BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
1.	Ponto	Jeremy	2301891525

Course Code : COMP6571　　　　**Course Name** : Programming Languages

Class : L2AC　　　　**Name of Lecturer(s)** : Jude Joseph Lamug Martinez

Major : Computer Science

Title of Assignment (if any) : Cashie

Type of Assignment : Final Project

Submission Pattern

Due Date : 20-06-2020　　　　**Submission Date** : 20-06-2020

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1.   Jeremy Ponto

TABLE OF CONTENTS

1. Project Specifications

Cashie is a cashier application which is used for calculating transactions between the shopper and the market or shop owner. This application is made for markets or shops which do not have a cashier application. The cashier staff can calculate prices easily by simply determining the quantity for each product and calculate the subtotal, tax and total price by clicking the button for calculating the total price. After calculating the total price, the cashier staff can input the amount of money paid by the shopper and calculate the change to be given back to the shopper by clicking the button for calculating the change. Once the transaction information is complete, the cashier staff can print the receipt for the evidence of the transaction.

NOTE: Printing the receipt is not possible since this is only a desktop application. Instead, the receipt will be shown after transaction information is complete. Besides, the products shown in the application are only samples.

2. Solution Design

2.1. Class Diagram
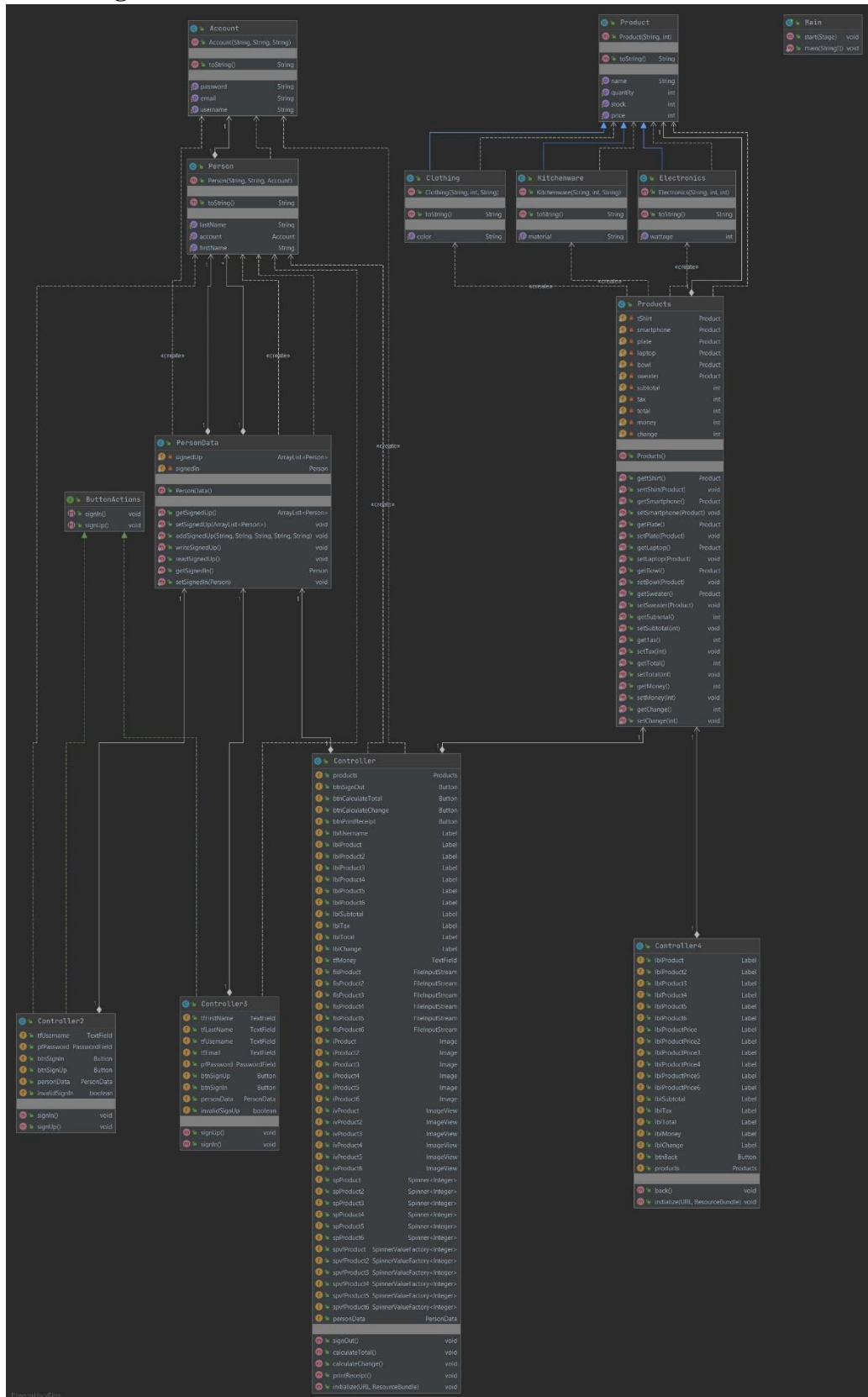
2.2. A Discussion about What is Implemented and How It Works

Before making the application, classes for products are needed to store their details. Therefore, an abstract class named Product is created to store general details of the products.

```java
package sample;

// Define an abstract class named Product as a template for more
specific products.
// This class contains 4 instance attributes.
// stock is for counting how many items left for the corresponding
product.
// name is for the Product's name.
// quantity is for counting how many items are bought for each
transaction.
// price is for the Product's price.
// This class has an overloaded constructor which accepts 2 parameters
which are name and price respectively.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from the inheritors of this
class.
public abstract class Product {
    private int stock;
    private String name;
    private int quantity;
    private int price;

    public Product(String name, int price) {
        this.stock = 100;
        this.name = name;
        this.quantity = 0;
        this.price = price;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
```

```java
    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Product{" +
                "stock=" + stock +
                ", name=" + name +
                ", quantity=" + quantity +
                ", price=" + price +
                '}';
    }
}
```

After the creation of the abstract class for storing general details of products, different classes can be made according to the product type. For this application, three samples of product types are made. Since these product types are specific products, these product types will inherit the Product abstract class.

First, there is the Clothing class which is one of the product types. People can choose various colors of clothing. Thus, color will be the instance attribute specific to this class.

```java
package sample;

// Define a class named Clothing which inherits the Product abstract
class.
// This class contains 5 instance attributes.
// stock, name, quantity and price are inherited from the Product
abstract class.
// color is the Clothing's color.
// This class has an overloaded constructor which accepts 3 parameters
which are name, price and color respectively.
// name and price use the inherited constructor.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from this class.
// Since Clothing is a Product, the object's stock, name, quantity and
price use the inherited toString method.
public class Clothing extends Product {
    private String color;

    public Clothing(String name, int price, String color) {
        super(name, price);
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
```

```
    }

    @Override
    public String toString() {
        return "Clothing{" + super.toString() +
                ", color=" + color +
                '}';
    }
}
```

Second, there is the Electronics class as another type of product. Different electronics have different wattage (power needed to use or charge the electronics), so wattage will be the instance attribute for this class.

```
package sample;

// Define a class named Electronics which inherits the Product
abstract class.
// This class contains 5 instance attributes.
// stock, name, quantity and price are inherited from the Product
abstract class.
// wattage is the Electronics' wattage.
// This class has an overloaded constructor which accepts 3 parameters
which are name, price and wattage respectively.
// name and price use the inherited constructor.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from this class.
// Since Electronics is a Product, the object's stock, name, quantity
and price use the inherited toString method.
public class Electronics extends Product {
    private int wattage;

    public Electronics(String name, int price, int wattage) {
        super(name, price);
        this.wattage = wattage;
    }

    public int getWattage() {
        return wattage;
    }

    public void setWattage(int wattage) {
        this.wattage = wattage;
    }

    @Override
    public String toString() {
        return "Electronics{" + super.toString() +
                ", wattage=" + wattage +
                '}';
    }
}
```

Finally, there is the Kitchenware class as the last product type. When people buys kitchenware, people sometimes checks what the kitchenware is made of. Hence, material become the instance attribute for this class.

```
package sample;
```

```java
// Define a class named Kitchenware which inherits the Product
abstract class.
// This class contains 5 instance attributes.
// stock, name, quantity and price are inherited from the Product
abstract class.
// material is the Kitchenware's material.
// This class has an overloaded constructor which accepts 3 parameters
which are name, price and material respectively.
// name and price use the inherited constructor.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from this class.
// Since Kitchenware is a Product, the object's stock, name, quantity
and price use the inherited toString method.
public class Kitchenware extends Product {
    private String material;

    public Kitchenware(String name, int price, String material) {
        super(name, price);
        this.material = material;
    }

    public String getMaterial() { return material; }

    public void setMaterial(String material) { this.material =
material; }

    @Override
    public String toString() {
        return "Kitchenware{" + super.toString() +
                ", material=" + material +
                '}';
    }
}
```

After these specialized class have been created for specifying product types, class for storing products is needed so transaction details can be stored for showing receipts. Because of that, products created must be class attributes.

```java
package sample;

// Define a class named Products which takes the Product abstract
class as its own class attribute for printing receipts.
// This class contains 11 class attributes.
// tShirt, smartphone, plate, laptop, bowl and sweater are Products
which contain quantity and price to be calculated.
// subtotal is for calculating the total price before tax calculation.
// tax is for calculating the total price's tax.
// total is for calculating the total price with tax included.
// money is for the amount of money the shopper pays.
// change is for the excess amount of money if the shopper pays
greater than the total.
// This class has a default constructor which accepts no parameters.
// Each class attribute has its own mutator and accessor for setting
and getting the value of each one.
public class Products {
    private static Product tShirt = new Clothing("T-shirt", 50000,
"white");
    private static Product smartphone = new Electronics("Smartphone",
```

```java
5000000, 40);
    private static Product plate = new Kitchenware("Plate", 150000,
"ceramic");
    private static Product laptop = new Electronics("Laptop",
10000000, 120);
    private static Product bowl = new Kitchenware("Pot", 250000,
"stainless steel");
    private static Product sweater = new Clothing("Sweater", 150000,
"red");
    private static int subtotal = 0;
    private static int tax = 0;
    private static int total = 0;
    private static int money = 0;
    private static int change = 0;

    public Products() {
    }

    public static Product gettShirt() {
        return tShirt;
    }

    public static void settShirt(Product tShirt) {
        Products.tShirt = tShirt;
    }

    public static Product getSmartphone() {
        return smartphone;
    }

    public static void setSmartphone(Product smartphone) {
        Products.smartphone = smartphone;
    }

    public static Product getPlate() {
        return plate;
    }

    public static void setPlate(Product plate) {
        Products.plate = plate;
    }

    public static Product getLaptop() {
        return laptop;
    }

    public static void setLaptop(Product laptop) { Products.laptop =
laptop; }

    public static Product getBowl() {
        return bowl;
    }

    public static void setBowl(Product bowl) {
        Products.bowl = bowl;
    }

    public static Product getSweater() {
        return sweater;
```

```
        }

    public static void setSweater(Product sweater) {
        Products.sweater = sweater;
    }

    public static int getSubtotal() {
        return subtotal;
    }

    public static void setSubtotal(int subtotal) {
        Products.subtotal = subtotal;
    }

    public static int getTax() {
        return tax;
    }

    public static void setTax(int tax) {
        Products.tax = tax;
    }

    public static int getTotal() {
        return total;
    }

    public static void setTotal(int total) {
        Products.total = total;
    }

    public static int getMoney() { return money; }

    public static void setMoney(int money) { Products.money = money; }

    public static int getChange() {
        return change;
    }

    public static void setChange(int change) {
        Products.change = change;
    }
}
```

Classes for storing products have been created. The staff needs an account because the application needs a sign in to be used. Now, a class named account is created so that user can sign up for signing in to the application.

```
package sample;

import java.io.Serializable;

// Define a class named Account which implements the Serializable
interface.
// The interface must be implemented in order to serialize the object
into a stream so that it can be saved in a file.
// This class contains 3 instance attributes.
// username is the Account's username.
// email is the Account's email.
// password is the Account's password.
// This class has an overloaded constructor which accepts 3 parameters
```

```java
which are username, email and password respectively.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from this class.
public class Account implements Serializable {
    private String username;
    private String email;
    private String password;

    public Account(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Account{" +
                "username=" + username +
                ", email=" + email +
                ", password=" + password +
                '}';
    }
}
```

Account is not on its own. A staff must have its own account. For this case, a class named Person must be created so the staff can type his/her first and last name and the account can be stored in a Person object.

```java
package sample;

import java.io.Serializable;

// Define a class named Person which implements the Serializable
interface and takes the Account class as its own instance attribute.
```

```java
// The interface must be implemented in order to serialize the object
into a stream so that it can be saved in a file.
// This class contains 3 instance attributes.
// firstName is the Person's first name.
// lastName is the Person's last name.
// account is the Person's account.
// This class has an overloaded constructor which accepts 3 parameters
which are firstName, lastName and account respectively.
// Each instance attribute has its own mutator and accessor for
setting and getting the value of each one.
// This class has an overridden toString method which returns the
information of the object instantiated from this class.
public class Person implements Serializable {
    private String firstName;
    private String lastName;
    private Account account;

    public Person(String firstName, String lastName, Account account)
{
        this.firstName = firstName;
        this.lastName = lastName;
        this.account = account;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Account getAccount() {
        return account;
    }

    public void setAccount(Account account) {
        this.account = account;
    }

    @Override
    public String toString() {
        return "Person{" +
                "firstName=" + firstName +
                ", lastName=" + lastName +
                ", account=" + account +
                '}';
    }
}
```

All signed up accounts must be stored for later use of the application. This is the reason why PersonData class is created. The ArrayList which stores all signed up accounts will be written to a file each time there is a new account.

```java
package sample;

import java.io.*;
import java.util.ArrayList;

// Define a class named PersonData for storing signed up Person's
accounts and currently signed in Person's account.
// This class contains 2 class attributes.
// signedUp is for storing signed up Person's accounts.
// signedIn is for storing currently signed in Person's account.
// This class has a default constructor which accepts no parameters.
// Each class attribute has its own mutator and accessor for setting
and getting the value of each one.
// This class has 3 custom methods.
// addSignedUp is for adding a new Person's account into signedUp.
// writeSignedUp is to write signedUp to a file so signed up Person's
accounts so they can be used later to sign in to the application.
// readSignedUp is to read signed up Person's accounts from a file so
they can be stored again in signedUp and used to sign in to the
application.
public class PersonData {
    private static ArrayList<Person> signedUp = new ArrayList<>();
    private static Person signedIn;

    public PersonData() {
    }

    public static ArrayList<Person> getSignedUp() {
        return signedUp;
    }

    public static void setSignedUp(ArrayList<Person> signedUp) {
        PersonData.signedUp = signedUp;
    }

    public static void addSignedUp(String firstName, String lastName,
String username, String email, String password) {
        signedUp.add(new Person(firstName, lastName, new
Account(username, email, password)));
    }

    public static void writeSignedUp() {
        try {
            FileOutputStream fos = new
FileOutputStream("signedUpAccounts.txt");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(signedUp);
            oos.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void readSignedUp() {
```

```
        try {
            FileInputStream fis = new
FileInputStream("signedUpAccounts.txt");
            ObjectInputStream ois = new ObjectInputStream(fis);
            signedUp = (ArrayList<Person>) ois.readObject();
            ois.close();
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static Person getSignedIn() {
        return signedIn;
    }

    public static void setSignedIn(Person signedIn) {
        PersonData.signedIn = signedIn;
    }
}
```

Before the start of making controller classes, there will be a button for singing in and up in the sign in scene (sample2.fxml) and the sign up scene (sample3.fxml). Those two buttons in those scenes will have the same name and signature. Therefore, we need to make an interface named ButtonActions so that the methods can be overridden according to each scene's needs (polymorphism).

```
package sample;

// Define an interface name ButtonActions as a function template for
signing in and up at sample2.fxml and sample3.fxml.
// This interface has 2 custom functions which will be overridden in
classes Controller2 and Controller3, the controller classes for each
of the scene mentioned above respectively.
// signIn is for signing in to application from sign in scene
(sample2.fxml) to product counting and price calculation scene
(sample.fxml) in the Controller2 class.
// signIn is also for changing scene from sign up scene (sample3.fxml)
to sign in scene (sample2.fxml) if a new staff has created a new
account in the Controller3 class.
// Changing scene is the reason why signIn must throw Exception.
// signUp is for signing up a new account for a new staff in the
Controller3 class.
// signUp is also for changing scene from sign in scene (sample2.fxml)
to sign up scene (sample3.fxml) if a new staff has not created a new
account in the Controller2 class.
// Changing scene is the reason why signUp must throw Exception.
public interface ButtonActions {
    public void signIn() throws Exception;

    public void signUp() throws Exception;
}
```

All necessary classes have been created. Now, it is the time to make controller classes for each scene. First, we make a controller class for the product counting and price calculation scene (sample.fxml) named Controller. This class

will control all product counting that the shopper buys, subtotal, tax, total, and change received by the shopper from the excess money that the shopper pays.

```java
package sample;

import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

import java.io.FileInputStream;
import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

// Define a class named Controller which implements the Initializable
interface and takes the Products and PersonData class as its own
instance attribute.
// This class is necessary for controlling the product counting and
price calculation scene (sample.fxml) controls.
// The interface must be implemented in order to override the
initialize method for setting up necessary controls in the product
counting and price calculation scene (sample.fxml) on the start of the
scene.
// The class contains all necessary controls and instance attributes
for the product counting and price calculation scene (sample.fxml).
// products is for setting up the products for display.
// btnSignOut is for signing out from the application.
// btnCalculateTotal is for calculating the subtotal, tax and total of
bought products.
// btnCalculateChange is for calculating the change if the shopper
pays greater than the total.
// btnPrintReceipt is for printing the receipt.
// Labels are for displaying the staff's username, products' name and
price for each item, subtotal, tax, total and change.
// tfMoney is for getting the amount of money the shopper pays.
// FileInputStreams are for obtaining files from the project's folder.
// Images are for storing files as images.
// ImageViews are for displaying products' images.
// Spinners are for counting the number of products the shopper
bought.
// SpinnerValueFactories are for setting the Spinners' values.
// personData is for getting the staff's username when the staff signs
in to and signs out from the application.
// This class has 4 custom methods for each button.
// signOut is for signing out from the application when the staff
clicks btnSignOut.
// signOut must throw Exception in order to change scene to the sign
in scene (sample2.fxml).
// calculateTotal is for calculating the subtotal, tax and total of
bought products when the staff clicks btnCalculateTotal.
// calculateChange is for calculating the change if the shopper pays
greater than the total when the staff clicks btnCalculateChange.
// printReceipt is for printing the receipt when the staff clicks
btnPrintReceipt.
// printReceipt must throw Exception in order to change scene to the
```

```
receipt scene (sample4.fxml).
public class Controller implements Initializable {
    public Products products;

    public Button btnSignOut, btnCalculateTotal, btnCalculateChange,
btnPrintReceipt;
    public Label lblUsername, lblProduct, lblProduct2, lblProduct3,
lblProduct4, lblProduct5, lblProduct6, lblSubtotal, lblTax, lblTotal,
lblChange;
    public TextField tfMoney;
    public FileInputStream fisProduct, fisProduct2, fisProduct3,
fisProduct4, fisProduct5, fisProduct6;
    public Image iProduct, iProduct2, iProduct3, iProduct4, iProduct5,
iProduct6;
    public ImageView ivProduct, ivProduct2, ivProduct3, ivProduct4,
ivProduct5, ivProduct6;
    public Spinner<Integer> spProduct = new Spinner();
    public Spinner<Integer> spProduct2 = new Spinner();
    public Spinner<Integer> spProduct3 = new Spinner();
    public Spinner<Integer> spProduct4 = new Spinner();
    public Spinner<Integer> spProduct5 = new Spinner();
    public Spinner<Integer> spProduct6 = new Spinner();
    public SpinnerValueFactory<Integer> spvfProduct = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.gettShirt().getStock(), 0);
    public SpinnerValueFactory<Integer> spvfProduct2 = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.getSmartphone().getStock(), 0);
    public SpinnerValueFactory<Integer> spvfProduct3 = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.getPlate().getStock(), 0);
    public SpinnerValueFactory<Integer> spvfProduct4 = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.getLaptop().getStock(), 0);
    public SpinnerValueFactory<Integer> spvfProduct5 = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.getBowl().getStock(), 0);
    public SpinnerValueFactory<Integer> spvfProduct6 = new
SpinnerValueFactory.IntegerSpinnerValueFactory(0,
products.getSweater().getStock(), 0);

    public PersonData personData;

    // When the staff clicks btnSignOut, this method will be called.
    // The staff will be signed out from the application and the scene
will change to sample2.fxml.
    public void signOut() throws Exception {
        personData.setSignedIn(new Person("", "", new Account("", "",
"")));


lblUsername.setText(personData.getSignedIn().getAccount().getUsername(
));

        Stage primaryStage;
        Parent root;

        primaryStage = (Stage) btnSignOut.getScene().getWindow();
        root =
```

```java
FXMLLoader.load(getClass().getResource("sample2.fxml"));

        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }

    // When the staff clicks btnCalculateTotal, this method will be
called.
    // The staff will get the subtotal, tax and total.
    // If no product is counted, a warning will appear that the staff
must count the number of products the shopper bought.
    // btnCalculateTotal must be clicked first in order to set tfMoney
enabled and btnCalculateChange visible.
    // The reason is that because the subtotal, tax and total must be
calculated first before inputting the shopper's money and calculating
the change.
    public void calculateTotal() {
        products.gettShirt().setQuantity(spProduct.getValue());
        products.getSmartphone().setQuantity(spProduct2.getValue());
        products.getPlate().setQuantity(spProduct3.getValue());
        products.getLaptop().setQuantity(spProduct4.getValue());
        products.getBowl().setQuantity(spProduct5.getValue());
        products.getSweater().setQuantity(spProduct6.getValue());

        products.setSubtotal(spProduct.getValue() *
products.gettShirt().getPrice() + spProduct2.getValue() *
products.getSmartphone().getPrice() + spProduct3.getValue() *
products.getPlate().getPrice() + spProduct4.getValue() *
products.getLaptop().getPrice() + spProduct5.getValue() *
products.getBowl().getPrice() + spProduct6.getValue() *
products.getSweater().getPrice());

        if(products.getSubtotal() == 0) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Please count products to calculate
subtotal!");
            a.show();
            return;
        }

        products.setTax(products.getSubtotal() / 10);
        products.setTotal(products.getSubtotal() + products.getTax());

        lblSubtotal.setText("Subtotal: Rp" + products.getSubtotal());
        lblTax.setText("Tax (10%): Rp" + products.getTax());
        lblTotal.setText("Total: Rp" + products.getTotal());

        tfMoney.setDisable(false);

        btnCalculateChange.setVisible(true);
    }

    // When the staff clicks btnCalculateChange, this method will be
called.
    // The staff will get the change after the staff inputs the amount
of money the shopper pays.
    // If there is no input, a warning will appear that the staff must
input an amount of money the shopper pays.
```

```java
    // If the input contains other than numeric, a warning will appear
that the staff must input a valid amount of money which is only
numeric.
    // If the amount of money inputted is smaller than the total, a
warning will appear that the staff must input an amount of money
greater than the total.
    // btnCalculateChange must be clicked first in order to set
btnPrintReceipt visible.
    // The reason is because the change must be calculated first
before printing the receipt.
    public void calculateChange() {
        if(tfMoney.getText().isEmpty()) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Please input an amount of money!");
            a.show();
            return;
        } else if(!tfMoney.getText().matches("[0-9]+")) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Please input a valid amount of money!");
            a.show();
            return;
        } else if(Integer.parseInt(tfMoney.getText()) -
products.getTotal() < 0) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Please input a sufficient amount of
money!");
            a.show();
            return;
        }

        products.setMoney(Integer.parseInt(tfMoney.getText()));
        products.setChange(products.getMoney() - products.getTotal());

        lblChange.setText("Total: Rp" + products.getChange());

        btnPrintReceipt.setVisible(true);
    }

    // When the staff clicks btnPrintReceipt, this method will be
called.
    // The staff will get the complete information of the shopper's
transaction which will change the scene to sample4.fxml.
    public void printReceipt() throws Exception {
        products.gettShirt().setStock(products.gettShirt().getStock()
- spProduct.getValue());

products.getSmartphone().setStock(products.getSmartphone().getStock()
- spProduct2.getValue());
        products.getPlate().setStock(products.getPlate().getStock() -
spProduct3.getValue());
        products.getLaptop().setStock(products.getLaptop().getStock()
- spProduct4.getValue());
        products.getBowl().setStock(products.getBowl().getStock() -
spProduct5.getValue());

products.getSweater().setStock(products.getSweater().getStock() -
```

```
spProduct6.getValue());

        Stage primaryStage;
        Parent root;

        primaryStage = (Stage) btnPrintReceipt.getScene().getWindow();
        root =
FXMLLoader.load(getClass().getResource("sample4.fxml"));

        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }

    // When the scene of this controller (sample.fxml) is shown, this
method will be called automatically.
    // This method will display the staff's username, all products'
image, name, price and quantity which will be shown by all products'
spinners.
    @Override
    public void initialize(URL location, ResourceBundle resources) {

lblUsername.setText(personData.getSignedIn().getAccount().getUsername(
));
        try {
            fisProduct = new FileInputStream("tShirt.jpg");
            fisProduct2 = new FileInputStream("smartphone.jpg");
            fisProduct3 = new FileInputStream("plate.jpg");
            fisProduct4 = new FileInputStream("laptop.jpg");
            fisProduct5 = new FileInputStream("bowl.jpg");
            fisProduct6 = new FileInputStream("sweater.jpg");
        } catch (IOException e) {
            e.printStackTrace();
        }
        iProduct = new Image(fisProduct);
        iProduct2 = new Image(fisProduct2);
        iProduct3 = new Image(fisProduct3);
        iProduct4 = new Image(fisProduct4);
        iProduct5 = new Image(fisProduct5);
        iProduct6 = new Image(fisProduct6);
        ivProduct.setImage(iProduct);
        ivProduct2.setImage(iProduct2);
        ivProduct3.setImage(iProduct3);
        ivProduct4.setImage(iProduct4);
        ivProduct5.setImage(iProduct5);
        ivProduct6.setImage(iProduct6);
        lblProduct.setText(products.gettShirt().getName() + " @ Rp" +
products.gettShirt().getPrice());
        lblProduct2.setText(products.getSmartphone().getName() + " @
Rp" + products.getSmartphone().getPrice());
        lblProduct3.setText(products.getPlate().getName() + " @ Rp" +
products.getPlate().getPrice());
        lblProduct4.setText(products.getLaptop().getName() + " @ Rp" +
products.getLaptop().getPrice());
        lblProduct5.setText(products.getBowl().getName() + " @ Rp" +
products.getBowl().getPrice());
        lblProduct6.setText(products.getSweater().getName() + " @ Rp"
+ products.getSweater().getPrice());
        spProduct.setValueFactory(spvfProduct);
        spProduct2.setValueFactory(spvfProduct2);
```

```
        spProduct3.setValueFactory(spvfProduct3);
        spProduct4.setValueFactory(spvfProduct4);
        spProduct5.setValueFactory(spvfProduct5);
        spProduct6.setValueFactory(spvfProduct6);
    }
}
```

Next, we need to set up the sign in scene (sample2.fxml). The application requires the staff's sign in from an existing account. To control this scene, a controller class named Controller2 is made. This class will control the username and password input which is necessary for signing in to this application. If the staff does not have an existing account, he/she needs to sign up for a new account.

```java
package sample;

import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.io.File;

// Define a class named Controller2 which takes the PersonData class
as its own instance attribute.
// This class is necessary for controlling the sign in scene
(sample2.fxml) controls.
// The class contains all necessary controls and instance attributes
for the sign in scene (sample2.fxml).
// tfUsername is for inputting the staff's username.
// pfPassword is for inputting the staff's password.
// btnSignIn is for signing in to the application.
// btnSignUp is for signing up a new account for a new staff.
// personData is for getting signedUp and setting signedIn to the
staff's account when the staff signs in to the application.
// invalidSignIn is for checking whether username or password input is
invalid.
// This class has 2 custom methods for each button.
// signIn is for signing in to the application when the staff clicks
btnSignIn.
// signIn must throw Exception in order to change scene to the product
counting and price calculation scene (sample.fxml).
// signUp is for signing up a new account for a new staff when the
staff clicks btnSignUp.
// signUp must throw Exception in order to change scene to the sign up
scene (sample3.fxml).
public class Controller2 implements ButtonActions {
    public TextField tfUsername;
    public PasswordField pfPassword;
    public Button btnSignIn, btnSignUp;

    public PersonData personData;

    public boolean invalidSignIn;

    // When the staff clicks btnSignIn, this method will be called.
    // The staff will be signed in to the application after signing in
```

```java
using the staff's existing account.
    // If one of the text fields are empty, a warning will appear that
all text fields must be filled.
    // If there is no existing account, a warning will appear that the
staff must sign up for a new account.
    // If there are text fields which has invalid input, a warning
will appear depending on which fields are invalid.
    @Override
    public void signIn() throws Exception {
        invalidSignIn = false;

        File file = new File("signedUpAccounts.txt");

        if(file.length() != 0) {
            personData.readSignedUp();
        }

        String username = tfUsername.getText();
        String password = pfPassword.getText();

        if(username.isEmpty() || password.isEmpty()) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("All data must be filled!");
            a.show();
            invalidSignIn = true;
        } else if(personData.getSignedUp().isEmpty()) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("A new account must be signed up!");
            a.show();
            invalidSignIn = true;
        } else {
            for(Person person : personData.getSignedUp()) {
                if
(!person.getAccount().getUsername().equals(username) &&
!person.getAccount().getPassword().equals(password)) {
                    Alert a = new Alert(Alert.AlertType.WARNING);
                    a.setTitle("Warning");
                    a.setContentText("Invalid username and
password!");
                    a.show();
                    invalidSignIn = true;
                    break;
                } else
if(!person.getAccount().getUsername().equals(username)) {
                    Alert a = new Alert(Alert.AlertType.WARNING);
                    a.setTitle("Warning");
                    a.setContentText("Invalid username!");
                    a.show();
                    invalidSignIn = true;
                    break;
                } else
if(!person.getAccount().getPassword().equals(password)) {
                    Alert a = new Alert(Alert.AlertType.WARNING);
                    a.setTitle("Warning");
                    a.setContentText("Invalid password!");
                    a.show();
                    invalidSignIn = true;
```

```java
                        break;
                    }
                }
            }

            if(invalidSignIn) {
                return;
            }

            for(Person person : personData.getSignedUp()) {
                if(person.getAccount().getUsername().equals(username) &&
person.getAccount().getPassword().equals(password)) {
                    personData.setSignedIn(person);
                }
            }

            Stage primaryStage;
            Parent root;

            primaryStage = (Stage) btnSignIn.getScene().getWindow();
            root = FXMLLoader.load(getClass().getResource("sample.fxml"));

            primaryStage.setScene(new Scene(root));
            primaryStage.show();
    }

    // When the staff clicks btnSignUp, this method will be called.
    // The staff can sign up for a new account after changing scene to
the sign up scene (sample3.fxml).
    @Override
    public void signUp() throws Exception {
        Stage primaryStage;
        Parent root;

        primaryStage = (Stage) btnSignUp.getScene().getWindow();
        root =
FXMLLoader.load(getClass().getResource("sample3.fxml"));

        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }
}
```

We also need to configure the sign up scene (sample.fxml). When a new staff works, he/she does not have an existing account. This is the reason why we need a controller class named Controller3. This class controls the new staff's first and last name, new username, email and password so he/she will have an account for signing in to the application.

```java
package sample;

import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
```

```java
import java.io.File;

// Define a class named Controller3 which takes the PersonData class
as its own instance attribute.
// This class is necessary for controlling the sign up scene
(sample3.fxml) controls.
// The class contains all necessary controls and instance attributes
for the sign up scene (sample3.fxml).
// tfFirstName is for inputting the new staff's first name.
// tfLastName is for inputting the new staff's last name.
// tfUsername is for inputting the new staff's username.
// tfEmail is for inputting the new staff's email.
// pfPassword is for inputting the new staff's password.
// btnSignUp is for signing up a new account for a new staff.
// btnSignIn is for signing in to the application.
// personData is for getting signedUp and setting signedIn to the
staff's account when the staff signs in to the application.
// invalidSignUp is for checking whether first name, last name,
username, email or password input is invalid.
// This class has 2 custom methods for each button.
// signUp is for signing up a new account for a new staff when the
staff clicks btnSignUp.
// signIn is for signing in to the application when the staff clicks
btnSignIn.
// signIn must throw Exception in order to change scene to the sign in
scene (sample2.fxml).
public class Controller3 implements ButtonActions {
    public TextField tfFirstName, tfLastName, tfUsername, tfEmail;
    public PasswordField pfPassword;
    public Button btnSignUp, btnSignIn;

    public PersonData personData;

    public boolean invalidSignUp;

    // When the staff clicks btnSignUp, this method will be called.
    // A new staff have a chance to sign up for a new account for
logging in to the application.
    // If one of the text fields are empty, a warning will appear that
all text fields must be filled.
    // If there is no email domain or email name, a warning will
appear that the inputted email is invalid.
    // If the inputted password contains less than 8 characters, a
warning will appear that the password must contain at least 8
characters.
    // If the inputted password doesn't contain at least a lowercase
letter, an uppercase letter and a digit, a warning will appear that
the inputted password must contain at least a lowercase letter, an
uppercase letter and a digit.
    // If the inputted username is the same as one of the existing
accounts' username, a warning will appear that the inputted username
has been taken.
    @Override
    public void signUp() {
        invalidSignUp = false;

        File file = new File("signedUpAccounts.txt");

        if(file.length() != 0) {
```

```java
            personData.readSignedUp();
        }

        String firstName = tfFirstName.getText();
        String lastName = tfLastName.getText();
        String username = tfUsername.getText();
        String email = tfEmail.getText();
        String password = pfPassword.getText();

        if(firstName.isEmpty() || lastName.isEmpty() ||
username.isEmpty() || email.isEmpty() || password.isEmpty()) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("All data must be filled!");
            a.show();
            invalidSignUp = true;
        } else if(!email.endsWith("@gmail.com") &&
!email.endsWith("@yahoo.com") || email.length() <= 10) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Invalid email!");
            a.show();
            invalidSignUp = true;
        } else if(password.length() < 8) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Password must contain at least 8
characters!");
            a.show();
            invalidSignUp = true;
        } else if(!password.matches("^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-
9]).+$")) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Password must contain at least a
lowercase letter, an uppercase letter and a digit!");
            a.show();
            invalidSignUp = true;
        } else {
            for (Person person : personData.getSignedUp()) {
                if
(person.getAccount().getUsername().equals(username)) {
                    Alert a = new Alert(Alert.AlertType.WARNING);
                    a.setTitle("Warning");
                    a.setContentText("Username has been taken!");
                    a.show();
                    invalidSignUp = true;
                    break;
                }
            }
        }

        if(invalidSignUp) {
            return;
        }

        personData.addSignedUp(firstName, lastName, username, email,
password);
```

```
        personData.writeSignedUp();

        Alert a = new Alert(Alert.AlertType.INFORMATION);
        a.setTitle("Information");
        a.setContentText("Sign Up Successful!");
        a.show();
    }

    // When the staff clicks btnSignIn, this method will be called.
    // The staff can log in to the application with an existing
account after changing scene to the sign in scene (sample2.fxml).
    @Override
    public void signIn() throws Exception {
        Stage primaryStage;
        Parent root;

        primaryStage = (Stage) btnSignIn.getScene().getWindow();
        root =
FXMLLoader.load(getClass().getResource("sample2.fxml"));

        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }
}
```

After three controllers above are set, all that is left to do is setting up the controller for showing the receipt scene (sample4.fxml). The controller class for this scene is Controller4. This class controls all transaction information for display in the receipt so there is an evidence of the transaction.

```
package sample;

import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.stage.Stage;

import java.net.URL;
import java.util.ResourceBundle;

// Define a class named Controller4 which implements the Initializable
interface and takes the Products class as its own instance attribute.
// This class is necessary for controlling the receipt scene
(sample4.fxml) controls.
// The interface must be implemented in order to override the
initialize method for setting up necessary controls in the receipt
scene (sample4.fxml) on the start of the scene.
// The class contains all necessary controls and instance attributes
for the receipt scene (sample4.fxml).
// Labels are for displaying the products' name, quantity and price,
subtotal, tax, total, the shopper's money and change.
// btnBack is for going back to calculate new transaction.
// products is for getting the products' transaction for receipt
printing.
// This class has a custom method for btnBack.
// back is for going back to calculate new transaction when the staff
clicks btnBack.
```

```java
// back must throw Exception in order to change scene to the product
counting and price calculation scene (sample.fxml).
public class Controller4 implements Initializable {
    public Label lblProduct, lblProduct2, lblProduct3, lblProduct4,
lblProduct5, lblProduct6, lblProductPrice, lblProductPrice2,
lblProductPrice3, lblProductPrice4, lblProductPrice5,
lblProductPrice6, lblSubtotal, lblTax, lblTotal, lblMoney, lblChange;
    public Button btnBack;

    public Products products;

    // When the staff clicks btnBack, this method will be called.
    // The staff can go back to calculate further transactions after
changing scene to the product counting and price calculation scene
(sample.fxml).
    public void back() throws Exception {
        Stage primaryStage;
        Parent root;

        primaryStage = (Stage) btnBack.getScene().getWindow();
        root = FXMLLoader.Load(getClass().getResource("sample.fxml"));

        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }

    // When the scene of this controller (sample4.fxml) is shown, this
method will be called automatically.
    // This method will display all products' name, quantity and
price, subtotal, tax, total, the shopper's money paid for the
transaction and change for the receipt.
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        lblProduct.setText(products.gettShirt().getName() + " x " +
products.gettShirt().getQuantity());
        lblProduct2.setText(products.getSmartphone().getName() + " x "
+ products.getSmartphone().getQuantity());
        lblProduct3.setText(products.getPlate().getName() + " x " +
products.getPlate().getQuantity());
        lblProduct4.setText(products.getLaptop().getName() + " x " +
products.getLaptop().getQuantity());
        lblProduct5.setText(products.getBowl().getName() + " x " +
products.getBowl().getQuantity());
        lblProduct6.setText(products.getSweater().getName() + " x " +
products.getSweater().getQuantity());
        lblProductPrice.setText("Rp" +
(products.gettShirt().getQuantity() *
products.gettShirt().getPrice()));
        lblProductPrice2.setText("Rp" +
(products.getSmartphone().getQuantity() *
products.getSmartphone().getPrice()));
        lblProductPrice3.setText("Rp" +
(products.getPlate().getQuantity() * products.getPlate().getPrice()));
        lblProductPrice4.setText("Rp" +
(products.getLaptop().getQuantity() *
products.getLaptop().getPrice()));
        lblProductPrice5.setText("Rp" +
(products.getBowl().getQuantity() * products.getBowl().getPrice()));
        lblProductPrice6.setText("Rp" +
```

```
(products.getSweater().getQuantity() *
products.getSweater().getPrice()));
        lblSubtotal.setText("Rp" + products.getSubtotal());
        lblTax.setText("Rp" + products.getTax());
        lblTotal.setText("Rp" + products.getTotal());
        lblMoney.setText("Rp" + products.getMoney());
        lblChange.setText("Rp" + products.getChange());
    }
}
```

All controllers for each scene have been configured. Now, what is needed is just a class for running the application. Finally, a class named Main is created so that the application can be run and launched.

```
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

// Define a class name Main which inherits the Application abstract
class.
// This class is necessary to run the application.
// The application shows the sign in scene (sample2.fxml) as the first
scene on stage since signing in is necessary to use the application.
public class Main extends Application {

    // This method will be called automatically when the staff runs
the application.
    // The application shows the sign in scene (sample2.fxml) as the
first scene on stage since signing in is necessary to use the
application.
    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("sample2.fxml"));
        primaryStage.setTitle("Cashie");
        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }

    // This method will be called automatically when the staff runs
the application.
    // The application will launch if the staff runs the application.
    public static void main(String[] args) {
        Launch(args);
    }
}
```

3. **Evidence of Working Program**
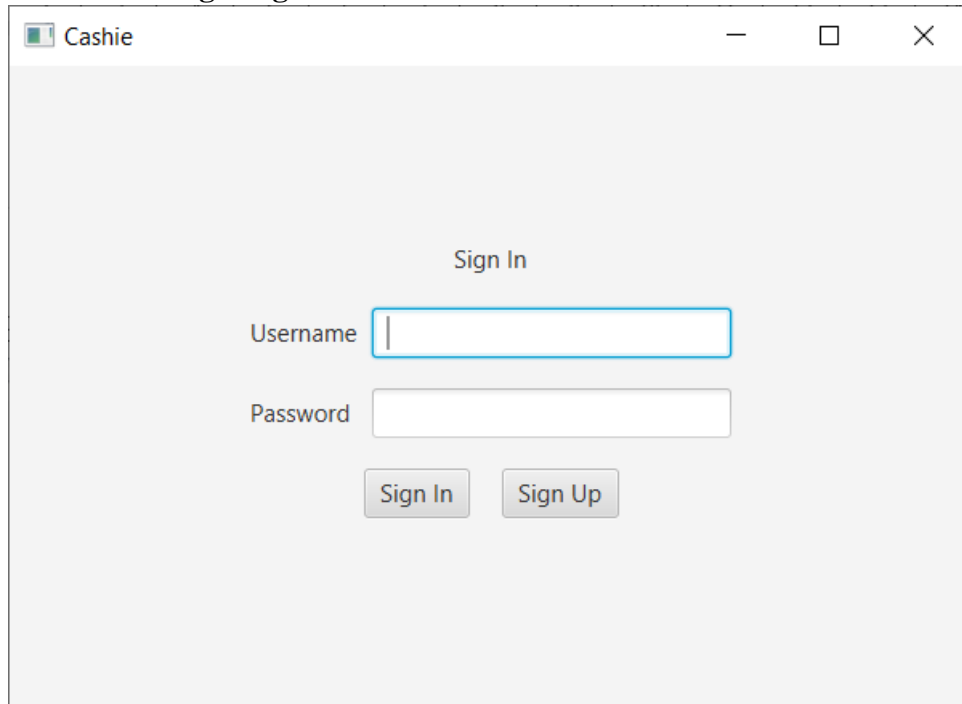
Screenshot of the sign in scene.
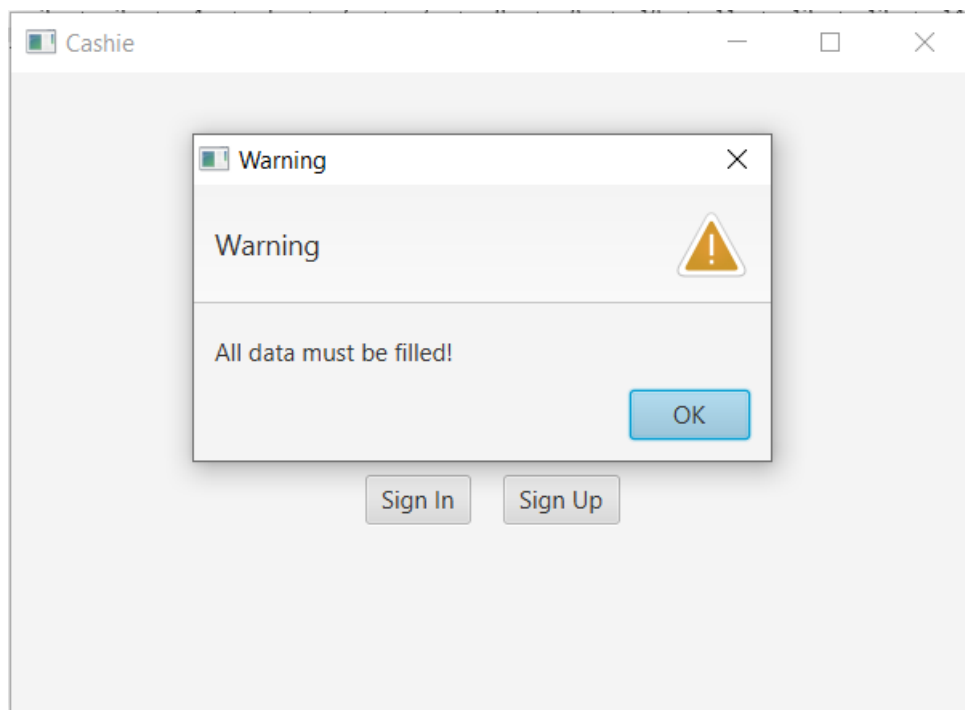
Screenshot of warning that all data must be filled to sign in.
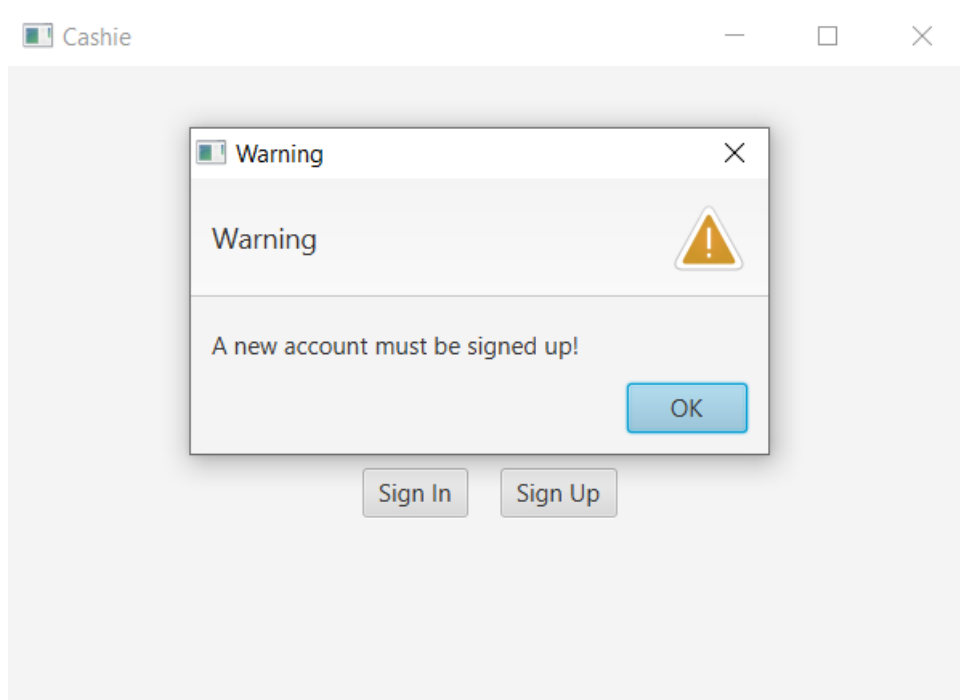
Screenshot of warning that there is no existing account.
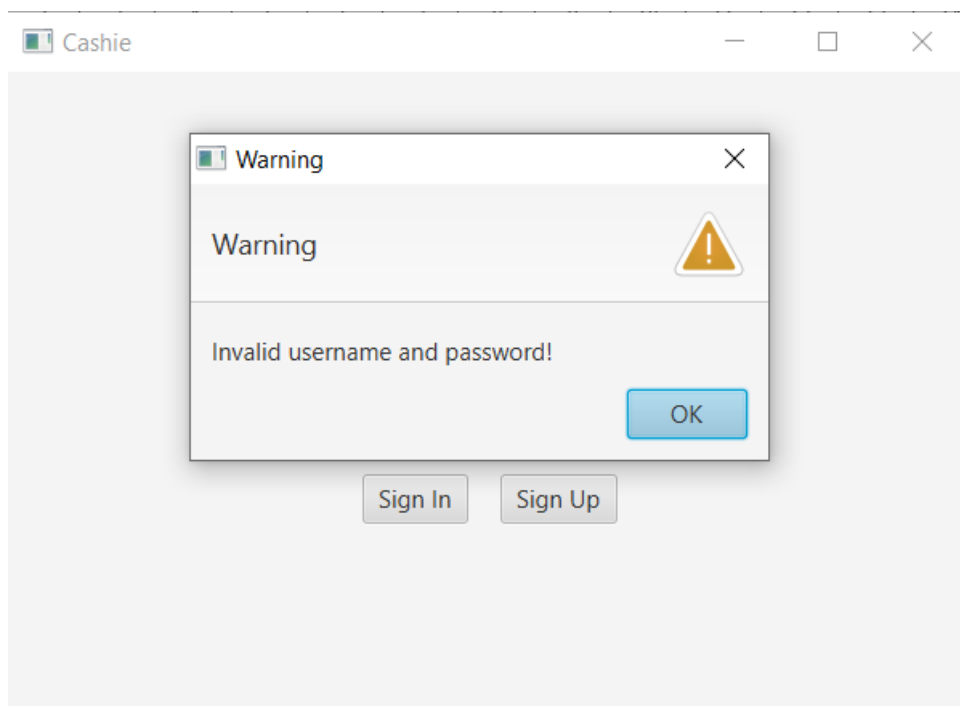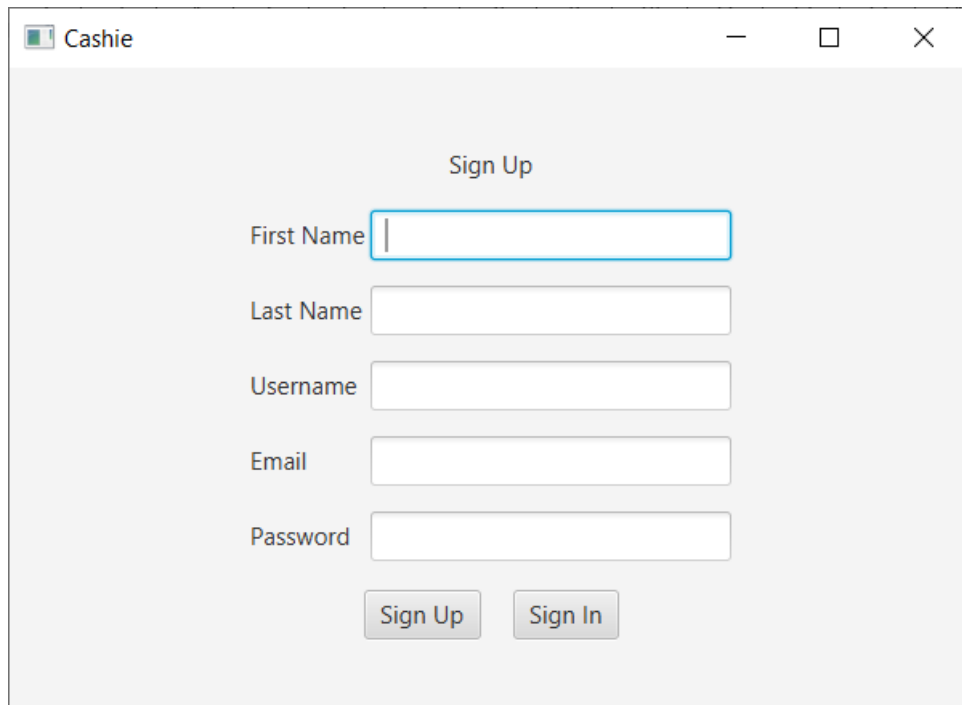
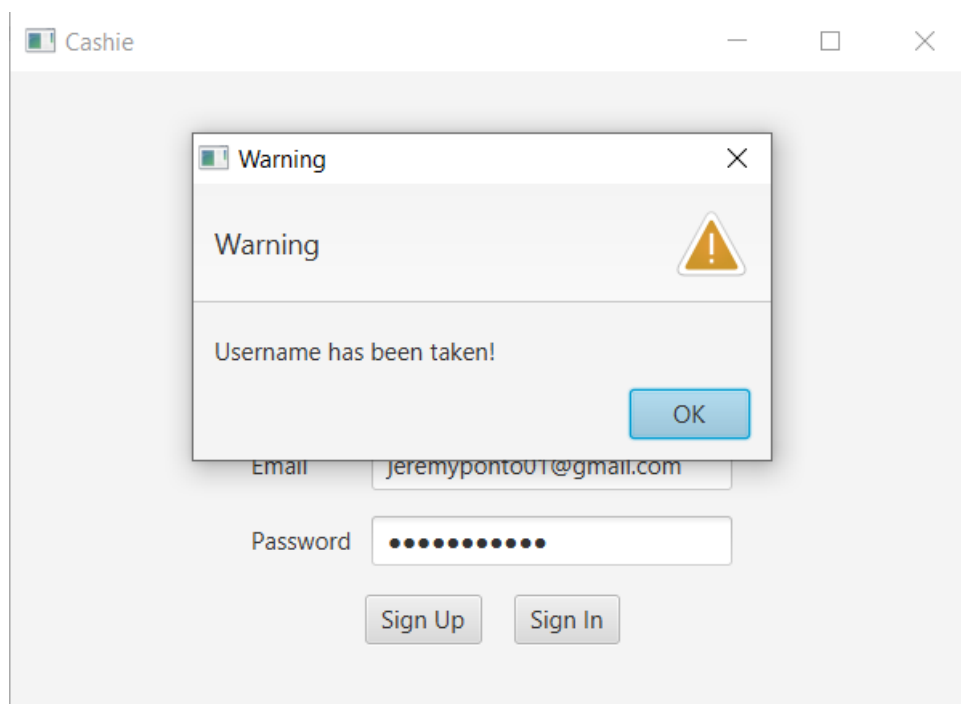Screenshot of warning that the staff has inputted invalid username and password.
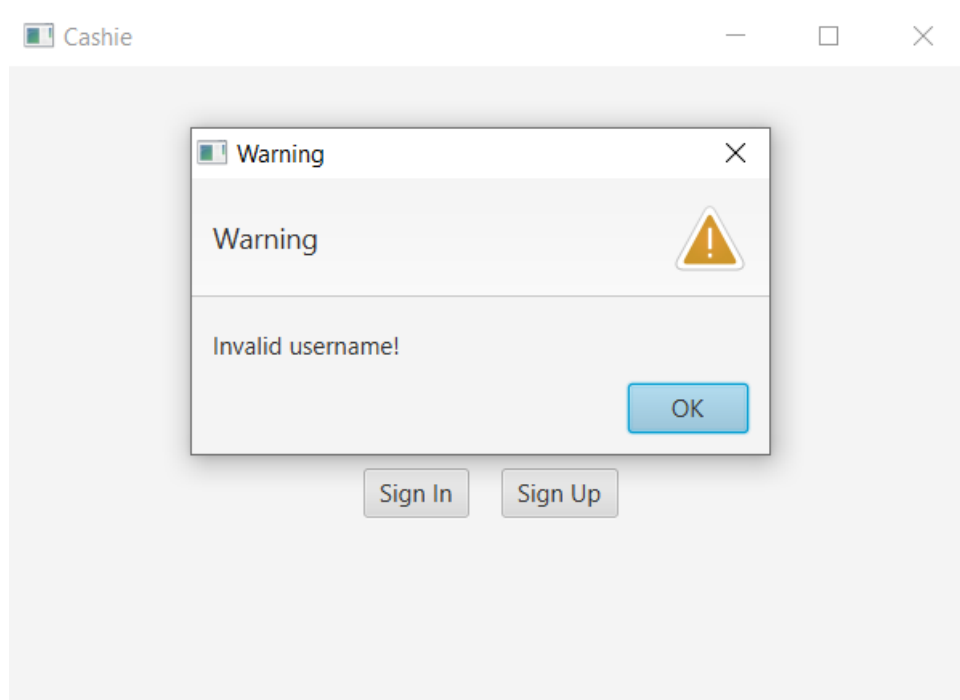
Screenshot of warning that the staff has inputted an invalid username.
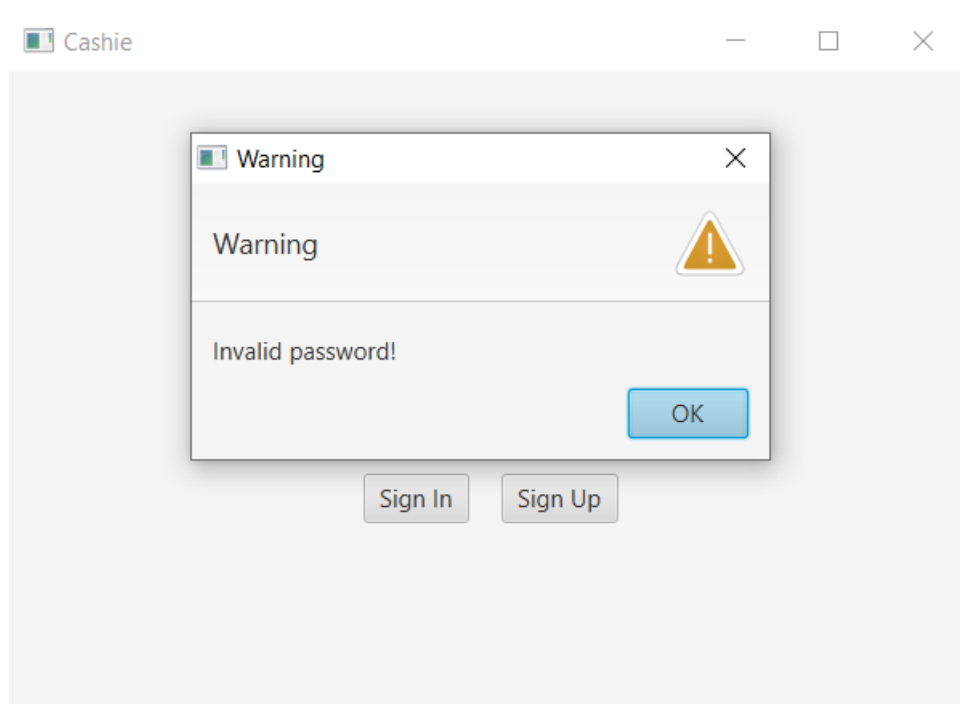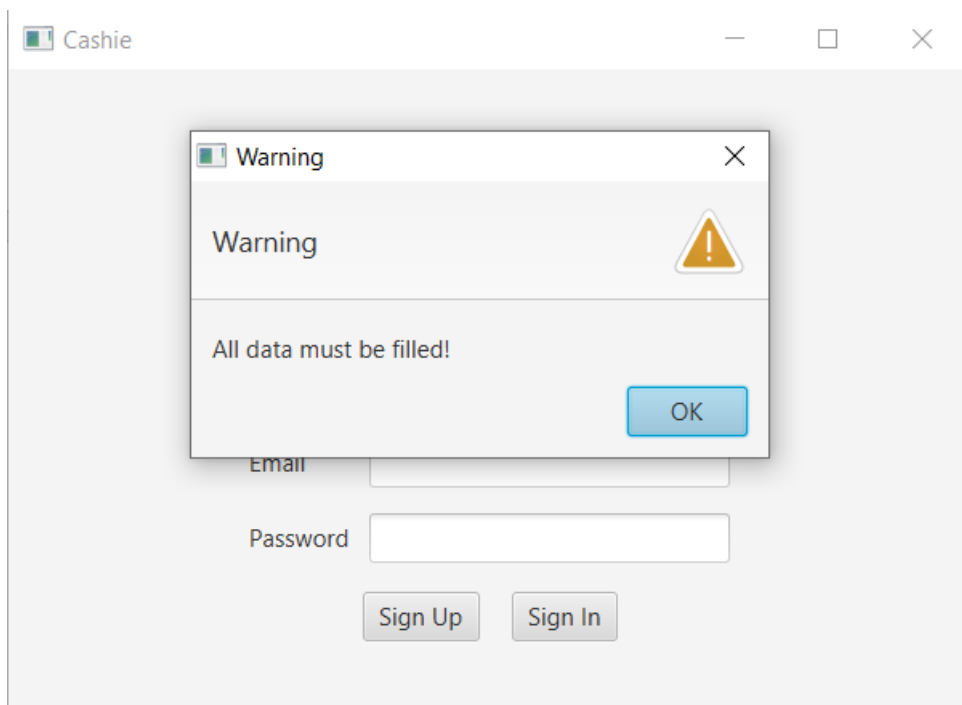
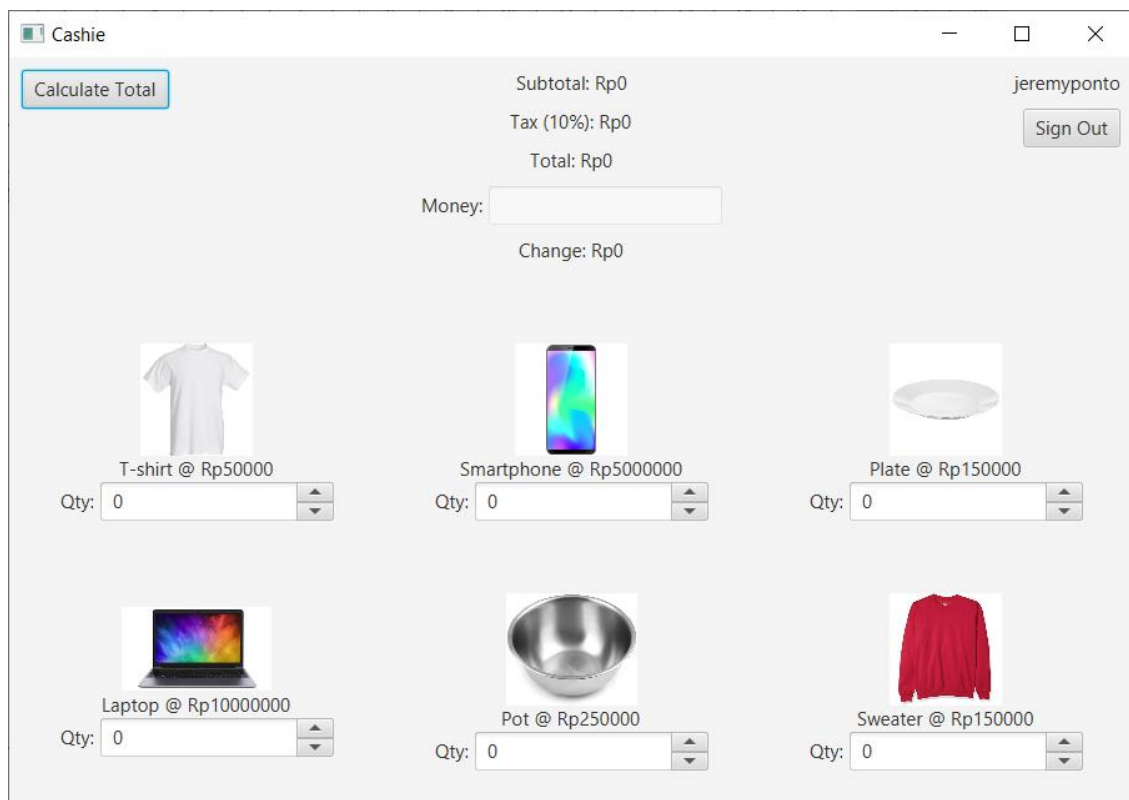Screenshot of warning that the staff has inputted an invalid password.

Screenshot of the sign up scene.

Screenshot of warning that all data must be filled to sign up.
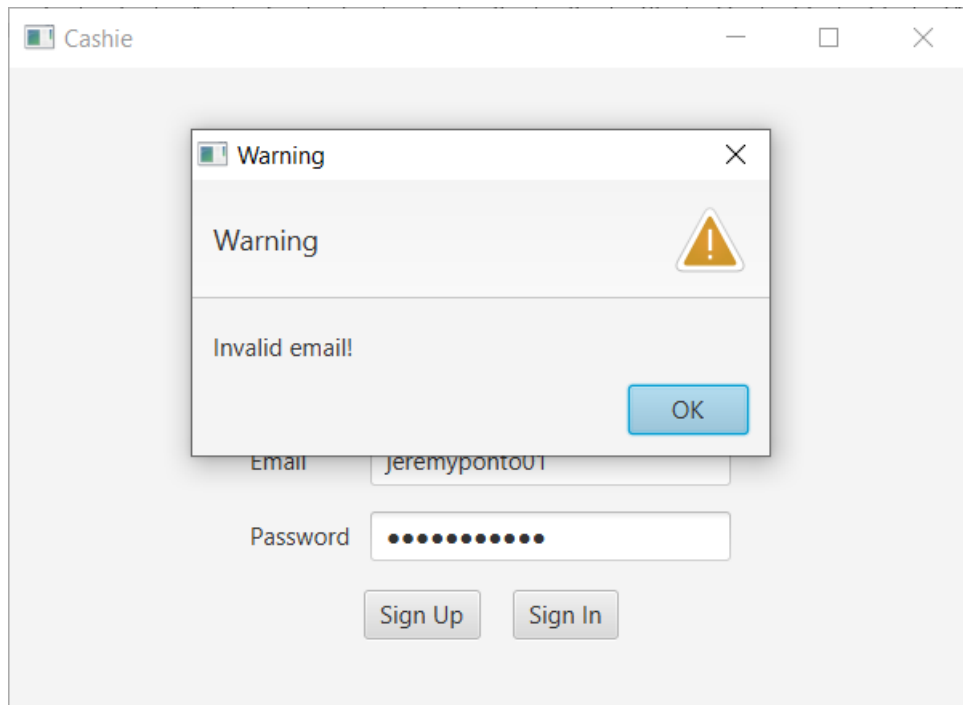
Screenshot of warning that the staff has inputted an invalid email.
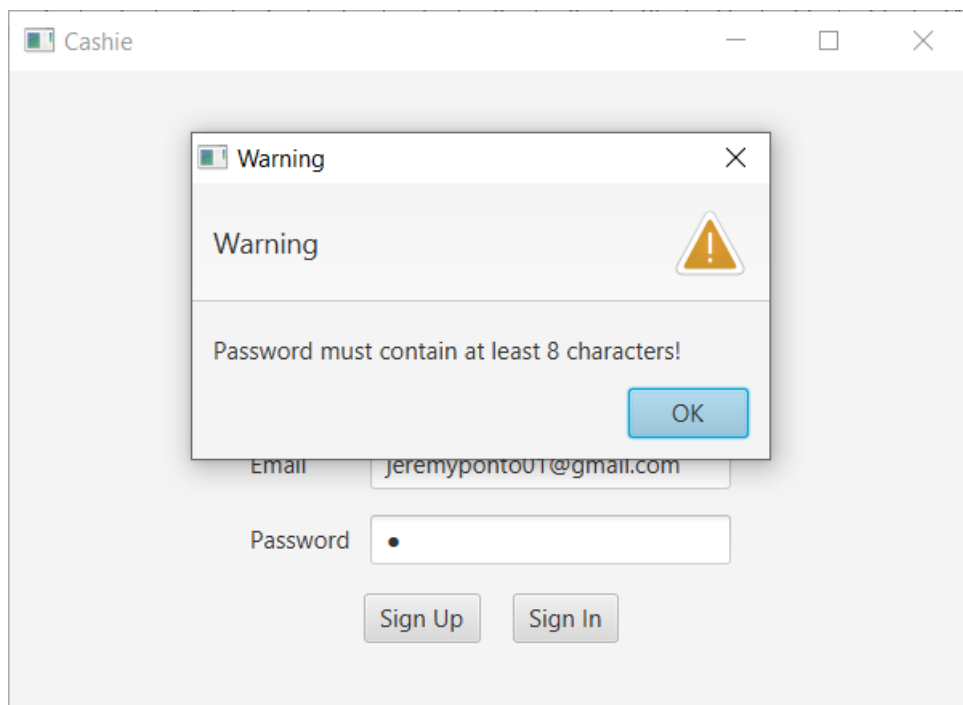
Screenshot of warning that the staff has inputted a password with less than 8 characters.
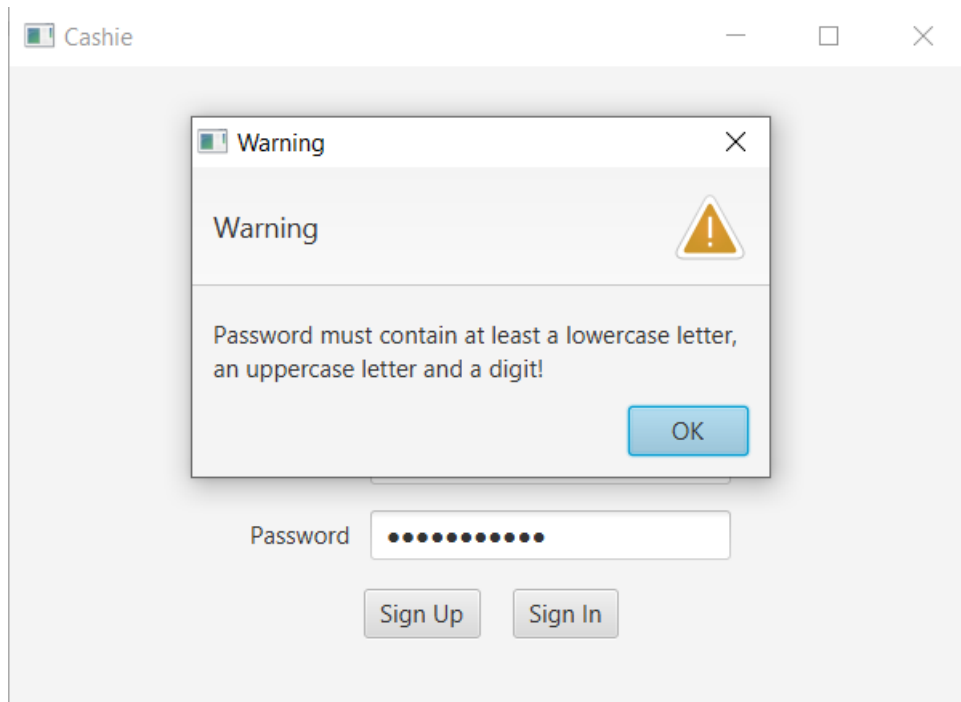
Screenshot of warning that the staff has inputted a password with no lowercase letters, uppercase letters or digit.

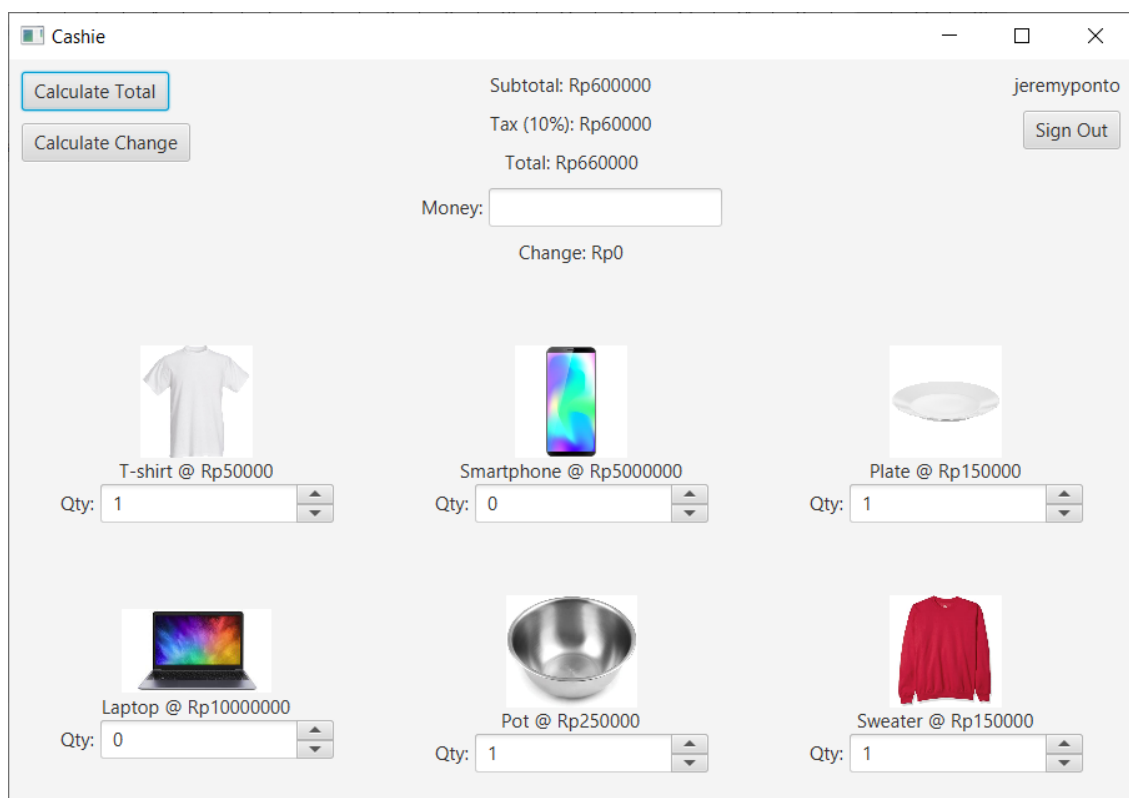Screenshot of warning that the staff has inputted the same username as one of the existing accounts' username.
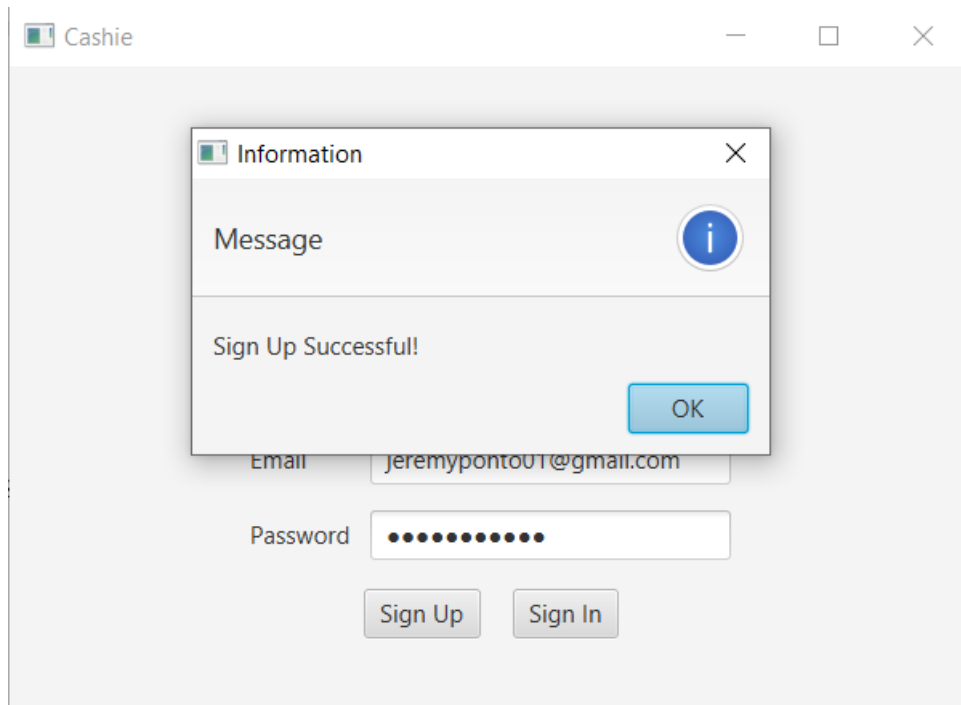
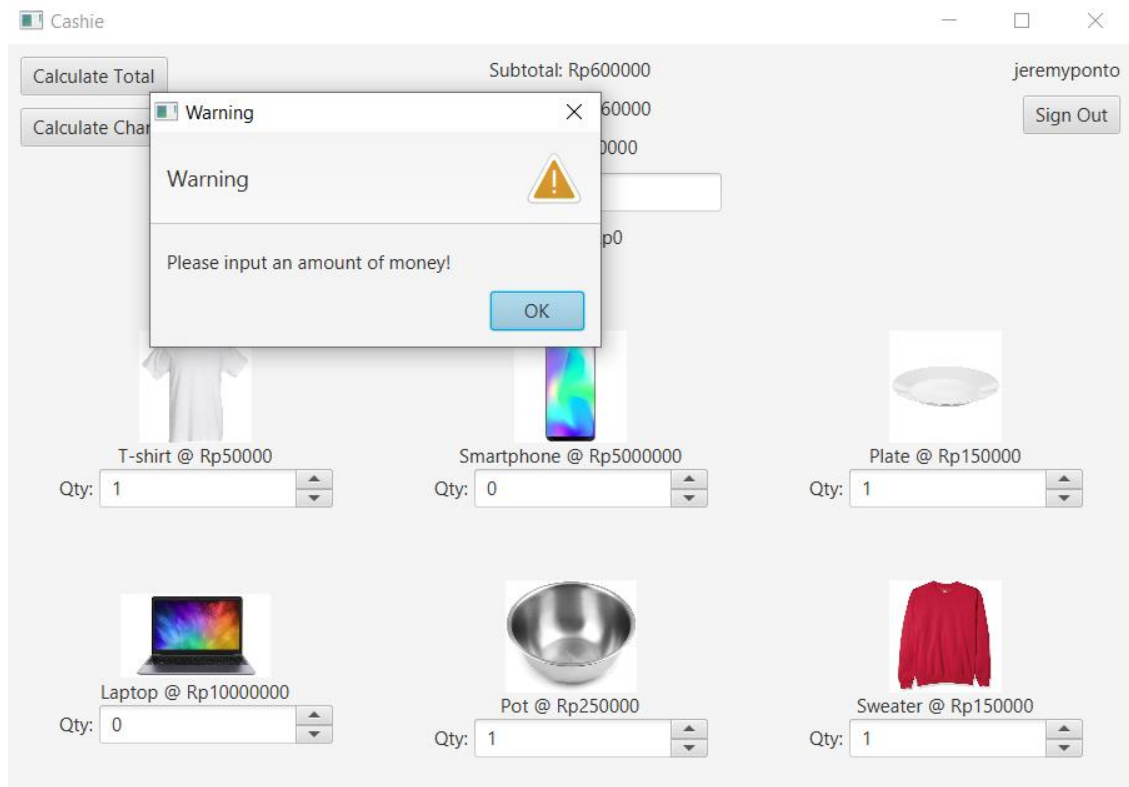Screenshot of information that the staff has signed up a new account successfully.

Screenshot of the price calculation scene after a successful sign in.
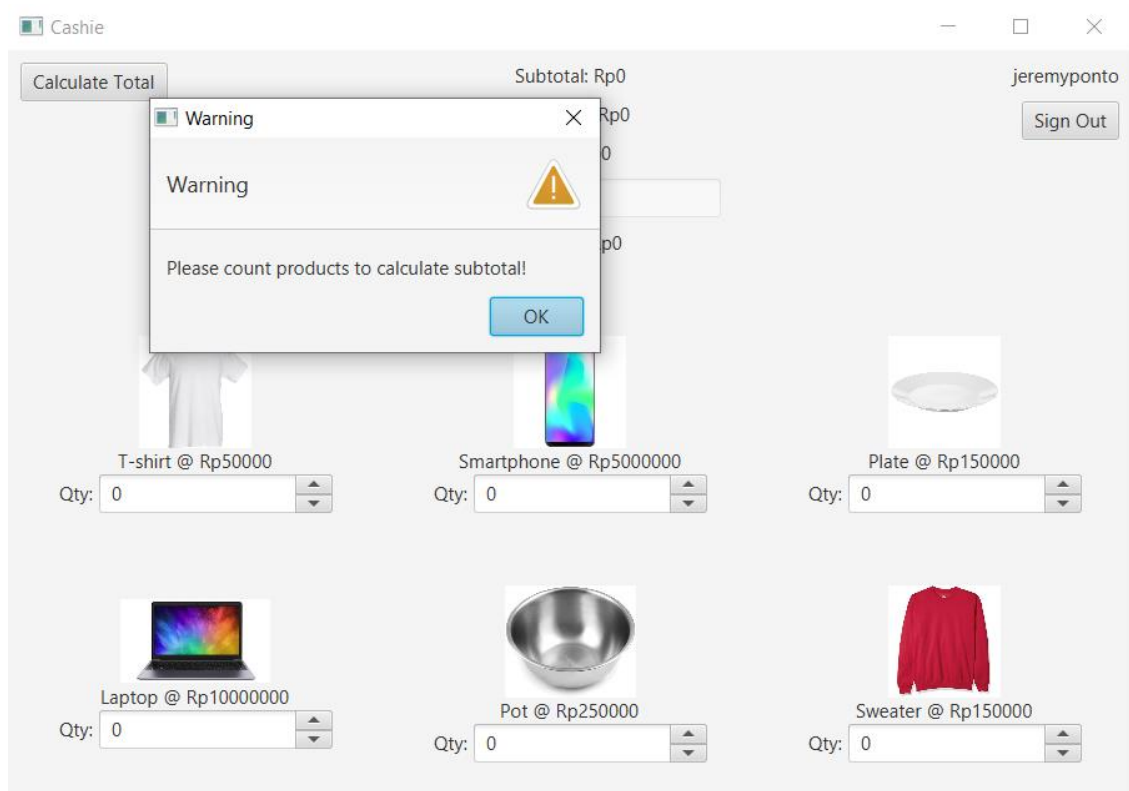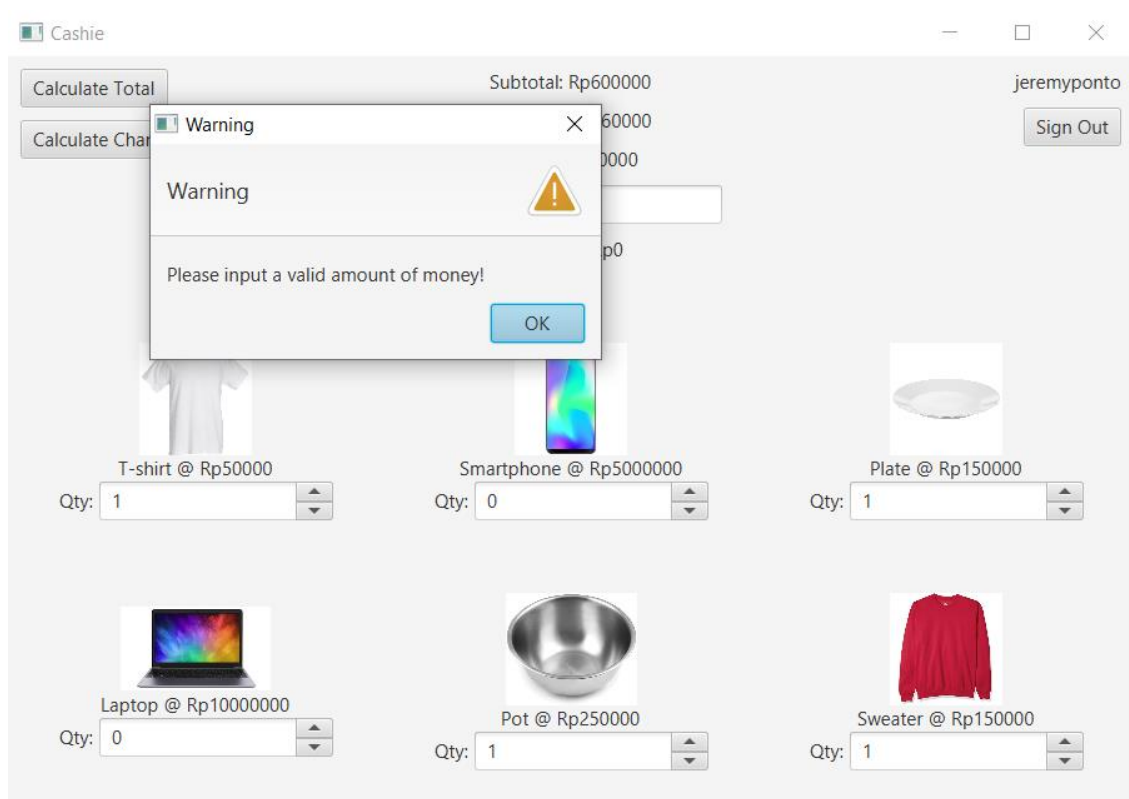
Screenshot of warning that the staff has not counted products that the shopper bought.
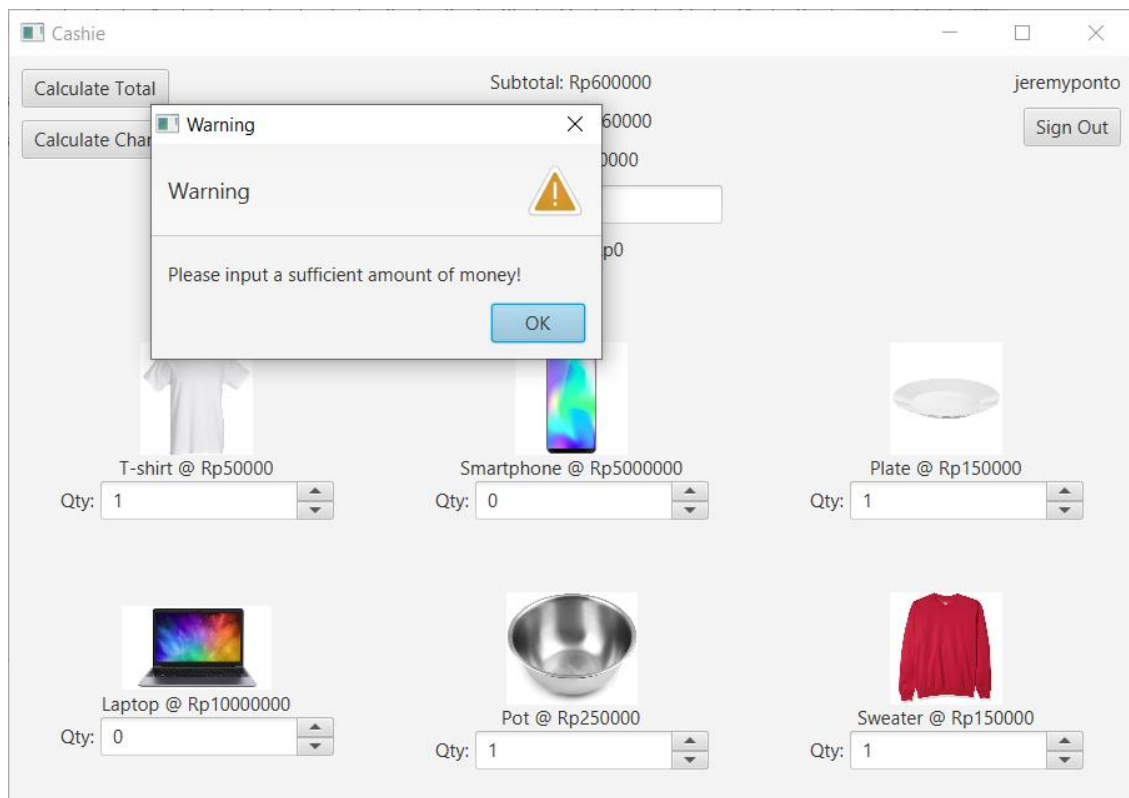

Screenshot of the subtotal, tax and total price after a successful total calculation.
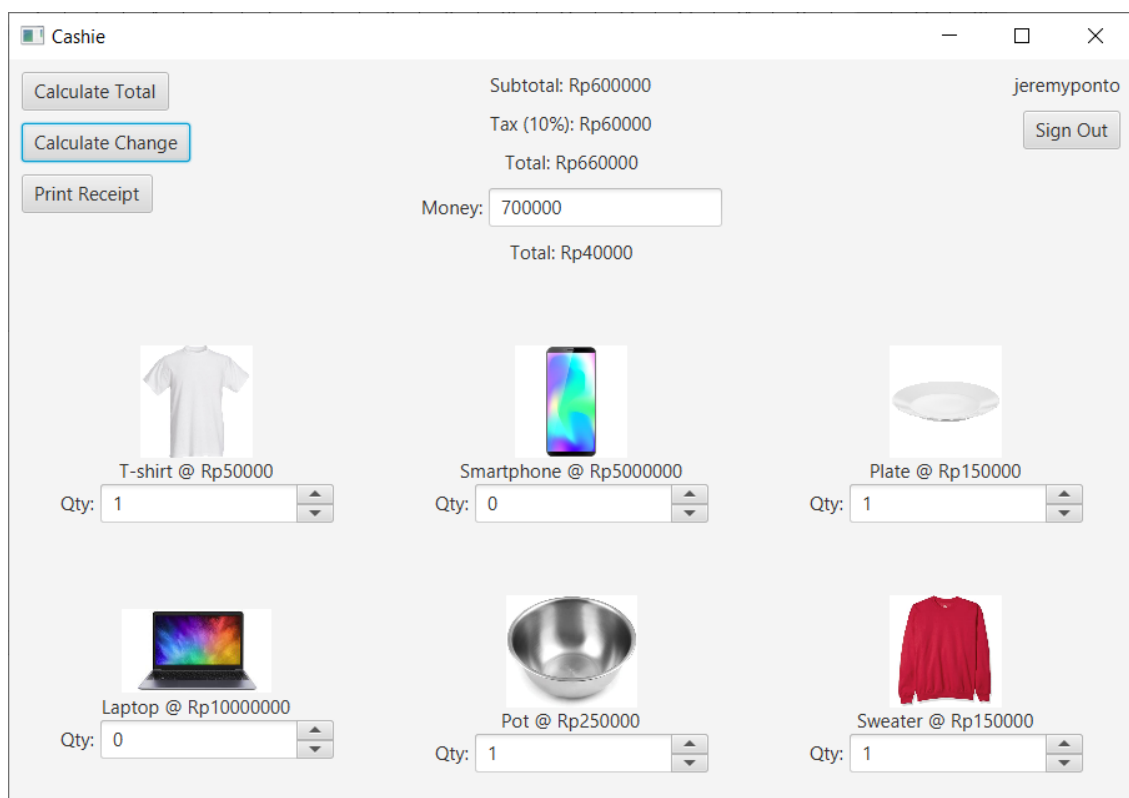
Screenshot of warning that the staff has not inputted the amount of money the shopper pays.

Screenshot of warning that the staff has inputted an invalid amount of money (other than digits).

Screenshot of warning that the staff has inputted an amount of money less than the total price.

Screenshot of the change after inputting the amount of money the shopper pays and a successful change calculation.

Cashie		□	×
	Receipt		
T-shirt x 1		Rp50000	
Smartphone x 0		Rp0	
Plate x 1		Rp150000	
Laptop x 0		Rp0	
Pot x 1		Rp250000	
Sweater x 1		Rp150000	
Subtotal		Rp600000	
Tax (10%)		Rp60000	
Total		Rp660000	
Money		Rp700000	
Change		Rp40000	
	Back		

Screenshot of the receipt shown after a complete transaction information with the shopper.

REFERENCES

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Alert.AlertType.html

https://gist.github.com/pethaniakshay/302072fda98098a24ce382a361bdf477

https://stackoverflow.com/questions/10751271/accessing-fxml-controller-class

https://stackoverflow.com/questions/16111496/java-how-can-i-write-my-arraylist-to-a-file-and-read-load-that-file-to-the

https://stackoverflow.com/questions/7190618/most-efficient-way-to-check-if-a-file-is-empty-in-java-on-windows/7190664

https://medium.com/@joshwickham/creating-an-integer-spinner-in-javafx-f8fda8d12ae5

https://www.tutorialspoint.com/javafx/javafx_images.htm

https://stackoverflow.com/questions/15111420/how-to-check-if-a-string-contains-only-digits-in-java/15111450

https://stackoverflow.com/questions/40336374/how-do-i-check-if-a-java-string-contains-at-least-one-capital-letter-lowercase