**DOCUMENTATION GRUPP 9**

**GENERAL STRUCTURE OF THE PROJECT**

The project’s GUI is built by using JavaFX.

Scene Builder has been used to produce the GUI layout. As a result, the src/main folder contains a /resource folder containing the .fxml files produced by Scene Builder. Some elements in these files are accessed as objects in the program code, and buttons are linked to certain functions using the onAction-tag in the .fxml files.

The project’s source code is found in src/main/java/com/example/pos\_system\_version\_xx/.

The project’s main() function is found within GUIApplication.java, which is the file the program is run from.

**RUNNING THE PROGRAM**

In order to run the program, the user should start the various peripherals (CashBox, CardReader, ProductCatalog etc.) by running their respective .jar files. Java8 has been used by Grupp 9 to run these files, as the newest versions of Java are not compatible.

When the necessary peripherals have been started, the program may be started by running GUIApplication.java.

**STARTUP**

When the program is running it will first call the JavaFX launch() function. This will eventually return control back to the GUIApplication by running the start() function. Here the GUIs are made visible, and assigned to objects that can be used to call the functions attached to each GUI window.

By using the getResource() function, the FXMLLoader objects are linked to their respective .fxml files.

The cashier object is given an eventHandler, such that the events fired by the CashierGUI can be handled in GUIApplication.

**CASHIER GUI**

The CashierGUI is the GUI handling the sales taking place. The CustomerGUI is only used to display information to the customer as the products are scanned and at last paid for.

When the program starts, the Product Catalog table is loaded with all the products in the ProductCatalog.jar database.

**USING CASHIER GUI**

There are two methods of adding products to the Shopping Cart table:

1. Enter the barcode of an existing product in the textbox under the “Scan barcode” label, and then press the “Scan” button.
2. Click on one of the products in the Product Catalog table, and then press the “Add” button.

If the Product Catalog table contains many products, it may be useful to search for a product by its name or keyword. This can be done by entering a name or barcode in the textbox under the “Search By Keyword” label. Pressing “Search” causes the Product Catalog table to only contain products with that name or keyword.

To make the Product Catalog table contain all products again: enter an empty string or a \* in the textbox, and then press “Search”.

If the cashier wishes to remove a product from the Shopping Cart table, this is done by clicking on one of the products in that table, and then pressing the “Remove” button.

When a product has been added or removed to/from the Shopping Cart table, the Total will be updated in both the Cashier and Customer GUIs. The CustomerGUI will also update the Shopping Cart table whenever the cashier has added or removed a product.

It is possible for the cashier to place discounts on products, by using the “Discount” button. Before pressing the button, the cashier should write a number representing the percentage to remove from a product’s price, and then click on the product (in the Shopping Cart table) which is given the discount. Only once that is done, should the cashier actually press the “Discount” button.

The “Shelf” button in the top-right can be used to store an ongoing sale, which can be returned to later. While a sale is shelfed, the cashier can serve other customers. If the cashier has shelfed a sale, they can get it back by pressing “Shelf” again. No ongoing sale should be taking place at that time. The “Shelf” functionality is then essentially an on/off button for shelfing and retrieving a shelfed sale. Only one sale can be shelfed at a time.

**STARTING PAYMENT**

To complete the sale, the customer should pay for the products.

To start the payment, the cashier should enter the amount of cash the customer paid in the textbox under the “Cash Received” label. If the customer wishes to pay it all by card, and therefore pays no cash, the cashier must enter an empty string or 0.0 in the textbox to indicate this. Then the cashier may press the “Start payment” button, which will start the payment process.

If the customer has paid enough cash to cover the cost, the amount of money the cashier should return is written next to the “Change” label.

If the customer must pay a portion by card, the Card Reader will be active, such that a credit card may be scanned by it.

When the sale has been finished, this is indicated by a green text string.

In order to start a new sale, the cashier should press the “Reset” button. This will clear the shopping cart, and set the total back to 0.0

A receipt is automatically stored as an image file after each sale is completed. The receipts are stored at the highest level of the project’s file hierarchy.

**SALESMAN GUI**

The SalesmanGUI is handling the ability to adjust the price of a product. The GUI displays a table of all items retrieved from the Product Catalog.

When the program starts, the Product Catalog table is loaded with all the products in the ProductCatalog.jar database.

**USING SALESMAN GUI**

To adjust the price of a product, you “click” (double click) on the price cell of the product you want to change and then proceed to edit the cell to a value of your choice. To confirm the change you just made you need to press “enter”. The new prices are updated in the CashierGUI after you use the reset button (which updates the tables).

**WHAT HAPPENS WHEN YOU PRESS A BUTTON IN CASHIER GUI?**

When a button is pressed in the CashierGUI, it runs the function specified in the button’s onAction-tag in the .fxml file. For the CashierGUI, these functions are found in the CashierGUI.java file, and most of the functions begin with the word “request”, such as requestAddProduct(). Each such function is supposed to fire an event, which may take some data from the GUI’s textboxes, such as a barcode.

The event is then handled in GUIApplication, which contains an eventHandler. These functions often begin with the word “on”, such as onAddProductRequested, in our example.

These GUIApplication functions might then perform some logic on the arguments, make some requests to the various peripherals, and then call GUI functions to update them depending on the changes.

When a request is to be made, or certain logic regarding a sale/order be made, GUIApplication calls a function in GUIController. GUIController is primarily responsible for storing orders, which contain the products and total price for a sale; and communication with peripherals.

Communication with peripherals is done by calling functions in the RequestHandler class, which handle the HTTP requests to the peripherals and return the response after a request has been made.

In the case of adding a product, a request is made to the ProductCatalog, to find a product by the barcode specified. If a product is found with that barcode, the product is returned in xml form. An xml-reader creates an actual product, which can then be used in GUIApplication and the rest of the program. At this stage, GUIApplication calls the functions addProduct(), which exists both in CashierGUI and CustomerGUI, which update their respective Shopping Cart tables to list the newly added product.

In an abstract form, the flow following a button click, can be described like this:

Button is clicked -> CustomerGUI fires an event -> event is handled -> GUIApplication handles the event [requests and logic is performed] -> the GUIs update their tables or labels if necessary.

**ADDITIONAL REMARKS REGARDING CLASSES**

There are two types of product classes. One named Product used in the backend (GUIApplication and GUIController), and one named PRODUCT\_TEST\_CLASS used in the frontend (GUI classes). This is because the GUI tables require objects of type Simple<something>Property (etc.) which are different from ordinary <something> objects, such as String and double.