Clothing Store Point of Sale System
Test Plan

Prepared by: Katherine Nemits, Huu-Khoa Nguyen, Alan Shami

March 24, 2023

System Description

This mobile app works across multi-platforms (iOS and Android) and is intended for internal use only by company employees. The app is designed for a clothing store with multiple locations across the city and has two levels of users, employees and managers. Employees can make sales and issue refunds, taking into consideration sales tax in both cases. Items can be identified by scanning the barcode using the device camera as a scanner. The app can also accept different payment methods using an external payment processor. Refunds will be cash only. After each transaction (whether a purchase or a sale transaction), the store's inventory should be updated. All store chains' inventory data will be stored and backed up on a public cloud provider using a relational database. This will enable the app to allow employees to look up items by ID, item name, or date added to the inventory. Another relational database will record transaction history and sales numbers but will be accessible to managers only.

UML Class Diagram

Origional Description:

The main classes for the system include Item, Inventory, Transaction History, Employee, Admin, and Camera.

Item:

This class holds all the variables relating to aspects of the product being sold in the store including name, ID number, price, size, and color. It also holds methods to return those individual values relating to the Item. This class is used by the Inventory, Employee, and Transaction History classes.

Inventory:

Inventory adds or removes items from the inventory of products and has the attributes “date added” and “quantity”. It uses the Item class and is updated by the Employee class.

Transaction History:

Transaction History has operations to update the transaction history and allows the administrator to search for a particular transaction using the sale number or date. Transaction History is only accessible by the Admin class

Employee:

The attributes of the Employee class include tax and quantity sold. The operations of the Employee class cover the aspects of making a sale, updating the inventory and transaction history, and making refunds at the store. It utilizes the Item class and the Camera class, and it regularly updates the Inventory.

Admin:

The Admin class is the only class with the ability to view the transaction history. It contains the attribute of the sale numbers to find a transaction.

Camera:

The Camera class connects the system to the camera of the device which it uses to scan barcodes on items to retrieve the ID number of the item. It is used by the Employee class when making transactions.

Updates:

Employee class:

- (int, int, int) date was added as a field
- quantitySold, tax, and date were added as parameters to makeSale()
- quantitySold was added as a parameter to refundItem()
- quantitySold and tax were added as parameters to calculatePrice()

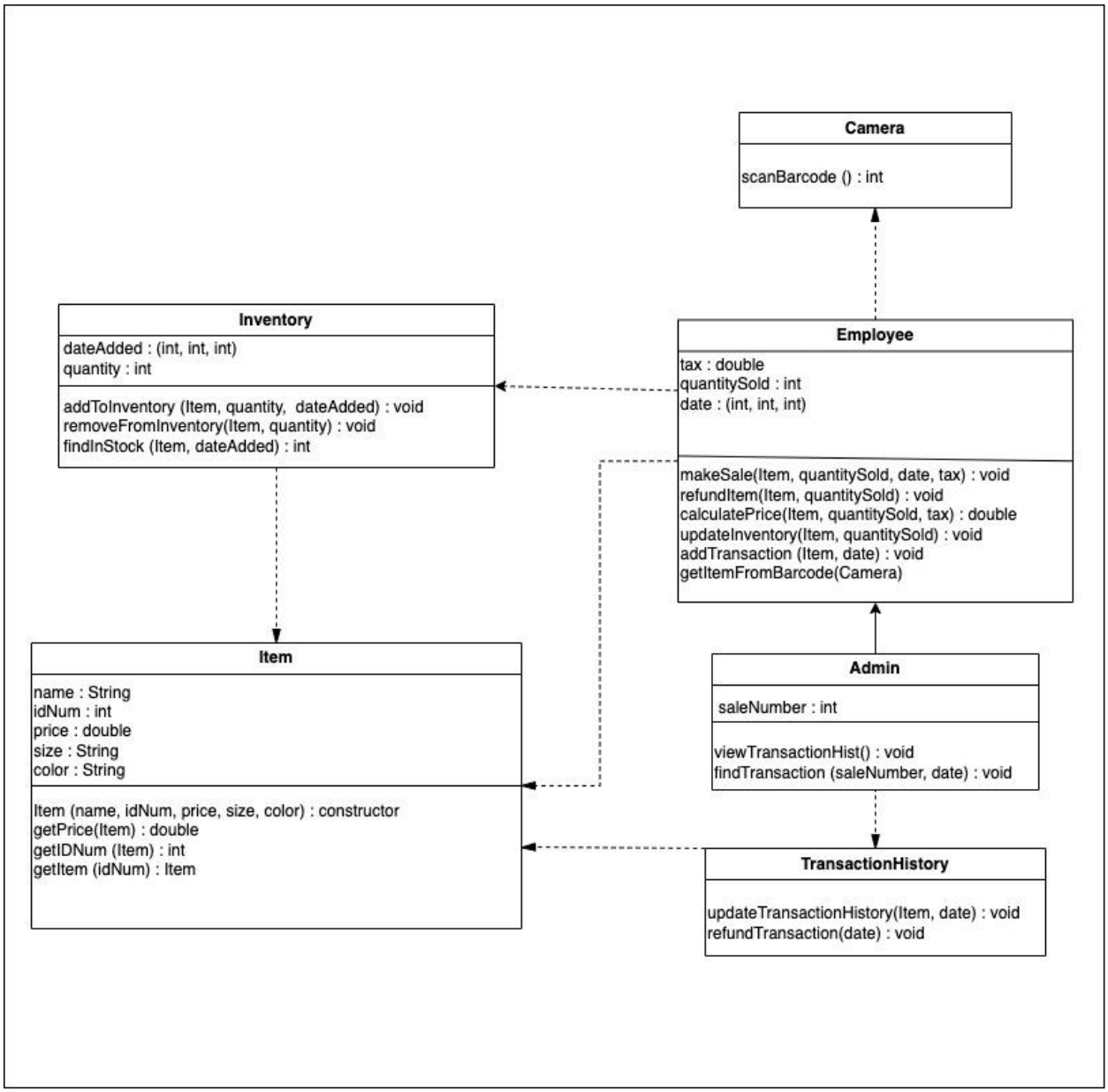
Admin class:

- findTransaction() was removed from TransactionHistory class and added to the Admin class

TransactionHistory class:

- refundTransaction() was added and takes in the parameter date which should be the date of the original transaction that needs to be refunded

All edits made were to create a better flow through the system and allow for the correct variables to be accessible where they need to be.



Verification Test Plan

Unit Test 1:

This test tests the getPrice() function located in the Item class. The method takes in the Item object and returns a double which is the price of the Item inputted.

1. Create Item object “Shirt”:
    - Name: Shirt
    - ID number - 1234567
    - Price - 12.99
    - Size - Medium
    - Color - grey
2. Call method getPrice() from the Item class while putting the Shirt object into the parameters.
    - getPrice(Shirt)
3. Expected Output:
    - System should return:

Price: $12.99

Unit Test 2:

This test is for the getItem() function located in the Item class. The method takes in an ID number and returns the Item object that the ID number corresponds to. It should return the Item’s name and all corresponding information about the Item including its ID number, price, size, and color.

1. Create Item object “Shirt”:
    - Name: Shirt
    - ID number: 1234567
    - Price: 12.99
    - Size: Medium
    - Color: grey
2. Call method getItem() from the Item class while putting “1234567” into the parameters as the ID Number.
    - getItem(1234567)
3. Expected Output:
    - System should return:

Shirt
ID number: 1234567
Price: $12.99
Size: Medium
Color: grey

Integration Test 1:

This test targets the addToInventory() function which is a part of the Inventory class and uses the Item object created in the Item class. The method takes in an Item object, the quantity of that Item, and the date it was added to the inventory.

1. Create Item object “Shirt”:
    - Name: Shirt
    - ID number: 1234567
    - Price: 12.99
    - Size: Medium
    - Color: grey
2. Initialize variable dateAdded in Inventory class to:
    - (01, 01, 2023)
3. Call method addToInventory() putting in the parameters shown:
    - addToInventory(Shirt, 100, dateAdded)
4. Expected Output:
    - The inventory should now appear to have 100 shirts added on 01/01/2023. This can be checked by calling the method findInStock(Shirt, dateAdded) which should return the number of shirts:

Shirt
Quantity: 100

Integration Test 2:

This test is for the method updateTransactionHistory() located in the class, TransactionHistory. The method takes in an Item object and the date of the transaction. The output of the method is void but it can be checked if the transaction was recorded by using the method viewTransaction() in the Admin class.

1. Create Item object “Shirt” from the Item class:
    - Name: Shirt
    - ID number: 1234567
    - Price: 12.99
    - Size: Medium
    - Color: grey
2. Initialize a variable “(int, int, int) date” as (01, 01, 2023)
3. Call the method updateTransactionHistory() with the parameters as shown:
    - updateTransactionHistory(Shirt, date)
4. Expected Output:
    - There should now be a transaction in the Transaction History for one shirt. The date of the transaction should be 01/01/2023. This can be checked by calling the method viewTransactionHist() in the Admin class which shows the user any and

all transactions put into the transaction history. The transaction just put in should be present as:

01/01/2023

1 Shirt
ID number: 1234567
Price: $12.99
Size: Medium
Color: grey

Transaction Total: $13.93
Sale Number: 00000001

System Test 1:

This test walks through the system, testing the process of an employee making a sale. This test uses the Employee, Item, Inventory, Admin, and TransactionHistory classes.

1. Create Item object “Shirt” from the Item class:
    - Name: Shirt
    - ID number: 1234567
    - Price: 12.99
    - Size: Medium
    - Color: grey
2. Initialize variable dateAdded in Inventory class to:
    - (01, 01, 2023)
3. Call the method addToInventory() using the following parameters:
    - addToInventory(Shirt, 50, dateAdded)
4. Initialize the field date in the Employee class to:
    - (01, 01, 2023)
5. Initiialize field quantitySold to 1
6. Call the method makeSale() from the Employee class inputting the following parameters:
    - makeSale(Shirt, quantitySold, date, tax)
7. Expected Output:
    - Calling the makeSale() results in the following:
    - calculatePrice(Shirt, quantitySold, tax) from the Employee class is called to calculate the price of the shirt with tax factored in. The price should be:

Price with Tax: $13.93

    - Using the external payment system the buyer’s card information is inputted and the payment is processed.

- updateInventory(Shirt, quantitySold) from Employee class is called which results in calling removeFromInventory(Shirt, quantitySold) from the Inventory class. Using the method findInStock(Shirt, dateAdded) the inventory should read:

Shirt
Quantity: 49

- addTransaction(Shirt, date) from the Employee class is called which in turn calls updateTransactionHistory(Shirt, date) from the TransactionHistory class which adds a transaction for 1 shirt being performed on 01/01/2023. This can be viewed using viewTransactionHist() from the admin class and the transaction should appear as:

01/01/2023

1 Shirt
ID number: 1234567
Price: $12.99
Size: Medium
Color: grey

Transaction Total: $13.93
Sale Number: 00000001

- After this the sale is complete.

System Test 2:

This test runs through the process of making a refund. The main method involved is refundItem() from the Employee class which also uses methods in the Item, Inventory, Admin, and TransactionHistory classes.

1. Create Item object “Shirt” from the Item class:
    - Name: Shirt
    - ID number: 1234567
    - Price: 12.99
    - Size: Medium
    - Color: grey
2. Initialize variable dateAdded in Inventory class to:
    - (01, 01, 2023)
3. Call the method addToInventory() using the following parameters:
    - addToInventory(Shirt, 50, dateAdded)
4. Initialize date field in Employee class to:
    - (01, 01, 2023)

5. Call method addTransaction(Shirt, date) which calls updateTransactionHistory(Shirt, date), adding a transaciton for one shirt tha to took place on 01/01/2023. This can be viewed by calling the method viewTransactionhist() from the Admin class:

01/01/2023

1 Shirt
ID number: 1234567
Price: $12.99
Size: Medium
Color: grey

Transaction Total: $13.93
Sale Number: 00000001

6. Call method refundItem() with the following parameters:
    - refundItem(Shirt, 1)
7. Expected Output:
    - Calling refundItem() results in the following:
    - calculatePrice(Shirt, 1, tax) is called which calculates the amount needed to be refunded and should result in:

Refund Price: $13.93

    - The refund will be paid in cash by the cashier.
    - Once complete, updateInventory(Shirt, 1) from the Employee class is called which calls addToInventory(Shirt, 1, dateAdded) in the Inventory class. There should be one shirt added to the inventory which can be checked using findInStock(Shirt, dateAdded). It should output the following:

Shirt
Quantity: 51

    - refundTransaction() is called inputting the date of the original purchase into the parameters. This should label the origioinal transaction as refunded. This can be checked using viewTransactionHist() in the Admin class:

01/01/2023

REFUNDED
1 Shirt
ID number: 1234567
Price: $12.99
Size: Medium

Color: grey

Transaction Total: $13.93
Sale Number: 00000001

- The refund is complete.