# Clothing Store Point of Sale System
# Software Design Specification Test Plan

By Angela Lee and Anthony Kim
March 22, 2024

# 1. Software Design Specification:

## System Description:

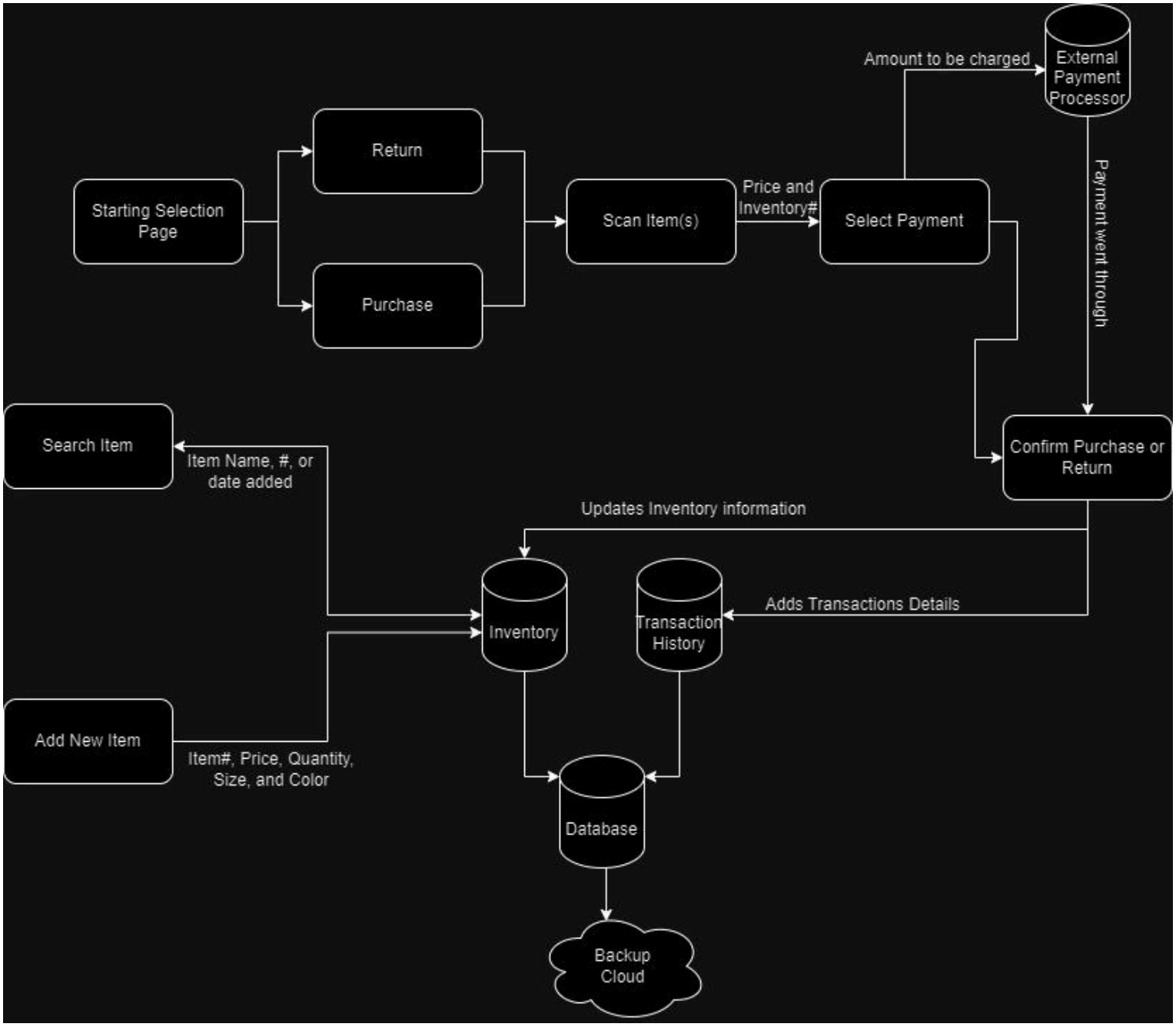
The system software design that we are specifying in this document is a clothing store point of sale system. This point of sale system should serve all of the needs of our clothing store client, as it will be designed to encompass essential functions such as inventory handling and processing transactions and returns.

The system will be integrated throughout all locations of the clothing store, and will be supported electronically on handheld devices by apple and android operating systems. All inventory information will be centrally stored in our client's cloud database, which will be synced across all store locations.

Employees will have access to inventory information and transaction/return processing capabilities, while administrators have additional access to transaction histories and data, thereby maintaining appropriate access control within the system. Customer interactions with the point of sale system will be done through the employees with access to this system, who will be able to make transactions and returns for customers and search for their desired items.

## Software Architecture Overview:

### SWA Diagram:



## SWA Diagram Description:

The Software Architecture Diagram provides a clear view of the system's components and how they interact with one another. The diagram above demonstrates how there are different processes for three main cases. These cases include returns and purchases, searching items, and adding items.

**1) Returns and Purchases:**

The process for returning and purchasing items can go through a total of 9 components.

1. Starting selection page
2. Selection of either a return or purchase. The choice is then sent to the Scan Item(s) component to signal whether the transaction is a purchase or return.
3. Scan Item(s) will read the item information (price, name, color, and size) from the barcode or the ID# of items. The data is then sent to the Select Payment component.
4. Select Payment will take data given from Scan Item(s) and send only the total price of all the items scanned to the External Payment Processors if credit/debit is selected, it then sends the rest of the scanned item's data to the Confirm Purchase or Return component. If payment was cash or a return was made, it sends Confirmation Purchase or Return the items' data and the transaction information.
5. External Payment Processor takes the data from Select Payment and charges the amount onto the credit/debit card. Once the payment has gone through it sends Confirm Purchase or Return component that the payment has been processed.
6. Confirm Purchase or Return takes the data received from Select Payment and External Payment Processor and confirms that payment has been made, once that has been done it sends all the items' data and transaction details of the purchase or return to two different components. Items' data will be sent to be stored in Inventory and transaction details will be sent to Transaction History.
7. Inventory component will take the item's details (ID#, name, color, size, price) and will add or remove the item from inventory depending on if it was a return or purchase. The Inventory Component will then send a copy of its updated data to the Database component.
8. Transaction History component will take the transaction details given from Confirm Purchase or Return and update itself. After that is done it will send a copy of its updated information to the Database component.
9. Lastly, the Database Component will send its inventory and transaction history data to the Backup Cloud.

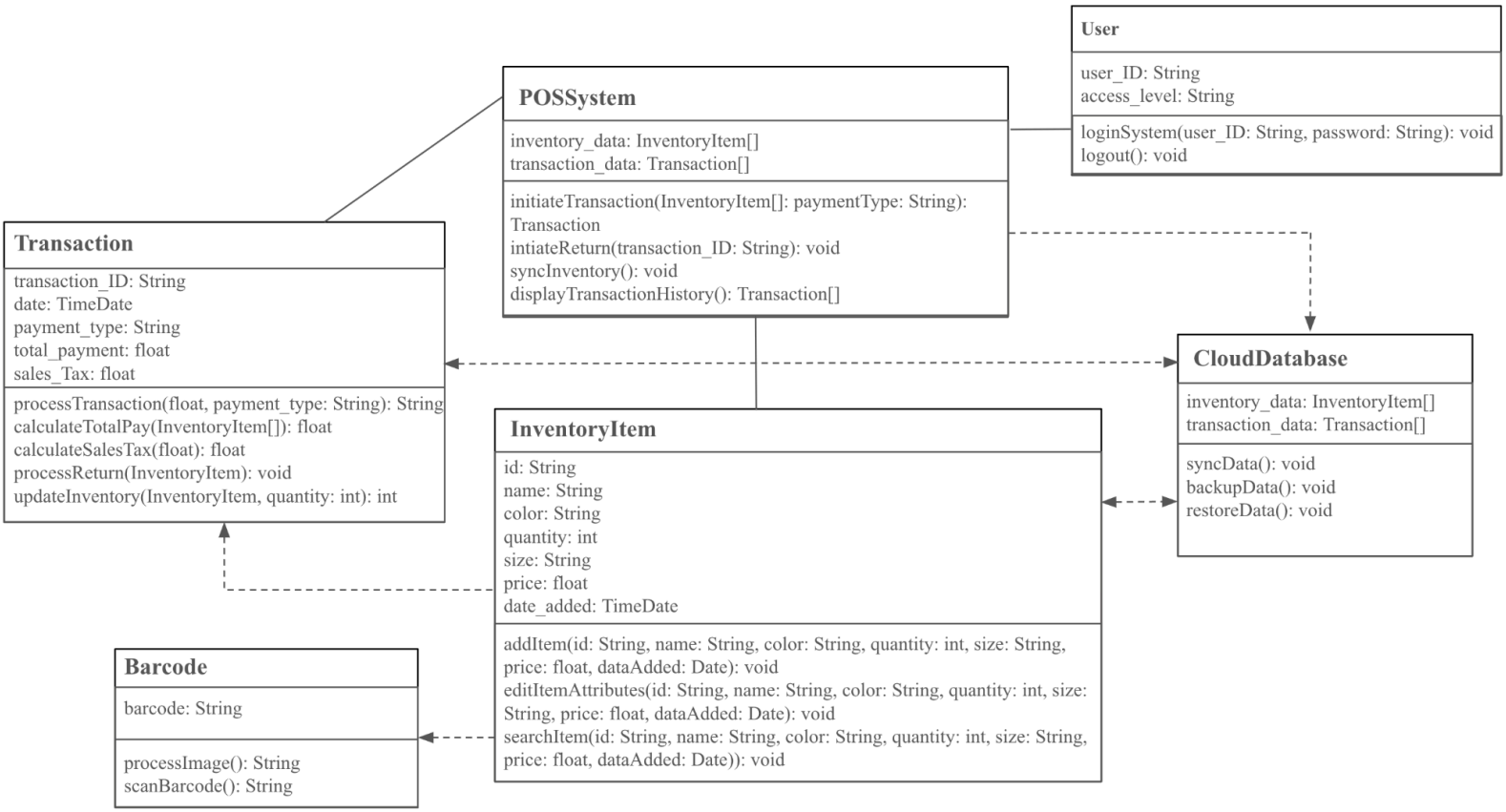
**2) Searching Items:**

1. Search Item receives an item's name, number, or date added. This data is then sent to the Inventory component.
2. Inventory then takes the data and searches through its data to find if there is a match. Inventory will then send back whether the item they are searching for is available or not to the Search Item component.

**Adding Items:**

1. Add New Item will receive the necessary data such as item#, price, quantity, size, and color. This data will then be sent to the Inventory component.
2. The Inventory component then takes the data given by Add New Item and updates itself with the new item's data.

## UML Class Diagram:



## UML Diagram Description:

**Description of Classes:**

This UML diagram illustrates six key classes of our point of sale system. This includes the Point of Sale System class (POSSystem), the User class, the Transaction class, the CloudDatabase class, the InventoryItem class, and the Barcode class.

1. The POSSystem class encompasses functionalities that involve overall system operations, and also stores inventory and transaction data. This class relies on the Transaction, User, and InventoryItem classes, and it utilizes the CloudDatabase class.
2. The CloudDataBase class is a central hub containing synchronized inventory and transaction data that is securely backed up. It has a reciprocal relationship with the Transaction and InventoryItem classes.
3. The User class defines individuals who are using the system (based on their level of access), along with their login information.
4. The Transaction class outlines how the system processes transactions and handles returns.
5. The InventoryItem class contains and manages inventory information and related functions. It utilizes the barcode class.
6. The Barcode class stores data that is necessary for scanning item barcodes.

**Description of Attributes:**

- POSSystem class:
    - inventory_data (InventoryItem[]): Inventory_data is an array of InventoryItem objects that represents the current inventory data of the system.
    - transaction_data (Transaction[]): transaction_data is an array of Transaction objects that represents recorded item transactions.
- CloudDatabase class:
    - inventory_data (InventoryItem[]): inventory_data stores inventory data/information into the client's cloud database, which allows for central data management and synching across all store locations.
    - transaction_data (Transaction[]): transaction_data holds a record of every transaction made through the point of sale system.
- User class:
    - user_ID (String): Identifies users of the POS system.
    - access_level (String): Determines level of user's access permissions within the system, depending on whether employee or administrator status.
- Transaction class:
    - transaction_ID (String): Uniquely identifies a transaction.
    - date (TimeDate): The date and time of when the transaction occurred.
    - payment_type (String): The chosen method of payment, such as credit, debit, or cash.

    - total_payment (float): The total amount that will be paid by the customer in the transaction.
    - sales_Tax (float): The sales tax amount to be added to the transaction.
- InventoryItem class:
    - id (String): A unique identifier for each and every item within the inventory.
    - name (String): The name of a clothing item.
    - color (String): The color of a clothing item.
    - quantity (int): The number of items available in stock for a particular item in the inventory.
    - size (String): The size of a clothing item.
    - price (float): The retail price of a clothing item.
    - date_added (TimeDate): The date and time when an item was added to the inventory.
- Barcode class:
    - barcode (String): The unique barcode associated with every item in the inventory.

**Description of Operations:**

- POSSystem class:
    - initiateTransaction: This begins a new item transaction using a list of items (array) and a specified payment type (String).
    - initiateReturn: Handles the return process for an item(s) from its unique transaction ID (String).
    - syncInventory: Synchronizes the inventory data across the entire point of sale system, which ensures that all locations have up-to-date information on the inventory.
    - displayTransactionHistory: Returns record of item transactions for review, which is only accessible to administrators.
- CloudDatabase class:
    - syncData: This ensures that a certain point of sale system store location's data is synchronized with the client's cloud database.
    - backupData: To maintain safety, this creates backups of the database to prevent loss of data in case of an error.
    - restoreData: Restores the data from data backups, which provides security against possible data corruption/data loss.
- User class:
    - loginSystem: Authenticates a user of the system, which will allow the user access to the system's functions after taking in their user ID (String) and password (String).

    - logout: Ends a user's session in the system, ensuring that access is secured and safely managed.
- Transaction class:
    - processTransaction: This processes the transaction by using details of the sale, such as the transaction amount (float) and payment type (String).
    - calculateTotalPay: Calculates and returns the total amount to be paid (float), including sales tax, based on the items involved in the transaction (array).
    - calculateSalesTax: Calculates and returns the sales tax amount (float) applicable to the subtotal (float) of the transaction.
    - processReturn: Handles the return process of an item by updating the inventory and the transaction history accordingly (InventoryItem).
    - updateInventory: Adjusts the inventory count (int) based on transactions and/or returns (InventoryItem) to ensure that item inventory levels are up-to-date.
- InventoryItem class:
    - addItem: Adds a new item to the inventory with details including the item's ID (String), name (String), color (String), quantity (int), size (String), and price (float). This will update the inventory data immediately.
    - editItemAttributes: Allows item attributes, such as the item ID (String), name (String), color (String), quantity (float), size (String), and price (float), to be modified, which ensures the inventory is accurate and up-to-date.
    - searchItem: Looks for an item within the inventory, which is done using attributes such as item ID (String), name (String), color (String), quantity (int), size (String), and price (float). This is useful for employees when they need to quickly look up and find items for customers.
- Barcode class:
    - processImage: Processes and returns the image of an item's barcode as a String captured by the handheld device's camera.
    - scanBarcode: Interprets the item's barcode from an image to return the associated string of characters.

**Update:

- processTransaction method was updated to return a String (transaction_ID)
- updateInventory method was updated to return an int (quantity)

## Development Plan and Timeline:

**List of tasks:**

- ☐ Designing: Angela
  - ☐ Based on the SWA and UML diagrams, we can start designing the system's software specifications. This includes working on the UI, UX, data structures and algorithms, etc.
- ☐ Development/implementation: Anthony
  - ☐ Writing the system's software code for interface and other components, developing system based on hardware requirements, identifying risks, implementing interaction protocols, working with the cloud system database
- ☐ Testing: Angela
  - ☐ Running through various unit/integration/system test cases and scenarios, verifying and validating elements of the system, identifying and correcting bugs and errors
- ☐ Overview and Deployment: Both
  - ☐ Review of the POS system final product and deployment, implementation of cloud system, software interface, maintenance of security/safety measures and data access restrictions
- ☐ Maintenance: Anthony
  - ☐ Monitor system performance and watch out for issues, carefully access system components, update and improve functionalities, adapt system to new requirements/technological advancements

**Timeline:**

**Start-2 weeks:**

The design of software based on SWA and UML diagrams should be completed with a significant amount of detail.

**1.5 months later:**

Development/implementation will be completed. This will include coding the software's various classes and connecting them to interact with each other properly.

**1 month later:**

Testing of software will be completed with detailed test cases that test each feature of the system. After test cases are completed any issues with the code will be fixed.

**5 days later:**

Overview and Deployment will be completed. Will do a thorough overview to ensure the security and reliability of the system. Once the overview is finished deployment of the system will begin.

**Deployment-Onward:**

Maintenance will begin to check on the stability and security of the system.

# 2. Verification Test Plan:

## Test Set 1 - Inventory Management:

1. **Unit Test:**

Target Feature: Updating Inventory

Input:

```
InventoryItem invt1 = new InventoryItem();
invt1.addItem(T-Shirt,Blue,1000,Large,19.99,12);
invt1.editItem(T-Shirt,Blue,100,Large,19.99,12)
if(invt1.searchItem(T-Shirt,Blue,100,Large,19.99,12)== true){
Return True
}
else
Return False
```

Expected Output:

True

Explanation:

- Test Description: This is a unit test that will validate the functionality of updating inventory after an item has been added.
- Features Tested: This test focuses mainly on all three of InventoryItem's class methods. The methods are addItem, editItemAttributes, and searchItem.
- Test Sets/Vectors:
    - Test Input Parameters:
        - [T-Shirt, Blue,1000, Large,19.99,12]
        - [T-Shirt, Blue,100, Large,19.99,12]
    - Test InventoryItem' Object Invt1
        - Used to perform addItem,editItemAttributes, and searchItem.
- Coverage: This covers InventoryItem's methods altogether as it tests the functionality of addItem, editItemAttributes, and searchItem. The test passing means that all the methods stated before are functioning as intended.

2. **Integration Test:**

Target Feature: Integration of Transaction and Inventory Item

Input:

```
InventoryItem Inventory = new InventoryItem;
Transaction trans1 = new Transaction;
Inventory.addItem(T-Shirt, Blue,1000, Large,19.99,12);
trans1.updateInventory(Inventory,500);
if(Inventory.searchItem(T-Shirt, Blue,500, Large,19.99,12))==true){
Return True;
}
Else
Return False;
```

Expected output:

True

Explanation:

- Test Description: This is an integration test that validates the functionality of updateInventory, addItem, and searchItem. This test will prove that Transaction can correctly update Inventory.
- Features Tested: This test focused on one method from Transaction and two methods from InventoryItem to test their integration with one another. The methods tested in this test were the ability to update inventory with the updateInventory method, and we tested this with addItem and searchItem.
- Test Sets/Vectors:
    - Test Input Parameters
        - T-Shirt, Blue,1000, Large,19.99,12
        - Inventory,500
        - T-Shirt, Blue,500, Large,19.99,12
    - Test InventoryItem object Inventory
        - Used to perform addItem and searchItem to create and find the updated item in inventory.
    - Test Transaction object Trans1
        - Used to perform updateInventory to integrate with InventoryItem.
- Coverage: This covered the interaction between Transaction and InventoryItem. It does so by adding an item to be tested, then updating the item through Transaction, and searching the item to see if Inventory has been updated to the correct amount.

3. **System Test:**

   Target Feature: Returning Process

   Test Case:

   - Verify Inventory Management for a Return
   - Simulate customer making a return

     An employee enters their PIN into the system. Then scan the garment's barcode or manually enter its ID number. The employee then confirms that these items are being returned. Then the system updates the inventory information and prints a receipt with the correct amount refunded to the customer. This process should have no errors or delays throughout the whole process.

   Explanation:

- Test Description: This is a system test that verifies the processing of a return from start to finish. This includes employees' putting their credentials in, inputting the item id, processing the return, and confirming the completion of the return.
- Features Tested: The test confirms the completion of the integration between POSSystem, User, Barcode, inventoryItem, Transaction, and CloudDataBase
- Test Sets/Vectors: This test involves every component of the system. The input is the employee scanning items, then processing the return, updating the inventory, which then should print a receipt that shows that the return had been completed and the proper amount of money had been refunded, and finally syncing the cloud database with the transaction and inventory data.
- Test Coverage: As stated above this test covers every component of the system. This is because the process of making a return involves all six components to make a correct and efficient return.

## Test Set 2 - Transaction Processing:

1. **Unit Test:**

   Input:

   Target Feature: Processing a purchase transaction

   Transaction processTransaction(float, payment_type: String): String

   Transaction a
   total_payment = $37.62
   payment_type = credit
   a.processTraction(total_payment, payment_type);

   if (a.processTransaction(float, payment_type: String) == transaction_ID)

   return Pass

   else
   return Fail

   Expected Output:

   Pass

Explanation:

- Test Description: This unit test is designed to validate the functionality of the processTransaction method within the Transaction class to test that it can successfully handle processing a purchase transaction.
- Features Tested: The test primarily evaluates the functionality of the processTransaction method to ensure that it accurately processes purchase transactions based on the provided parameters total_payment and payment_type
- Test Sets/Vectors:
  - Test Input Parameters:
    - total_payment: Represents the total amount of the transaction to be paid.
    - payment_type: Indicates the type of payment method used for the transaction (credit card, debit card, cash).
  - Test transaction object (a):
    - Used to perform the processTransaction method.

- Coverage: This covers the processTransaction method within the Transaction class, ensuring that the method's functionality is carefully tested. While the test primarily focuses on validating purchase transaction processing, it indirectly verifies other

important aspects as well, such as parameter handling, method invocation, and return value correctness.

2. **Integration Test:**
   Target Feature: Integration of transaction processing and inventory management
   Input:
   Transaction updateInventory(InventoryItem, quantity: int): int

   InventoryItem b
   b.id = A0123456
   b.name = Poppy dress
   b.color = red
   b.quantity = 517
   b.size = small
   b.price = 24.99
   date_added = 11.3.23
   quantity = b.quantity

   Transaction p
   if (p.updateInventory(p, 516) == quantity - 1)

   return Pass

   else
   return Fail

   Expected Output:
   Pass

Explanation:

- Description: This integration test is used to validate the integration between the transaction processing and inventory management features within the system. It focused on verifying if the updateInventory method correctly updates the quantity of an inventory item after a transaction is processed based on transaction details.
- Features Tested:
  - Integration of Transaction Processing and Inventory Management: The test evaluates how effectively the updateInventory method is integrated within the overall transaction processing flow to manage the item inventory.
  - Functionality of Inventory Update: This test ensures that the updateInventory method accurately adjusts an inventory item based on the transaction details.
- Test Sets/Vectors:
  - Test InventoryItem Object (b):

        - id: An identifier for an inventory item.
        - name: The name of an item.
        - color: The color of an item.
        - quantity: The initial quantity of a specified item in the item inventory.
        - size: The size of an item.
        - price: The price of an item.
        - date_added: The date an item was added to the item inventory.
    - Test Transaction Object (p):
        - Simulates a transaction object to perform the updateInventory method.
    - Test Item Quantity Value (516):
        - Represents the quantity value used to update an inventory item.
- Coverage: This test covers the interaction between the transaction processing and inventory management features of the system. It validates the accuracy of the updateInventory method within the context of a transaction, ensuring that data remains consistent and up-to-date.

3. **System Test:**

   Target Feature: Initiating/processing a purchase transaction

   Test Case:

   - Verify transaction processing flow for a purchase transaction.
     - Simulated user interaction:

       A customer walks up to the register to check out their items, where the employee either scans or enters the id of each of their clothing items. The total payment amount will be calculated for the customer to pay. The customer selects their payment method, and the transaction is processed. The transaction is processed successfully, the item inventory is updated, the transaction data is recorded into the system's transaction history, and the payment is processed without any errors.

   Explanation:

   - Test Description: This test simulates a complete purchase transaction. This covers everything from customer item selection, payment processing, and database synchronization. The goal of this test is to ensure that all components involved in the transaction processing flow efficiently and effectively work smoothly together.
   - Features Tested: The test verifies whether integration of the Transaction processing, Inventory management, Payment processing, and Database synchronization functions of the system is successful.
   - Test Sets/Vectors: This test involves simulating user interaction to choose items for purchasing, selecting a customer payment method, processing the payment, updating the item inventory, recording the transaction into the transaction history, and synchronizing the transaction data with the Cloud Database.
   - Coverage: The test covers the entire transaction processing flow, including item selection, payment processing, inventory management, and data synchronization, ensuring end-to-end functionality and data consistency.