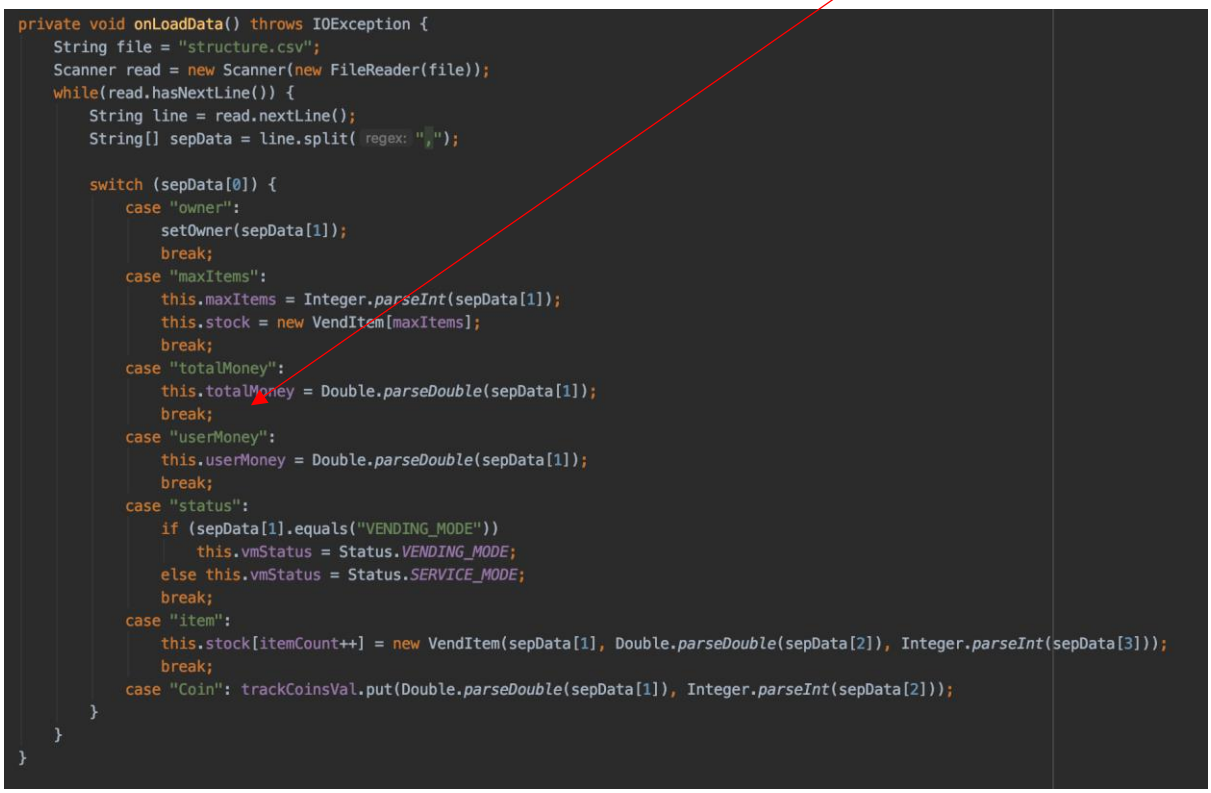
This method is used to help me determine the change that a user is due. How it works is, it iterates through a double array, which lists the coins. In a descending order. Which will compare if the change is larger than the coin type. If true, it will then try how many times if will go into this change. For example, 0.50 goes into 1.20 twice. By working this out and going from a Descending order, what is the most efficient way of giving the change back towards the user. However, once we find a coin that is smaller than the change, we also check if the amount availability of that coin is greater than 0. If it is, this will reduce the change amount by the amount of times if can fit within the change, multiplied by the coin. Every time a coin is taken away depending on what coin is used. However, if the algorithm goes throughout the array and finds that the change is greater than 0. If Will print out that the change is not possible to give back due to coin availability.

```
private String change(double change) {
    String res = "";
    ArrayList<Double> list = new ArrayList<>();

    while (change > 0) {
        for (int i = coinType.length - 1; i >= 0; i--) {
            if (change >= coinType[i] && trackCoinsVal.get(coinType[i]) > 0) {
                int count = (int) (change / coinType[i]);
                change = round( value: change - count * coinType[i], places: 2);
                res += count + "x " + coinType[i]+"| ";
                for(int j = 0; j < count; i++)
                    trackCoinsVal.put(coinType[i], trackCoinsVal.get(coinType[i]) - 1);
            }
        }
        if (change > 0) {
            res = "The amount of coins available, doesn't create the nomination of the change that is due, Please ask Engineer for support";
            return res;
        }
    }
    return res;
}
```

OnloadData Method

The purpose of this method is to load the data that has been previously saved onto the 'structure.csv' file. Therefore, the Vending machine will try to load the data if the file is found, however if it is not. It will load the default settings. The way this method works, is that is will read the file and separate it with commas. Depending on first word found, this will determine what the remaining data will do on the same line. For example, "UserMoney, 1.0". The method will read and find out that if the first word is UserMoney, then It will look the second part of the splitted line and store that value within the variable userMoney of the vending Machine object.



```
private void onLoadData() throws IOException {
    String file = "structure.csv";
    Scanner read = new Scanner(new FileReader(file));
    while(read.hasNextLine()) {
        String line = read.nextLine();
        String[] sepData = line.split( regex: ",");

        switch (sepData[0]) {
            case "owner":
                setOwner(sepData[1]);
                break;
            case "maxItems":
                this.maxItems = Integer.parseInt(sepData[1]);
                this.stock = new VendItem[maxItems];
                break;
            case "totalMoney":
                this.totalMoney = Double.parseDouble(sepData[1]);
                break;
            case "userMoney":
                this.userMoney = Double.parseDouble(sepData[1]);
                break;
            case "status":
                if (sepData[1].equals("VENDING_MODE"))
                    this.vmStatus = Status.VENDING_MODE;
                else this.vmStatus = Status.SERVICE_MODE;
                break;
            case "item":
                this.stock[itemCount++] = new VendItem(sepData[1], Double.parseDouble(sepData[2]), Integer.parseInt(sepData[3]));
                break;
            case "Coin": trackCoinsVal.put(Double.parseDouble(sepData[1]), Integer.parseInt(sepData[2]));
        }
    }
}
```

OnStoreData Method

The OnStoreData is used to store the data, whenever the program is closing using the menu of vending machine. The purpose of this class, as this will save the structure of the vending machine back towards the'structure.csv 'file. Therefore, overwriting the data. For example, if the user creates an object for an item. The next time the program is loaded, it will load the data that was stored the last time the program was closed. Therefore, the data will never be lost.

```
public void onStoreData() throws FileNotFoundException {
    PrintWriter writer = new PrintWriter( fileName: "structure.csv");

    writer.write( s: "owner,"+this.owner+"\n");
    writer.write( s: "maxItems,"+this.maxItems+"\n");
    writer.write( s: "totalMoney,"+this.totalMoney+"\n");
    writer.write( s: "userMoney,"+this.userMoney+"\n");
    writer.write( s: "status,"+this.vmStatus+"\n");

    for(int i = 0; i < itemCount; i++)
        writer.write( s: "item,"+stock[i].getName()+","+stock[i].getPrice()+","+stock[i].getQty()+"\n");

    for(int i = 0; i < coins.size(); i++)
        writer.write( s: "Coin,"+coinType[i]+","+trackCoinsVal.get(coinType[i])+"\n");
    writer.close();
}
```

```
~~ [Machine Is Now In 'Service Mode' And Has Now Shut Down]
~~ Saving data...
~~ Vending machine structure has been saved
```

Round Method

This method is used to fix the error that was occurring within the change method.
As working with precise numbers can cause an error, such '1.4-0.2'. Due to IEEE floating point format, the result of this might end like '1.19999999997' Therefore, using this method has allowed me to round my change to two decimal places. The value is what gets taken and rounded, the places parameter takes into how many places it gets rounded by.

```
public static double round(double value, int places) {
    if (places < 0) throw new IllegalArgumentException();

    long factor = (long) Math.pow(10, places);
    value = value * factor;
    long tmp = Math.round(value);
    return (double) tmp / factor;
}
```