



Faculty of Engineering and Applied Science
SOFE 3650U - Software Design and Architecture
Exercise/Tutorial 3

Exercise/Tutorial Group Number: 20

Group Member 1

Name: William Zdenek Chamberlain

Student ID: 100846922

Group Member 2

Name: Jason Manarrou

Student ID: 100825106

Group Member 3

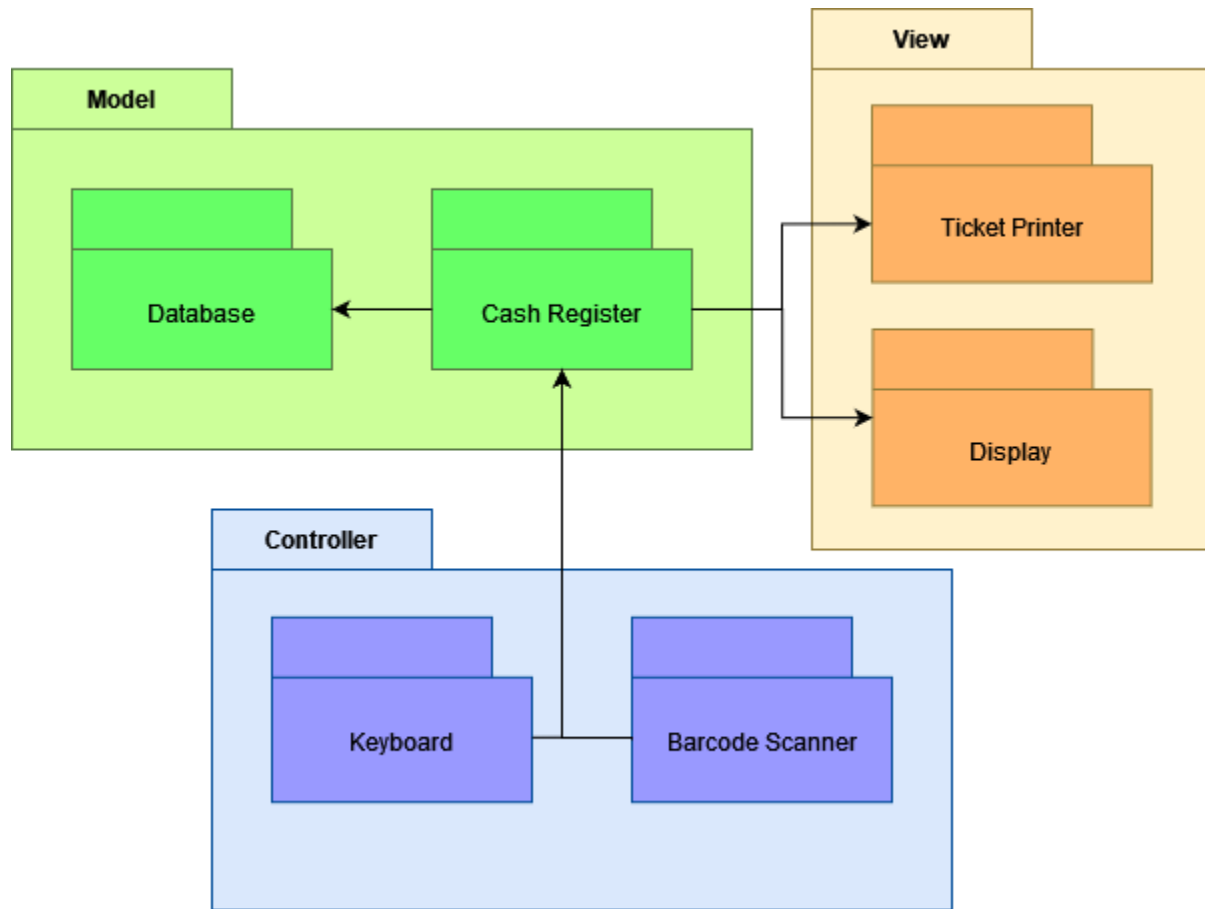
Name: Trent Jordan

Student ID: 100831853

Date: September 29th 2023

Tutorial CRN: 43510

UML Architecture Diagram

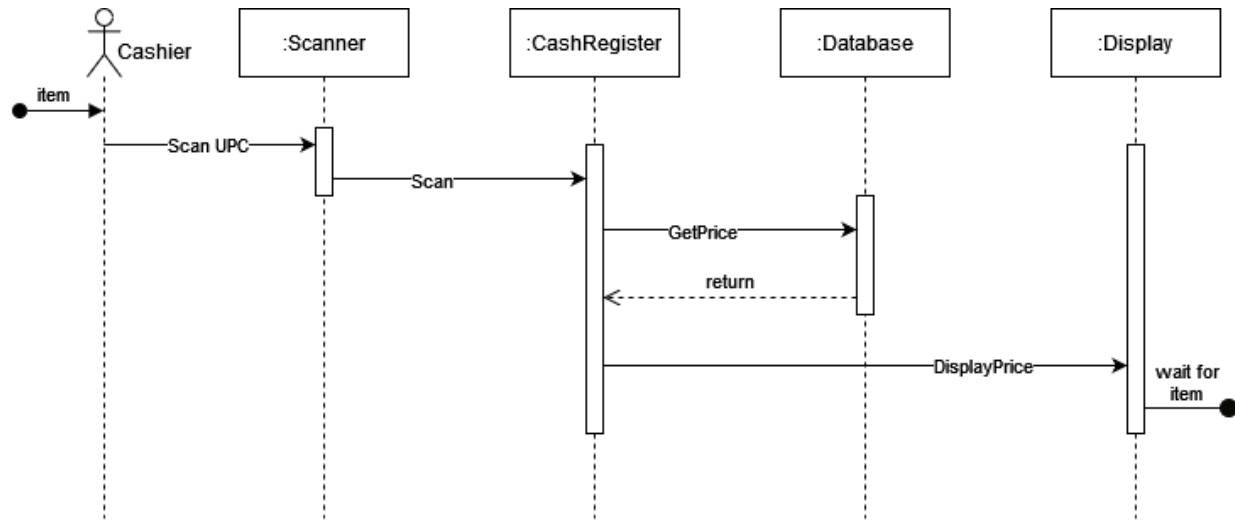


The View package contains the Ticket Printer and the Display, as these are the output/viewing devices of the cashregister design. These devices fulfill the classic interactions of the View package components as they are updated by the Model and then query the model for the Updated state (like an Observer).

The Controller package contains the Keyboard and the Barcode Scanner, as these are the devices that are capable of updating the state of the Model package and updating the View (either directly with the keyboard or indirectly through the scanner and cash register).

The Model package contains the Cash Register and the Database as these are responsible for the business logic and are never interacted with directly by the user. The Cash Register is updated by the Keyboard and Barcode Scanner (Controller package), and updates the Ticket Printer and Display (View package).

UML Sequence Diagram



Here the cashier (an actor), interacts with the scanner to scan an object. The scanner then calls upon the cash register with the info scanned. The cash register then calls upon the database for the price and name of the product associated with the UPC. The cash register then calls upon the display to notify it of the state change. The system then waits for the next input.

Implementation

```

Driver [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (Sep 29, 2023, 6:15:35 PM)
036000291452
Current Working DIR: /home/jason/eclipse-workspace/Software-Design-Tutorials/EX3
-----
Apples_Red
-----
036000291466
Current Working DIR: /home/jason/eclipse-workspace/Software-Design-Tutorials/EX3
-----
Apples_Red
Bananas
-----
036000291700
Current Working DIR: /home/jason/eclipse-workspace/Software-Design-Tutorials/EX3
-----
Apples_Red
Bananas
Cucumbers
-----
    
```

Above is the **OUTPUT** ! In light blue, are some of my inputs handled by the keyboard, as you can see the system looks for the UPC (Universal Product Code) in the DB and then adds the item to the display.

We've used a **Driver** Class to instantiate the CashRegister, and call the `run()` method, this is to simulate an environment for multiple cash registers, EG instantiating **CR2** & **CR3** !

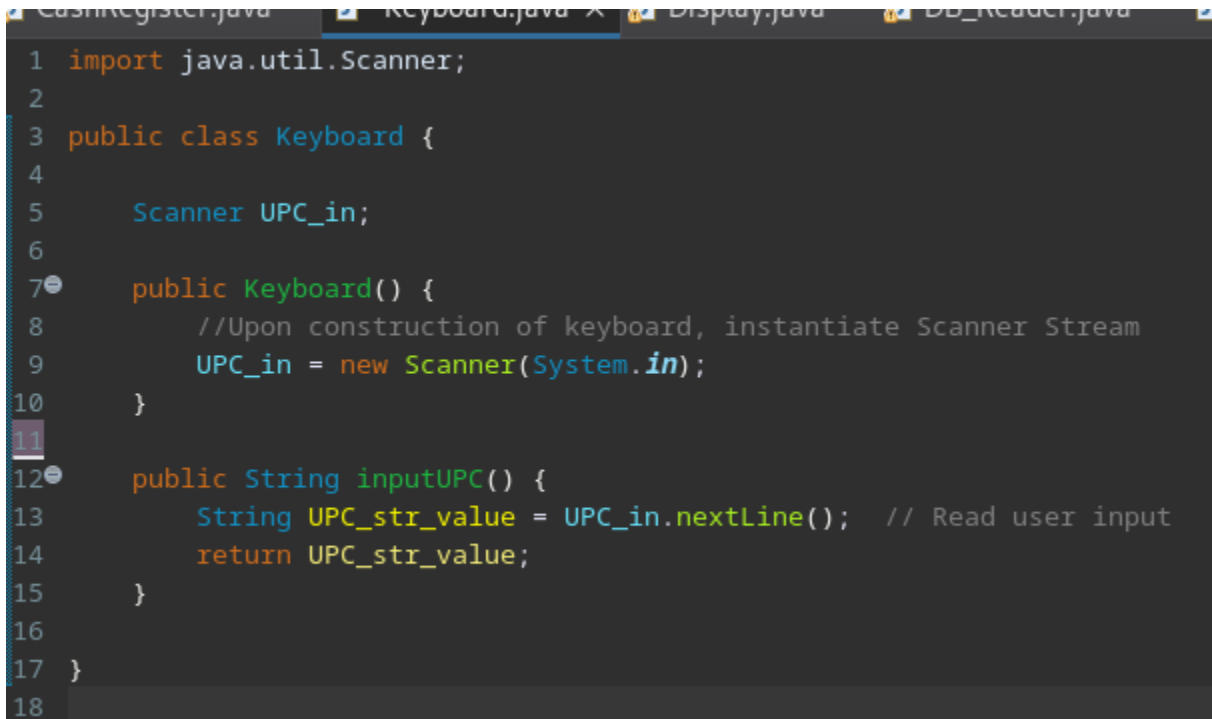
```

1 public class Driver {
2
3     public static void main(String[] Args) {
4         CashRegister CR1 = new CashRegister();
5         CR1.run();
6     }
7
8 }
9
    
```

Here's the **Cash Register** class, the Run method is always listening for Keyboard inputs, (*Line 12*) and then once a UPC is inputted, the Product Name (*UPC_name*) is fetched from the Database, as it searches for the UPC and then returns the Name of the Product, which is then stored in the display object, so we can keep track of all the items inputted on the screen.

```
1  import java.util.Scanner;
2
3  public class CashRegister {
4
5      Keyboard keyboard1 = new Keyboard();
6      Display display1 = new Display();
7
8      public void run() {
9
10         while(true) {
11
12             String Input_UPC = keyboard1.inputUPC();
13             String UPC_Name = DB_Reader.getItemFromDatabase(Input_UPC);
14
15             display1.Add(UPC_Name);
16             display1.Display();
17
18         }
19
20     }
21
22 }
23
```

Here's the **Keyboard** class, upon construction it constructs the Input Stream Object, and stores it UPC_in. When `inputUPC()` is called, it just prompts the user (*input stream*) and returns the imputed value.



```
1 import java.util.Scanner;
2
3 public class Keyboard {
4
5     Scanner UPC_in;
6
7     public Keyboard() {
8         //Upon construction of keyboard, instantiate Scanner Stream
9         UPC_in = new Scanner(System.in);
10    }
11
12    public String inputUPC() {
13        String UPC_str_value = UPC_in.nextLine(); // Read user input
14        return UPC_str_value;
15    }
16
17 }
18
```

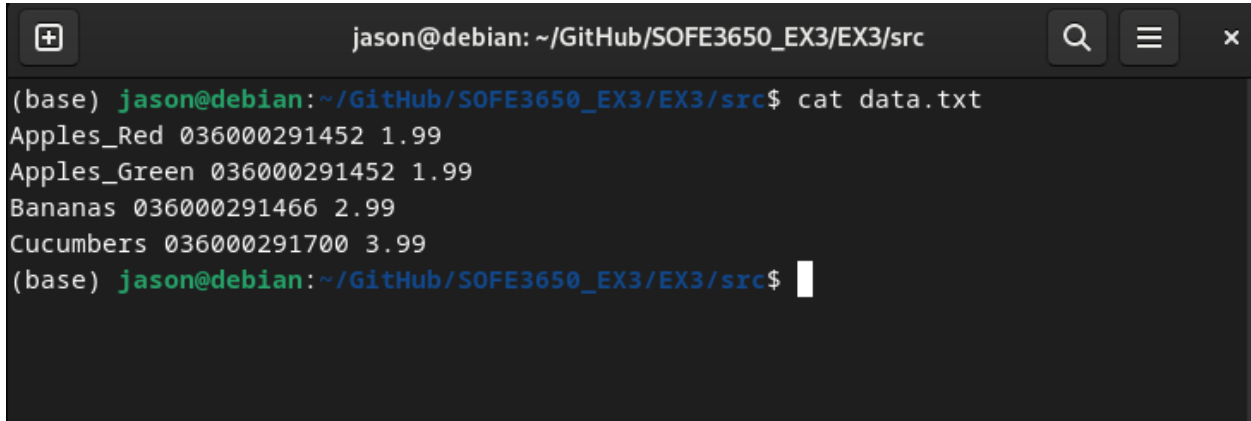
Here's the **Display** class, where you can Add Items to the list of Strings to be displayed, then also be able to display it (*System Out*). In the CashRegister we get an input, find the UPC code and respective name, then append that name into the Items list via [Add\(\)](#), and then [Display\(\)](#).

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3
4 public class Display {
5
6     ArrayList<String> Items = new ArrayList<String>();
7
8     public void Add(String Item) {
9         Items.add(Item);
10    }
11
12    public void Display() {
13        System.out.println("-----");
14
15        for (String item : Items) {
16            System.out.println(item);
17        }
18
19        System.out.println("-----");
20    }
21 }
22
```

The **DB_Reader** class essentially reads in files from the Database (*data.txt*) so that an inputted UPC String Code is searched for, and the associated Product Name is found and returned.

```
14 import java.io.File;
5
6 public abstract class DB_Reader {
7
8     public static String getItemFromDatabase(String inputUPC) {
9         System.out.println("Current Working DIR: " + System.getProperty("user.dir"));
10        ArrayList<String> readLines = new ArrayList();
11
12        try {
13
14            //Read File & Scan it
15            File myObj = new File("src/data.txt");
16            Scanner myReader = new Scanner(myObj);
17
18            while (myReader.hasNextLine()) {
19                String data = myReader.nextLine();
20                readLines.add(data);
21            }
22
23            myReader.close();
24        } catch (FileNotFoundException e) {
25            System.out.println("An error occurred.");
26            e.printStackTrace();
27        }
28
29        //Actually parsing thru the Read Lines, and outputting
30        for(int i = 0; i < readLines.size(); i++) {
31
32            String[] parts = readLines.get(i).split(" ");
33
34            String productName = parts[0];
35            String productUPC = parts[1];
36
37            if(inputUPC.equals(productUPC)) {
38                return productName;
39            }
40        }
41        return "";
42    }
43 }
44
```


Also, here is the ``src/data.txt`` that it's reading (Database):

A terminal window with a dark background. The title bar shows 'jason@debian: ~/GitHub/SOFE3650_EX3/EX3/src'. The terminal content shows a command prompt '(base) jason@debian:~/GitHub/SOFE3650_EX3/EX3/src\$' followed by 'cat data.txt'. The output lists four items: 'Apples_Red 036000291452 1.99', 'Apples_Green 036000291452 1.99', 'Bananas 036000291466 2.99', and 'Cucumbers 036000291700 3.99'. The prompt returns to '(base) jason@debian:~/GitHub/SOFE3650_EX3/EX3/src\$' with a cursor.

```
(base) jason@debian:~/GitHub/SOFE3650_EX3/EX3/src$ cat data.txt
Apples_Red 036000291452 1.99
Apples_Green 036000291452 1.99
Bananas 036000291466 2.99
Cucumbers 036000291700 3.99
(base) jason@debian:~/GitHub/SOFE3650_EX3/EX3/src$
```

This is the data.txt file, acting as the Database of our implementation. As you can see in the source code, we just use **Element[0]** (*Product Name*) and **Element[1]** (*UPC*)

Source Code LINK:

https://github.com/jasonmzx/SOFE3650_EX3/tree/main/EX3/src