1- PURPOSE OF THE DOCUMENT

The purpose of this document is to explain the “FoodHorse Supermarket Chain” project given as an assignment in the CS202 course. In the document

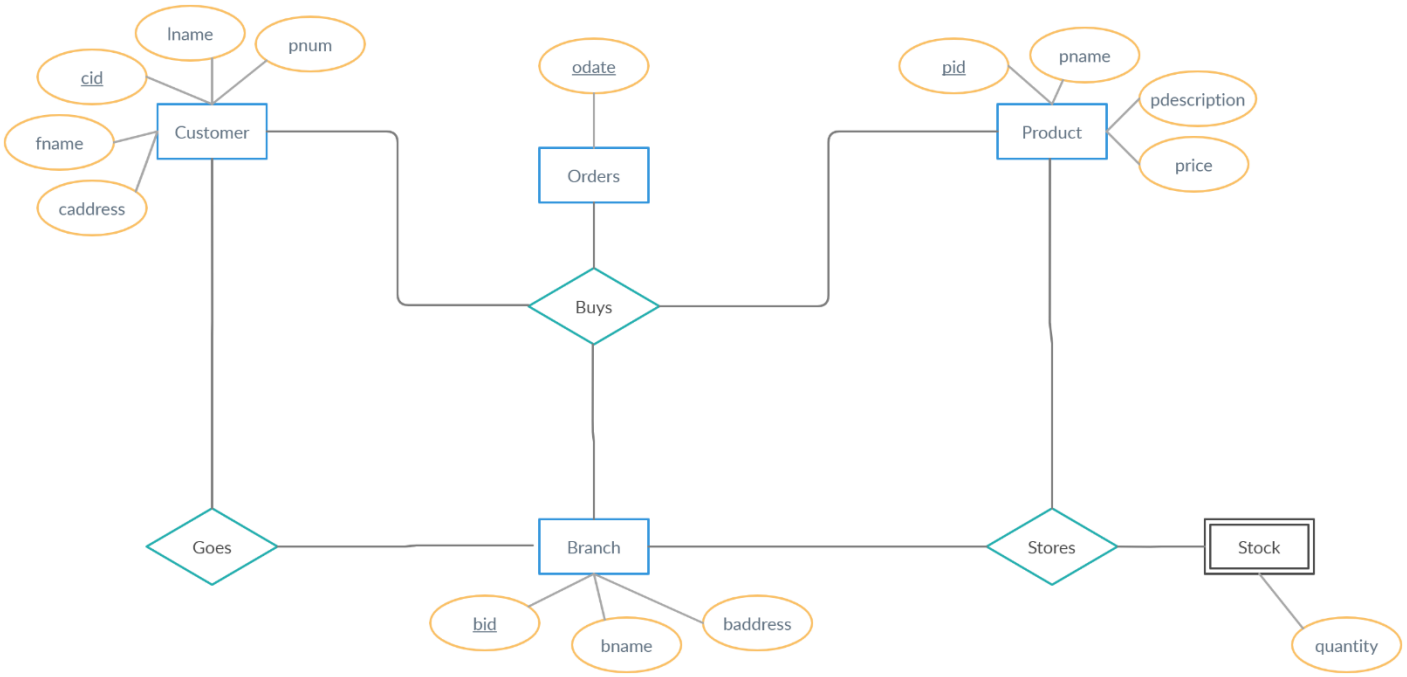
- ER Diagram (prepared in order to facilitate the database design)
- DDL & DML Statements (prepared to create tables in the database and insert sample data into it)
- Java Console Application (prepared to interact with the database by taking input from the user)

will be examined and explained in detail.

2- PROJECT DESCRIPTION

First of all, in the project, I was asked to prepare a database design for the FoodHorse supermarket chain. In the design, I was asked to keep information about customers, products, branches, shopping done by customers, and stocks in branches, and the design of customers, products, and branches was given to me as an example beforehand. Therefore, in addition to these three tables, I added the "Orders" table to keep the information of shopping made by the customers and the "Stock" table to keep the information of stocks in the branches.

2.1 ER Diagram



In the diagram I have prepared above, you can see 5 entities. These are all used as tables in my database. Customer entity has 5 attributes and this entity has a primary key attribute named "cid" (customer id) so that each customer can be uniquely identified. Branch entity has 3 attributes, 1 of which "bid" (branch id) is a primary key. Product entity has 4 attributes and one of them (pid) (product id) is the primary key. Since these entities I have listed are not determined by me, I will not give much detail about them.

Again, I mentioned above that I have created a table named Orders in order to keep information about customers' purchases in the database. When creating "Orders" I first looked into what should be kept in this table. Attributes that should be kept in "Orders" were customer id, product id, branch id and quantity. When creating "Orders", I first

looked into what should be kept in this table. Attributes that should be kept in "Orders" were customer id, product id, branch id and quantity. That's why I thought that both customer, product and branch entities should be bound to "Orders". But I couldn't decide whether "Orders" should be a relationship or an entity. Therefore, I checked the lecture notes in LMS and compared the status of "Orders" to the “semester” example in “Fall2020CS202Week2-ERPracticeUpdated.pdf”, and I decided that "Orders" should be an entity because all orders, not recent orders, were required to be kept in the database. Then, I linked customer, branch and product entities to "Orders" entity with the "buys" relationship. I used the "cid", "pid" and "bid" from other entities that are connected to the "Orders" entity as both foreign key and primary key in the "Orders" entity. I made sure that the “odate” (order date) attribute in the "Orders" entity is also a primary key because if this attribute was not a primary key, a customer could only buy a product from a branch once.

Later, I decided whether "Stock" should be relationship or entity in the same way. But when I was designing "Stock", I decided that the quantity attribute should not be a primary key because if this attribute was a primary key, different quantities could be given to the same product in the same branch and since I did not define this attribute as primary key, I had to design the "Stock" entity as a weak entity. Then, paying attention to the cardinality constrains, I drew my ER diagram as above but I am just not sure of the cardinality of the links between the "Orders" entity and the “Buys” relationship and between the "Stock" entity and “Stores” relationship.

2.2 DDL Statements

While I was creating my DDL statements, I checked the MySQL documentation from Google.

```
1 •   CREATE DATABASE IF NOT EXISTS FoodHorse;
```

In the documentation, the keyword "if not exists" was used. So I decided to use it too. After creating my database with the above code, I started creating my tables. I decided to write as *Foodhorse.Customer* because I couldn't find it even though I searched how to set the default database in DDL on google.

```
3 • CREATE TABLE IF NOT EXISTS FoodHorse.Customer(
4       cid int AUTO_INCREMENT PRIMARY KEY,
5       fname varchar(30) NOT NULL,
6       lname varchar(30) NOT NULL,
7       caddress varchar(30) NOT NULL,
8       pnum varchar(11) NOT NULL
9   );
```

“Not null” keyword also has been used in the documentation and it made sense to me that the attributes I wrote "not null" next to cannot be null, so I also used this keyword. Similarly, I created the product and branch tables.

```
11 • CREATE TABLE IF NOT EXISTS FoodHorse.Branch(
12      bid int AUTO_INCREMENT PRIMARY KEY,
13      bname varchar(50) NOT NULL,
14      baddress varchar(30) NOT NULL
15  );
```

```
17 • CREATE TABLE IF NOT EXISTS FoodHorse.Product(
18       pid int AUTO_INCREMENT PRIMARY KEY,
19       pname varchar(50) NOT NULL,
20       pdescription varchar(100) NOT NULL,
21       price float NOT NULL
22   );
```

Then I created the "Orders" and "Stock" tables.

```
24 • CREATE TABLE IF NOT EXISTS FoodHorse.Orders(
25       odate varchar(19) NOT NULL,
26       cid int,
27       pid int,
28       bid int,
29       FOREIGN KEY (cid) REFERENCES FoodHorse.Customer(cid),
30       FOREIGN KEY (pid) REFERENCES FoodHorse.Product(pid),
31       FOREIGN KEY (bid) REFERENCES FoodHorse.Branch(bid),
32       PRIMARY KEY (odate, cid, pid, bid)
33   );
34
35 • CREATE TABLE IF NOT EXISTS FoodHorse.Stock(
36       quantity int NOT NULL,
37       pid int,
38       bid int,
39       FOREIGN KEY (pid) REFERENCES FoodHorse.Product(pid),
40       FOREIGN KEY (bid) REFERENCES FoodHorse.Branch(bid),
41       PRIMARY KEY (pid, bid)
42   );
```

Inside my "Orders" entity, I created the "odate" attribute as varchar (19) because I wanted the entered value to match the format "yyyy-mm-dd hh: mm: ss". I have defined cid, pid and bid attributes as both foreign and primary keys. Finally, I created the "Stock" table as you can see.

2.3 DML Statements

The only interesting part I can mention here is that I found on google that I don't have to write insert into each time to insert multiple rows.

```sql
INSERT INTO FoodHorse.Customer(fname, lname, caddress, pnum)
VALUES
    ('Seçkin', 'Ağırbaş', 'Alibeyköy', '05001112233'),
    ('Mahmut', 'Acar', 'Kartal', '05002221133'),
    ('Merve', 'Özkan', 'Beşiktaş', '05003332211'),
    ('Aysel', 'Tekin', 'Kadıköy', '05002223311'),
    ('Orhan', 'Yavuz', 'Ataşehir', '05003331122');

INSERT INTO FoodHorse.Branch(bname, baddress)
VALUES
    ('Alibeyköy Center', 'Alibeyköy'),
    ('Kadıköy Center', 'Kadıköy'),
    ('Kadıköy Pier', 'Kadıköy'),
    ('Beşiktaş Square', 'Beşiktaş'),
    ('Taksim Square', 'Taksim');

INSERT INTO Foodhorse.Product(pname, pdescription, price)
VALUES
    ('FoodHorse Olive Oil', 'FoodHorse brand olive oil 1L', 11.50),
    ('FoodHorse Rice', 'FoodHorse brand rice 2.5kg', 13.75),
    ('FoodHorse Milk', 'FoodHorse brand whole milk 1L', 3.75),
    ('FoodHorse Kosher Dill Pickles', 'FoodHorse brand pickles 680g', 4.45),
    ('FoodHorse Strawberry Jam', 'FoodHorse Strawberry Jam 380g', 6.45);
```

2.4 Java Console Application

Firstly, I watched the video Emir Arditi uploaded to LMS to connect this application to my MySQL database.

```java
    private static final String URL = "jdbc:mysql://localhost:3306/FoodHorse?useSSL=false";
    private static final String username = System.getenv( name: "UNAME");
    private static final String password = System.getenv( name: "PWORD");
    private static final Scanner scanner = new Scanner(System.in);
    private static int userInput = 0;
    private static Connection connection = null;
```

Then, as Emir wrote in the video, I defined a global variable named URL. Since I don't want my username and password to appear, I saved them in environment variables and accessed their with *System.getenv()* method. Then, every time I got input from the user, I did not want to close the scanner because once I closed, I could not receive input from the user again. So I defined the scanner as a global variable and closed it with the *scanner.close()* method before the application ended. Likewise, I have defined connection and *userInput* as global variables.

```java
public static void establishConnection(){
    try{
        connection = DriverManager.getConnection(URL, username, password);
    }
    catch(SQLException e){
        System.out.println("There was an error establishing the connection");
    }
}

public static void closeConnection(){
    try{
        if(connection != null)
            connection.close();
    }
    catch(SQLException e){
        System.out.println("There was an error closing the connection");
    }
}
```

Later, I added the *establishConnection()* and *closeConnection()* methods that Emir gave in the video to my application in this way.

```java
public static void displayMenu(){
    System.out.println("-------------------------------------------");
    System.out.println("(1) Customer registration");
    System.out.println("(2) Buying a product");
    System.out.println("(3) List customers");
    System.out.println("(4) List a customer's purchases");
    System.out.println("(5) List a customer's most recent purchases");
    System.out.println("(6) List branches");
    System.out.println("(7) List a branch's stock");
    System.out.println("(8) Add new branch store");
    System.out.println("(9) Add new product");
    System.out.println("(10)Add product to branch's stock");
    System.out.println("(11)Search customer by their phone number");
    System.out.println("(12)Remove a user from system");
    System.out.println("(13)Exit");
    System.out.println("-------------------------------------------");
}
```

Then I created the function named *displayMenu()* to present the options to the user.

```java
public static void getUserInput(){
    System.out.print("Please enter a number between 1-13: ");
    String tempInput = scanner.nextLine();
    while(true){
        try{
            userInput = Integer.parseInt(tempInput);
            if(userInput>=1 && userInput<=13) {
                break;
            }
            else{
                System.out.print("Your input is not between 1-13. Please enter a number between 1-13: ");
                tempInput = scanner.nextLine();
            }
        }
        catch(Exception e) {
            System.out.print("Your input was invalid, please enter a number between 1-13: ");
            tempInput = scanner.nextLine();
        }
    }
}
```

Later, I wrote the *getUserInput()* function to enable the user to access these options that I have presented. This function firstly checks if the user input is a number, then if it is a number, it checks whether the number is between 1 and 13.

```java
public static void main(String[] args){
    try {
        establishConnection();
        while(userInput != 13){
            displayMenu();
            getUserInput();
            handleFunctions();
        }
        scanner.close();
        closeConnection();
    }
    catch(Exception e){
        e.printStackTrace();
        System.out.println("There is a problem in the main function.");
    }
}
```

In the main method, the connection is first established, then it is checked whether the user input is 13. Since *userInput* is defined as 0 among global variables, the program definitely gets into the while loop for the first time. Then, in the while loop, options are presented to the user with the *displayMenu()* function and the user is expected to enter a number with the *getUserInput()* function. The while loop continues to run until the user enters 13, and when the user enters 13, the while loop is exited, first the scanner then the connection is closed and the program ends. The handleFunctions () method, which is the only method I haven't explained in the screenshot above, decides which function will be called according to the user's input. For example, if the user enters 1, the function handle1 () will be called.

2.4.1 *handle1();* “Customer Registration”

```
724     public static void handle1(){
725         String customerName = getCustomerInfo("name");
726         String customerSurname = getCustomerInfo("surname");
727         String customerAddress = getCustomerInfo("address");
728         String customerPhoneNumber = getCustomerInfo("phone number");
729         addCustomer(customerName, customerSurname, customerAddress, customerPhoneNumber);
730     }
```

In this function, firstly, name, surname, address and phone number are obtained from the user with the help of *getCustomerInfo()* method. Within the *getCustomerInfo()* method, the values entered by the user are checked with utmost care and if the entered values are not compatible, the user is asked to enter values again. Then, a customer with these values is created in the database with the *addCustomer()* method. You can see a test I've done below.

```
Please enter a number between 1-13: 1
Please enter customer name: oqw1ueoq1usdjlkwjlda1usdlakwldkauselakjsdlasjdqow1uasd
Name can be up to 30 letters.
Please enter customer name: John
Please enter customer surname: qlkwjalksjdlkweklausdlrfmnasdfo1wejlksaflahwejkahskaşlkds
Surname can be up to 30 letters.
Please enter customer surname: Doe
Please enter customer address (only city for simplicity): lqkwjelkajsdkljwqlue1laudlakjdlkauldeuaskdljlawkjda
Address can be up to 30 letters.
Please enter customer address: Izmir
Please enter customer phone number: Van
Phone number must be 11 digits. Like (05551112233)
Please enter customer phone number: onbirharfli
Your input is invalid.
Please enter customer phone number: 01112223344
Customer has been added to the db successfully.
```

2.4.2 *handle2();* “Buying a Product”

```
733     public static void handle2(){
734         int productId = Integer.parseInt(getProductInfo("id"));
735         int branchId = Integer.parseInt(getBranchInfo("id"));
736         if(checkStock(productId, branchId)){
737             int customerId = Integer.parseInt(getCustomerInfo("id"));
738             String orderDate = getOrderDate();
739             addOrder(customerId, productId, branchId, orderDate);
740             int quantity = getQuantity(productId, branchId);
741             uptadeStock(productId, branchId ,quantity);
742         }
743         else{
744             System.out.println("Sorry, the product you have specified is not in the stock of the" +
745                     " branch you have specified.");
746         }
747     }
```

In this method, *productId* and *branchId* are first obtained from the user. While these values are taken, the user input is carefully sanitized and it is checked whether the entered id is in the database. Then, if the entered *branchId* and *productId* are in the database, it is checked whether the product specified in the specified branch has a stock or not

by the *checkStock()* method. If it is in stock, the *customerId* and *orderDate* are taken from the user and the order is created first and then the quantity in stock is reduced by one. If it is not available in stock, then the user is told that there is no stock with the entered *productId* and *branchId*, and then go back to the beginning. You can see the tests performed below.

```
Please enter a number between 1-13: 2
Please enter product id: van
Your input was invalid, please enter a number for product id: -1
Your product id is not in the database. Please enter a valid id: 5
Please enter branch id: van
Your input was invalid, please enter a number for branch id: -1
Your branch id is not in the database. Please enter a valid id: 3
Sorry, the product you have specified is not in the stock of the branch you have specified.
```

```
Please enter a number between 1-13: 2
Please enter product id: 1
Please enter branch id: 1
Please enter customer id: van
Your input was invalid, please enter a number for customer id: -1
Your customer id is not in the database. Please enter a valid id: 4
Please enter order date. For ex(2020-01-27 17:51:07): van
Order date must be 19 characters. Like 1964-03-18 07:03:54
Please enter order date: 2020-30-10 11:05:12
Your month is not between 01-12
Please enter order date: 2020-07-39 18:25:17
Your day is not between 01-31
Please enter order date: 2020-09-11 99:00:42
Your hour is not between 0 and 24
Please enter order date: 2020-01-12 22:65:44
Your minute is not between 0 and 60
Please enter order date: 2020-03-09 20:45:79
Your second is not between 0 and 60
Please enter order date: 2020-11-05 17:12:05
Order has been added successfully.
Stock has been updated successfully.
```

2.4.3 *handle3();* "List Customers"

```java
public static void handle3() { listCustomerInfo(""); }
```

This function prints all the information of registered customers on the screen. You can see the output below.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CUSTOMER INFORMATION
ID:1    Name:Seçkin Surname:Ağırbaş Address:Alibeyköy   PhoneNumber:05001112233
ID:2    Name:Mahmut Surname:Acar    Address:Kartal  PhoneNumber:05002221133
ID:3    Name:Merve  Surname:Özkan   Address:Beşiktaş    PhoneNumber:05003332211
ID:4    Name:Aysel  Surname:Tekin   Address:Kadıköy PhoneNumber:05002223311
ID:7    Name:John   Surname:Doe Address:Izmir   PhoneNumber:01112223344
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

2.4.4 *handle4();* "List a Customer's Purchases"

```java
public static void handle4(){
    int customerId = Integer.parseInt(getCustomerInfo("id"));
    listCustomersOrders(customerId);
}
```

This method gets the *customerId* from the user and then checks if that id exists in the database. If it exists in the database, then the user's orders are printed on the screen, if it is not in the database, the user is told that the id entered is not in the database. You can see the test below.

```
Please enter a number between 1-13: 4
Please enter customer id: van
Your input was invalid, please enter a number for customer id: 12
Your customer id is not in the database. Please enter a valid id: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ORDER INFORMATION
OrderDate:2003-10-01 12:12:12   ID:1    ProductID:1 BranchID:1
OrderDate:2010-05-30 12:35:05   ID:1    ProductID:1 BranchID:1
OrderDate:2013-09-12 19:34:53   ID:1    ProductID:1 BranchID:1
OrderDate:2015-03-17 19:33:25   ID:1    ProductID:1 BranchID:1
OrderDate:2018-02-30 11:26:45   ID:1    ProductID:1 BranchID:1
OrderDate:2020-01-27 17:51:07   ID:1    ProductID:1 BranchID:1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

2.5.5 *handle5();* "List a Customer's Most Recent Purchases"

```
761     public static void handle5(){
762         int customerId = Integer.parseInt(getCustomerInfo("id"));
763         ArrayList<String> customerDates = getOrderDatesFromDb(customerId);
764         Collections.sort(customerDates);
765         Collections.reverse(customerDates);
766         listRecentOrders(customerDates, customerId);
767     }
```

This function first gets *customerId* from the user. It then uses this *customerId* to take the *orderDates* from the "Orders" table and save them in an arraylist. It then sorts the dates in ascending order with the *Collections.sort()* function. Then, these dates are converted into descending order with *Collection.reverse()* function. Then, the received *customerId* and these listed dates are put as input to the *listRecentOrders()* function and the last 5 orders of the specified customer are printed on the screen. You can see the tests performed below.

```
Please enter a number between 1-13: 5
Please enter customer id: van
Your input was invalid, please enter a number for customer id: 1
OrderDate:2020-01-27 17:51:07   CustomerId:1    ProductId:1 BranchId:1
OrderDate:2018-02-30 11:26:45   CustomerId:1    ProductId:1 BranchId:1
OrderDate:2015-03-17 19:33:25   CustomerId:1    ProductId:1 BranchId:1
OrderDate:2013-09-12 19:34:53   CustomerId:1    ProductId:1 BranchId:1
OrderDate:2010-05-30 12:35:05   CustomerId:1    ProductId:1 BranchId:1
```

2.4.6 *handle6();* "List Branches"

```
770     public static void handle6() { listBranchInfo(); }
```

Branch table is printed on the screen in this function. You can see the output below.

```
Please enter a number between 1-13: 6
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BRANCH INFORMATION
BranchID:1  BranchName:Alibeyköy Center BranchAddress:Alibeyköy
BranchID:2  BranchName:Kadıköy Center   BranchAddress:Kadıköy
BranchID:3  BranchName:Kadıköy Pier BranchAddress:Kadıköy
BranchID:4  BranchName:Beşiktaş Square  BranchAddress:Beşiktaş
BranchID:5  BranchName:Taksim Square    BranchAddress:Taksim
BranchID:6  BranchName:Agora Center BranchAddress:Izmir
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

2.4.7 *handle7();* “List Branch’s Stock”

```
public static void handle7(){
    int branchId = Integer.parseInt(getBranchInfo("id"));
    listBranchsOrders(branchId);
}
```

In this method, firstly, the *branchId* is obtained from the user. Then it is checked whether this id is in the database or not. If not, this user is notified. Then, in the *listBranchOrders()* function, the orders of the products taken from this branch are printed on the screen by using this *branchId*.

```
Please enter a number between 1-13: 7
Please enter branch id: van
Your input was invalid, please enter a number for branch id: 17
Your branch id is not in the database. Please enter a valid id: 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ORDER INFORMATION
OrderDate:2003-10-01 12:12:12   ID:1    ProductID:1 BranchID:1
OrderDate:2010-05-30 12:35:05   ID:1    ProductID:1 BranchID:1
OrderDate:2013-09-12 19:34:53   ID:1    ProductID:1 BranchID:1
OrderDate:2015-03-17 19:33:25   ID:1    ProductID:1 BranchID:1
OrderDate:2018-02-30 11:26:45   ID:1    ProductID:1 BranchID:1
OrderDate:2020-01-27 17:51:07   ID:1    ProductID:1 BranchID:1
OrderDate:2020-11-05 17:12:05   ID:4    ProductID:1 BranchID:1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

2.4.8 *handle8();* “Add New Branch Store”

```
public static void handle8(){
    String branchName = getBranchInfo("name");
    String branchAddress = getBranchInfo("address");
    createBranch(branchName, branchAddress);
}
```

In this function, *branchName* and *branchAddress* are received from the user. Then, using this information, a new branch is created with the *createBranch()* function. You can see the tests performed below.

```
Please enter a number between 1-13: 8
Please enter branch name: qwkjakldsjlkwjkldajlksdjwlkjadlkajsdlkjalkdjaklsjdklajdlkajsdlajldkjwklasdmwlkjdalksjdlwjdlkajsdlakjw
Name can be up to 50 letters.
Please enter branch name: Migros Center
Please enter branch address (only city for simplicity): qşlaşlksdşlqwklşakdlşkwqşlkdalşskşlkwdşlakdşlaksşldkawşdklşalksd
Address can be up to 30 letters.
Please enter branch address: Izmir
Branch has been created successfully.
```

2.4.9 *handle9();* “Add New Product”

```
788     public static void handle9(){
789         String productName = getProductInfo("name");
790         String productDescription = getProductInfo("description");
791         float productPrice = Float.parseFloat(getProductInfo("price"));
792         createProduct(productName, productDescription, productPrice);
793     }
```

In this method, a new product is created by taking the product name, description and price from the user and using the *createProduct()* function. You can see the tests performed below.

```
Please enter a number between 1-13: 9
Please enter product name: FoodHorse Kitkat
Please enter product description: FoodHorse Chocolate Bar 250g
Please enter product price: van
Your input was invalid, please enter a float number: -1
Price must be greater than 0. Please enter price: 4.32
Product has been created successfully.
```

2.4.10 *handle10();* “Add Product to Branch’s Stock”

```
796     public static void handle10(){
797         int productId = Integer.parseInt(getStockInfo("product"));
798         int branchId = Integer.parseInt(getStockInfo("branch"));
799         int quantity = Integer.parseInt(getStockInfo("quantity"));
800         addStock(quantity, productId, branchId);
801     }
```

In this function, a new stock is created with the *addStock()* function by taking *productId*, *branchId* and *quantity* from the user. You can see the tests performed below.

```
Please enter a number between 1-13: 10
Please enter product id: van
Your input was invalid, please enter a number for product id: -1
Your product id is not in the database. Please enter a valid id: 2
Please enter branch id: van
Your input was invalid, please enter a number for branch id: -2
Your branch id is not in the database. Please enter a valid id: 4
Please enter quantity: van
Your input was invalid, please enter a number for quantity: -10
Your quantity must be greater than 0. Please enter quantity:  28
Stock has been added successfully
```

2.4.11 *handle11();* “Search Customer By Their Phone Number”

```
803     public static void handle11(){
804         String customerPhoneNumber = getCustomerInfo("phone number");
805         listCustomerInfo(customerPhoneNumber);
806     }
```

In this function, *customerPhoneNumber* is taken from the user and the information of the user is printed on the screen with *listCustomerInfo()* function. You can see the tests performed below.

```
Please enter a number between 1-13: 11
Please enter customer phone number: van
Phone number must be 11 digits. Like (05551112233)
Please enter customer phone number: onbirharfli
Your input is invalid.
Please enter customer phone number: 05001112233
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CUSTOMER INFORMATION
ID:1    Name:Seçkin Surname:Ağırbaş Address:Alibeyköy   PhoneNumber:05001112233
------------------------------------------
```

2.4.12 *handle12();* “Remove a User From System”

```
809     public static void handle12(){
810         int customerId = Integer.parseInt(getCustomerInfo("id"));
811         deleteCustomersInfo(customerId);
812     }
```

In this function, *customerId* is taken from the user and it is checked whether this id is in the database. If it is not in the database, this user is reported. If it exists in the database, with the *deleteCustomersInfo()* function, the customer's orders are deleted first and then from the database. You can see the tests performed below.

```
Please enter a number between 1-13: 11
Please enter customer phone number: van
Phone number must be 11 digits. Like (05551112233)
Please enter customer phone number: onbirharfli
Your input is invalid.
Please enter customer phone number: 05001112233
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CUSTOMER INFORMATION
ID:1    Name:Seçkin Surname:Ağırbaş Address:Alibeyköy   PhoneNumber:05001112233
------------------------------------------
```

2.4.13 *handle13();* “Exit”

```
813     public static void handle13(){
814         System.out.println("THANK YOU HOPE TO SEE YOU AGAIN!");
815         System.out.println(
816                 "88                                    \n" +
817                 "88                                    \n" +
818                 "88                                    \n" +
819                 "88,dPPYba,  8b       d8  ,adPPYba,   \n" +
820                 "88P'    \"8a `8b     d8' a8P_____88   \n" +
821                 "88       d8  `8b   d8'  8PP\"\"\"\"\"\"\"  \n" +
822                 "88b,   ,a8\"   `8b,d8'   \"8b,   ,aa   \n" +
823                 "8Y\"Ybbd8\"'      Y88'     `\"Ybbd8\"'   \n" +
824                 "                d8'                   \n" +
825                 "               d8'                    ");
826     }
```

In this method, the user is bid farewell. You can see the output below.

```
Please enter a number between 1-13: 13
THANK YOU HOPE TO SEE YOU AGAIN!
88
88
88
88,dPPYba,  8b       d8  ,adPPYba,
88P'    "8a `8b     d8' a8P_____88
88       d8  `8b   d8'  8PP"""""""
88b,   ,a8"   `8b,d8'   "8b,   ,aa
8Y"Ybbd8"'      Y88'     `"Ybbd8"'
                d8'
               d8'
```

Tuna Tuncer

S018474