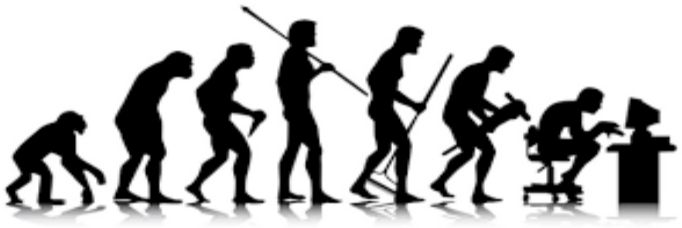
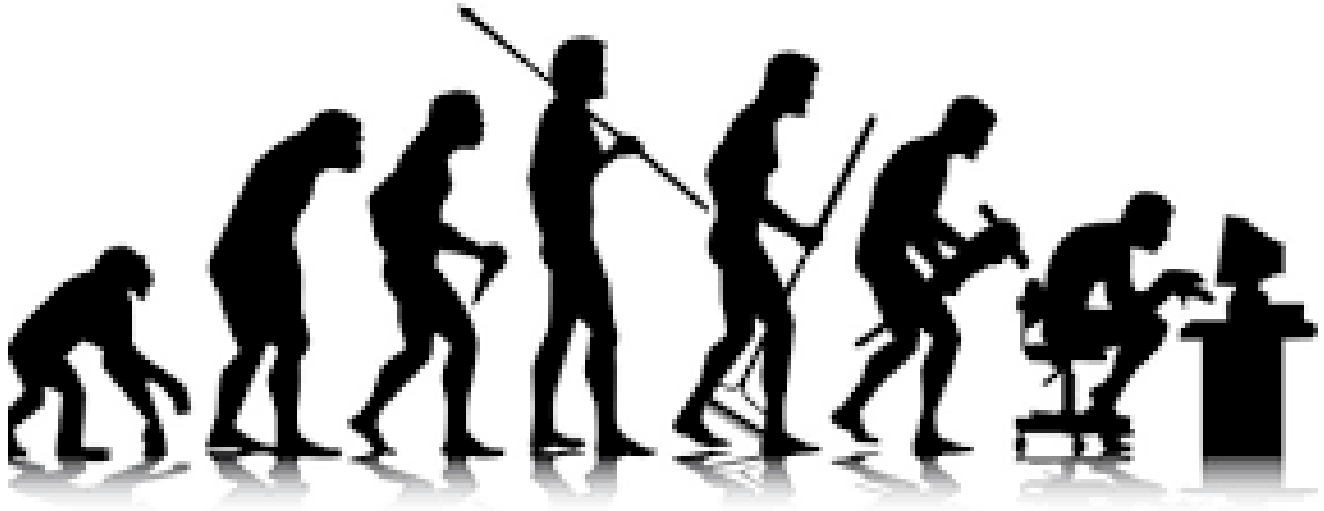
Typewriter Monkeys

Tyler Webb, Logan McAbee, Kris Heiskell

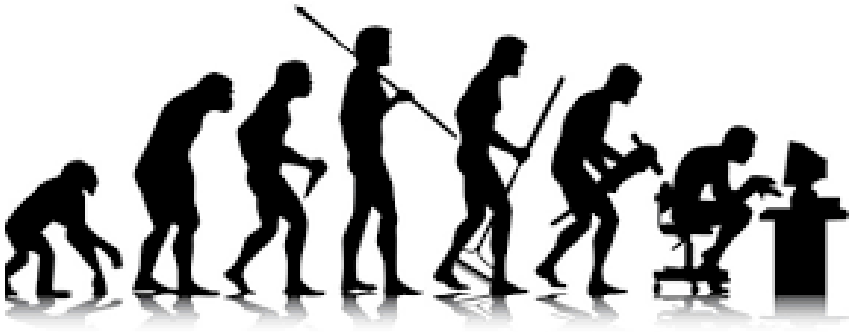


Introduction



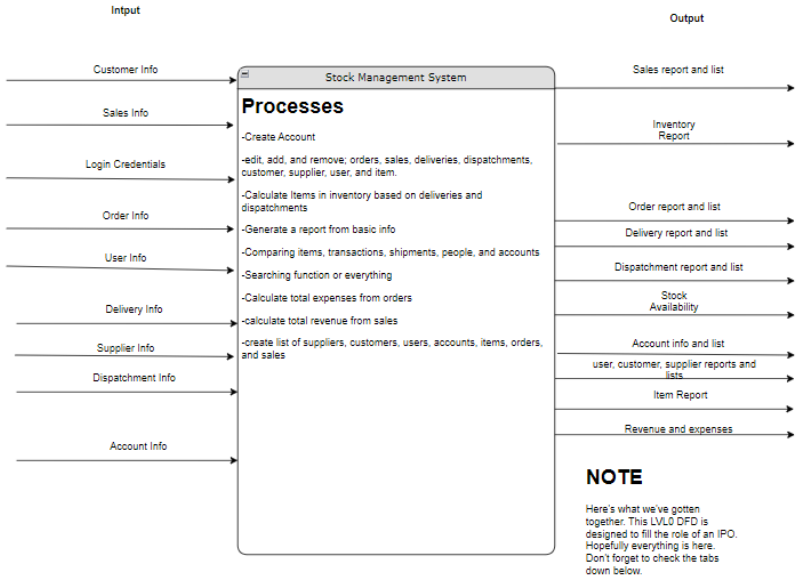
- The purpose of this project was to design an inventory management system using software. The functions of this software program include specific and secure user accounts, procurement of new inventory via purchases, relinquishment of current inventory via sales, inventory traceability, and keeping records of customers and suppliers.

Requirements



1) Create New User Accounts and Perform Credential Checking
    a. Create, remove, and edit user accounts.
    b. Account validation via username and password crosschecking with the account file.
    c. Save and load user accounts and credentials.
2) Procurement of New Inventory via Purchases
    a. Create new purchase orders to increase current inventory.
    b. Remove past purchase orders from history.
    c. Generate report of all past purchase orders.
    d. Save and load details of procurement of new inventory.
3) Relinquishment of Current Inventory via Sales
    a. Create new sales orders to decrease current inventory.
    b. Remove past sales orders from history.
    c. Generate report of all past sales orders.
    d. Save and load details of relinquishment of current inventory.
4) Inventory Traceability
    a. Create, remove, and edit inventory items.
    b. Display all current inventory with live updates.
    c. Save and load inventory items and specifics.
5) Customer and Supplier Record Keeping and Management
    a. Create, remove, and edit customer and supplier information.
    b. Display all current customers and suppliers with live updates.
    c. Save and load customer and supplier information.

Analysis



User Info

name, phone#, email, account(s), User ID

Supplier Info

business name, address, phone#, email, category, Supplier Id

Delivery Info

Item(s) info(minus location and ID), Supplier, Arrival date, Delivery ID

Login Credentials

password and username

Customer Info

name, phone#, email, address, Customer ID

Order Info

Item(s) info(minus location and ID), total cost, Supplier, Order ID

Dispatchment Info

Item(s), Customer, Dispatchment ID, Date (to be) shipped

Sales Info

Item info, revenue, Customer, Sales ID

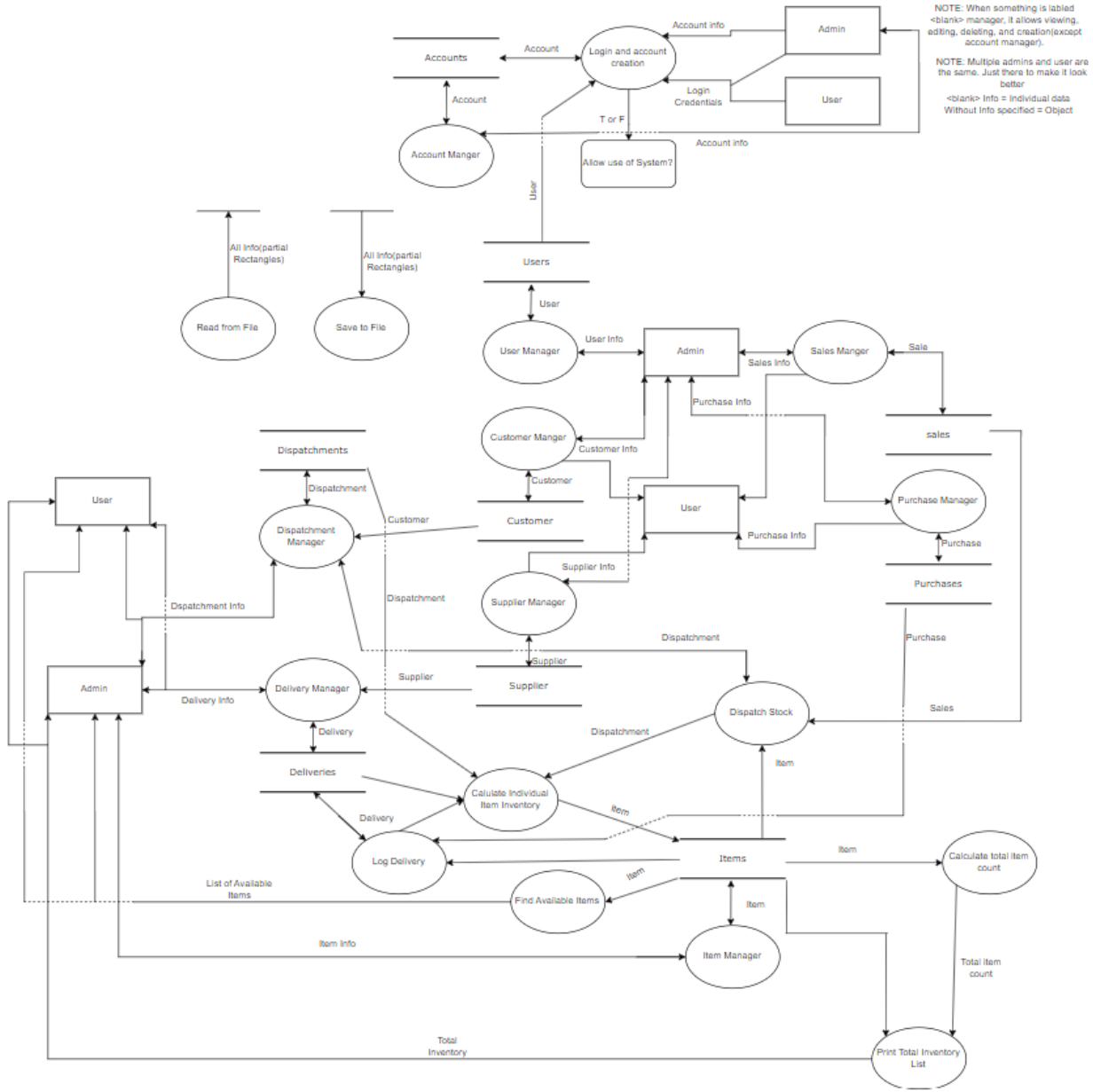
Account Info

password, username, account ID, Owner

Item Info

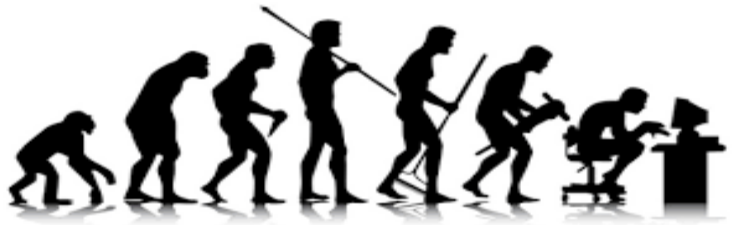
Name, count, measurement , category, location, price, Item ID

Note: ID is determined when an item is logged in the system and location when it is stored



Changes Made to Analysis

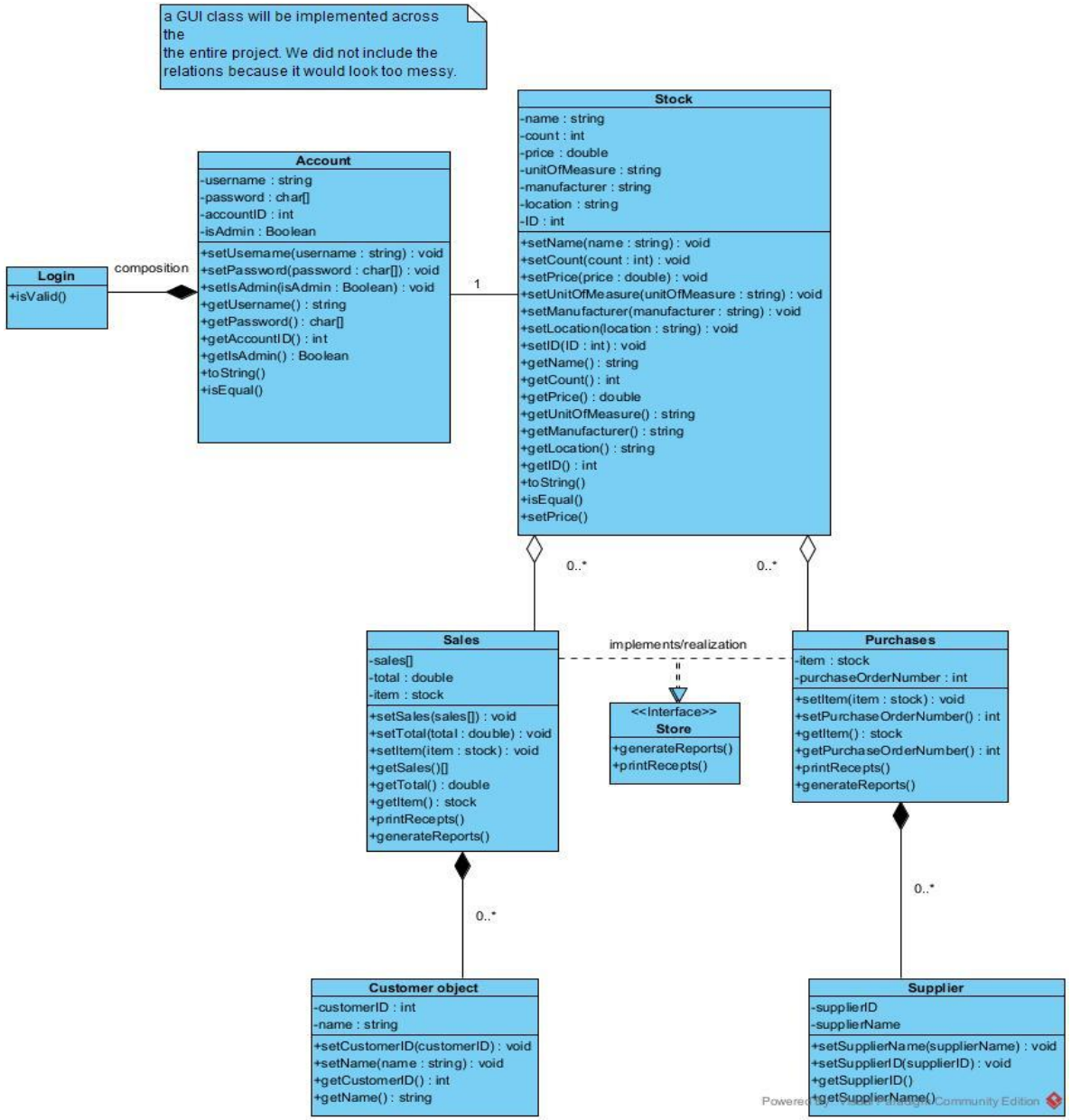
- Removed Delivery and Dispatchment Class
  - Merged into sales and purchases classes due to similar functionality.
- Simplified functionality of class processes.
  - Removed small specifics not immediately relevant to the user.
    - Calculating aggregate of monetary exchanges.
    - Calculating aggregate of inventory movements.



Design CRC

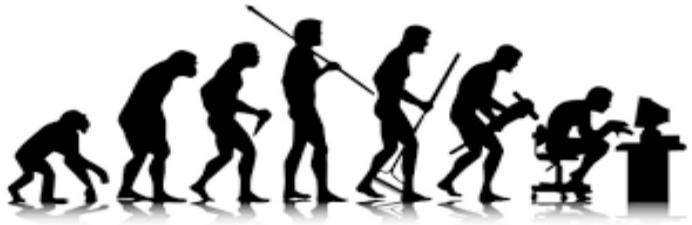

Customer
- Build, maintain, & update objects of class
- Make info displayable
- Sale
- GUI
- StockManager
Stock
- Stock Manipulation
- Store stock info
- Check Availability
- Generate/print stock report
- StockManager
- Store
- GUI
- Sales
- Purchases
Store (interface)
- Print Receipts
- Generate Reports
- GUI
- StockManager
- Stock
- Account
- Store
- Sales
- Purchases
Purchases (implements)
- Perform Purchase Functions
- Store
- GUI
- StockManager
- Stock
- Supplier
Sales (implements)
- Perform Sales Fnction
- Store
- GUI
- StockManager
- Customer
StockManager (Main)
- Create Objects
- Use Methods
- Establish an order of operation
- GUI
- Stock
- Login
- Account
- Store
GUI
- Build Window
- Accept User Input
- Visualize Info
- Account
- StockManager
Supplier
- Build, Maintain, & Update objects of class
- Make info displayable
- Purchases
- GUI
- StockManager
Login
1) Accept Username & Password
2) Credential Input Validation
3) Display Login GUI
4) Self terminating
- Account
- StockManager
- GUI
Account
- Store Account Info
- Verify Login Credentials
- Organize/Sort Accounts
- Display Account info
- Detail Access Permissions
- StockManager
- GUI

Design UML


a GUI class will be implemented across the
the entire project. We did not include the relations because it would look too messy.
Stock
-name : string
-count : int
-price : double
-unitOfMeasure : string
-manufacturer : string
-location : string
-ID : int
+setName(name : string) : void
+setCount(count : int) : void
+setPrice(price : double) : void
+setUnitOfMeasure(unitOfMeasure : string) : void
+setManufacturer(manufacturer : string) : void
+setLocation(location : string) : void
+setID(ID : int) : void
+getName() : string
+getCount() : int
+getPrice() : double
+getUnitOfMeasure() : string
+getManufacturer() : string
+getLocation() : string
+getID() : int
+toString()
+isEqual()
+setPrice()
Account
-username : string
-password : char[]
-accountID : int
-isAdmin : Boolean
+setUsername(username : string) : void
+setPassword(password : char[]) : void
+setIsAdmin(isAdmin : Boolean) : void
+getUsername() : string
+getPassword() : char[]
+getAccountID() : int
+getIsAdmin() : Boolean
+toString()
+isEqual()
Login
+isValid()
composition
1
0..*
0..*
Sales
-sales[]
-total : double
-item : stock
+setSales(sales[]) : void
+setTotal(total : double) : void
+setItem(item : stock) : void
+getSales()[]
+getTotal() : double
+getItem() : stock
+printRecepts()
+generateReports()
implements/realization
<<Interface>>
Store
+generateReports()
+printRecepts()
Purchases
-item : stock
-purchaseOrderNumber : int
+setItem(item : stock) : void
+setPurchaseOrderNumber() : int
+getItem() : stock
+getPurchaseOrderNumber() : int
+printRecepts()
+generateReports()
0..*
0..*
Customer object
-customerID : int
-name : string
+setCustomerID(customerID) : void
+setName(name : string) : void
+getCustomerID() : int
+getName() : string
Supplier
-supplierID
-supplierName
+setSupplierName(supplierName) : void
+setSupplierID(supplierID) : void
+getSupplierID()
+getSupplierName()

Changes to Design

- We assumed that there would be more collaborations than were needed.
  - For example, Login was the only needed collaboration for the Main class StockManager.
- We changed methods of certain classes to better their functionality.
  - Creation of new instance variables and methods for Sales and Purchases.
  - Creation of new methods for Customer and Supplier.
  - Swapped a method from Login to Account.



Implementation

```
} else if (actionEvent.getSource() == saveEdit) {
    if (currentSelectionFlag == Category.ACCOUNTS) {//<-- Operation to perform when Accounts is selected

        Account.editAccount(Integer.parseInt(idField.getText()), passwordField.getText().toCharArray(),
                nameField.getText(),Boolean.parseBoolean(adminField.getText()));
        textDisplayArea.setText(Account.accountListToString());
        textDisplayArea.setText("ID, USERNAME, PASSWORD, ADMIN\n" + Account.accountListToString());

    } else if (currentSelectionFlag == Category.INVENTORY) {//<-- Operation to perform when Inventory is selected

        stock.editStock(nameField.getText(), Integer.parseInt(countField.getText()), Double.parseDouble(priceField.getText()),
                unitOfMeasureField.getText(), manufacturerField.getText(), locationField.getText(),
                Integer.parseInt(idField.getText()));
        textDisplayArea.setText("NAME, COUNT, PRICE UNIT OF MEASURE, MANUFACTURER, LOCATION, ID\n"
                + stock.stockListToString());

    } else if (currentSelectionFlag == Category.CUSTOMERS) {//<-- Operation to perform when Customers is selected

        customer.editCustomer(Integer.parseInt(idField.getText()), nameField.getText());
        textDisplayArea.setText("NAME, ID" + customer.customerListToString());

    } else if (currentSelectionFlag == Category.SUPPLIERS) {//<-- Operation to perform when Suppliers is selected

        supplier.editSupplier(Integer.parseInt(idField.getText()), nameField.getText());
        textDisplayArea.setText("NAME, ID" + supplier.supplierListToString());
```

```
if (actionEvent.getSource() == accounts) { //<-- Operation to perform when Accounts is selected

    currentSelectionFlag = Category.ACCOUNTS;//<-- sets the category flag
    hideLabelsAndTextFields();
    toggleAdminPrivileges();
    textDisplayArea.setText("ID, USERNAME, PASSWORD, ADMIN\n" + Account.accountListToString());

    nameLabel.setBounds( x: 10,  y: 70,  width: 100,  height: 30);
    passwordLabel.setBounds( x: 10,  y: 105,  width: 100,  height: 30);
    adminLabel.setBounds( x: 10,  y: 140,  width: 100,  height: 30);
    idLabel.setBounds( x: 10,  y: 175,  width: 100,  height: 30);

    nameField.setBounds( x: 100, y: 70, width: 200, height: 30);
    passwordField.setBounds( x: 100, y: 105, width: 200, height: 30);
    adminField.setBounds( x: 100, y: 140, width: 200, height: 30);
    idField.setBounds( x: 100, y: 175, width: 200, height: 30);

    nameLabel.setVisible(true);
    passwordLabel.setVisible(true);
    adminLabel.setVisible(true);
    idLabel.setVisible(true);

    nameField.setVisible(true);
    passwordField.setVisible(true);
    adminField.setVisible(true);
    idField.setVisible(true);

    saveEdit.setText("Edit");

    SwingUtilities.updateComponentTreeUI(mainFrame); //<-- updates the frame
```

Implementation (Continued)

```java
/**
 * Takes the data in accountList and saves it to the Accounts.txt file.
 * @throws IOException
 */
3 usages
public static void saveAccountInfo() throws IOException {
    BufferedWriter bWriter = new BufferedWriter(new FileWriter(accountsFile));
    bWriter.write(accountListToString());
    bWriter.flush();
    bWriter.close();

}
/**
 * Reads the Accounts.txt file and inputs the data into the accountList arrayList in a position based on it's ID
 * @throws FileNotFoundException
 * @throws IOException
 * @throws ClassNotFoundException
 */

1 usage
public static void loadAccountInfo() throws FileNotFoundException, IOException, ClassNotFoundException,
        InputMismatchException {//<-- should I just have these handle the exceptions?
    Scanner reader = new Scanner(accountsFile);
    reader.useDelimiter( pattern: ", |\n");
    while(reader.hasNext()){
        int throwawayInt = reader.nextInt();
        String username = reader.next();
        char[] password = (reader.next()).toCharArray();
        boolean admin = reader.nextBoolean();
        Account loadedAccount = new Account(password,username,admin);
        accountList.add(loadedAccount);
    }
    Collections.sort(accountList, new Account());//<-- uses tim sort
    reader.close();
}
```

```java
/**
 * This method will add a stock to the sales history for generating reports and receipts.
 * @param stock The stock to be added to the history.
 */
1 usage
public void addSalesHistory(Stock stock, int count) {
    Stock controlStock = new Stock();
    controlStock.setID(stock.getID());
    controlStock.setName(stock.getName());
    controlStock.setPrice(stock.getPrice());
    controlStock.setCount(count);
    controlStock.setUnitOfMeasure(stock.getUnitOfMeasure());
    controlStock.setManufacturer(stock.getManufacturer());
    controlStock.setLocation(stock.getLocaton());
    salesHistory.add(controlStock);
}
```

Testing Plans

Tyler Webb, Kris Heiskell, Logan McAbee

Testing

1) Run the program.
2) Enter the number 1 to enter Stock menu.
    a. Enter the number 1 to list the current stock.
        i. Expected:

1 : Apple | Count : 53 | price : $3.47 | lbs | Monkey Orchard | Produce Section #D14 | ID: 49913

---

2 : Wrench | Count : 7 | price : $7.23 | lbs | Monkey Machanics | Tools Section #F11 | ID: 34357

---

3 : Shampoo | Count : 12 | price : $7.5 | oz | Monkey Salon | Hygine Secion #T03 | ID: 56638

---

4 : Tommy Gun | Count : 387 | price : $1238.32 | lbs | Monkey With A Machine Gun Co. | Outdoors Secion #G01 | ID: 16942

Welcome to the stock tester please select from the options below (Input validation has not been implemented yet so be careful!)

    b. Enter the number 2 to add stock.
        i. Enter *Mr.Ward* for stock name.
        ii. Enter *1* for the stock count.
        iii. Enter *1.01* for the stock price.
        iv. Enter *each* for the stock unit of measure.
        v. Enter *Mrs.Ward* for the stock manufacturer.
        vi. Enter *CETAS-160A* for the stock location.
        vii. Enter *1020* for the stock ID.
    c. Enter the number 2 to list the stock and see the newly added stock item.
        i. Expected:

1 : Apple | Count : 53 | price : $3.47 | lbs | Monkey Orchard | Produce Section #D14 | ID: 49913
---
2 : Wrench | Count : 7 | price : $7.23 | lbs | Monkey Machanics | Tools Section #F11 | ID: 34357
---
3 : Shampoo | Count : 12 | price : $7.5 | oz | Monkey Salon | Hygine Secion #T03 | ID: 56638
---
4 : Tommy Gun | Count : 387 | price : $1238.32 | lbs | Monkey With A Machine Gun Co. | Outdoors Secion #G01 | ID: 16942

Test Results

Tester: Daniel Wyatt

As stated, there is no input validation.

2
- A: Pressing 1 returns the expected results.
- B: After entering stock manufacturer it skips Stock Location to Stock ID
- C: The new item does not show the name

3
- A: Works as expected

4
- A: Works as expected

5
- A: Works as expected

6
- A: Works as expected
- B: Works as expected

7
- A: Works as expected
- B: Works as expected
- C: Works as expected
- D: Works as expected

8
- A: Works as expected
- B: Works as expected
- C: Works as expected
- D: Works as expected

9
- A:
- B: Does not list the accounts until after I add an account, it does not seem to be interacting with the Accounts file at all
- C: The 2 in the menu is not in line with the rest on the numbers, works otherwise
    - 1: Works as expected
    - 2: Works as expected
    - 3: Works as expected
- D: Works as expected for the account I added
- E: I had to readd the deleted user since it doesn't list any other accounts
    - 1: works as expected
    - 2: Works as expected
    - 3: Works as expected
    - 4: Works as expected