Test Plan

Clothing Store Point-of-Sale System

Spring 2024 Group 7

Triet Lieu, Software Engineer

Konrad Kapusta, Software Architect

Prepared for

CS 250: Introduction to Software System

Instructor: Bryan Donyanavard

I. UNIT TESTS

SET A: Transaction Class

Target	getTotalCost()
Expectation	Transaction's cart records the Items included in the transaction. getTotalCost() should return the total of the costs of Items in the transaction, for this total cost to be charged to the customer
Procedure	Get each Item's price → sum all such prices → return calculated sum
Sample Case	Transaction's cart contains Item entries below: Item: Mens Dress Shoes — price: $60.00 Item: Womens Blouse — price: $30.00
Check Output	transaction.getTotalCost() computes (30.0 + 60.0), returns **90.0**

Target	makeSales()
Expectation	Transaction's cart records the Items included in the transaction. getTotalCost() should return the total of the costs of Items in the transaction, for this total cost to be charged to the customer
Procedure	Call verifyPayment(card) and check the return if True Call getTotalCost() to calculate the amount to charge the customer Call functions to display the total cost and wait for customer payment
Sample Case	Transaction's cart contains Item entries below: Item: Mens Dress Shoes — price: $60.00 Item: Womens Blouse — price: $30.00
Check Output	**90** in the chosen currency (eg $) is displayed to the customer

SET B: Item Class

Target	getAttributeValue()
Expectation	Item's attributes are stored in a map eg {'size': 'SM', 'color': 'red'} getAttributeValue(string attrb) should return the value paired to attrb
Procedure	Call function → Get value of key attrb in map → Return attrb's value
Sample Input	Item 'Mens Pants' has attributes {'size': 'MD', 'style': 'casual'}
Check Output	pants.getAttributeValue('style') returns string **'casual'**

Target	Constructor: Item(float price, {map: attributes})
Expectation	Create new Item profile and set its attributes member to input map and set its price member to the input price
Procedure	Call new Item(price, {map}) with known attribute keys and values Check the Item has the assigned price Check each assigned key exists in the Item's attributes and has the assigned attribute value.
Sample Input	price = 50.0, attributes = {'size': 'MD', 'style': 'casual'}
Check Output	item.getPrice() == 50.0 && item.getAttributeValue('size') == 'MD' && item.getAttributeValue('style') == 'casual'

II. INTEGRATION TESTS

SET A: Item Table in Database—Integrates with Item

Target	Purchase management system can successfully read from and write data to the Item profiles stored in the database
Expectation	The database retrieves Item profiles in the database that meets the specified selector conditions and can likewise update the values of Items if a store manager requests data updates.
Procedure	Check database access credentials If access clears, use the specified selector conditions to select Items Programmatically create objects encapsulating data and return them
Sample Case	[{ 'name': 'Mens Jacket', 'attributes': {{'size': 'MD'}}}, {name': 'Womens Blouse', 'attributes': {{'size': '9'}}}] in Item Table. SELECT * FROM ITEM WHERE name LIKE "%Men%"
Check Output	1 Item instance: name == "Mens Jacket", attributes['size'] == 'MD'

SET B: Inventory Table in Database—Integrates with Transaction

Target	Transaction processing system can successfully read from and write data to Inventory Table in the database
Expectation	When a transaction order processes an Item purchase, the count of that Item in a Store's Inventory should be reduced. When a return is processed, the Item's count should be increased
Procedure	Transaction processing system makes sales and verifies customer's payment via card or manual input from cashier Transaction processor sends the list of Items to the Inventory Table Inventory Table deduct 1 from the count values of passed Items
Sample Case	[{ 'name': 'Mens Jacket', 'ID': 2, count: 6}, { 'name': 'Womens Blouse', 'ID': 4, count: 7}] UPDATE Inventory SET count = count – 1 WHERE ID IN (2, 4)
Check Output	[{ 'name': 'Mens Jacket', 'ID': 2, count: **5**}, { 'name': 'Womens Blouse', 'ID': 4, count: **6**}]

III. SYSTEM TESTS

SET A: Transaction—Card Payment Scan && Verify—Inventory Update

Target	Process an item's purchase and payment and if payment passes, update the count of that item in inventory
Expectation	Transaction's cart records the Items included in the transaction. getTotalCost() should return the total of the costs of Items in the transaction, for this total cost to be charged to the customer
Procedure	Display the total cost to the customer and request payment If a customer chooses to pay via card, scan that card Verify the card by making API request to card processor service If verification passes, reduces count of items in transaction by 1 each
Sample Case	transaction.cart: "Mens Jacket--$50", "Womens Blouse--$60" Call requestPayment and if card is provided, verifyPayment(card)
Check Output	If card passes verification, check that counts of "Mens Jacket" and "Womens Blouse" are both each 1 less than prior to transaction

SET B: Item Barcode Scan—Display Price to Output Device

Target	Integrates physical device to identify item via barcode and output device such as a screen that can receive price data and show it
Expectation	A connected barcode scanner passes visual data to be processed by the system. System passes along price to screen to show to customer
Procedure	Scanner scans the purchased item's barcode Scan processing system get scan and gets corresponding item profile Purchase management system gets price of that Item in database Purchase management system sends item's name and price to screen Screen displays the item's name and price
Sample Case	Assigned barcode image for {'name': 'Mens Jacket', 'price': 50.0}
Check Output	Item matched is 'Mens Jacket'. Price of Mens Jacket' is 50.0. Screen displays "Mens Jacket—$50.00"