**DOKUZ EYLÜL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

# CME 2210

**Object Oriented Analysis and Design**

**RETAIL MANAGEMENT SYSTEM**

**by**

**Berhan Türkü Ay - 2017510013**

**Gökhan Göksel Elpeze - 2015510127**

**Murat Taylan Şahin - 2017510093**

**Oktay Türkdağlı - 2017510101**

## CHAPTER ONE INTRODUCTION

Increasing productivity and speeding processes up has a direct effect on customer satisfaction in retailers. Technology plays an important role in terms of more comfortable and intuitive customer experience. Retail Management System (RMS) we develop keeps this principle in mind and lets you manage these processes in real-time. Thanks to this, it is possible to speed up processes easily ranging from receipts to products, inventory

management to price management, purchases to promotions, orders to transfers.

RMS is designed with needs of retailers selling a wide range of different products in

mind.

Using RMS, you can attract more customers and attend to existing customers in real-time, manage different branches of your shop, analyze your sale data and generate meaningful reports about your company's future. In addition to this you can offer promotions for customers and check your employees’ performance from this program.

**MAIN FUNCTIONALITIES AND USER GUIDE:**

Selling a Product:

* After Logging in with your username and password, click on SALES menu.
* In the screen that opens up, enter product id to the SEARCH bar. Click on the

correct product or press ENTER.

* Chosen product will appear on the Cart Preview Window.
* Repeat these steps for all other products in your customer’s cart.
* Click on GENERATE RECEIPT button.
* Pick the correct payment option (Cash or Credit Card) and press COMPLETE TRANSACTION button.
* You can press CANCEL on any of these steps in case you want to cancel the

transaction.

* Sale successful notification appears upon successful sale.

Adding a New Product:

* After login, if you have admin privileges PRODUCT tab will activate, click on PRODUCT menu.
* If you want to add a new item, leave SEARCH bar on the newly opened screen

blank.

* Product’s attributes are filled (Example TV attributes):
  + - Category
    - Name
    - Size
    - Build Date

o Name of Manufacturer

o Name of Distributor

o Price

o Quantity

* + - Date Product enters company (Passive)
    - Person adding the item (Passive)
* Click on SAVE.
* Item added successfully notification appears upon successful entry.

Removing a Product:

* After login, if you have admin privileges PRODUCT tab will activate, click on PRODUCT menu.
* Type id of the product that needs to be removed on the newly opened screen.

Click on the correct item or press ENTER

* Attributes of the product should fill automatically. Click on REMOVE button
* Click YES on the “Are you sure?” popup.
* Item successfully removed notification appears upon successful removal.

Editing an Existing Product:

* After login, if you have admin privileges PRODUCT tab will activate, click on PRODUCT menu.
* Type id of the product that needs to be edited on the newly opened screen. Click

on the correct item or press ENTER

* Attributes of the product should fill automatically. Modify the attributes you need

to change.

* Click on EDIT button.
* If the edited product is the same as an already existing product, “Would you like to merge these items?” notification appears. If you do not want to merge the

items, click no and edit items attributes again.

* Item successfully edited notification appears upon a successful edit..

Transfering Products:

* After login, if you have admin privileges PRODUCT tab will activate, click on PRODUCT menu.
* Pick TRANSFERS tab on the newly opened screen, Type item name or id to the

SEARCH bar and pick the correct item or press ENTER

* Pick the origin and destination branches.
* Enter the quantity (Quantity may not exceed origin branch’s maximum stock).
* “Transfer request is sent.” notification appears upon successful entry.
* An admin should approve of transfer request.
  + If user has logged in with an admin account, MANAGEMENT tab will

activate, click on MANAGEMENT tab.

* + Pick TRANSFER REQUESTS tab from newly opened window.
  + Transfer requests should be listed here. Pick the request you want to

approve and click APPROVE.

* + Transfer approved notification appears upon successful approval.

Adding a New Employee:

* After login, if you have admin privileges MANAGEMENT tab will activate,

click on MANAGEMENT menu.

* Navigate to EMPLOYEES tab on the newly opened window and pick ADD

option. Do not fill in the SEARCH bar.

* Following attributes of the Employee is filled
  + Employee ID (Passive) o National Identification Number
  + Taxpayer ID
  + Name
  + Surname
  + Branch id where employee works in o Position
  + Phone Number o Address o E-mail
  + Salary (If this is left empty base salary of the position will be applied)
  + Birth Date o Birthplace
  + Start date(Passive)
  + Current user (Passive)
* Click on SAVE button.
* Employee added notification appears upon successful entry.

Editing Employees:

* After login, if you have admin privileges MANAGEMENT tab will activate,

click on MANAGEMENT menu.

* Navigate to EMPLOYEES tab on the newly opened window and pick ADD

option. Fill in the SEARCH bar with employee id or name and pick the employee

or press ENTER.

* Employee attributes should automatically fill up, edit the attribute you want to

change.

* Click EDIT button.
* Edit successful notification appears upon successful edit.

Removing Employees:

* After login, if you have admin privileges MANAGEMENT tab will activate, click on MANAGEMENT menu.
* Navigate to EMPLOYEES tab on the newly opened window and pick ADD

option. Fill in the SEARCH bar with employee id or name and pick the employee

or press ENTER.

* Employee attributes should automatically fill up, check if employee is the one

you want to remove, then Click EDIT button.

* Answer yes to “Are you sure?” notification.
* Employee successfully removed notification appears upon successful removal.

Adding a new branch:

* After login, if you have admin privileges MANAGEMENT tab will activate,

click on MANAGEMENT menu.

* Navigate to BRANCHES tab on the newly opened window and pick ADD

option. Do not fill in the search bar.

* Fill in the following attributes of the new branch:
  + - Name
    - Phone number o E-mail o Address
    - Opening Date (Passive)
* Click on SAVE button
* Successfully saved notification appear upon successful entry.

Removing a Branch:

* After login, if you have admin privileges MANAGEMENT tab will activate,

click on MANAGEMENT menu.

* Navigate to BRANCHES tab on the newly opened window and pick REMOVE

option. Fill the search bar with branch id.

* Click on REMOVE button
* Answer yes to “Are you sure?” notification.
* Branch successfully removed notification appears upon successful removal.

Editing a Branch:

* After login, if you have admin privileges MANAGEMENT tab will activate,

click on MANAGEMENT menu.

* Navigate to BRANCHES tab on the newly opened window. Fill the search bar

with branch id.

* Attributes should fill automatically. Edit the attributes you want to change.
* Answer yes to “Are you sure?” notification.
* Branch successfully edited notification appears upon successful edit.

**CHAPTER TWO: REQUIREMENTS**

## CLASSES

1. **MANAGEMENT CLASSES**

|  |  |
| --- | --- |
|  | |
| ​Inventory Manager | |
|  | |
| Adds an already existing item to mentioned branch | |
| ​addItemToShop​(Shop shopToAdd​, int ​quantity, Item item)​: void | |
|  | |
| Removes an already existing item from a branch in specified quantity | |
| ​removeItemFromShop​(Shop shopToAdd​, int ​quantity, Item item)​: void | |
|  | |
| Creates a new Item | |
| ​addNewItem​(Item item)​: void | |
|  | |
| Removes an existing item from all branches and marks it as removed | |
| ​removeItem​(Item item)​: void | |
|  | |
| Transfers specified product from origin Shop to destination | |
| ​transferBetweenBranches​(Shop originShop​, ​Shop destinationShop​, ​Item | |
| transferredItem​, int ​quantity):​ void | |
|  | |
| Orders items in specified quantity from the item’s manufacturer | |
| ​orderItem​(Item item​, int ​quantity)​: void | |
|  | |
| Searches and returns an item from Inventory | |
| ​searchItem​(Item item)​: Item | |
|  | |
|  | |
| Sales Manager | |
|  | |
| Sells specified item in given quantity | |
| sellItem​(Item item​, int ​quantity)​: void | |
|  | |
| Generates final receipt of customer and saves it | |
| generateReceipt​(Item[] items)​: void | |
|  | |
| Finds and returns specified receipt | |
| searchReceipt​(int receiptID, Date date)​: Receipt | |
|  | |
| Returns an item and saves a return receipt |  |
| returnItem​(Item item​, int ​receiptID)​: void |  |

|  |
| --- |
|  |
| Reports Manager |
|  |
| Fetches specified report |
| displayReports​(int chosenReport) ​:void |
|  |
|  |
|  |

|  |
| --- |
|  |
| Staff Manager |
|  |
| Adds a new employee |
| addEmployee​(Employee employee) ​:void |
|  |
| Removes an employee |
| removeEmployee​(Employee employee) ​:void |
|  |
| Edits an employee |
| editEmployee​(Employee employee) ​:void |
|  |
| Finds and returns employee with specified ID |
| searchEmployee​(int employeeID) ​:Employee |
|  |

1. **PRODUCT CLASSES**

**Note: Attributes of Product classes are not final and they might change later on to be more detailed/realistic or to improve usability.**

**Our base class for products is “Item”**

|  |
| --- |
| public class ​Item { |
| ​private ​String ​brand​; |
| private ​String ​name​; |
| private double ​price​; |
| private int ​itemID​; |
| private ​Manufacturer ​manufacturer​; |

**We are extending new classes from Item class. First one is “Clothing”.**

1. **- CLOTHING**

|  |
| --- |
| public class ​Clothing ​extends ​Item{ |
| ​private ​String ​color​; |
| private ​String ​gender​; |

**We are extending new classes from Clothing class. They are “Footwear”, “Pants” and**

**“Tshirt”.**

**SUBCLASSES:**

* 1. **FOOTWEAR**

|  |
| --- |
| public class ​Footwear ​extends ​Clothing { |
| ​private double ​size​; |

* 1. **PANTS**

|  |
| --- |
| public class ​Pants ​extends ​Clothing { |
| ​private int ​sizeWidth​; |
| private int ​sizeLength​; |

* 1. **T-SHIRT**

|  |
| --- |
| public class ​Tshirt ​extends ​Clothing { |
| ​private ​String ​size​; |

**We are extending new classes from Item class. Second one is “Electronics”.**

1. **- ELECTRONICS**

|  |
| --- |
| public class ​Electronics ​extends ​Item { |
| ​private double ​size​; |
| private ​String ​resolution​; |

**We are extending new classes from Clothing class. They are “Computer”, “Phone” and**

**“Television”.**

* 1. **COMPUTER**

|  |
| --- |
| public class ​Computer ​extends ​Electronics { |
| ​private ​String ​processor;​ |
| private ​String ​graphicsCard​; |
| private ​String ​graphicsCapacity​; |
| private ​String ​ramType​; |
| private ​String ​ramSpeed​; |
| private ​String ​ramCapacity​; |
| private ​String ​storage​; |

* 1. **PHONE**

|  |
| --- |
| public class ​Phone ​extends ​Electronics{ |
| ​private ​String ​memory​; |
| private ​String ​ram​; |

* 1. **TELEVISION**

|  |
| --- |
| public class ​Television ​extends ​Electronics { |
| ​private boolean ​smart​; |
| private ​String ​screenType​; |

**We are extending new classes from Item class. Third one is “OtherProducts”. Other**

**products is for all the items and categories that don’t have a specific class.**

**3-) OTHER PRODUCTS**

|  |
| --- |
| public class ​OtherProducts ​extends ​Item { |
| ​private ​String ​categoryName​; |

**C) COMPANY CLASSES:**

**MAIN CLASS: COMPANY**

**1 – COMPANY**

|  |
| --- |
| public class ​Company { |
| ​private ​List<Shop> ​shops;​ |
| private int ​phoneNumber​; |
| private ​String ​emailAddress​; |
| private ​Address ​address​; |
|  |
|  |
| addNewShop​(Shop shop)​:void |
|  |
| removeShop​(Shop shop)​:void |
|  |
| addNewManufacturer​(Manufacturer manufacturer)​:void |
|  |
| removeManufacturer​(Manufacturer manufacturer)​:void |
|  |

1. **SHOP**

|  |
| --- |
| public class ​Shop { |
| ​private ​List<Employee> ​employees​; |
| private ​ArrayList<Item> ​products​; |
| private int ​phoneNumber​; |
| private ​String ​emailAddress​; |
| private ​Address ​address​; |
|  |

1. **EMPLOYEE**

|  |
| --- |
| public class ​Employee { |
|  |
| ​private ​String ​name​; |
| private ​Address ​address​; |
| private int ​phoneNumber​; |
| private int ​weeklyWorkHours​; |
| private ​Position ​position​; |
| private int ​shopID​; |
| private int ​employeeID​; |
| private ​String ​password​; |
| private int ​weeklyWageBonus​; |

1. **MANUFACTURER**

|  |
| --- |
| public class ​Manufacturer { |
| ​private ​String ​name​; |
| private int ​manufacturerID​; |
| private ​Address ​address​; |
| private ​String ​phoneNumber​; |
| private ​List<Item> ​soldItemList​; |

**UTILITY CLASSES:**

**1-ADDRESS**

|  |
| --- |
| public class ​Address { |
| ​private ​String ​city​; |
| private ​String ​town​; |
| private ​String ​street​; |

**2-DATE**

|  |
| --- |
| public class ​Date { |
| ​private int ​day​; |
| private int ​month​; |
| private int ​year​; |
|  |
| validateDate​(Date date) ​:boolean |

1. **POSITION**

|  |
| --- |
| public class ​Position { |
| ​private ​String ​positionName​; |
| private int ​baseWage​; |

1. **RECEIPT**

|  |
| --- |
| public class ​Receipt { |
| ​private ​Date ​date​; |
| private double ​totalPrice​; |
| private ​Employee ​cashier​; |
| private ​List<Item> ​cart​; |
| private int ​receiptID​; |
|  |

1. **PHONE NUMBER**

|  |
| --- |
| public class ​PhoneNumber { |
| ​private ​String ​countryCode​; |
| private int ​code​; |
| private int ​number​; |
| private ​String ​type​; |
|  |
| validatePhoneNumber​(PhoneNumber number)​:boolean |
|  |

**GUI CLASSES:**

**CONTROLLERS:**

**(Common Function) public void** ​**handleButtonAction**​**(ActionEvent event) {}:void**

**1– CompanyScreenController**

**2- HomeScreenController**

|  |
| --- |
|  |
| This function logs out from current user and opens Login Screen |
| logout​()​:void |
|  |
| This function saves all data to a file for next session |
| saveData​()​:void |
|  |
|  |

**3-LoginScreenController**

|  |
| --- |
|  |
| Checks username and password |
| loginCheck​() ​:boolean |
|  |
| Loads data from previous sessions to initialize the program |
| loadData​()​:void |
|  |

**4-SaleScreenController**

**5-ReportsScreenController FXMLs:**

**1-companyScreen**

**2-homeScreen**

**3-loginScreen**

**4-saleScreen**

**5-reportsScreen**

**CHAPTER THREE**

## UML DIAGRAMS

CLASS DIAGRAM:

Diagram shows all the classes inside the system and their attributes and functions. Item class and all its children represent product classes. Phone number, address, date, receipt and position represents utility classes. Company, shop, employee and manufacturer represents main classes. 4 interfaces correspond to management classes.

USE CASE DIAGRAM:

There are 4 main actors on the system. Administrator for managing the overall system and company, Branch manager for overseeing a branch and managing sales, sales associate for completing sales and a customer who makes purchases through the sales associate. Management Screen, Company screen, product screen, sale screen and menu represent 5 main screens of the system and other use cases under them shows their main functionalities. Blue dotted line is used to imply <<Extends>>, black dotted line is for <<Include>> and straight black line shows association.

ACTIVITY:

FIRE:

Employee resignes, program user removes their personal information from system, then removes his username and password.

HIRE

Employee comes to the company and gives his personal information for the hiring process. Program user enters this information to the system and generates unique username and password. Assigns him to a shop.

SALE

customer comes to register cashier searchs item from the system, shop checks inventory, then cashier adds item and repeats this until all customers items has been scanned, customer chooses payment option (cash or credit card), cashier generates receipt, shop updates inventory, cashier completes purchase.

STATE

EMPLOYEE

Employee is hired, his daily routine is selling items and returning items, then when employee quits his information is deleted.

MANAGER

manager is hired, his daily routine is selling, adding, returning, editing, transfering and removing items; adding, editing, removing employees. Adding, editing, removing branches. when manager quits his information is deleted.

SEQUENCE DIAGRAM:

ADD NEW EMPLOYEE

User logs in, enters the home screen. Activates employee function and contacts employee class a new user is added with addNewEmployee function, before this function finishes, checks if this employee has been added before.

ADD NEW ITEM

User logs in, enters the home screen. Activates product function and connects to product class. A new product is added with addNewItem, before this function ends, checks if this item has been added before.

SALE SCREEN

User logs in, enters the home screen. Starts saleScreen function and connects to SaleScreenController class. Adds items to sale screen with searchItem function, after all items are added generates receipt with generateReceipt function

TRANSFER SCREEN

User logs in, enters the home screen. Starts productScreen function and connects to product class. Adds a new transfer with transferBetweenBranches funtion, before this function ends coonects to Item class with searchItem function and adds necessary items.

Delete Informations from System

Delete Login Informations

Wants to Quit

Visit Company

Manager

Employee

ACTIVITY:

Employee Fire Activity

Enter Informations to System

Generate Username and Password for Employee

Assign to Shop

Provides Personal Informations

Visit Company

Manager

Employee

ACTIVITY:

Employee Hire Activity

Complete Purchase

Cash

Credit

Payment

Customer Comes to the Register

Search Item

Add Item

Update Inventory

Generate Receipt

Generate Bills

Check Inventory

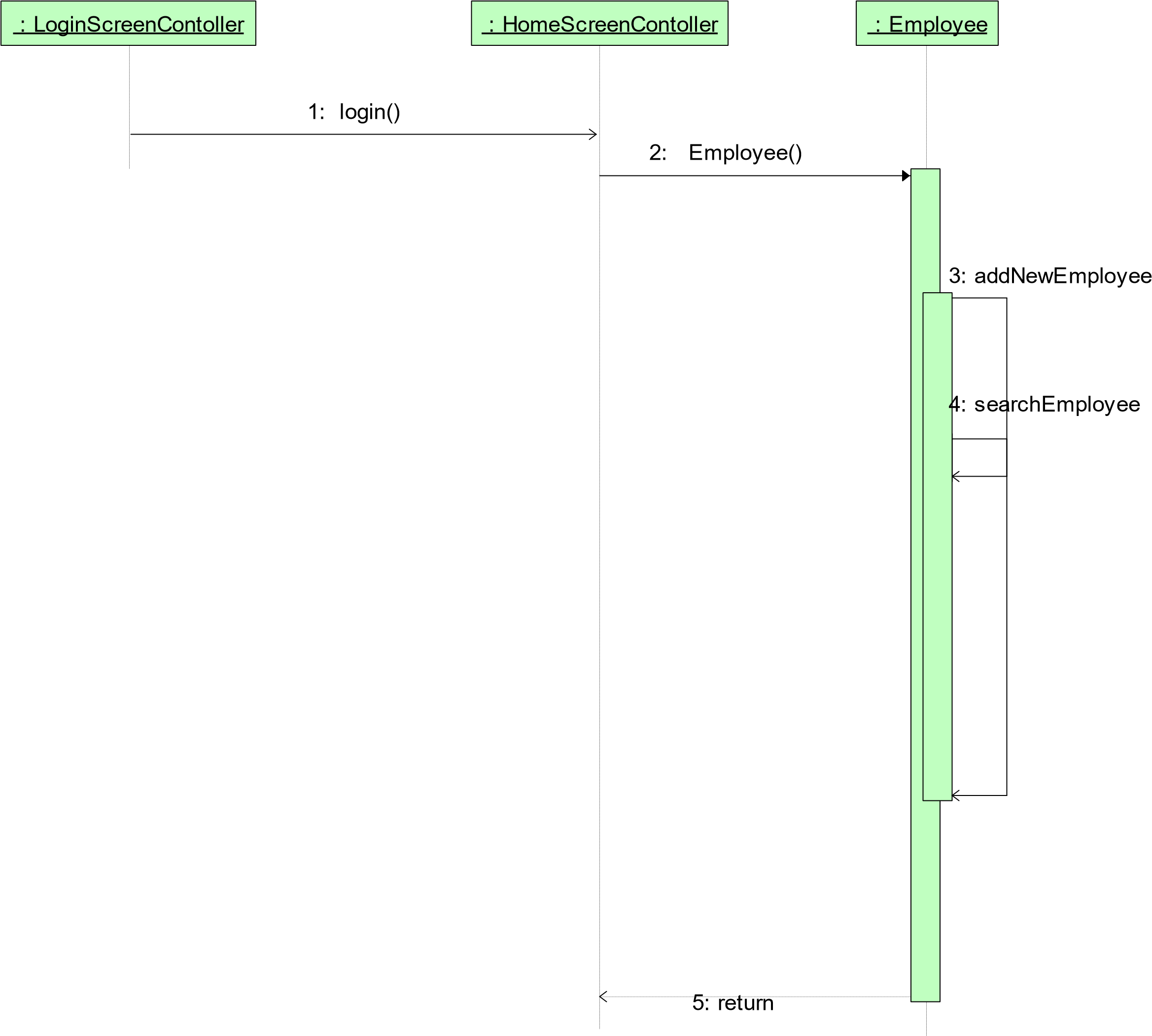
Some items are not added

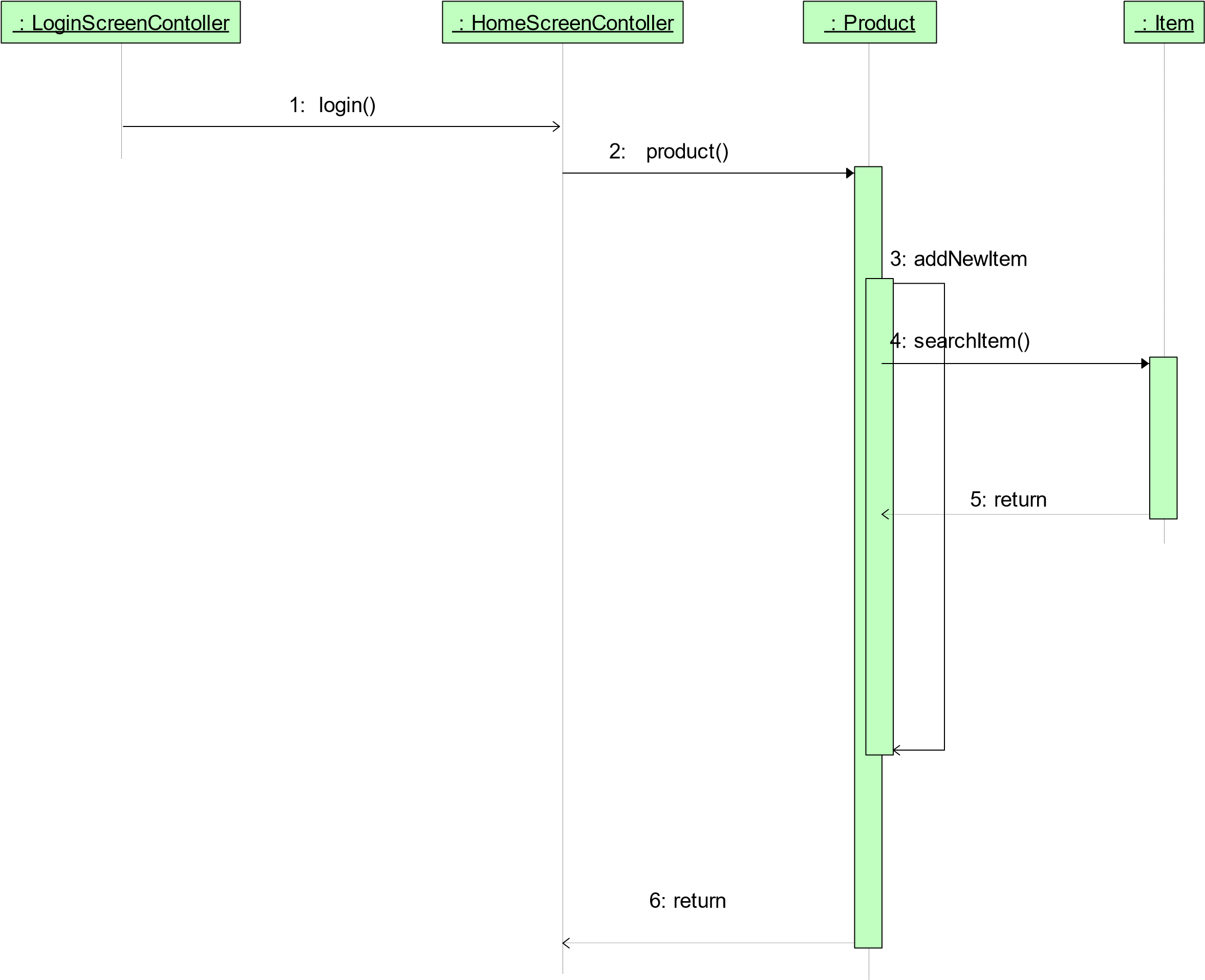
All items added

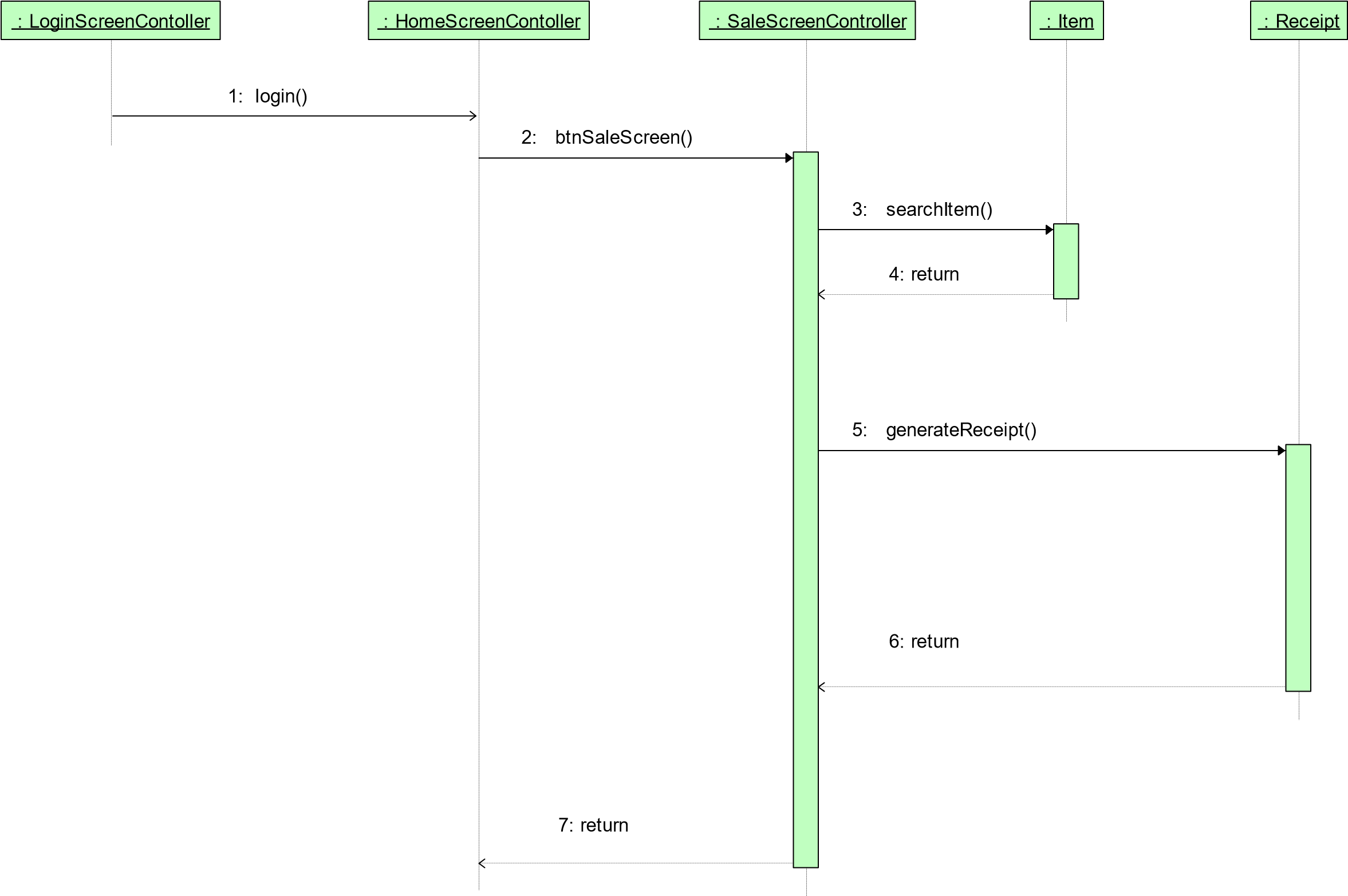
Customer

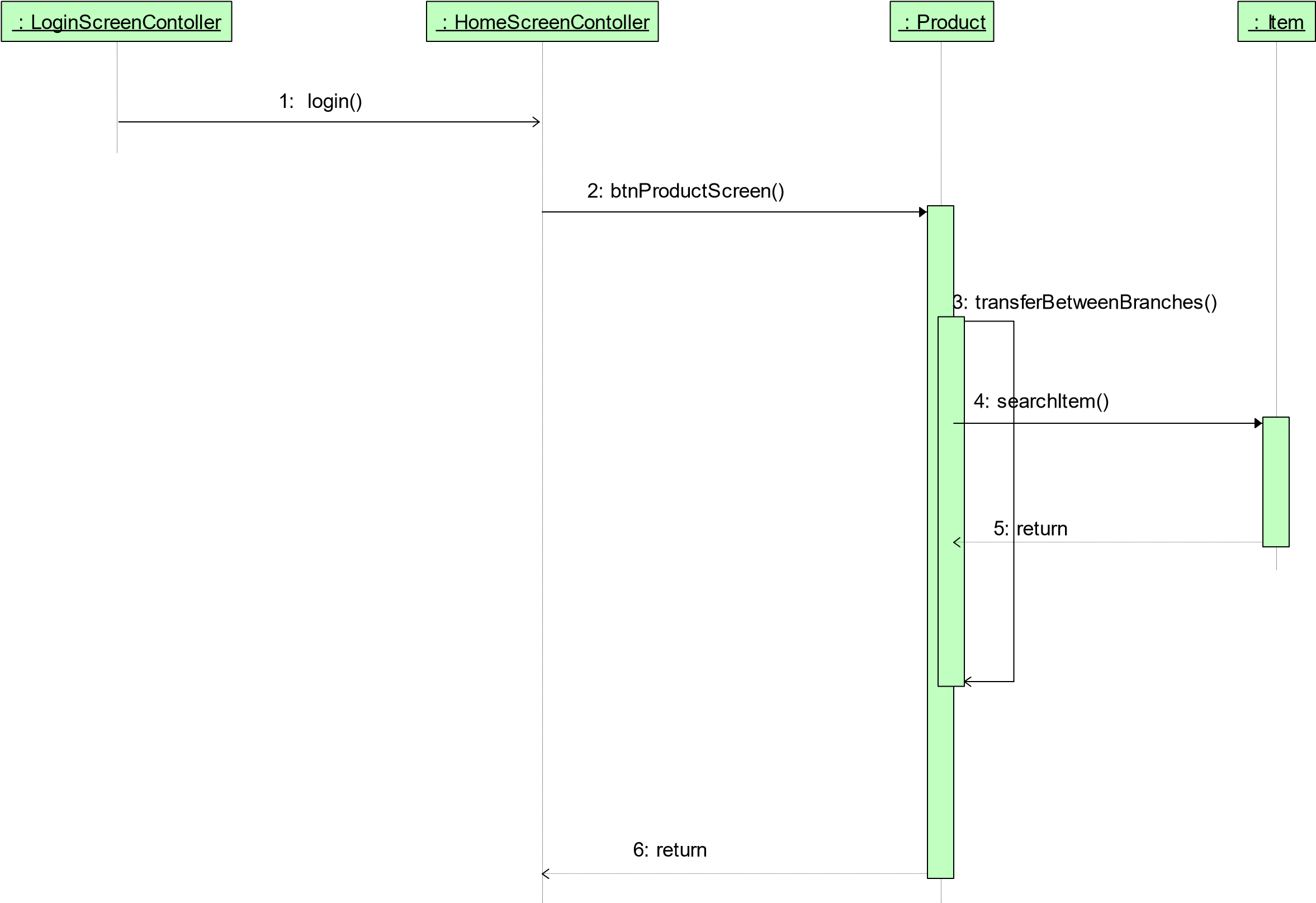
Cashier

Shop









OBJECT:

EMPLOYEE

Daily Routine

Employee logged in

Employee quits

Employee Hired

Employee information entered

Performing Sales

Sell Item

Return Item

Employee laid off

Employee informations deleted

Daily Routine

Manager Quits

OBJECT:

MANAGER

Manager Hired

Manager Information Entered

Manager laid off

Manager Informations Deleted

Manager Logged In

Performing Management

Sell Item

Return Item

Add New Item

Edit Item

Transfer Item

Remove Item

Add Employee

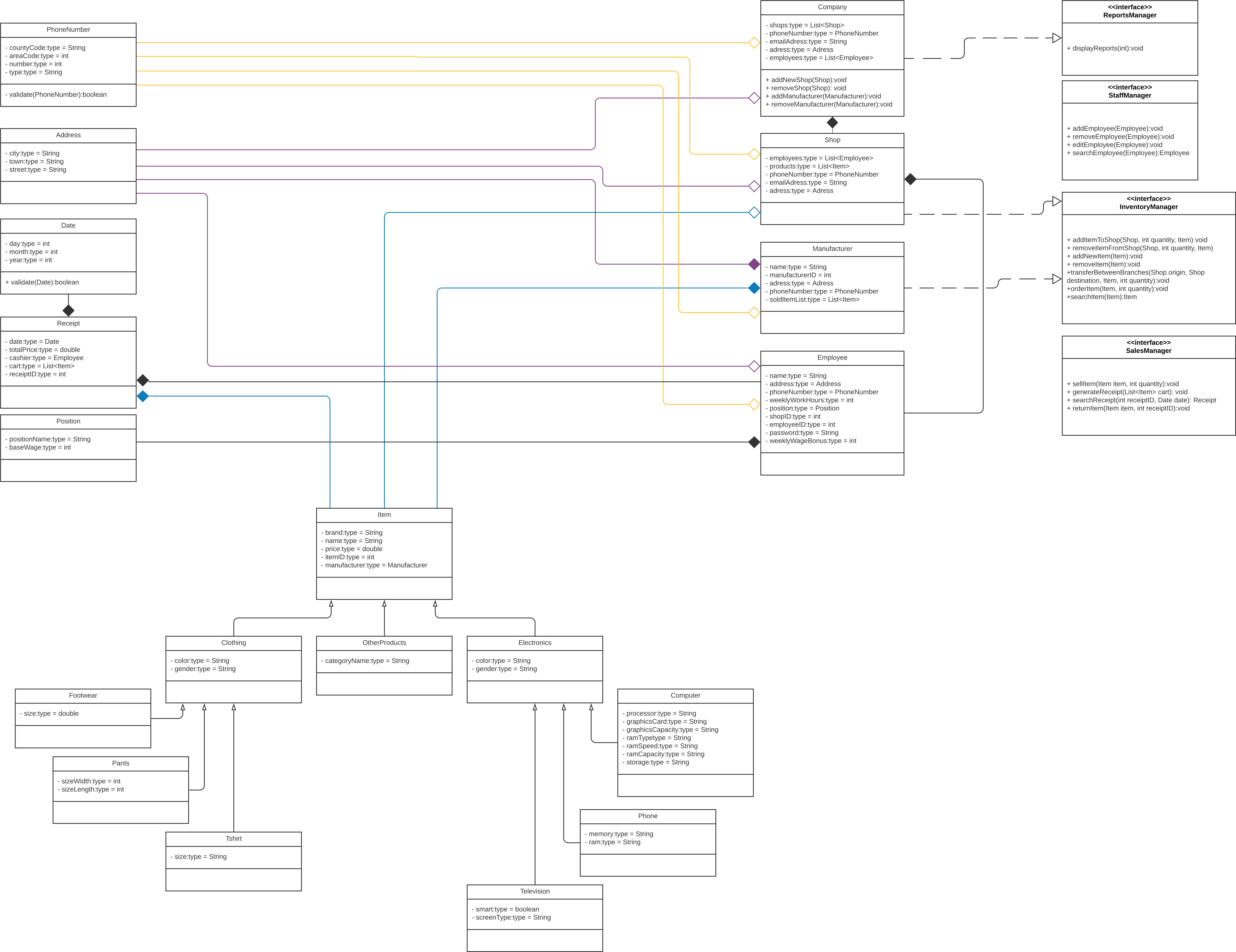
Edit Employee

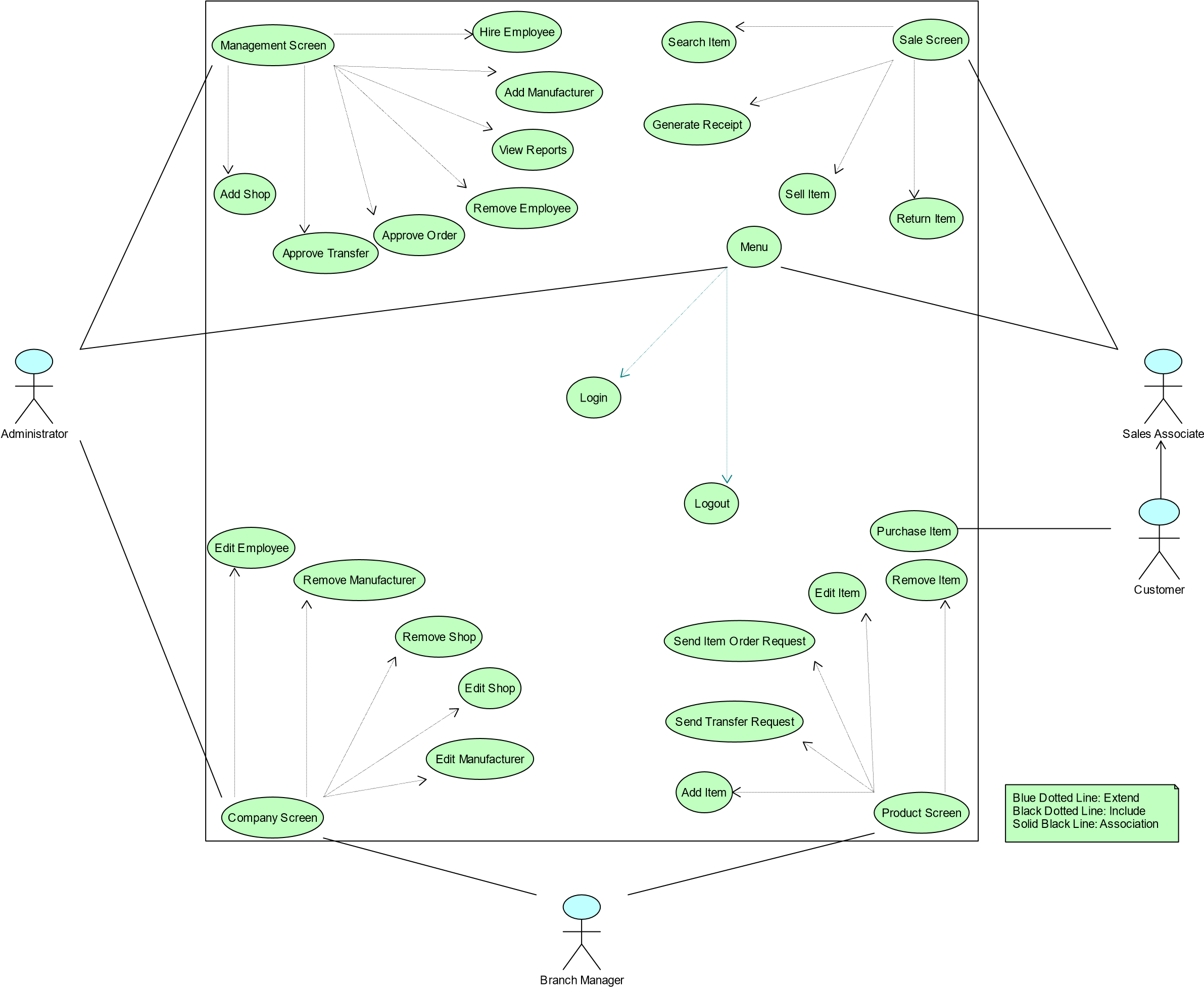
Remove Employee

Add New Branch

Edit Branch

Remove Branch





**CHAPTER FOUR**

## IMPLEMENTATION

There are 5 main types of classes in our program. They are Utility classes, Management classes, Company classes, Product classes and GUI classes.

Product classes are used for storing the information of various types of items sold in retail stores. Shared variables for all products are written in a root 'Item' class and all other product classes are its children. Under product classes are 3 categories, namely Electronics, Clothing and Other Products. These categories also include some specific products as their children.

Utility classes are used to store some specific data forms such as Address, Date, and Position etc.

Company classes include Shops, Manufacturers, Company, and Employee. These classes are used to model their real life counterparts. Manufacturer, Company and Shop all have an Inventory, therefore they inherit from Inventory Manager interface. This interface is required for all objects with an inventory.

Shop and Company classes implement Staff Manager interface. This interface has some methods to control employees such as giving them bonuses, hiring new employees or firing existing employees. All objects that include employees must implement this interface.

Company class inherits from Reports Manager interface. This interface has some functions to view various reports regarding a number of topics including but not limited to best-selling branches, top employees and most-least profitable products for each branch.

Shop implements Sales Manager which has some functions to deal with sales, returns and receipts.

**INTERFACES:**

**Inventory Manager:**

Inventory Manager class can do 6 operations

1. Adds an already existing item to mentioned branch with addItemToShop.
2. Removes an already existing item from a branch in specified quantity removeItemFromShop.
3. Creates a new Item with addNewItem.
4. Removes an existing item from all branches and marks it as removed with removeItem.
5. Transfers specified product from origin Shop to destination with transferBetweenBranches.
6. Orders items in specified quantity from the item’s manufacturer with orderItem.
7. Searches and returns an item from Inventor with searchItem.

**Sales Manager:**

Sale Manager class can do 4 operations

1. Sells specified item in given quantity with sellItem.
2. Generates final receipt of customer and saves it with generateReceipt
3. Finds and returns specified receipt with searchReceipt
4. Returns an item and saves a return receipt with returnItem.

**Staff Manager:**

Staff Manager class can do 4 operations

### Adds a new employee with addEmployee​.

### Removes an employee with removeEmployee.

### Edits an employee with editEmployee.

### Finds and returns employee with specified ID with searchEmployee.

**Reports Manager:**

Reports Manager class can do 1 operation

1. Fetches specified report with displayReports.

**CHAPTER FIVE**

**CONCLUSION AND FUTURE WORKS**

In this chapter, a summary and future works must be written.