BSc (Hons) Artificial Intelligence and Data Science

Module: CM1601 Programming Fundamentals

Coursework 2 Report

Module Leader: Mr. Iresh Bandara

RGU Student ID : 2313077

IIT Student ID : 20220577

Student Name : R.V.Wellalage

Executive Summary

In this project we are instructed to create a graphical user interface for an inventory system called john's internet café. For that I used javafx and scene builder to reach all the tasks perfectly. In this inventory system it has several methods to communicate with the user. Adding items, deleting items, updating items, saving items and viewing items in ascending order by their item code. In adding item part it will check all the validations for adding a correctly. Ans also the update.

There are 6 dealers for this internet café. John is going to select four dealers randomly among those six random dealers. And print those four random dealers into a table sorted by their location. And user should be able to give id number of a random dealer and get all the item details of that dealer.

As a brief description this is the basic idea of our project. Go ahead with the report to get more knowledge about this project.

Contents

Table of Figures

Flow chart

- ADD item

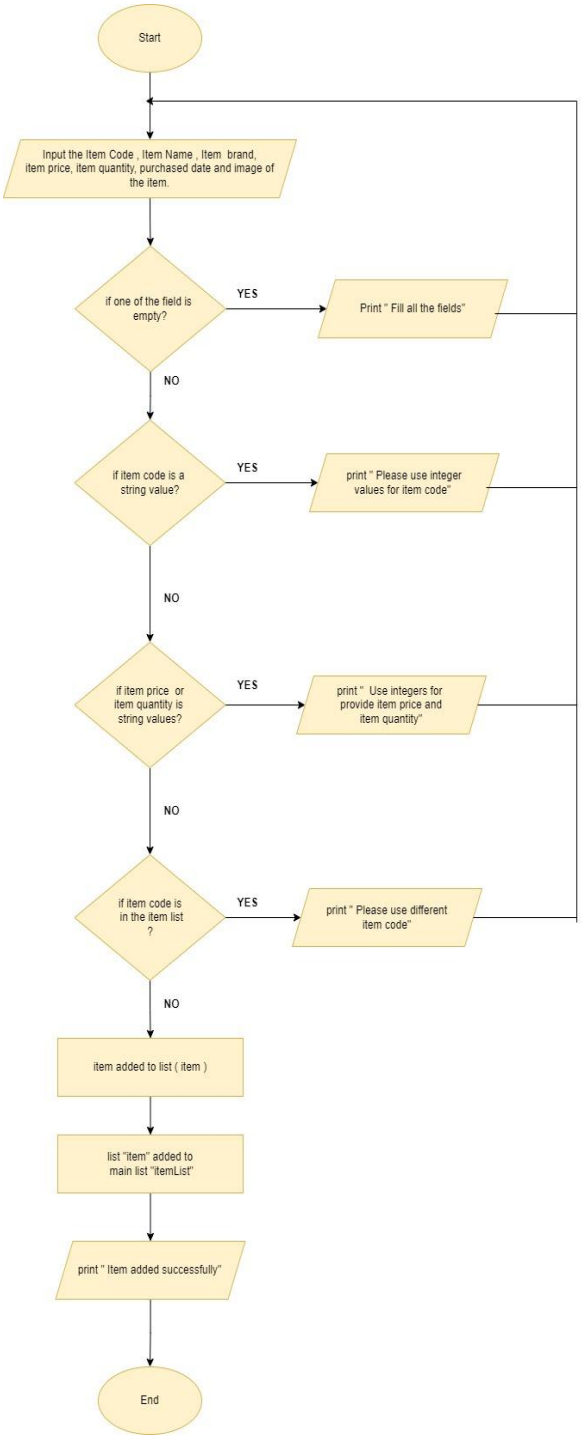


Figure 1

➢ DELETE item

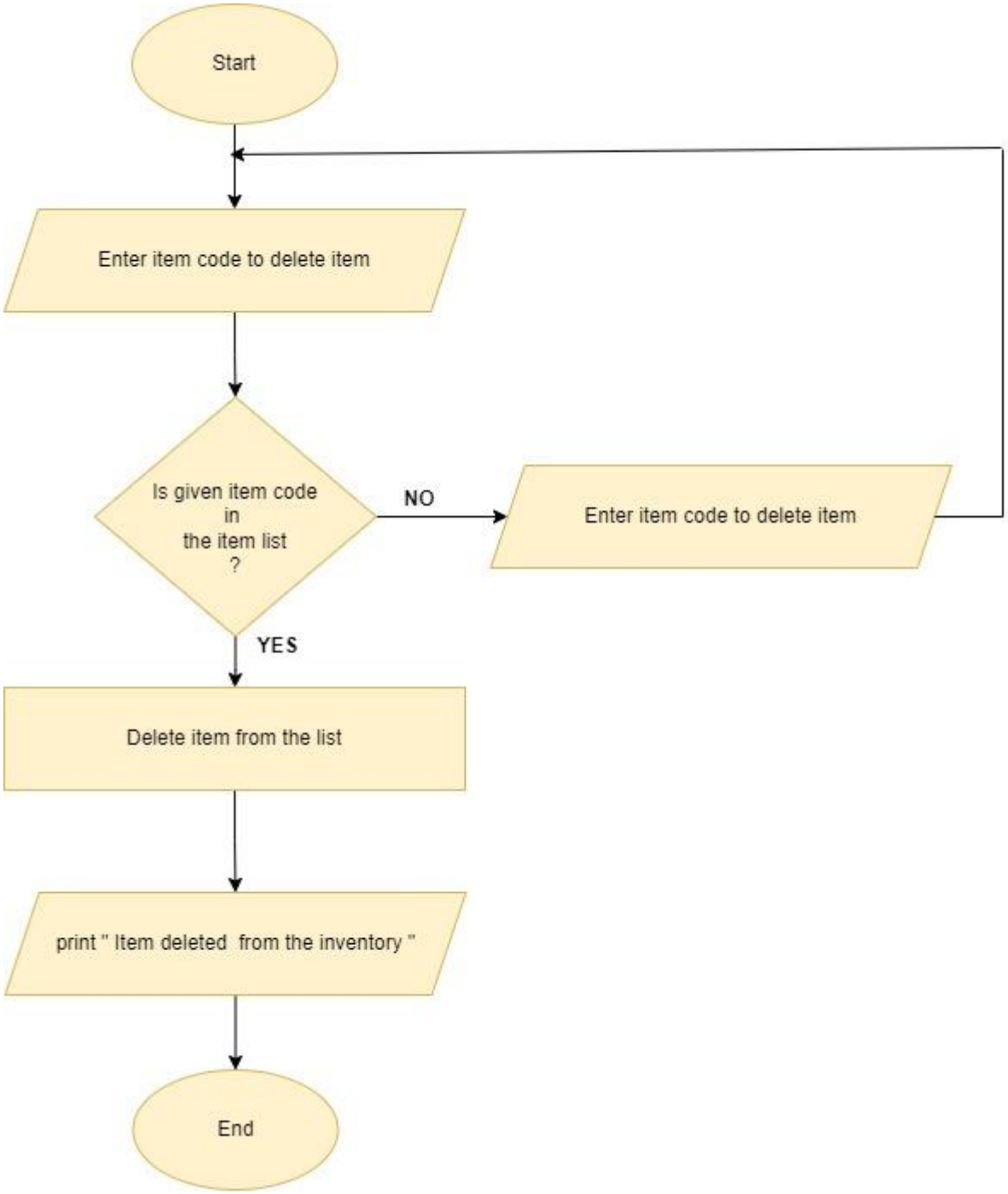


Figure 2

➢ UPDATE item

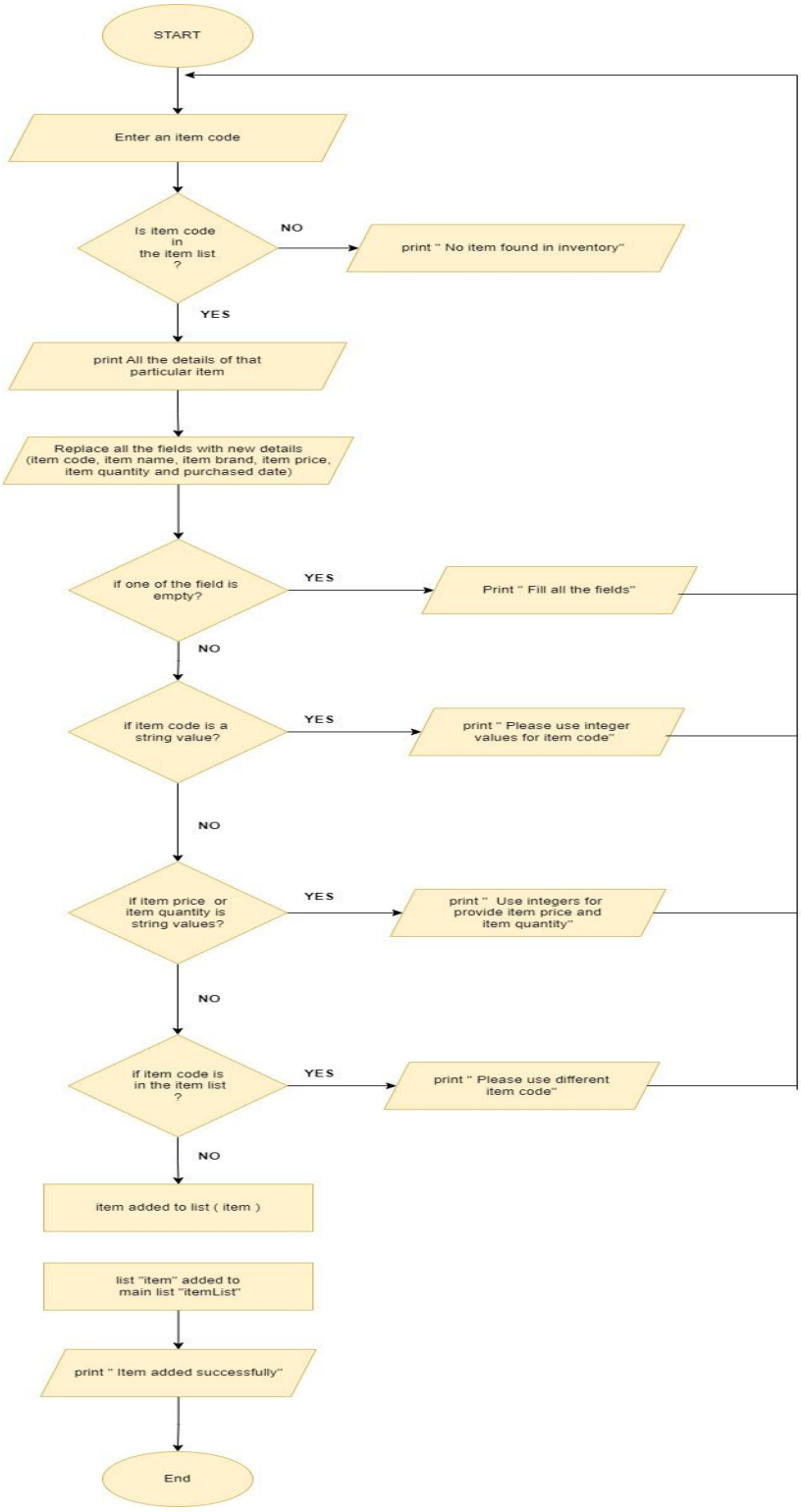


Figure 3

➢ VIEW items

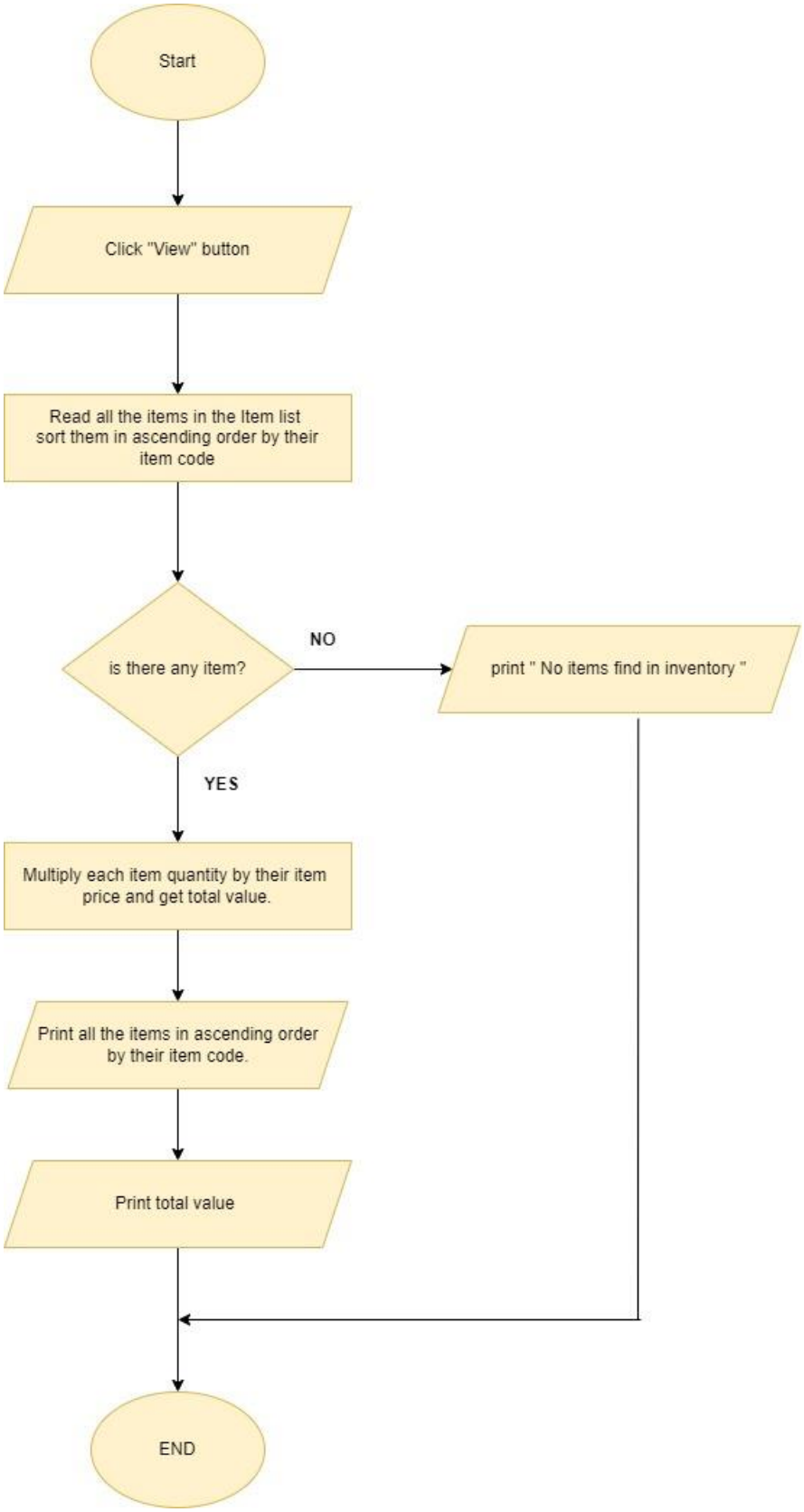


Figure 4

➢ SAVE item details

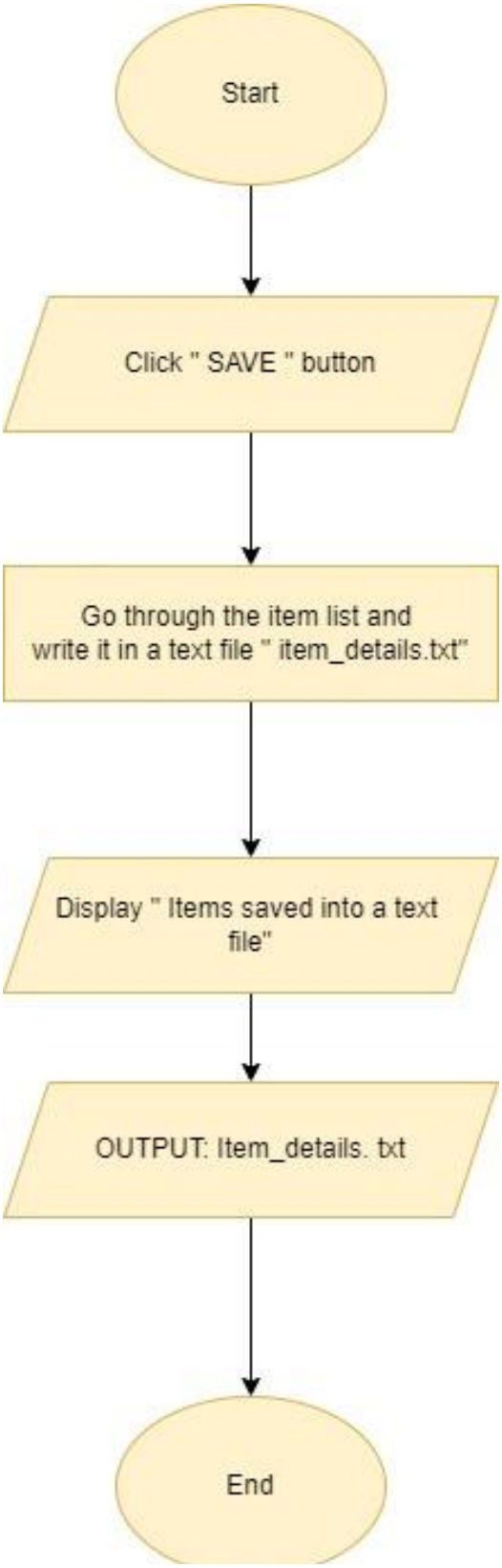


Figure 5

➢ SELECT RANDOM DEALERS

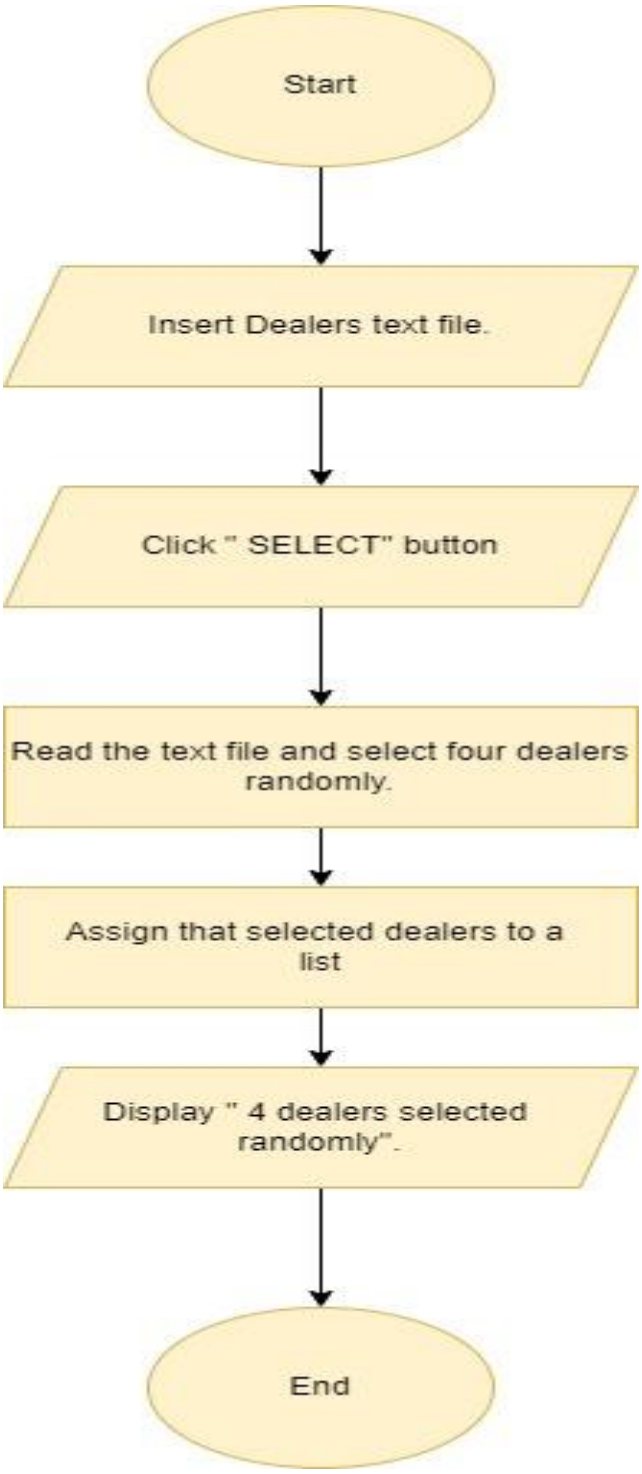


Figure 6

➢ DISPLAY RANDOM DEALERS

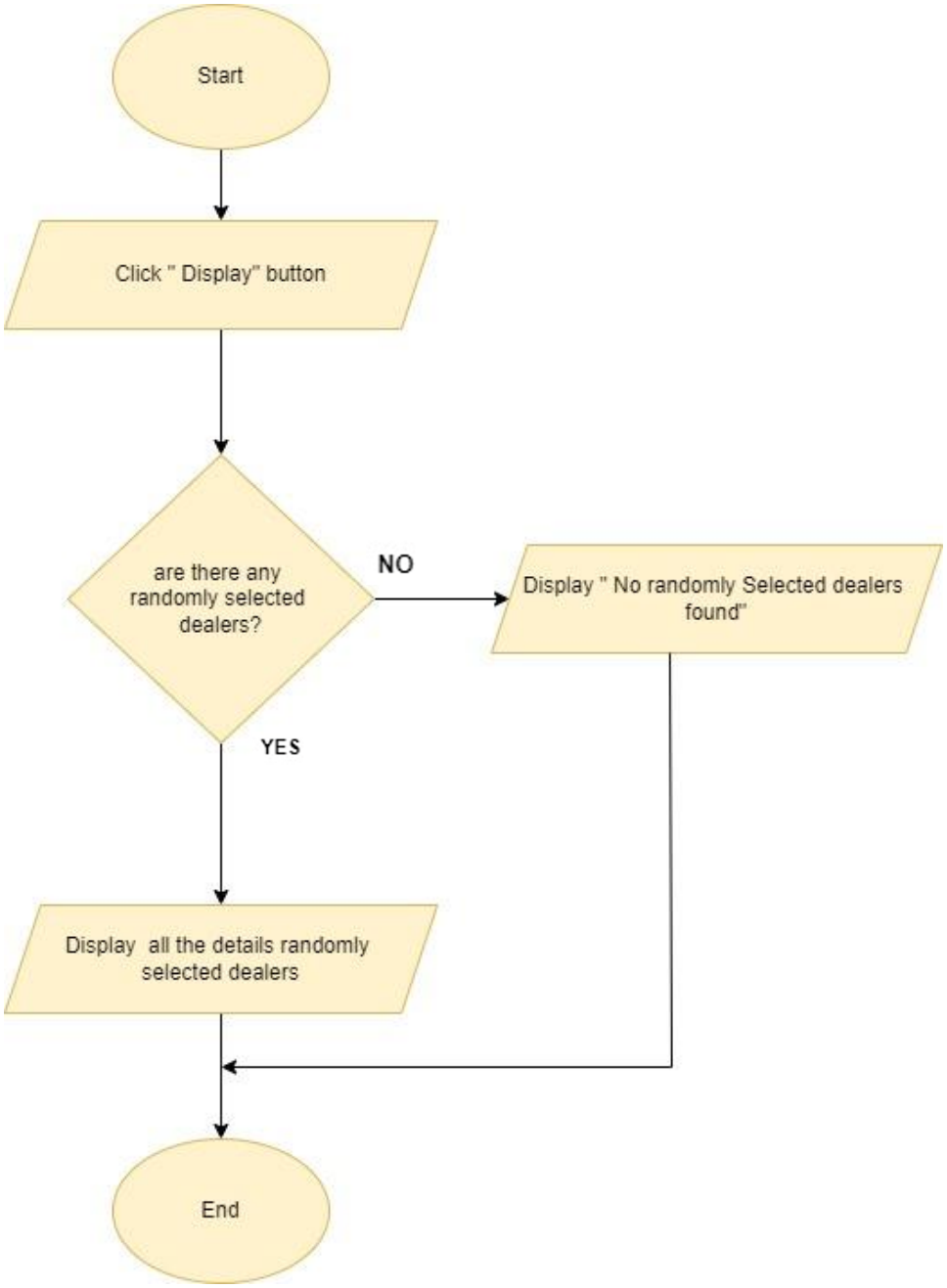


Figure 7

➢ DISPLAY DEALER DETAILS

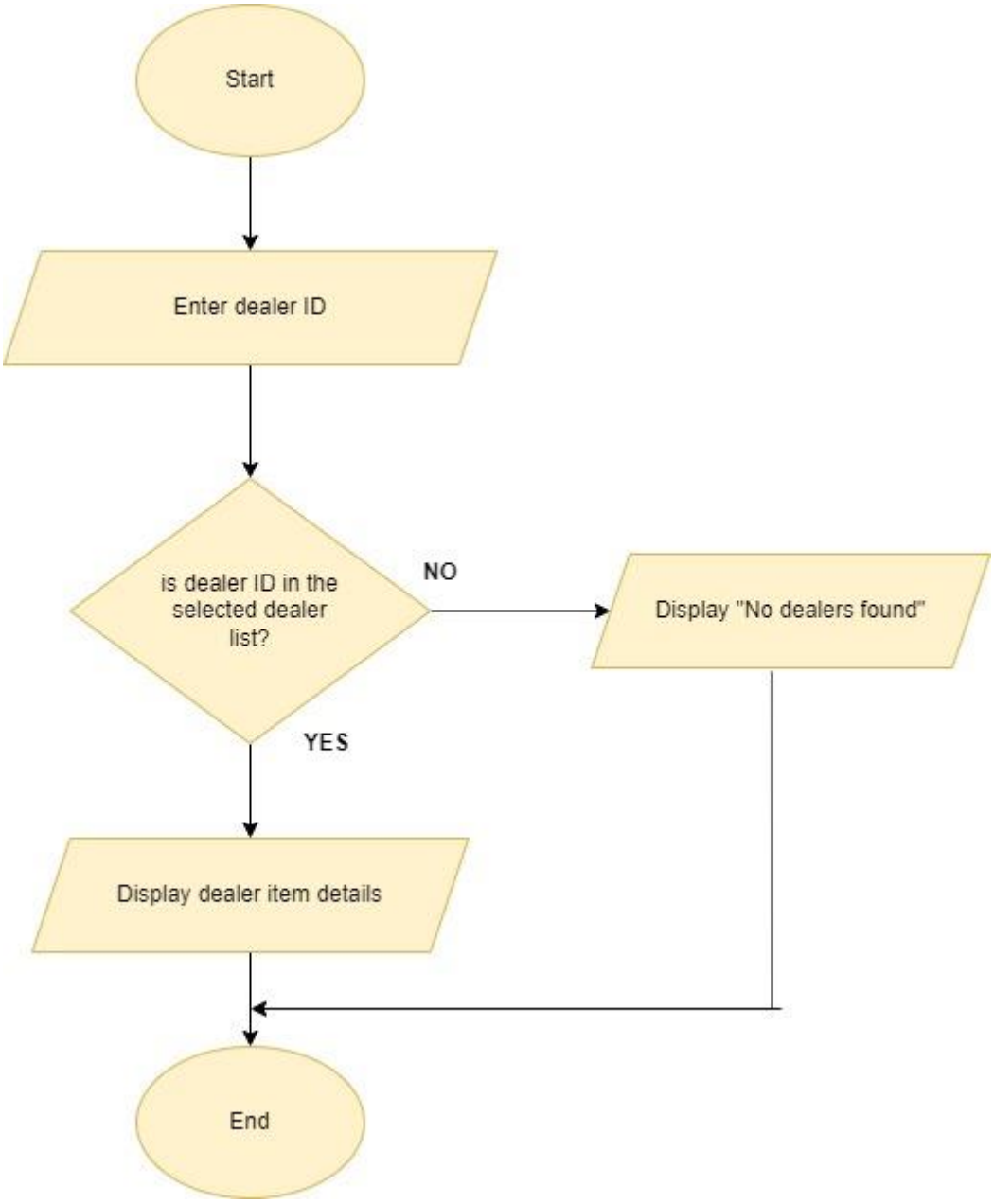


Figure 8

Introduction to functions with code

➢ STARTUP

User can get the startup from this menu tab.

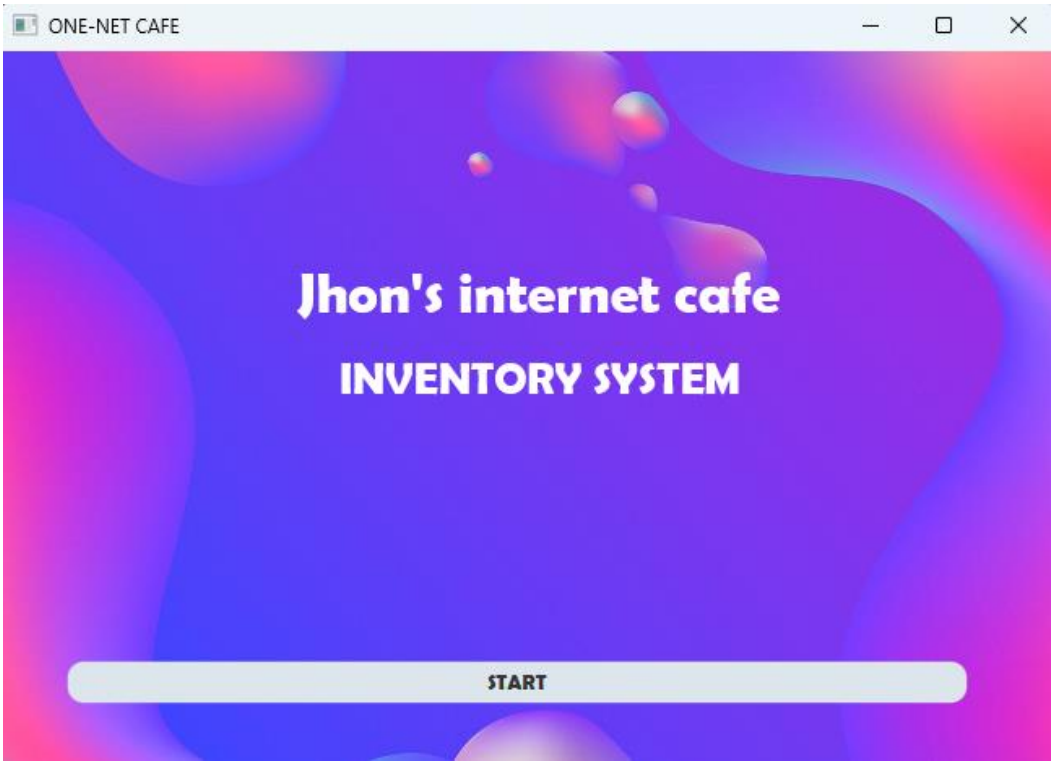


Figure 9

Source code:

```
public class HelloApplication extends Application {



    @Override
    public void start(Stage stage) throws IOException {


        FXMLLoader fxmlLoader = new    FXMLLoader(MENU.class.getResource("startup.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 650, 445);
        stage.setTitle("ONE-NET CAFE");
        stage.setScene(scene);
        stage.show();
    }



    public static void main(String[] args) {

        launch();
    }
}
```

This page will give the startup interface to the user. User can access to the system throughout this page. When user click the start button, it will load the menu file to the user to through the further requirements.

➢ MENU

From the menu file user can access to the every function of this café system.

Figure 10

Source code:

```
public void onStartButtonClick() throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("menu.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("MENU");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) startup_anc.getScene().getWindow();
    currentStage.close();
}
```

In menu file there is the every and each button to access the other functions.

➢ Adding Items

User can access to the add items page using the button which is in the menu controller. In this add items controller user can give an item code as an integer, item name, item brand, item price , item quantity and item purchased date. And there are several methods to check validations of the given fields.

```
public void onAddButtonClick() throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("ADD_item.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("ADD ITEMS");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) menuanc.getScene().getWindow();
    currentStage.close();
}
```

- **imageButton method**

```
public void imageButton(ActionEvent event) {

    FileChooser pathOfFile = new FileChooser();
    pathOfFile.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Images","*.png","*.jpg","*.jpeg","*.webp"));
    imagePath=pathOfFile.showOpenDialog(null);


    Image image = new Image(imagePath.getAbsolutePath());
    picture.setImage(image);


}
```

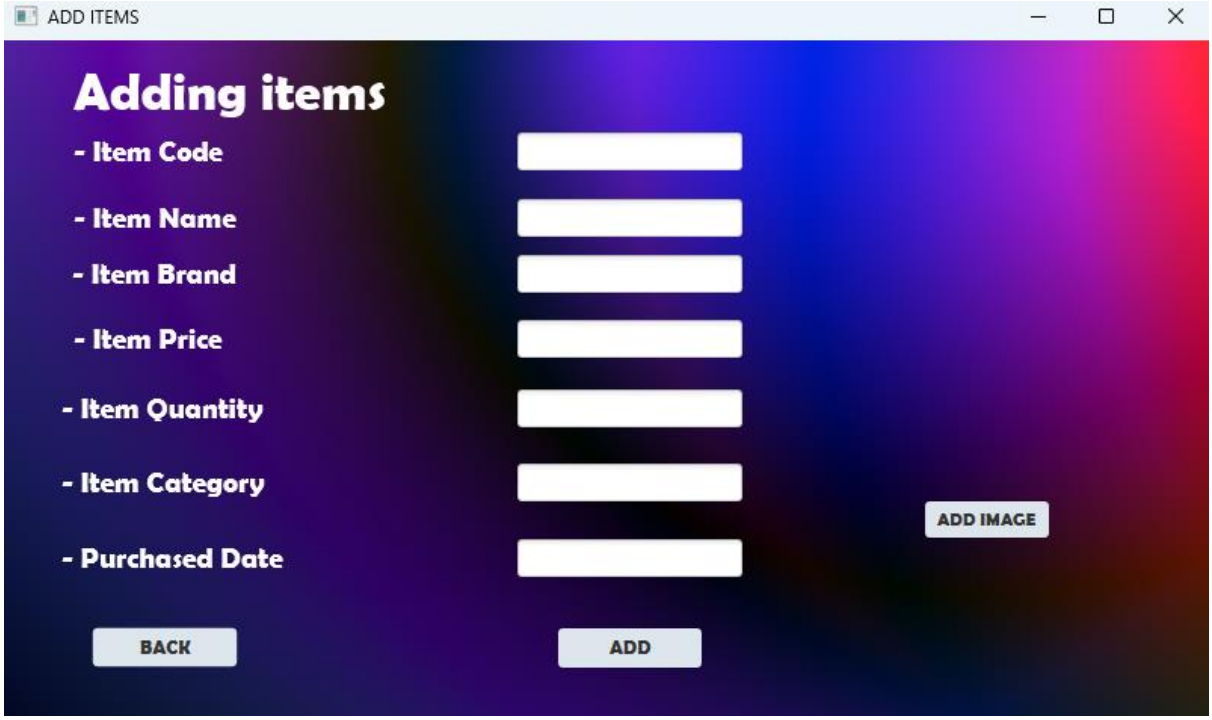


Figure 11

- **addButtonClick** method

```
public void addButtonClick() throws IOException {

    if (!isAllFieldsFilled()) {
        error("Please fill all fields.");
        return;
    }

    List<Object> item = new ArrayList<>();

    try {
        int a1 = Integer.parseInt(itemCode.getText().trim());
        String a2 = itemName.getText();
        Object a3 = itemBrand.getText();
        double a4 = Double.parseDouble(itemPrice.getText());
        int a5 = Integer.parseInt(itemQuantity.getText().trim());
        Object a6 = itemCategory.getText();
        Object a7 = date.getText();

        if (isValidInput(a1, a4, a5)) {
            item.add(a1);
            item.add(a2);
            item.add(a3);
            item.add(a4);
            item.add(a5);
            item.add(a6);
            item.add(a7);


            for (List<Object> itemm : itemList) {
                int existingItemCode = (int) itemm.get(0);
                if (existingItemCode == a1) {
                    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("error_add.fxml"));
                    Stage stage = new Stage();
                    Scene scene = new Scene(fxmlLoader.load(), 500, 150);
                    stage.setScene(scene);
                    stage.show();

                    return;

                }
            }

            itemList.add(item);

            FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("item_add_popup_msg.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmlLoader.load(), 500, 150);
            stage.setScene(scene);
            stage.show();

            clearTextFields();
        } else {
            error("Please enter correct numeric values for Item code,Price and Quantity.");
        }
    } catch (NumberFormatException e) {
        error("Please enter correct numeric values for Item code,Price and Quantity.");
        e.printStackTrace();
    }
}
```

This is the prior method of add items controller. In this method program will check whether the all input fields are filled or not, correct data types and if there are any repeating item codes validation of item price and item quantity.

- **isAllFieldsFilled** method

```
private boolean isAllFieldsFilled() {
    return !itemCode.getText().trim().isEmpty()
            && !itemName.getText().isEmpty()
            && !itemBrand.getText().isEmpty()
            && !itemPrice.getText().isEmpty()
            && !itemQuantity.getText().isEmpty()
            && !itemCategory.getText().isEmpty()
            && !date.getText().isEmpty();
}
```

In this method program will check whether all the fields are filled or not.

Figure 12

- **isValidInput** method

```
private boolean isValidInput(int code,double price, int quantity) {

    return price > 0 && quantity > 0 && code > 0;
}
```

Here is the method to check item price and item quantity. If user enters any String value to these text fields , the program will an output pop-up message describing the error.

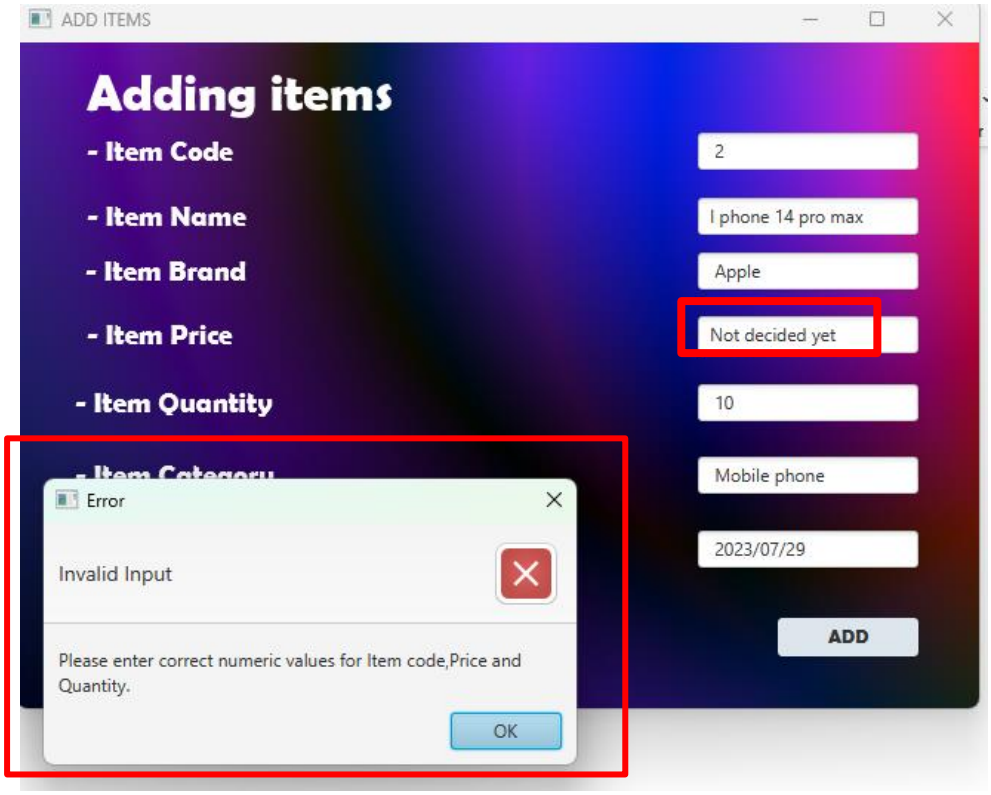


Figure 13

- **clearTextFields** method

This will clear all the fields after adding the item perfectly.

```
private void clearTextFields() {
    itemCode.clear();
    itemBrand.clear();
    itemName.clear();
    itemQuantity.clear();
    itemCategory.clear();
    itemPrice.clear();
    date.clear();

}
```

- **error** method

```
public void error(String erroMessage) {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Error");
    alert.setHeaderText("Invalid Input");
    alert.setContentText(erroMessage);

    alert.showAndWait();
    alert.close();
}
```

This method will give the all the error alerts in add items controller.

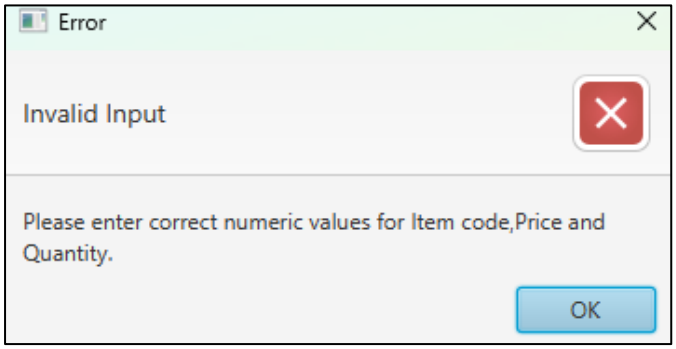


Figure 14

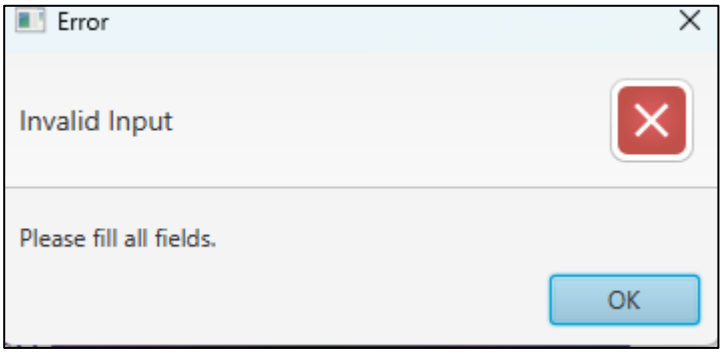


Figure 15

- **onBackButtonClick** method

```
public  void onBackButtonClick() throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(MENU.class.getResource("menu.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("MENU");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) add_anc.getScene().getWindow();
    currentStage.close();
}
```

This will load the menu page to the user. And close the add items page.

➢ Delete items

User can call this method from the menu controller. In here user can delete items by searching item code. When this method called it will check through the main array list of items and find the given item code. If there is that particular item code, it will delete it from the main list and give a message to the user that the deletion was successful.

```
public void onDelButtonClick() throws IOException {
    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("DELETE_item.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("DELETE ITEMS");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) menuanc.getScene().getWindow();
    currentStage.close();
}
```

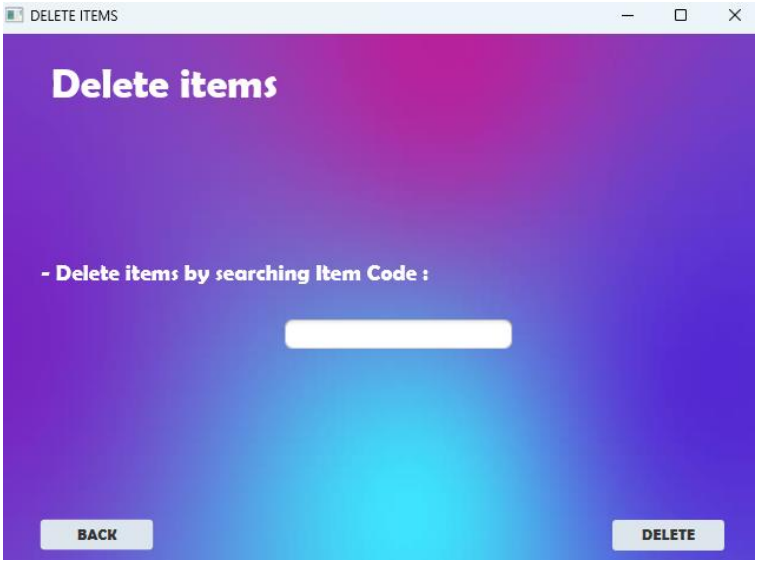


Figure 16

- **onDeleteButtonClick** method

```
public void onDeleteButtonClick() throws IOException{

    int code = Integer.parseInt(deleteField.getText().trim());
    if(add_items.itemList != null){
        deleteItem(code);
    }
    else {

        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("error_del.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 500, 150);
        stage.setScene(scene);
        stage.show();

    }
}
```

In here user has to give an item id to delete it from the inventory. And it will check whether it is in the item list.

- **deleteItem** method

```
public void deleteItem(int code) throws IOException {
    for (List<Object> item : add_items.itemList){
        int itemCode = (int) item.get(0);
        if (itemCode == code){
            add_items.itemList.remove(item);

            FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("item_del_popup_msg.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmlLoader.load(), 500, 150);
            stage.setScene(scene);
            stage.show();

            deleteField.clear();
        }

        else {
            FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("item_del2_popup_msg.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmlLoader.load(), 500, 150);
            stage.setScene(scene);
            stage.show();

            deleteField.clear();

        }
    }
}
```

Checking the item list and search for given item code. If it is there print “item deleted from inventory”. And if it is not print “item not found”.

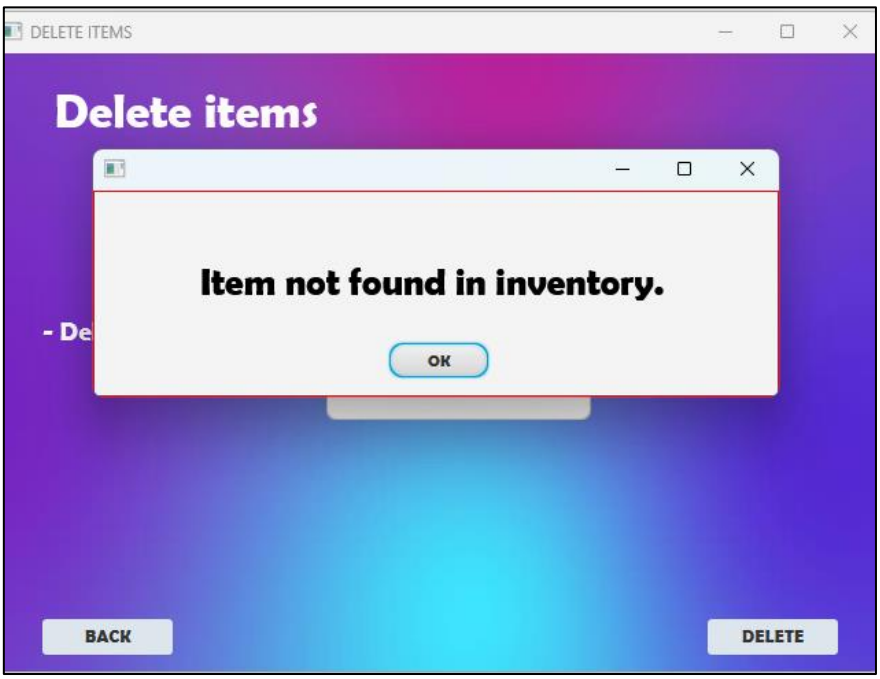


Figure 17

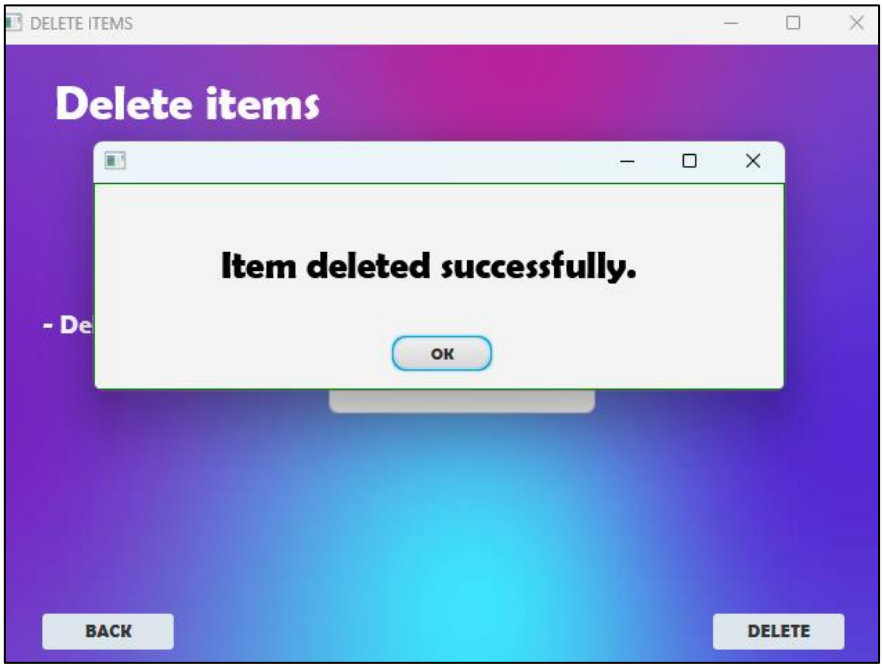


Figure 18

➢ Updating items

In this part user will be able to update item details. For this user has to observe the item list by giving the item code . When insert any item code , program should check whether that item is exist or not.

```
public void onUpButtonClick() throws IOException {
    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("UPDATE_item.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("UPDATE ITEMS");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) menuanc.getScene().getWindow();
    currentStage.close();
}
```

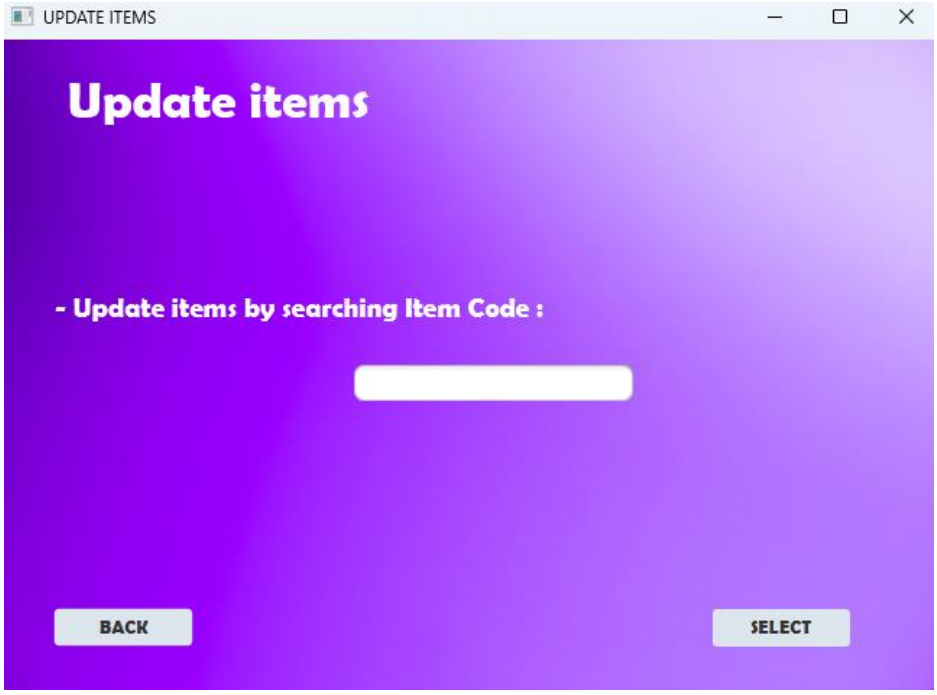


Figure 19

- **onSelButtonClick** method

```
public void onSelButtonClick() throws IOException {

    int code = Integer.parseInt(upLable.getText().trim());
    List<Object> selectedItem = updateItem(code);

    if( selectedItem != null){


        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("UPDATE2.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
        stage.setTitle("UPDATE ITEMS");
        stage.setScene(scene);
        stage.show();
```

```

        UPDATE_item2 controller = fxmlLoader.getController();
        controller.setSelectedItem(selectedItem);

        Stage currentStage = (Stage) up_anc.getScene().getWindow();
        currentStage.close();
    }
    else {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("item_del2_popup_msg.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 500, 150);
        stage.setScene(scene);
        stage.show();
    }
}
```

This checks the given item code is in the item list. If that so, it loads the another fxml file with controller to update details of the item. But if it does not fit with that item code it will pop up a fxml file to display that “item is not in inventory”.

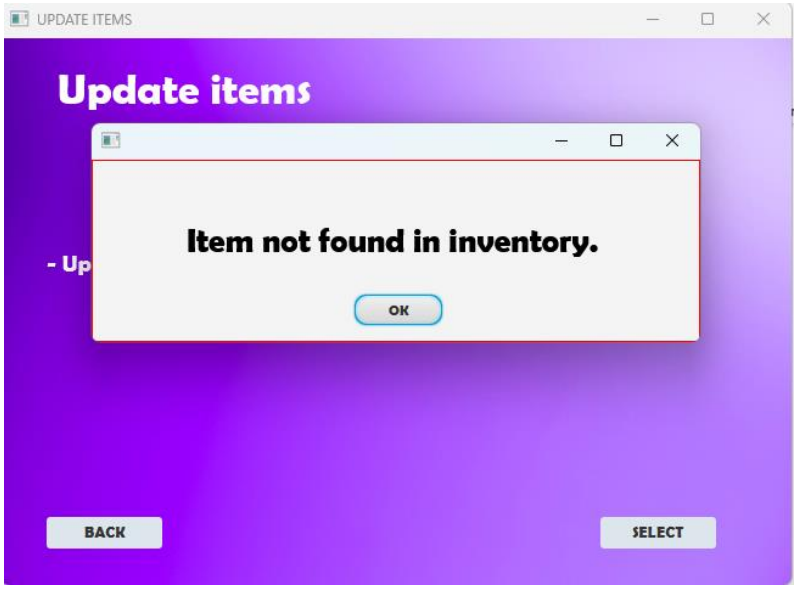


Figure 20

- **updateItem** method

this method will check whether the given item code is exists or not.

```
public List<Object> updateItem(int findCode){
    for (List<Object> item : add_items.itemList) {

        int itemCode = (int) item.get(0);
        if (itemCode == findCode) {
            return item;
        }
    }
    return null;

}
```

UPDATE_items2.java controller

From this controller it will identify the specific item code and go through the all indexes in the item list and get the saved details of that particular item.

- **setSelectedItem** method

```
public void setSelectedItem(List<Object> item){
    selectedItem = item;

    update_itemCode.setText(String.valueOf(selectedItem.get(0)));
    update_itemName.setText((String) selectedItem.get(1));
    update_itemBrand.setText((String) selectedItem.get(2));
    update_itemPrice.setText(String.valueOf(selectedItem.get(3)));
    update_itemQuantity.setText(String.valueOf(selectedItem.get(4)));
    update_itemCategory.setText((String) selectedItem.get(5));
    update_Date.setText((String) selectedItem.get(6));
}
```

- **setSelectedItem** method

When user call this method, user can get all the saved details about that particular details by going through this method.

```
public void setSelectedItem(List<Object> item){
    selectedItem = item;

    update_itemCode.setText(String.valueOf(selectedItem.get(0)));
    update_itemName.setText((String) selectedItem.get(1));
    update_itemBrand.setText((String) selectedItem.get(2));
    update_itemPrice.setText(String.valueOf(selectedItem.get(3)));
    update_itemQuantity.setText(String.valueOf(selectedItem.get(4)));
    update_itemCategory.setText((String) selectedItem.get(5));
    update_Date.setText((String) selectedItem.get(6));
}
```

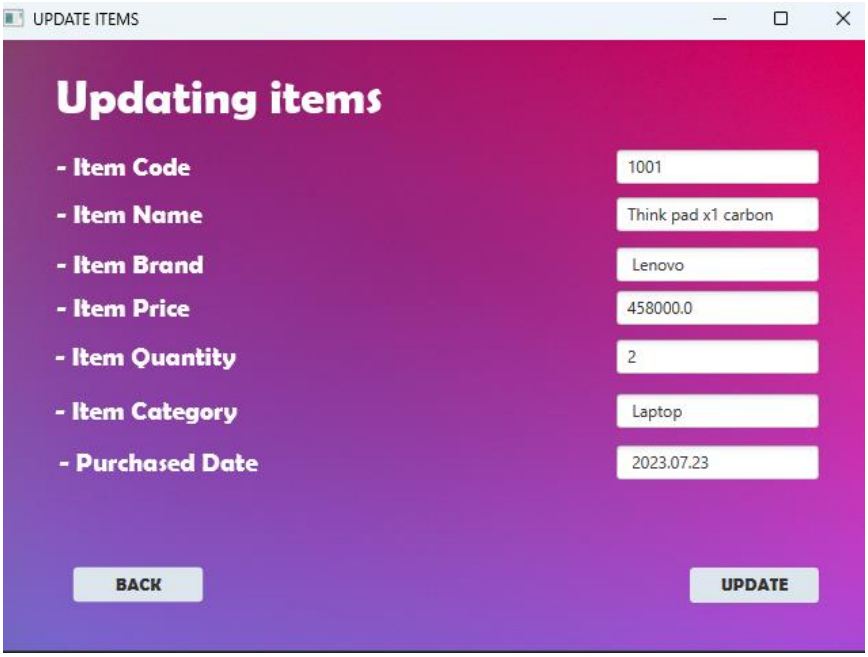


Figure 21

- **isAllFieldsFilled** method

in this method it will check whether any fields are filled or not.

```
private boolean isAllFieldsFilled() {
    return !update_itemCode.getText().trim().isEmpty()
            && !update_itemName.getText().isEmpty()
            && !update_itemBrand.getText().isEmpty()
            && !update_itemPrice.getText().isEmpty()
            && !update_itemQuantity.getText().isEmpty()
            && !update_itemCategory.getText().isEmpty()
            && !update_Date.getText().isEmpty();
}
```

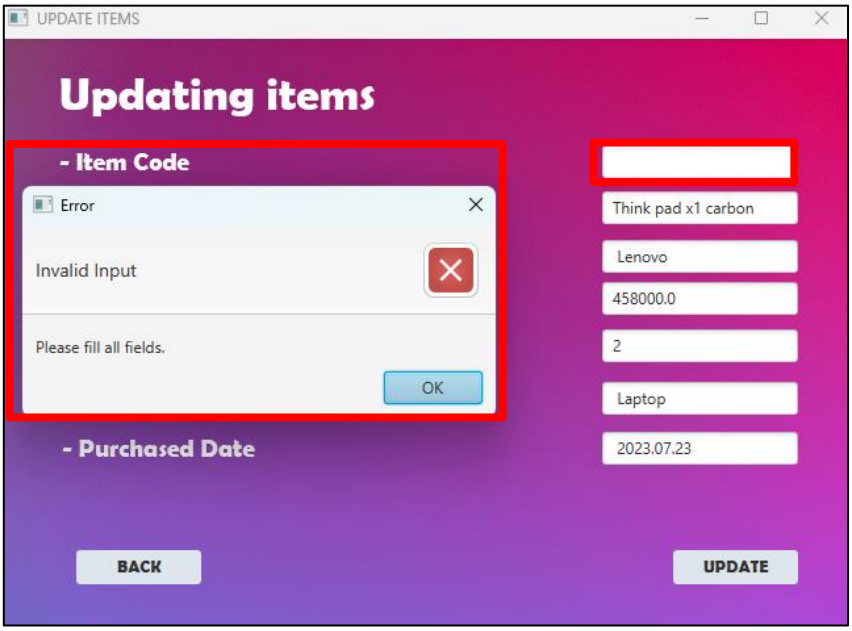


Figure 22

- **onUpButtonClick** method

```
public  void onUpButtonClick() throws IOException {

    if (!isAllFieldsFilled()) {
        error("Please fill all fields.");
        return;
    }

    int code = 0;
    String name = null;
    String brand = null;
    double price = 0;
    int quantity = 0;
    String category = null;
    String date = null;
    try {
        code = Integer.parseInt(update_itemCode.getText().trim());
        name = update_itemName.getText();
        brand = update_itemBrand.getText();
        price = Double.parseDouble(update_itemPrice.getText());
        quantity = Integer.parseInt(update_itemQuantity.getText().trim());
        category = update_itemCategory.getText();
        date = update_Date.getText();

        for (List<Object> itemm : itemList) {
            int existingItemCode = (int) itemm.get(0);
            if (existingItemCode == code) {
                FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("error_add.fxml"));
                Stage stage = new Stage();
                Scene scene = new Scene(fxmlLoader.load(), 500, 150);
                stage.setScene(scene);
                stage.show();

                return;

            }
        }

        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("item_update_popup_msg.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 500, 150);
        stage.setScene(scene);
        stage.show();


    } catch (NumberFormatException e) {
        error("Please enter correct numeric values for Item code,Price and Quantity.");
        e.printStackTrace();
    }

    // assing values to the above created variables...
    selectedItem.set(0, code);
    selectedItem.set(1, name);
    selectedItem.set(2, brand);
    selectedItem.set(3, price);
    selectedItem.set(4, quantity);
    selectedItem.set(5, category);
    selectedItem.set(6, date);

}
```

This is the final method to update all the details of an item. At First it calls the method IsAllFields fill method and it will gives user an error if any empty fields are there. Afte that it will assign all the values to their particular index number of the list. After successfully updated that item user can receive a message that telling “your updating is done”.

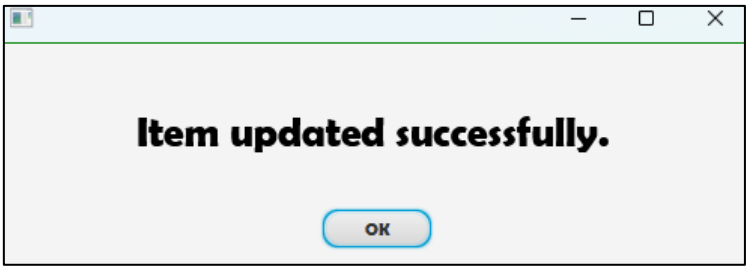


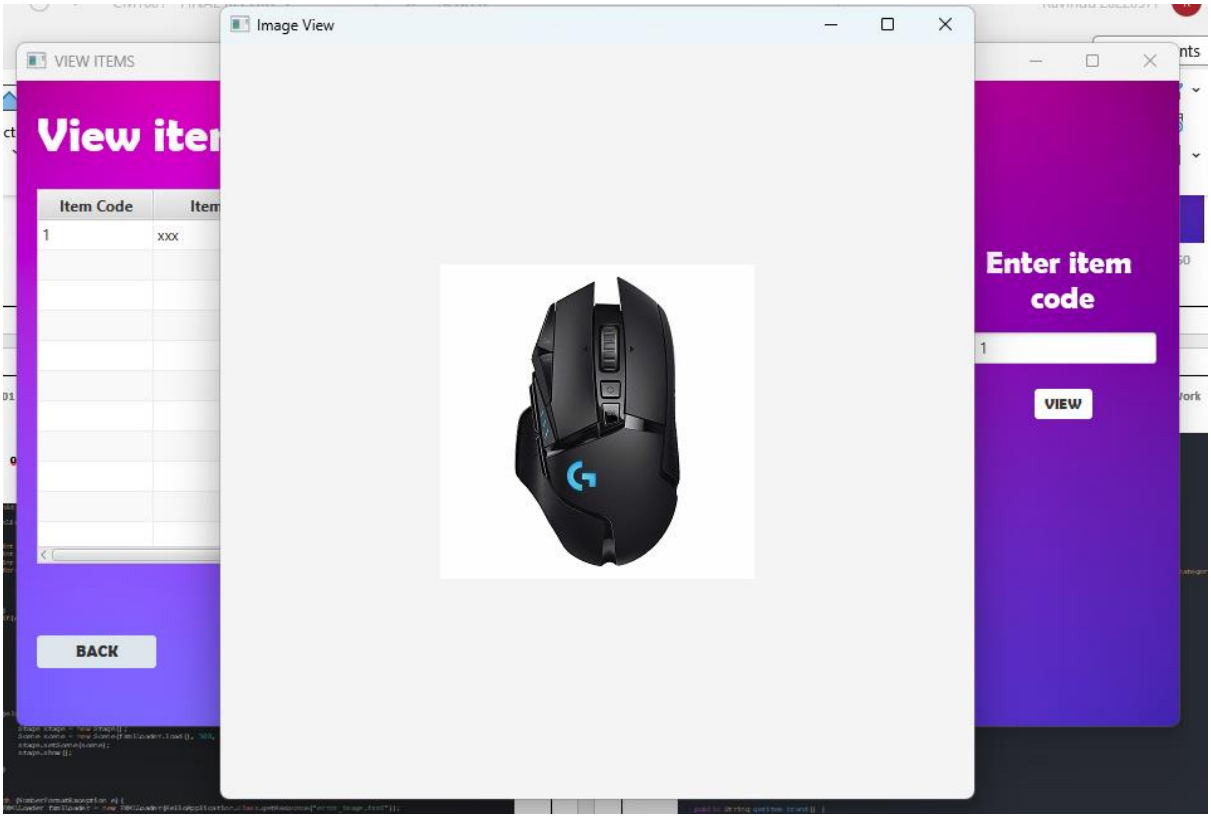
Figure 23

➢ View items

When user calls this method , it will go through the main item list for checking items. The code works with a custom data structure called items, which likely represents an item's properties, such as item code, name, brand, price, quantity, category, and date. An ObservableList<items> called listView is created to store items objects for display in the TableView. And at last it calculates total price of item by calculate their item quantity by their item price.

```
public void initialize(URL url, ResourceBundle resourceBundle) {
    List<List<Object>> itemList = add_items.itemList;

    ObservableList<items> listView = FXCollections.observableArrayList();

    c_itemCode.setCellValueFactory(new PropertyValueFactory<items,Integer>("item_code"));
    c_itemName.setCellValueFactory(new PropertyValueFactory<items,String>("item_name"));
    c_itemBrand.setCellValueFactory(new PropertyValueFactory<items,String>("item_brand"));
    c_itemPrice.setCellValueFactory(new PropertyValueFactory<items,Double>("item_Price"));
    c_itemQuantity.setCellValueFactory(new PropertyValueFactory<items,Integer>("item_quantity"));
    c_itemCategory.setCellValueFactory(new PropertyValueFactory<items,String>("item_category"));
    c_date.setCellValueFactory(new PropertyValueFactory<items,String>("date"));

    double total = 0;

    for(List<Object> i : itemList){
        total = total + ((Double)i.get(3)*(Integer)i.get(4));
        String x = String.format("%.2f",total);
        totalPrice.setText(String.valueOf(x));

    }


    for (List<Object> sample:itemList){
        items newItem = new items(
                (Integer) sample.get(0),
                (String) sample.get(1),
                (String) sample.get(2),
                (Double) sample.get(3),
                (Integer) sample.get(4),
                (String) sample.get(5),
                (String) sample.get(6)
        );
        listView.add(newItem);
    }


    listView.sort(Comparator.comparingInt(items::getItem_code));
    itemTableView.setItems(listView);



}
```

Figure 24



- **onView method**

```
public void onView() throws IOException {

    List<List<Object>> itemList = add_items.itemList;
    try{

        int no = Integer.parseInt(imageField.getText());
        int position =0;
        int codeExsist = 0;
        for(List<Object> i : itemList){
            if( i.get(0).equals(no)){
                position = itemList.indexOf(i);
                codeExsist++;
            }
        }
        if(codeExsist>0){
            Image image = new Image((String) itemList.get(position).get(7));
            ImageView imageView = new ImageView(image);
            Scene scene = new Scene(new javafx.scene.layout.StackPane(imageView),600,600);


            Stage imageStage = new Stage();
            imageStage.setTitle("Image View");
            imageStage.setScene(scene);
            imageStage.show();


        }else {
            FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("error_image.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmlLoader.load(), 500, 150);
            stage.setScene(scene);
            stage.show();

        }



    }catch (NumberFormatException e){
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("error_image.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 500, 150);
        stage.setScene(scene);
        stage.show();        }
}
```

- **Items** object

When viewing the items program uses object to view the items of the table.

```
package com.example.hello;

public class items {
    int item_code;
    String item_name;

    String item_brand;

    double item_Price;

    int item_quantity;

    String item_category;

    String date;


    public items(int item_code, String item_name, String item_brand, double item_Price, int item_quantity, String item_category,
String date) {
        this.item_code = item_code;
        this.item_name = item_name;
        this.item_brand = item_brand;
        this.item_Price = item_Price;
        this.item_quantity = item_quantity;
        this.item_category = item_category;
        this.date = date;
    }




    public int getItem_code() {
        return item_code;
    }

    public void setItem_code(int item_code) {
        this.item_code = item_code;
    }

    public String getItem_name() {
        return item_name;
    }

    public void setItem_name(String item_name) {
        this.item_name = item_name;
    }

    public String getItem_brand() {
        return item_brand;
    }

    public void setItem_brand(String item_brand) {
        this.item_brand = item_brand;
    }

    public double getItem_Price() {
        return item_Price;
    }

    public void setItem_Price(double item_Price) {
        this.item_Price = item_Price;
    }

    public int getItem_quantity() {
        return item_quantity;
    }

    public void setItem_quantity(int item_quantity) {
        this.item_quantity = item_quantity;
    }

    public String getItem_category() {
        return item_category;
    }

    public void setItem_category(String item_category) {
        this.item_category = item_category;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }
}
```

➢ Save items

When User calls this method it will run through the main item list and write the details to a text file called “Item_details.txt”. User can save the item details to a text file by clicking on the save button on the menu page.

- **onSaveButtonClick** method

```
public void onSaveButtonClick() {
    try {
        String fileName = "item_details.txt";
        BufferedWriter writer = new BufferedWriter(new FileWriter(fileName));

        for (List<Object> item : itemList) {
            writer.write(formatItemString(item));
            writer.newLine();
        }

        writer.close();

        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("item_save_popup_msg.fxml"));
        Stage stage = new Stage();
        Scene scene = new Scene(fxmlLoader.load(), 500, 150);
        stage.setScene(scene);
        stage.show();

    } catch (IOException e) {
        e.printStackTrace();

    }
}
```

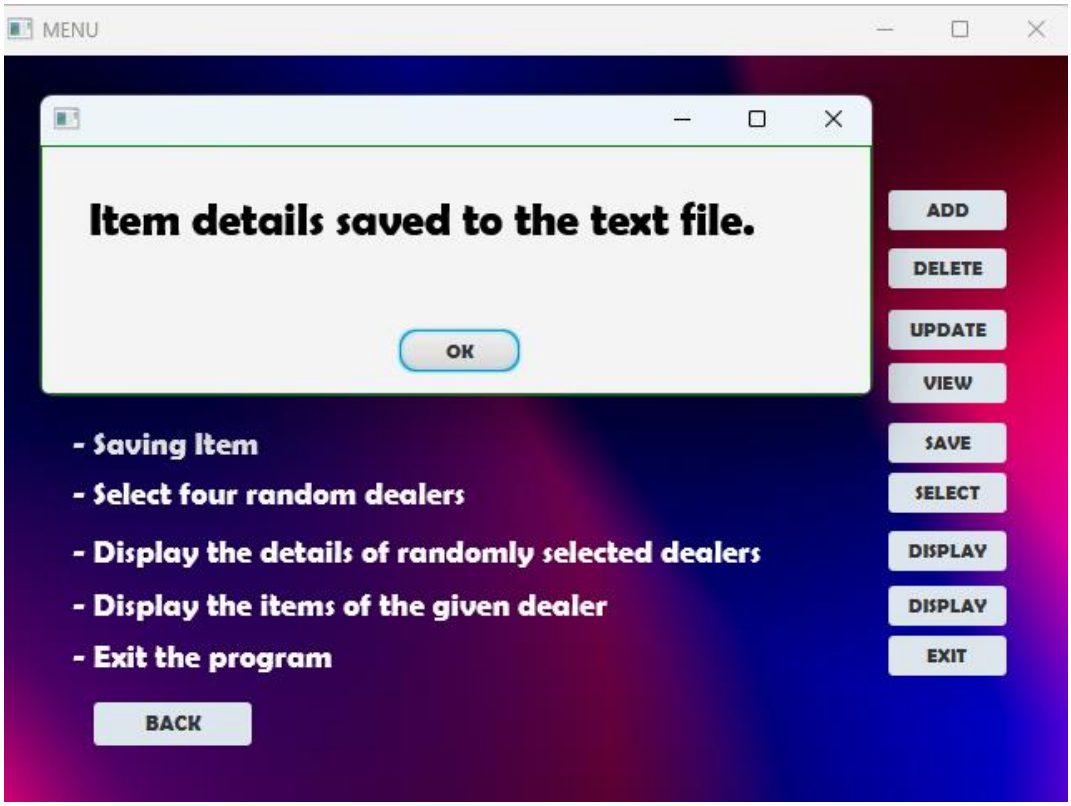


Figure 25

```
item_details
File Edit View

1001 || Think pad x1 carbon || Lenovo || 585000.0 || 10 || Laptop || 2023.03.17 ||
1002 || Inspiral pro 2 || Dell || 450000.0 || 10 || Laptop || 2023.07.18 ||
1003 || Samsung S23 ultra || Samsung || 253500.0 || 5 || Mobile Phone || 2023.07.19 ||
1004 || HP Eac || HP || 8500.0 || 3 || Mouse || 2023.04.07 ||
1005 || JBL head set || JBL || 9800.0 || 5 || Head Set || 2023.05.14 ||
```

➢ Select Random Dealers from the text file.

There is a list called allDealer to bare all the details. For that the program reads Dealer Reader controller and get "readDealerFromFile()" method to read the dealers from the file and select four random dealers.

```java
public void onSelectButtonclick() throws IOException {


    List<Dealer> allDealers = DealerReader.readDealersFromFile();
    randomDealers = DealerSelector.getRandomDealers(allDealers, 4);


    FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("selectedMsg.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 450, 150);
    stage.setTitle("SELECTED");
    stage.setScene(scene);
    stage.show();

}
```

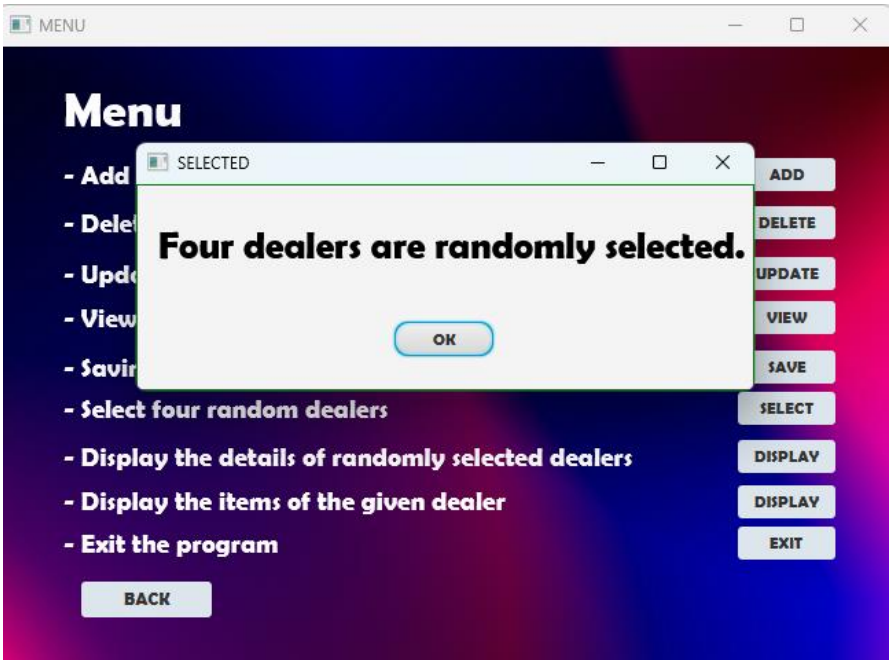


Figure 26

- **DealerReader.readDealersFromFile** method.

```
public static List<Dealer> readDealersFromFile() {
    List<Dealer> dealers = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader("dealers.txt"))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] attributes = line.split(",");
            if (attributes.length == 4) {
                String dealerId = attributes[0].trim();
                String dealerName = attributes[1].trim();
                String phone = attributes[2].trim();
                String location = attributes[3].trim();

                Dealer dealer = new Dealer(dealerId, dealerName, phone, location);
                dealers.add(dealer);
            }

        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println(dealers);

    return dealers;
}
```

Program has a controller called “DealerReader” to read dealers from text file. This reads the text file and store it into a list called “dealer”. And it will select four random dealers from that file randomly. And get their details (dealer ID, dealer name, phone number and dealer location).

- **DealerSelector.getRandomDealers** method

```
public static List<Dealer> getRandomDealers(List<Dealer> dealers, int count) {
    if (count >= dealers.size()) {
        return new ArrayList<>(dealers);
    }

    Set<Dealer> randomDealersSet = new HashSet<>();
    Random random = new Random();
    while (randomDealersSet.size() < count) {
        Dealer dealer = dealers.get(random.nextInt(dealers.size()));
        randomDealersSet.add(dealer);
    }

    List<Dealer> randomDealers = new ArrayList<>(randomDealersSet);

     for (int i = 0; i < randomDealers.size() - 1; i++) {
        for (int j = 0; j < randomDealers.size() - i - 1; j++) {
            if (randomDealers.get(j).getLocation().compareTo(randomDealers.get(j + 1).getLocation()) > 0) {
                Dealer temp = randomDealers.get(j);
                randomDealers.set(j, randomDealers.get(j + 1));
                randomDealers.set(j + 1, temp);
            }
        }
    }

    System.out.println(randomDealers);
    return randomDealers;
}
```

In here program has a controller called “DealerSelector” to select four dealers and sort them according to their location in ascending order.

➢ Displaying Randomly Selected Dealers

In this part user should be able to view the randomly selected dealer details.

- **onDis1ButtonClick** method

```
public void onDis1ButtonClick() throws IOException {

    FXMLLoader fxmlLoader = new FXMLLoader(MENU.class.getResource("RANDOM_DEALERS.fxml"));
    Stage stage = new Stage();
    Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
    stage.setTitle("RANDOMLY SELECTED DEALERS DETAILS");
    stage.setScene(scene);
    stage.show();

    Stage currentStage = (Stage) menuanc.getScene().getWindow();
    currentStage.close();
}
```

After clicking "DISPLAY" button it will print all the details of selected random dealers.

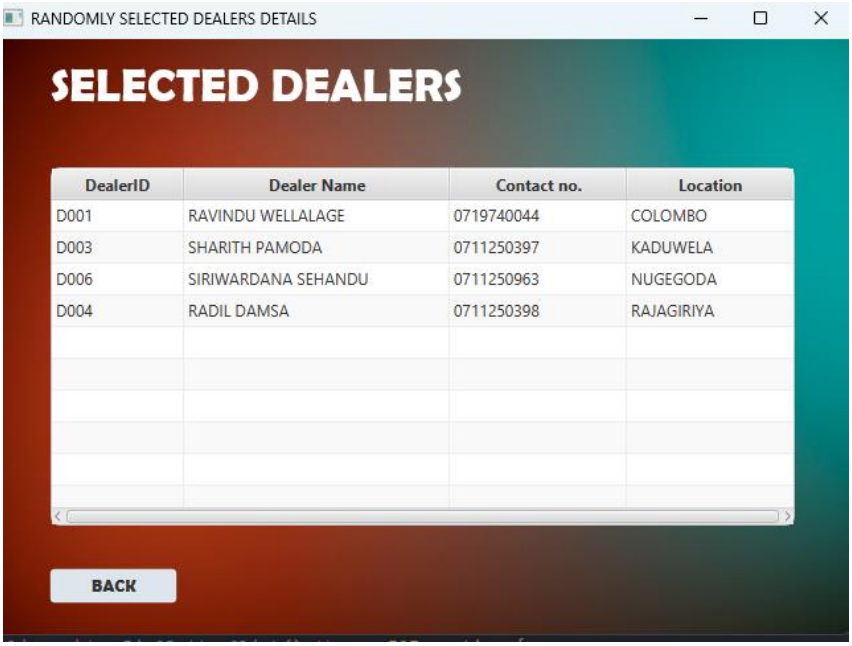


Figure 27

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {


    List<Dealer> selectedDealers = MENU.randomDealers;

    ObservableList<Dealer> dealerView = FXCollections.observableArrayList(selectedDealers);

    dealerID.setCellValueFactory(new PropertyValueFactory<Dealer, String>("dealerID"));
    dealerName.setCellValueFactory(new PropertyValueFactory<Dealer, String>("dealerName"));
    contact.setCellValueFactory(new PropertyValueFactory<Dealer, String>("phone"));
    location.setCellValueFactory(new PropertyValueFactory<Dealer, String>("location"));



    dealerTable.setItems(dealerView);
}
```

In this code "randomDealer" list will assign to the "selectedDealers" list. Then it will pass into a observable list. It uses the "Dealer" object to navigate elements clearly.

- **Dealer** object

```
package com.example.hello;

import javafx.beans.property.SimpleStringProperty;

public class Dealer {

    private String dealerID;
    private String dealerName;
    private String phone;
    private String location;

    @Override
    public String toString() {
        return "Dealer ID: " + dealerID + ", Dealer Name: " + dealerName + ", Phone: " + phone + ", Location: "
+ location;
    }

    public Dealer(String dealerID, String dealerName, String phone, String location) {
        this.dealerID = dealerID;
        this.dealerName = dealerName;
        this.phone = phone;
        this.location = location;
    }

    public String getDealerID() {
        return dealerID;
    }

    public void setDealerID(String dealerID) {
        this.dealerID = dealerID;
    }

    public String getDealerName() {
        return dealerName;
    }

    public void setDealerName(String dealerName) {
        this.dealerName = dealerName;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }
}
```

➢ Displaying Dealer Items

- **onSelectButtonClick** method

Then user will redirect to a new controller called “RANDOM_DEALER_DETAILS.java”. In that file programmer will get the dealer id from the user. When user inserted a dealer ID , It will check whether that dealer id is in the randomly selected dealer list. If there is that dealer id it will print that dealer’s item details. If there is not that dealer id it will print an error message.

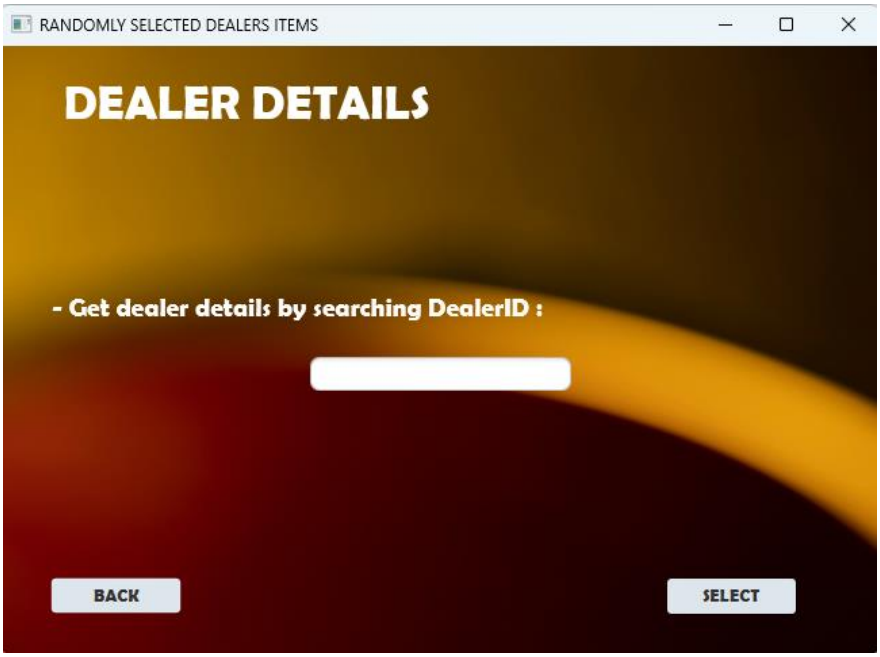


Figure 28

```
public void onSelectButtonClick() throws IOException {
    code = getDealerId.getText().trim().toUpperCase();
    List<Dealer> randomDealers = MENU.randomDealers;

    for (Dealer dealer : randomDealers) {
        if (dealer.getDealerID().equals(code)) {
            displayDealerItems(code);
            FXMLLoader fxmlLoader = new
FXMLLoader(MENU.class.getResource("RANDOM_DEALER_DETAIL.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmlLoader.load(), 650.0, 450.0);
            stage.setTitle("MENU");
            stage.setScene(scene);
            stage.show();

            Stage currentStage = (Stage) dis1.getScene().getWindow();
            currentStage.close();
            return; // Exit the method after showing the details
        }
    }

    System.out.println(dealerItem);

    showAlert("Dealer is not in the list.");
}
```

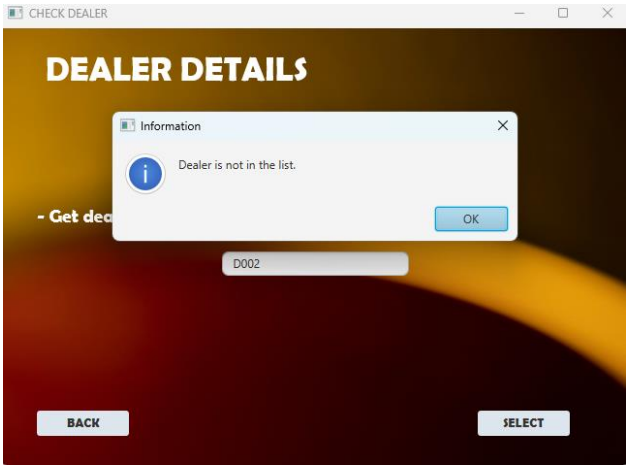


Figure 29

- **displayDealerItems** method

```
private void displayDealerItems(String dealerId) {
    try (BufferedReader br = new BufferedReader(new FileReader("dealers_items.txt"))) {
        String line;
        boolean foundDealer = true;
        while ((line = br.readLine()) != null) {
            line = line.replace("[","");
            line = line.replace("]","");
            String[] attributes = line.split(",");
            if (attributes.length == 5) {
                String itemDealerId = attributes[0].trim();
                String itemName = attributes[1].trim();
                String itemBrand = attributes[2].trim();
                String itemPrice = attributes[3].trim();
                String itemStock = attributes[4].trim();

                if (itemDealerId.equals(code)) {
                    List<Object> temp = new ArrayList<>();
                    temp.add(itemName);
                    temp.add(itemBrand);
                    temp.add(itemPrice);
                    temp.add(itemStock);
                    System.out.println(temp);
                    foundDealer = true;
                    dealerItem.add(temp);

                }
            }
        }

        if (!foundDealer) {
            System.out.println(foundDealer);
            showAlert("Dealer is in the list, but no items found for this dealer.");
        }
    } catch (IOException e) {
        showAlert("Error reading dealer items file.");
    }
}
```

In this method program will go through the items of that particular dealer. And get that dealer's items line by line and print it into a table view. To that I used an object to print all the dealer items to a table view neatly.

- **itemDealer** object

```java
package com.example.hello;

public class itemDealer {

    private String dddname;
    private String dddbrand;
    private String dddprice;
    private String dddquantity;

    public itemDealer(String dddname, String dddbrand, String dddprice, String dddquantity) {
        this.dddname = dddname;
        this.dddbrand = dddbrand;
        this.dddprice = dddprice;
        this.dddquantity = dddquantity;
    }

    public String getDddname() {
        return dddname;
    }

    public void setDddname(String dddname) {
        this.dddname = dddname;
    }

    public String getDddbrand() {
        return dddbrand;
    }

    public void setDddbrand(String dddbrand) {
        this.dddbrand = dddbrand;
    }

    public String getDddprice() {
        return dddprice;
    }

    public void setDddprice(String dddprice) {
        this.dddprice = dddprice;
    }

    public String getDddquantity() {
        return dddquantity;
    }

    public void setDddquantity(String dddquantity) {
        this.dddquantity = dddquantity;
    }
}
```

- **initialize** method

```java
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    itemName.setCellValueFactory(new PropertyValueFactory<itemDealer, String>("dddname"));
    itemBrand.setCellValueFactory(new PropertyValueFactory<itemDealer, String>("dddbrand"));
    itemPrice.setCellValueFactory(new PropertyValueFactory<itemDealer, String>("dddprice"));
    itemStock.setCellValueFactory(new PropertyValueFactory<itemDealer, String>("dddquantity"));
    ObservableList<itemDealer> dItems = FXCollections.observableArrayList();
    for (List<Object> oneItem: dealerItem){
        itemDealer item = new itemDealer(
                (String) oneItem.get(0),
                (String) oneItem.get(1),
                (String) oneItem.get(2),
                (String) oneItem.get(3)
        );
        dItems.add(item);
    }
    List<Dealer> selectedDealers = MENU.randomDealers;
    String dName = "";
    for(Dealer i: selectedDealers){
        if (i.getDealerID().equals(RANDOM_DEALER_2.name)){
            dName = i.getDealerName();
        }
    }
    dealerName.setText(dName);
    dealerID.setText(RANDOM_DEALER_2.name);
    itemDetails.setItems(dItems);
    dealerItem.clear();
}
```

In this code snippet program will check that given item by dealer id. If that dealer is in the selected list, program will check the items for that dealer item. If there are any identified items , it will neatly print in a table.

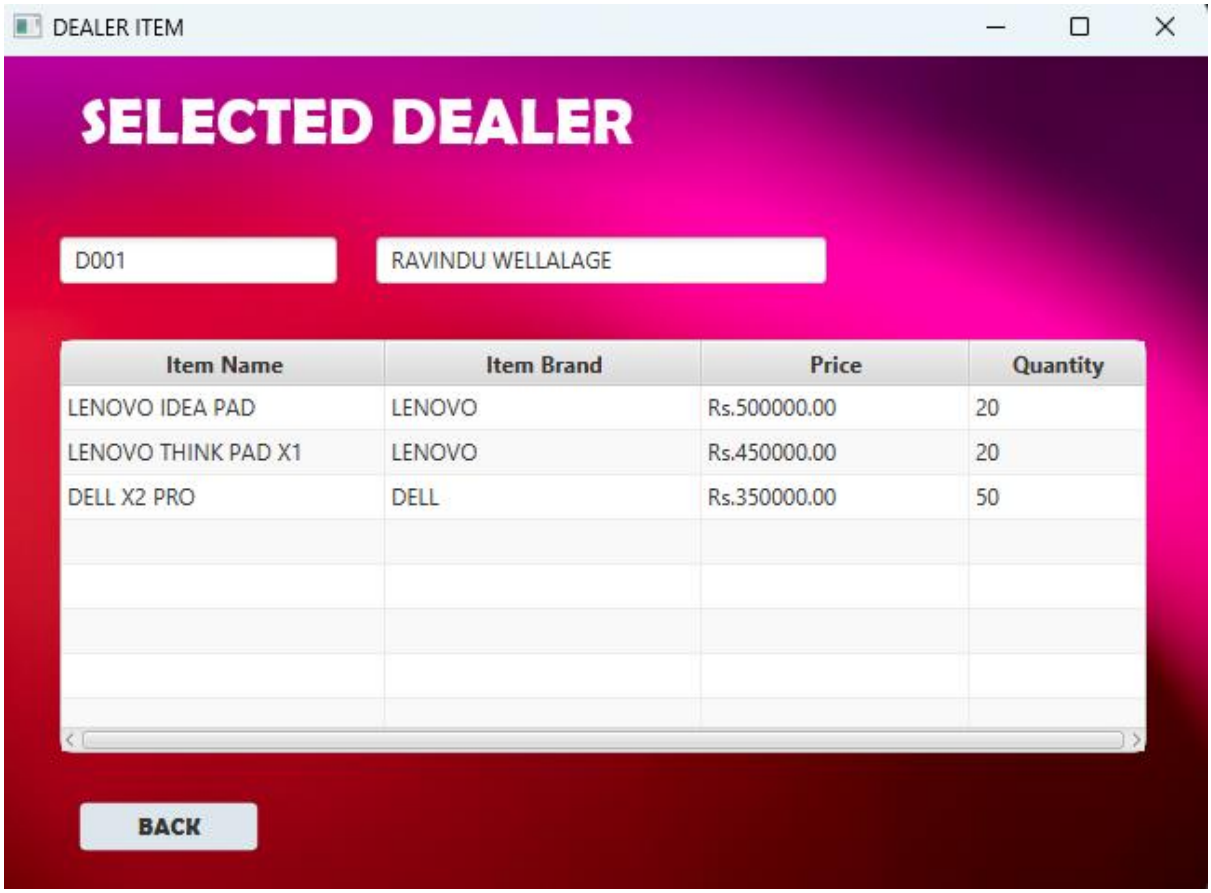


Figure 30

- **Exit** program

```
public void onExitButtonClick(ActionEvent event) {

    Platform.exit();
}
```

Test plan and test cases

Test case	User Input	Description	Expected Output	Actual Output	Pass/Fail
1	Click "ADD" button	Fill all the fields And press "ADD".	"Item added successfully"	"Item added successfully"	Pass
2	ADD – Item Code	Enter invalid Item Code (Not Integer or Empty)	"Enter correct numeric value for item code"	"Enter correct numeric value for item code"	Pass
3	ADD- Item Code	Entering existing Item Code	"Item code is already in use"	"Item code is already in use"	Pass
4	ADD- Item Price	Entering invalid Item Price (Not Integer or Empty)	"Enter correct numeric value for item price"	"Enter correct numeric value for item price"	Pass
5	ADD- Item Quantity	Entering invalid Item Quantity (Not Integer or Quantity)	"Enter correct numeric value for item quantity"	"Enter correct numeric value for item quantity"	Pass
6	ADD – Empty Fields	Keeping any of the input fields are empty including Item Code, Price, and Quantity	"Please fill all the fields"	"Please fill all the fields"	Pass
7	Click "DELETE" button	Entering valid Item Code	"Item deleted successfully"	"Item deleted successfully"	Pass
8	DELETE	Entering invalid Item Code (Not existing)	"Item not found In inventory"	"Item not found In inventory"	Pass
9	Click "UPDATE" button	Enter an item code Which is in inventory.	Giving all the details of that item	Giving all the details of that item	Pass
10	UPDATE	Entering invalid item code which is not in the inventory	"Item not found in inventory"	"Item not found in inventory"	Pass
11	UPDATE Item Code	Entering Existing item code	"Item code is already in use"	"Item code is already in use"	Pass

12	UPDATE Item Code Item price Item Quantity	Entering non numeric item code Item price Item quantity	"Enter correct numeric value for item code , price or quantity"	"Enter correct numeric value for item code , price or quantity"	Pass
13	UPDATE – Empty Fields	Keeping any of the input fields empty including Item Code, Price, and Quantity	"Please fill all the fields"	"Please fill all the fields"	Pass
14	VIEW	Click VID to view Items in ascending order.	Table displayed. Sorted according to the Item Code in ascending order	Table displayed. Sorted according to the Item Code in ascending order	Pass
15	SAVE	Saving files to the text file .	"Item saved into a text file"	"Item saved into a text file"	Pass
16	SELECT	Randomly selecting 4 dealers	"Four dealers are selected randomly"	"Four dealers are selected randomly"	Pass
17	DISPLAY Randomly selected dealer details.	Click Display button To display all the randomly selected dealers.	Display Dealers details sorted by their location.	Display Dealers details sorted by their location.	Pass
18	DISPLAY Details of given dealer	Display the user selected randomized dealer's items	Display item details .	Display item details .	Pass
19	DISPLAY Details of given dealer	Entering a dealer ID which is not in randomly selected dealers list	"Dealer is not in the list"	"Dealer is not in the list"	Pass
20	ESC	Exit the program	"Thank you for using John's internet café!"	"Thank you for using John's internet café!"	Pass

- **Test case – 01**

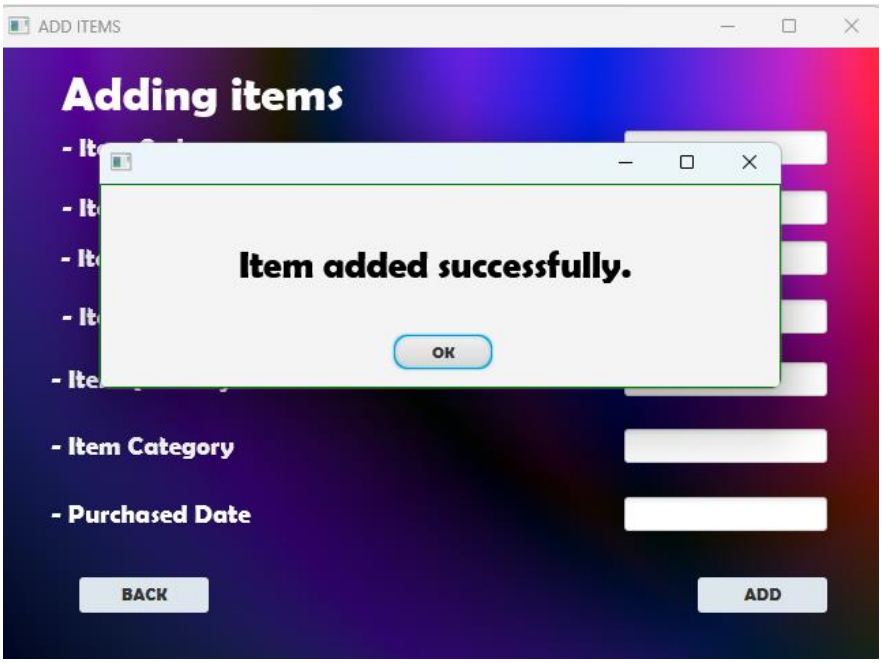


Figure 31

- **Test case – 02**

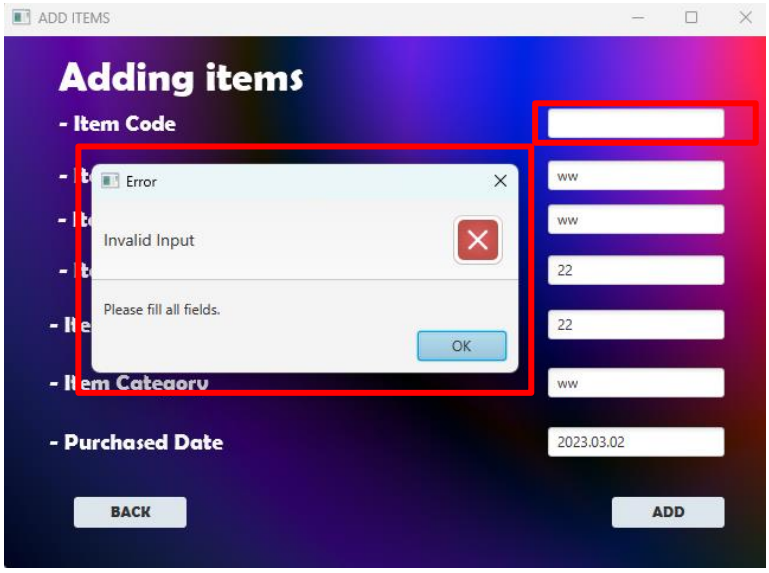


Figure 32

- **Test case – 03**

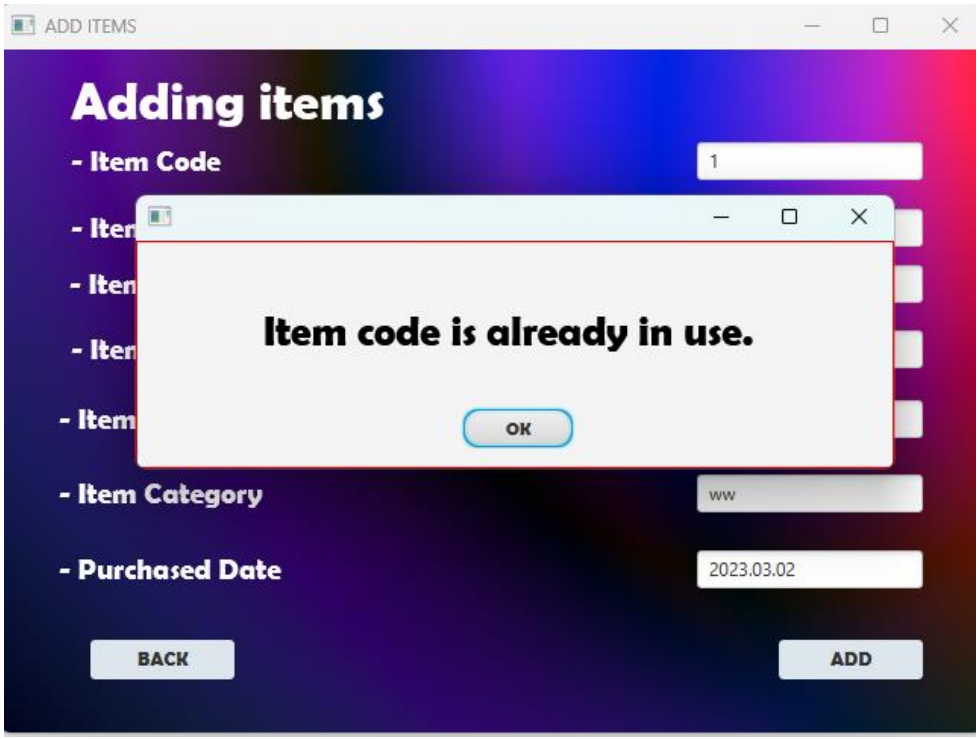


Figure 33

- **Test case – 04**

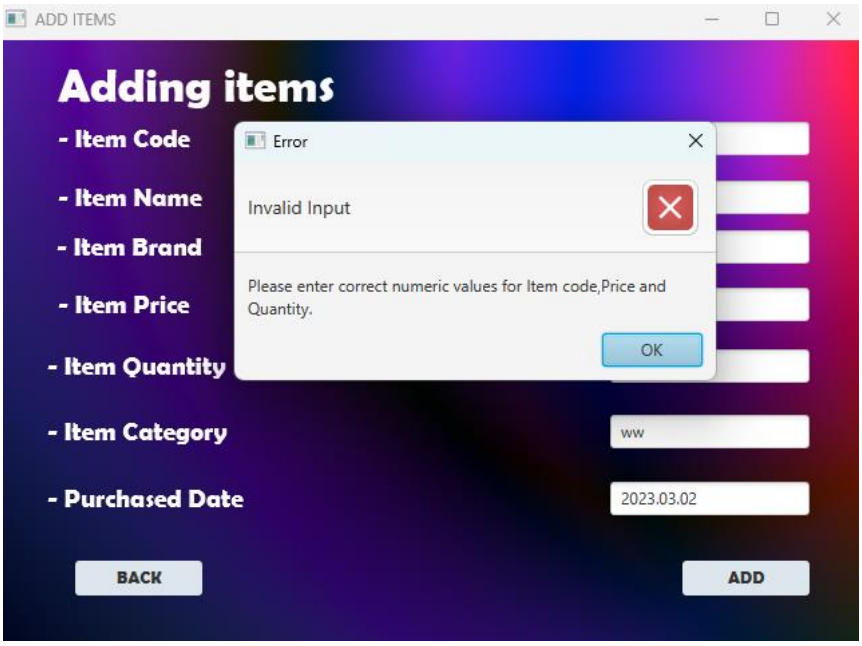


Figure 34

- **Test case – 05**

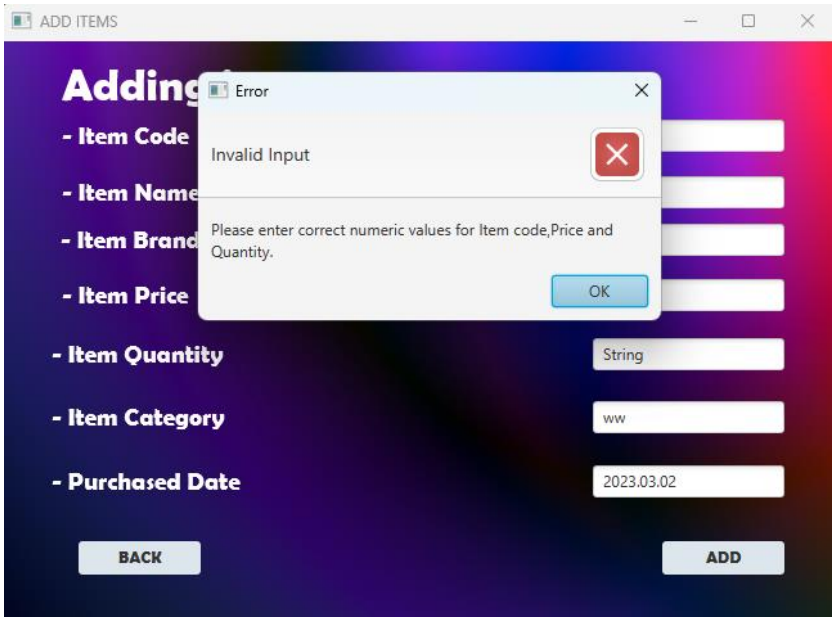


Figure 35

- **Test case – 06**

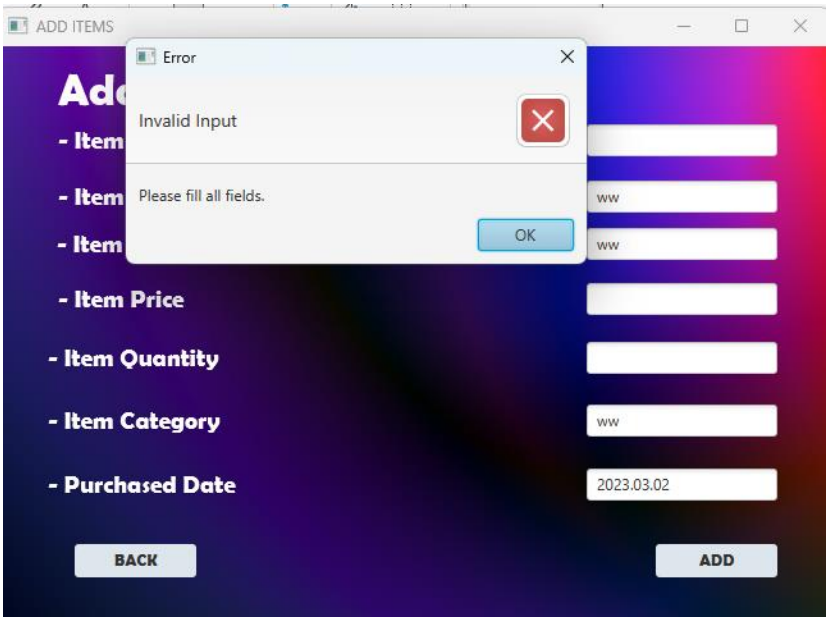


Figure 36

- **Test case – 07**

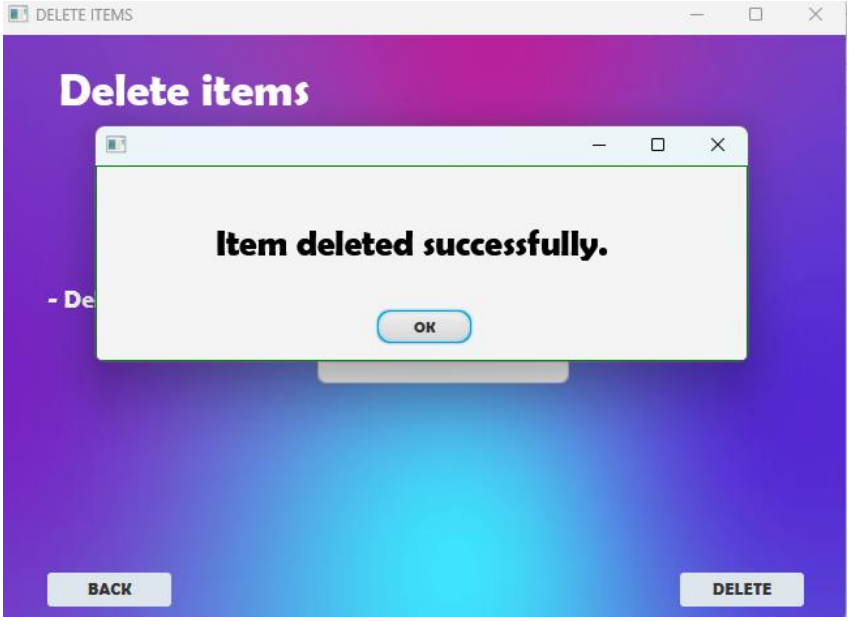


Figure 37

- **Test case – 08**

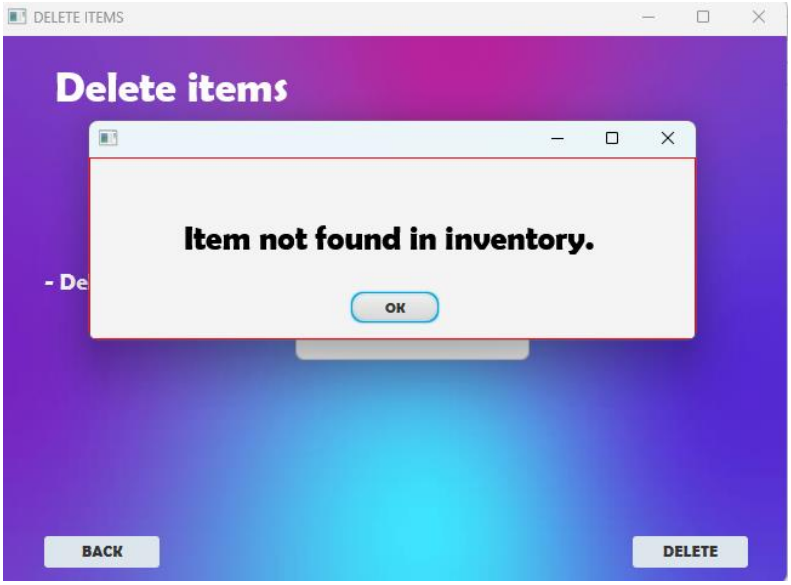


Figure 38

- **Test case – 09**

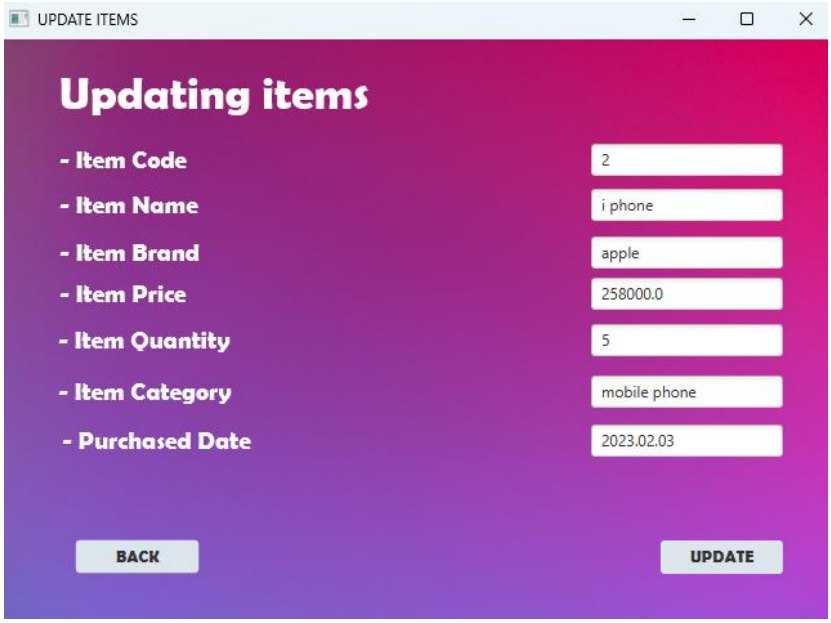


Figure 39

- **Test case – 10**

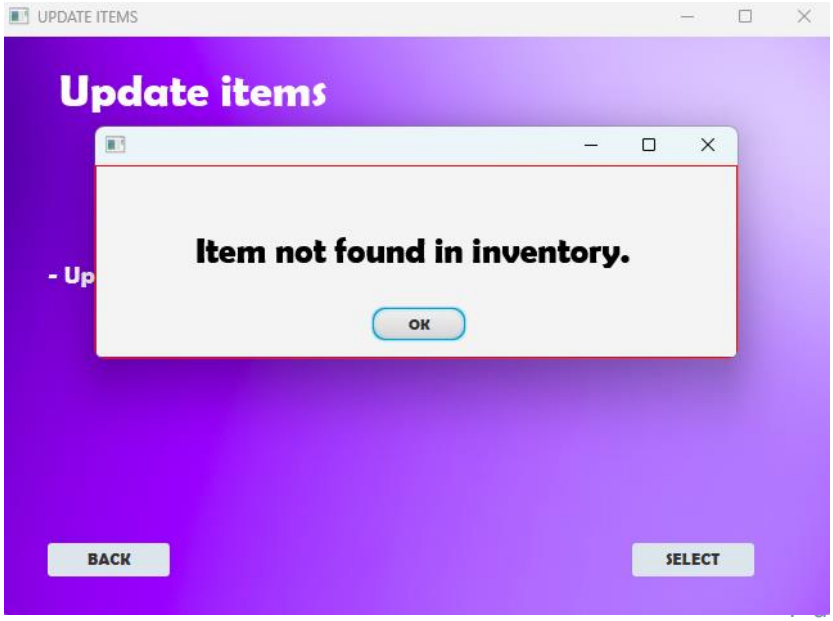


Figure 40

- **Test case – 11**

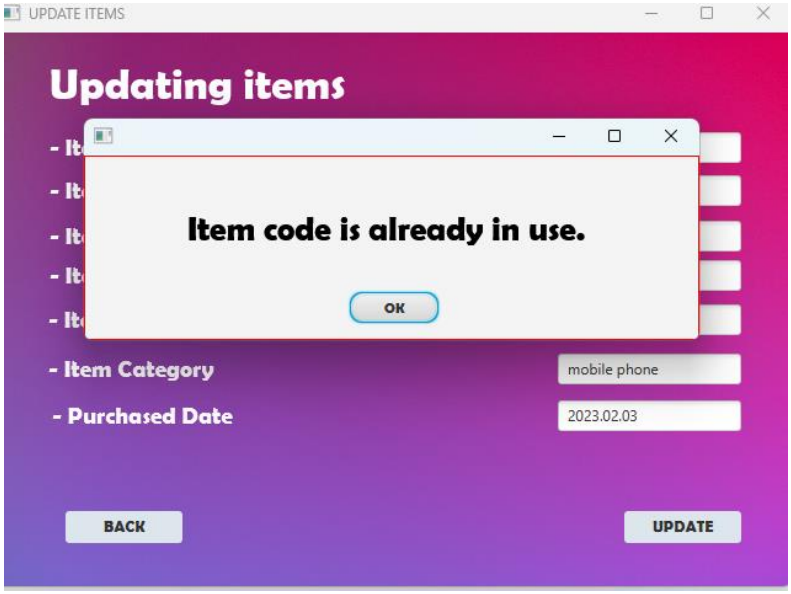


Figure 41

- **Test case – 12**

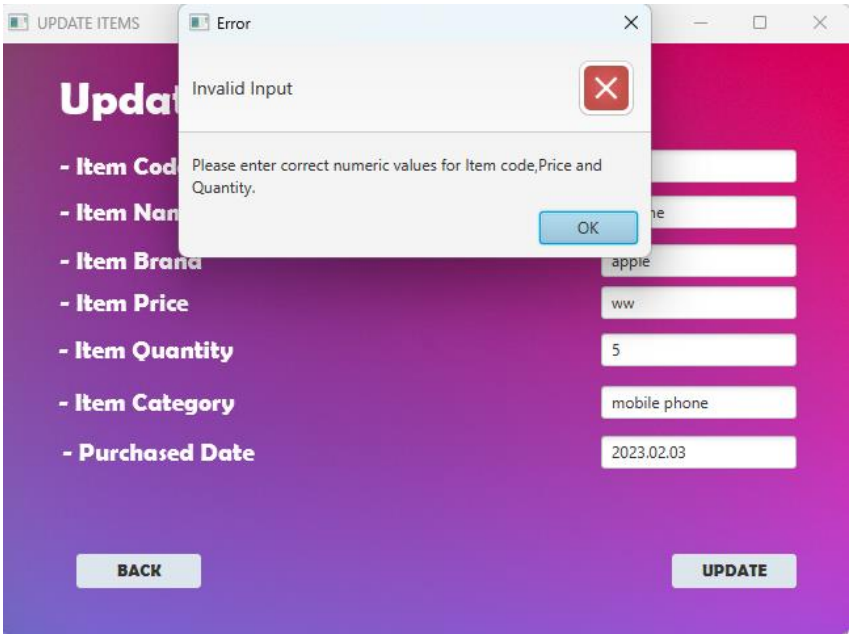


Figure 42

- **Test case – 13**

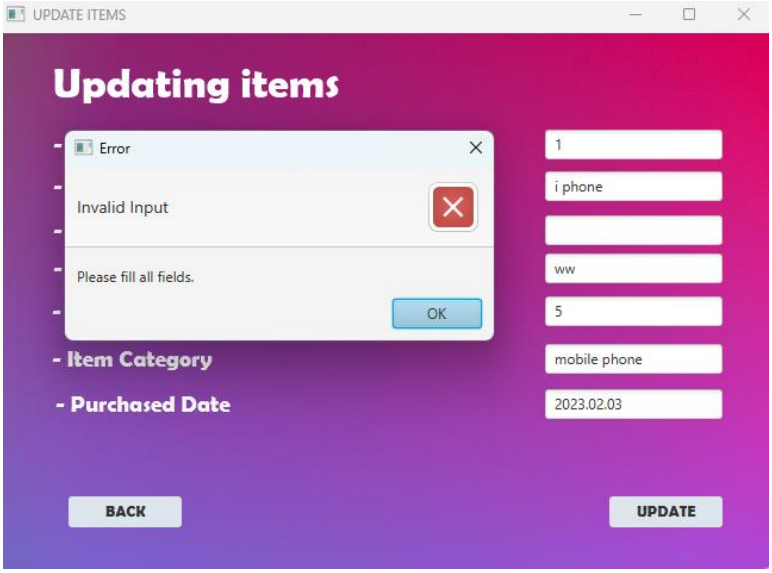


Figure 43

- **Test case – 14**

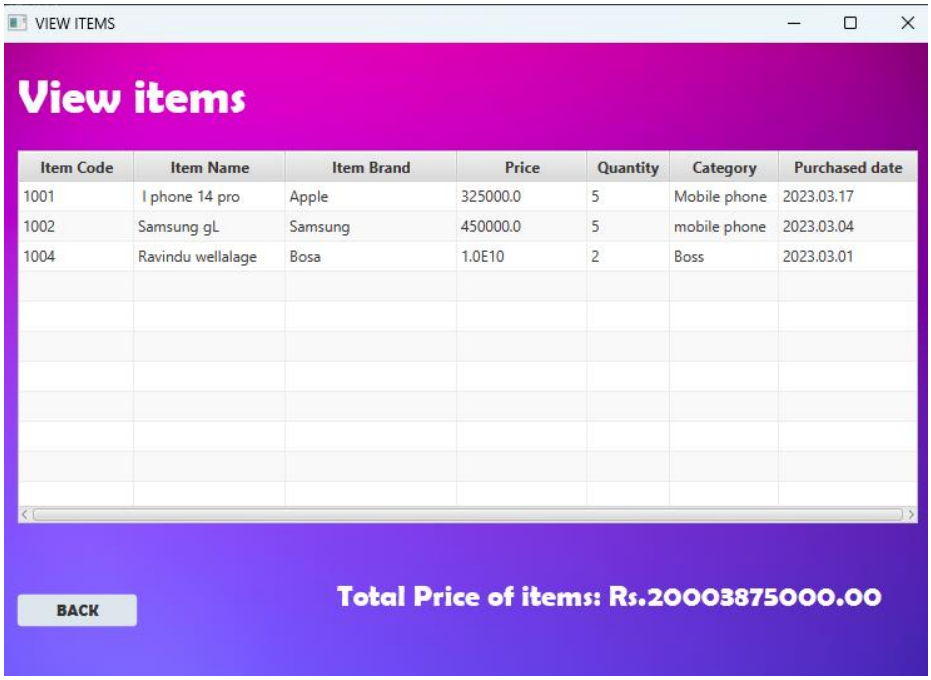


Figure 44

- **Test case – 15**

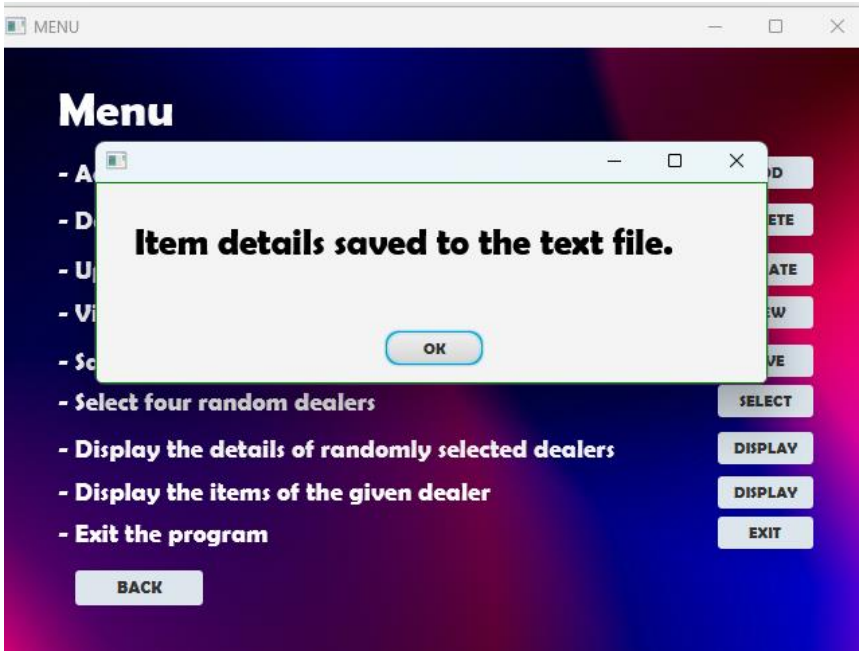


Figure 45

- **Test case – 16**

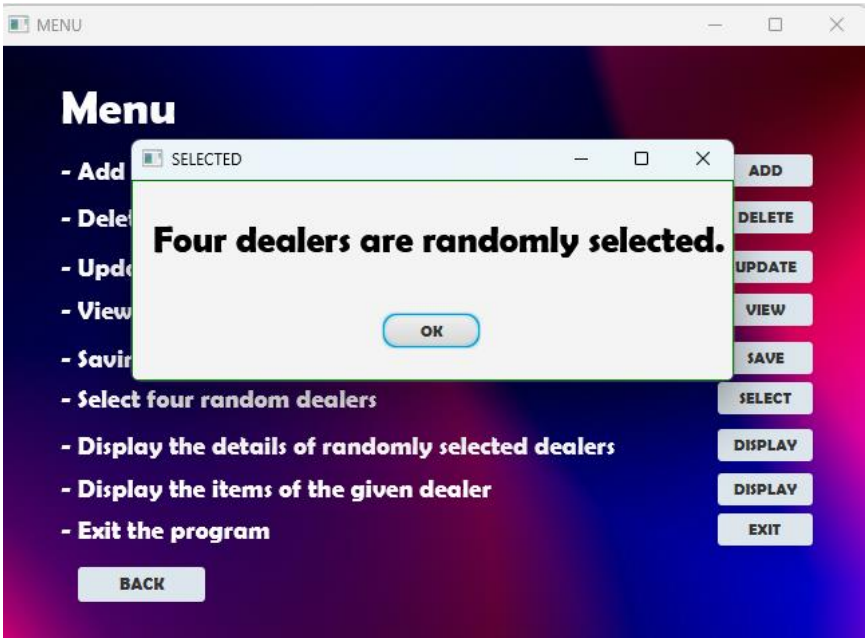
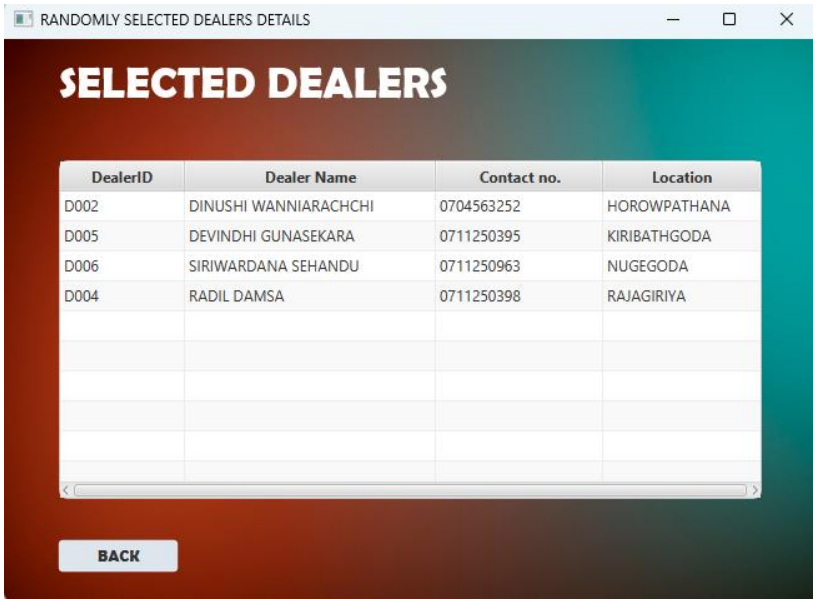


Figure 46

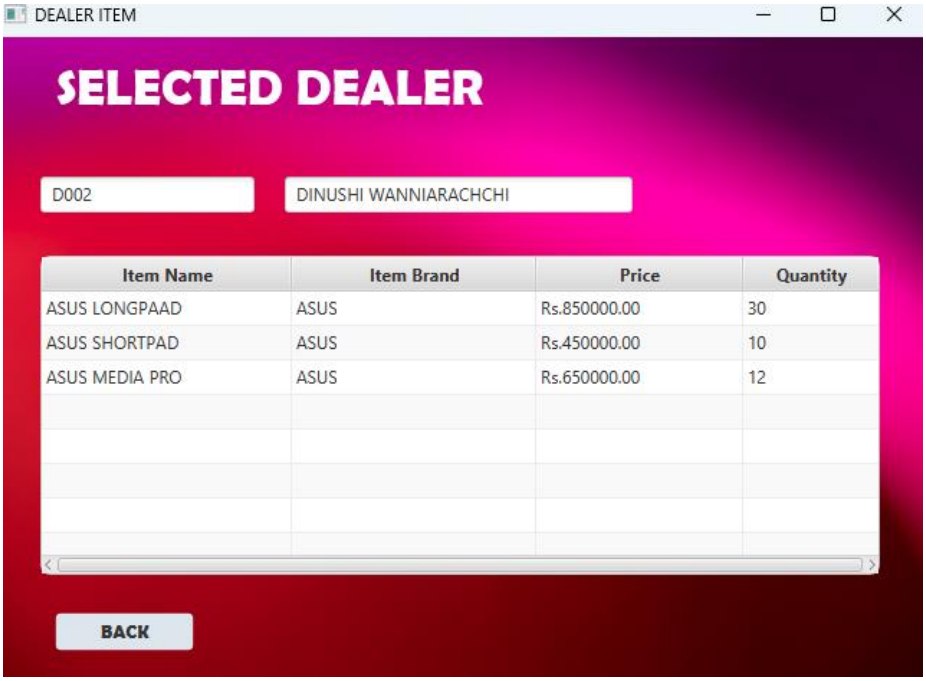
- **Test case – 17**



DealerID	Dealer Name	Contact no.	Location
D002	DINUSHI WANNIARACHCHI	0704563252	HOROWPATHANA
D005	DEVINDHI GUNASEKARA	0711250395	KIRIBATHGODA
D006	SIRIWARDANA SEHANDU	0711250963	NUGEGODA
D004	RADIL DAMSA	0711250398	RAJAGIRIYA

Figure 47

- **Test case – 18**



Item Name	Item Brand	Price	Quantity
ASUS LONGPAAD	ASUS	Rs.850000.00	30
ASUS SHORTPAD	ASUS	Rs.450000.00	10
ASUS MEDIA PRO	ASUS	Rs.650000.00	12

Figure 48

- **Test case – 19**

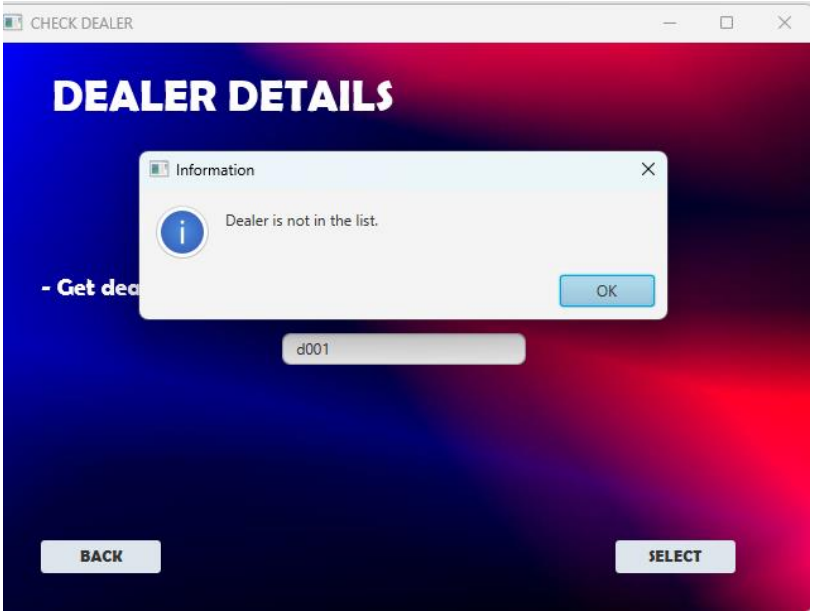


Figure 49

- **Test case – 20**

```
WARNING: Loading FXML document with JavaFX API of version 19 by JavaFX runtime of version 17.0.2-ea
Thank you for using John's internet café!

Process finished with exit code 0
```

Robustness and the maintainability

- Program has several exception handling methods to identify the errors occurred in running time. If user enters a different data type it will gives an alert to the user with a brief explanation.

- In this program there are lot of comments to understand the code easily. When anyone else trying to do the modification of this code, it will really easy to edit the code by reading the comments.

- I used meaningful variable names for assigning values to variables and meaningful names for lists , array lists to identify them easily. As a example, I have a list to store item details of the system. For that I used a list called “itemList” which can easily understand to anyone.

- In this code I used several methods to improve the maintainability of this code. And some methods are able to access as needed. By using these methods clearly user can experience a better quality of this program

Conclusions & assumptions.

➢ Assumptions

- For Item code user has to enter only integer values.
- For the item price user must enter an integer values
- For the item quantity also user has to enter an integer value.
- User must fill all the fields before adding an item.
- When user successfully added an item it will clear all fields and ready to get another item details.
- Every item details added to an array list and that particular array list will added to main array list.
- When deleting an item user needs to call item by it's code.
- When item deleted successfully user will get message.
- When updating an item user needs to call by it's item name.
- Before updating item details , user can see the previous item details in relevant fields.
- User need to add at least one item to delete and update items. Otherwise user will get an error.
- When user save item details to text file , it will not display titles of items (Ex: item name , item ID, item brand etc.) it will display only value of that titles.
- Program will select 4 random dealers by their dealer ID.
- When viewing dealer items user needs to enter randomly selected dealer's dealer ID.
- After enter a valid dealer ID user can see their dealer ID and dealer name.
- When user clicked exist button it will terminate the entire program.

➢ Conclusion

This program is about an inventory system of an internet café creating using javafx and scene builder. In john's café user can add items, delete items , update items , view items , save items into text file , select dealers , view dealer details as well as dealer items. When adding items system will check all the validations of those given details. Whether the item is an integer or not , item price and quantity has correct values. And is it a valid item code or not. And also in deleting items user will get interface to insert an item code. It checks all the validation of that item code and do the deletion. In updating part user has to insert item code to get the details for expected item. For updating part also system will check all the validations. After that system will update the all the lists according to the updated data.

In this inventory system user can select four dealers randomly from the provided dealer text file. Then user can retrieve selected dealers and user can view dealer items by giving selected dealer id to the given interface. And if user needs to terminate the program , there is a button to terminate the program. Used javafx, scenebuilder application, java classes ,objects and oop concepts fot this task.

Reference list

- (No date) *Java arraylist*. Available at: https://www.w3schools.com/java/java_arraylist.asp (Accessed: 30 July 2023).

- *JavaFX tutorial* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/javafx-tutorial/ (Accessed: 30 July 2023).

- *JavaFX: Textfield* (2022) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/javafx-textfield/ (Accessed: 30 July 2023).

- *JavaFX: Button with examples* (2019) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/javafx-button-with-examples/ (Accessed: 30 July 2023).

- *5 bind the GUI to the application logic* (no date) *5 Bind the GUI to the Application Logic (Release 2)*. Available at: https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/bind-ui-to-logic.htm#CHDDADJE (Accessed: 30 July 2023).

- *Bekwam courses* (no date) *Bekwam Courses - Beginning JavaFX TableView*. Available at: https://courses.bekwam.net/public_tutorials/bkcourse_simpletableapp.html (Accessed: 30 July 2023).

- *Gradient vectors, photos and PSD files: Free Download* (no date) *Freepik*. Available at: https://www.freepik.com/search?format=search&query=gradient (Accessed: 30 July 2023).

- *JavaFx + scene builder introduction and Interface Designing* (2020) *YouTube*. Available at: https://www.youtube.com/watch?v=ObQoMh4y_fM&t=182s (Accessed: 30 July 2023).

- Doesn't MatterDoesn't Matter 40544 gold badges1212 silver badges2323 bronze badges *et al.* (1959) *Bubblesort implementation, Stack Overflow*. Available at: https://stackoverflow.com/questions/11644858/bubblesort-implementation (Accessed: 30 July 2023).

Appendices

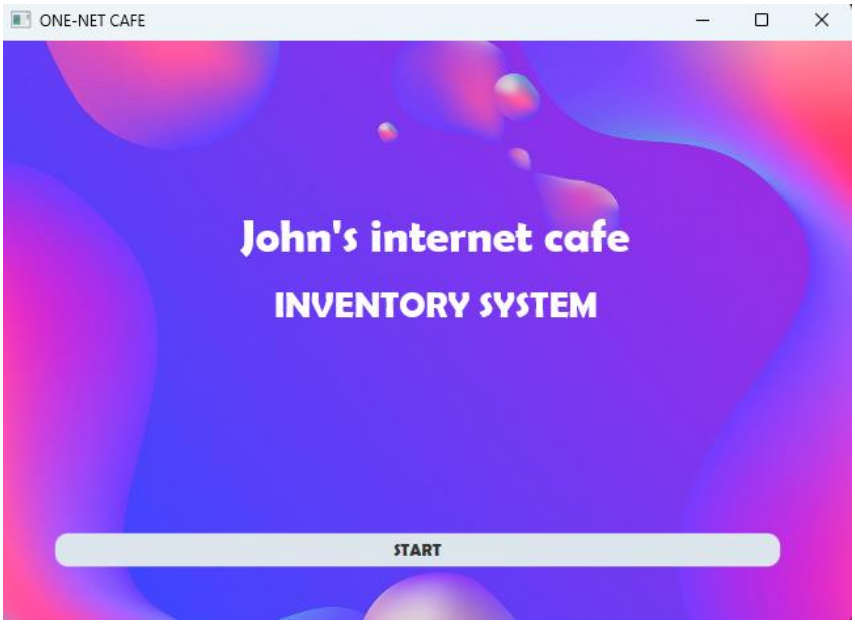


Figure 50

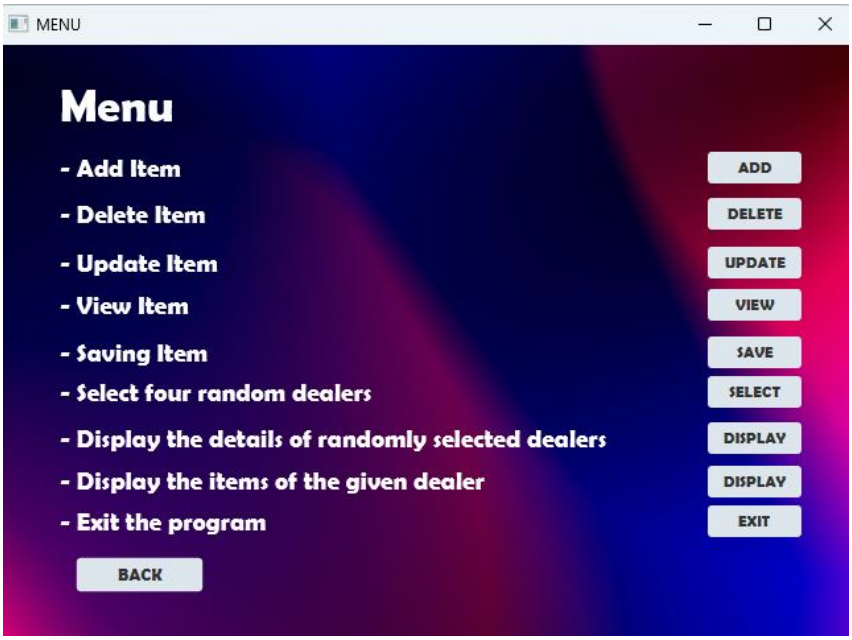


Figure 51

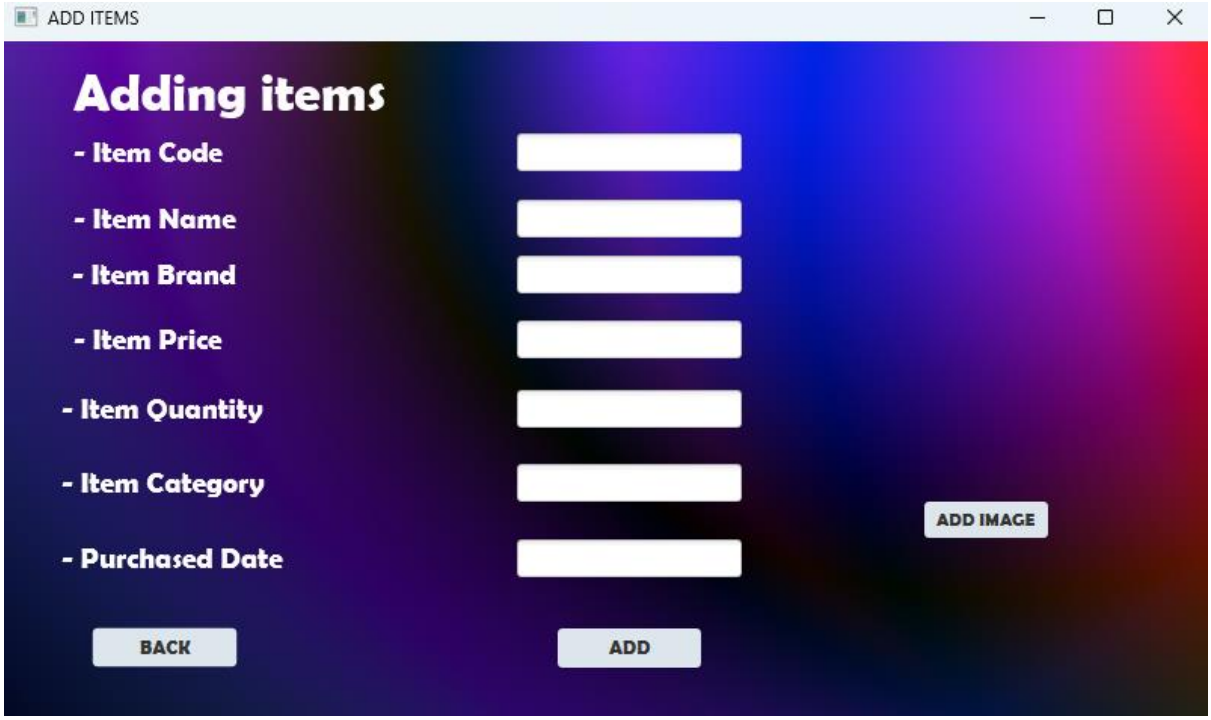


Figure 52

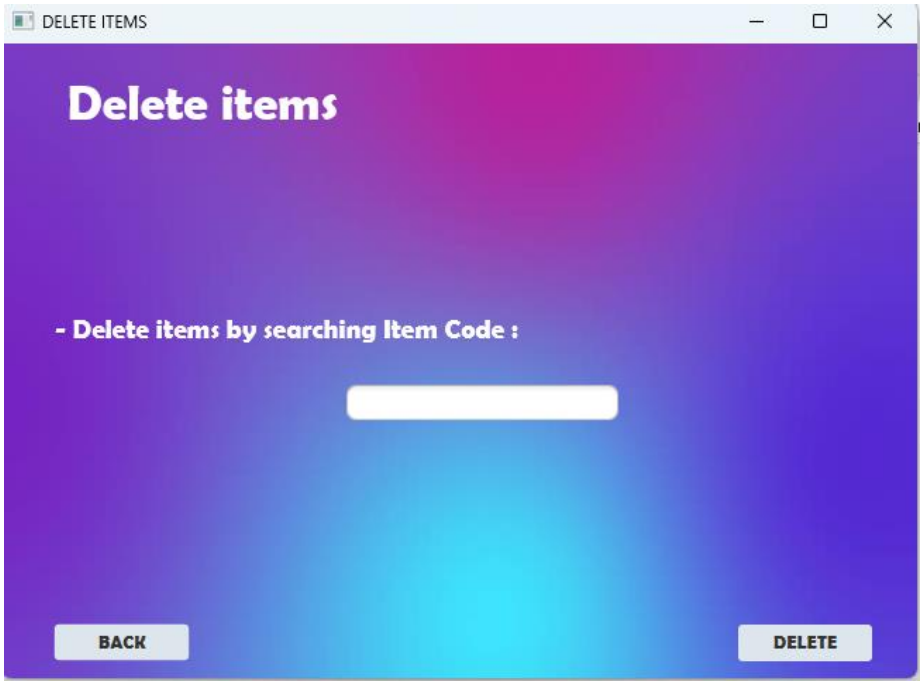


Figure 53

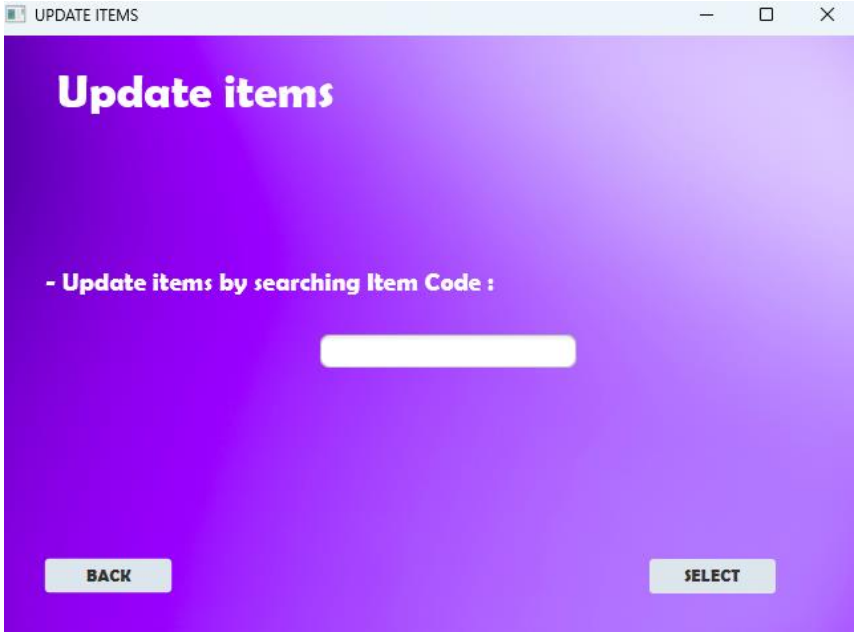


Figure 55

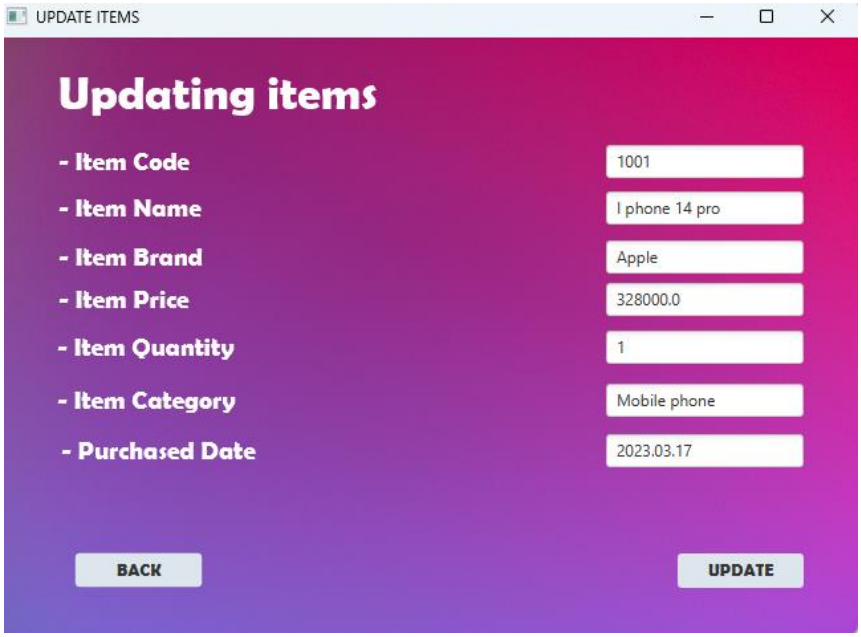


Figure 54


VIEW ITEMS
View items
Item Code
Item Name
Item Brand
Price
Quantity
Category
Purchased date
1
xxx
xxx
25000.0
50
xxx
2023.02.05
Enter item code
1
VIEW
BACK
Total Price of items: Rs.1250000.00

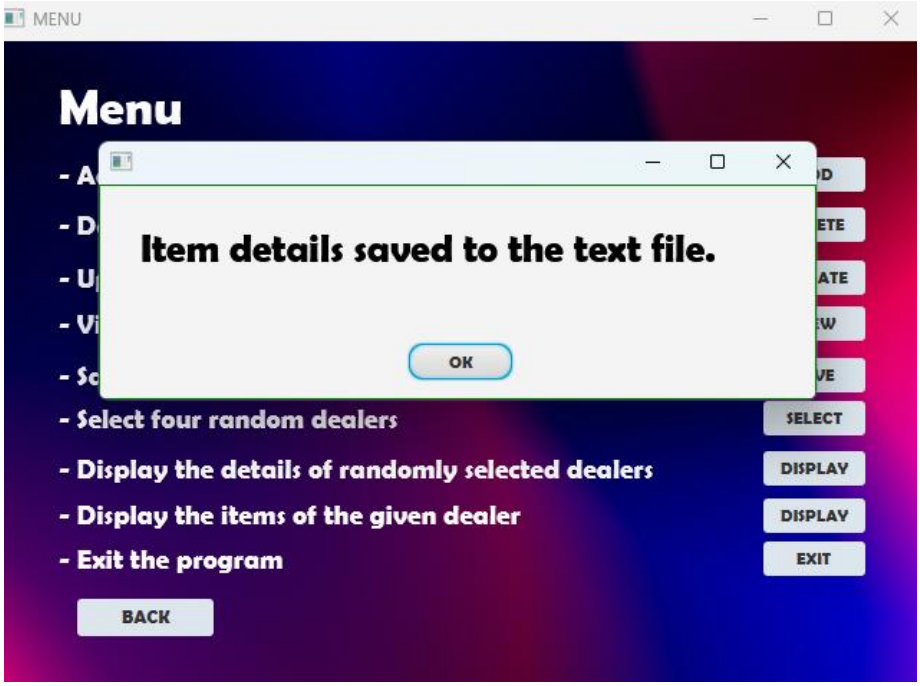

MENU
Menu
Item details saved to the text file.
OK
- Select four random dealers
SELECT
- Display the details of randomly selected dealers
DISPLAY
- Display the items of the given dealer
DISPLAY
- Exit the program
EXIT
BACK

Figure 57

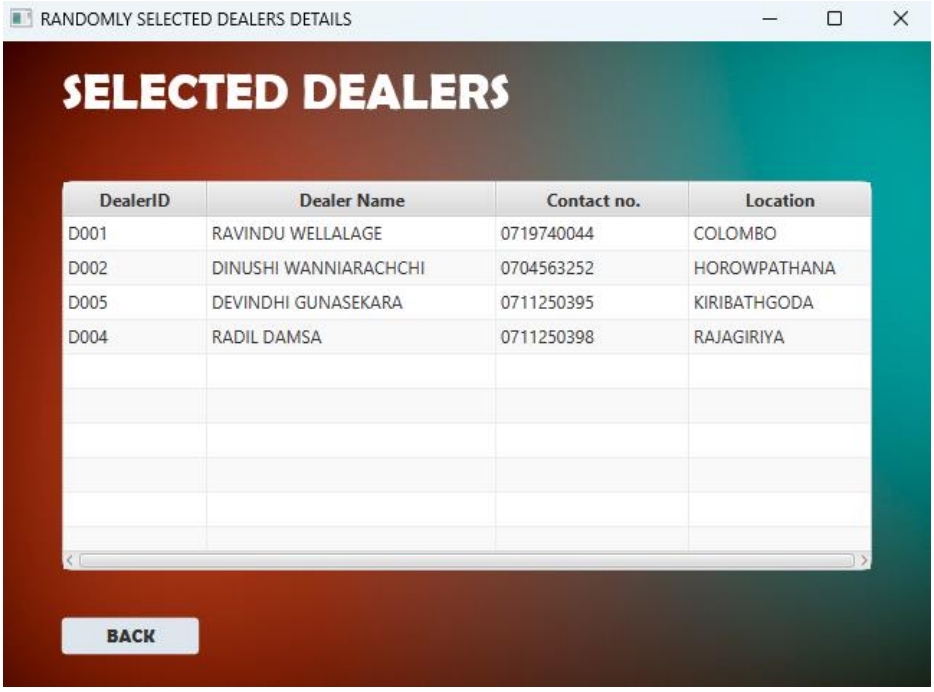


Figure 58

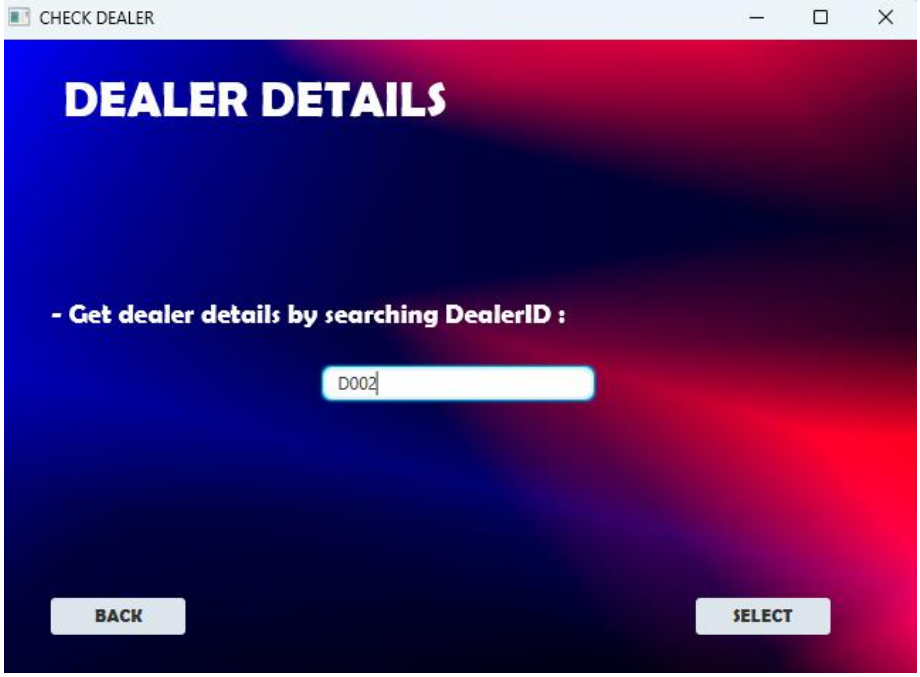


Figure 59

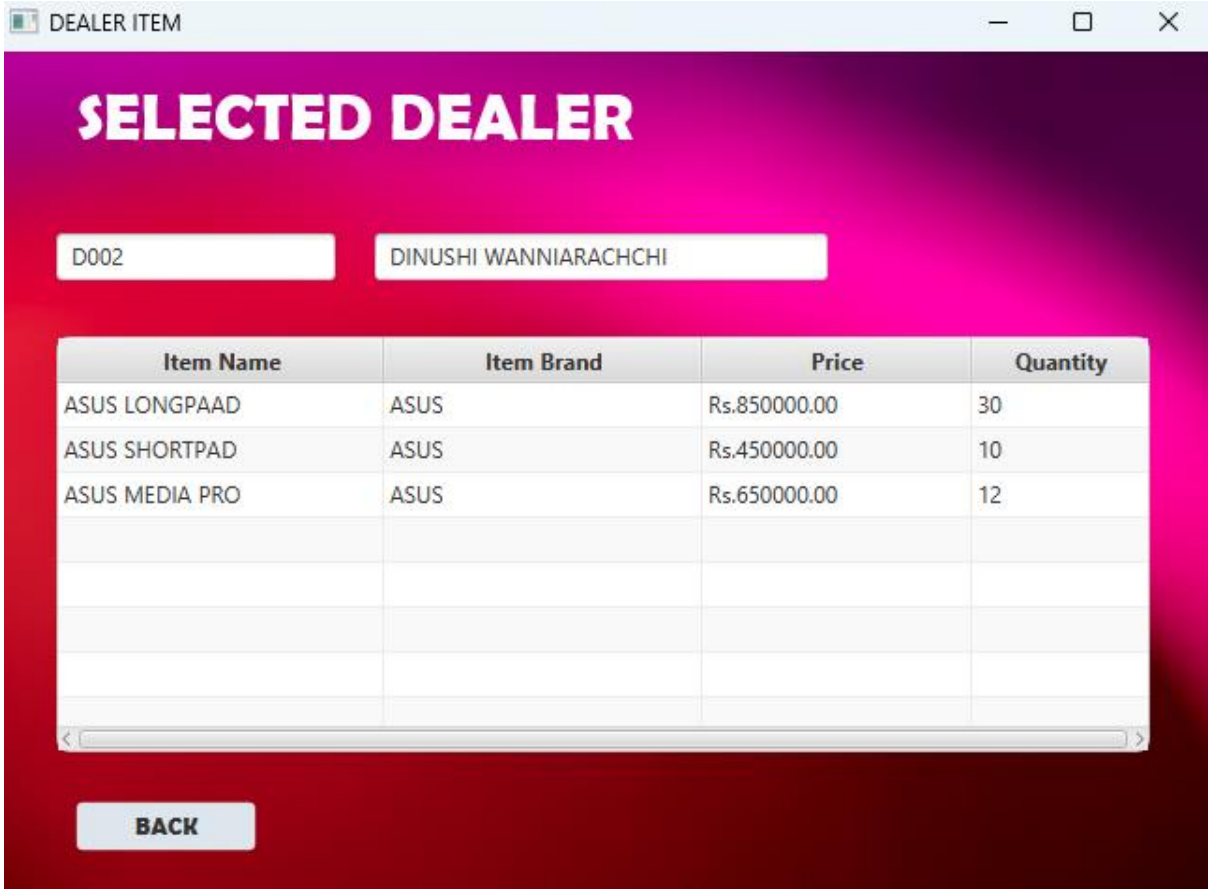

DEALER ITEM
SELECTED DEALER
D002
DINUSHI WANNIARACHCHI
Item Name
Item Brand
Price
Quantity
ASUS LONGPAAD
ASUS
Rs.850000.00
30
ASUS SHORTPAD
ASUS
Rs.450000.00
10
ASUS MEDIA PRO
ASUS
Rs.650000.00
12
BACK

Figure 60

!!!