16-4-2023

Test cases Integrative Task II

Daron Mercado,
Juan José Barrera,
Alexis Jaramillo
ICESI UNIVERSITY

Number	Class	Scenario
SetupStage1	ShopAppTest	An object in the ShopApp class with no products or orders.
SetupStage2	ShopAppTest	An object of the ShopApp class with no orders and two products. The products: 1. name=” Teclado gamer”, description=”Keyboard just for the pros”, price=100000, quantity=20, category=ELECTRONIC, purchaseTimes=12. 2. name=”HIT mango pet 500ml”, description=”Colombian 100% juice”, price=2500, quantity=15, category= FOOD_AND_DRINK, purchaseTimes=120.
SetupStage3	ShopAppTest	*An object of the ShopApp class with one order and three products: 1. name=”Camiseta Polo”, description=”Very elegant”, price=30000, quantity=15, c=category CLOTHES_AND_ACCESORIES, purchaseTimes=10. 2. name=”Marcador Sharpie”, description=”Draw the future”, price=2500, quantity=8, category= STATIONERY, purchaseTimes=24 3. name=”Agua 500 ml”, description=”Refreshing”, price=2500, quantity=15, category=FOOD_AND_DRINK, purchaseTimes=14.
SetupStage4	ShopAppTest	*An object of the ShopApp class with two orders and three products. Orders: 1. buyerName="Moon", productList="[Leggins, Pokemon T-shirt]", totalPrice=52000. purchaseDate="2023-04-02" 2. buyerName="Mariana", productList="[Leggins]", totalPrice=25000, purchaseDate="2023-04-02" The products: 1. name=”Leggins”, description=”Be cool and fresh”, price=”25000”, quantity=25, category=CLOTHES_AND_ACCESORIES, purchaseTimes=8. 2. name=”Soccer ball”, description=”¡Nothing is better!”, price=30000, quantity=22, category=SPORTS, purchaseTimes=15. 3. name=”Pokemón T-shirt”, description=”Let’s catch them”, price=27000, quantity=12, category= CLOTHES_AND_ACCESORIES, purchaseTimes=14.

Requirement 1: Register Product

Test Objective: Verify that the "registerProduct" method correctly registers different products to the ShopApp. The program will launch an exception in case non-numerical entries are entered in numerical fields or in case you want to register an already existing product.

Class	Method	Scenario	Input Values	Expected result
ShopApp	registerProduct	SetupStage1	name="PlayStation 8", description="The mythical PS8", price=8000000 quantity=5 category=ELECTRONIC purchaseTimes=8	The application now has a product called "PlayStation 8" with its various properties.
ShopApp	registerProduct	SetupStage1	name="Imagine Dragons T-shirt", description="Fell the power", price=70000 quantity=XXXX category=CLOTHES_AND_ACCESORIES purchaseTimes=45	The number of products remains the same and the program throws an exception saying that a letter was typed where a number was requested.
ShopApp	registerProduct	SetupStage2	name="Gamer keyboard" description="New keyboard", price=30000 quantity=20 category=ELECTRONIC purchaseTimes=14	The number of products remains the same and the program throws an exception saying that a product with that name already exists.

Requirement 2: Register Order

Test Objective: Verify that the "registerOrder" method correctly registers orders to the application and subtracts the number of available units of a product. In case a product from the os product listis not found, the program will throw an exception.

Class	Method	Scenario	Input Values	Expected result
ShopApp	registerOrder	SetupStage3	buyerName="Alexander" productList="[Water 500ml","Polo L T-shirt", "Sharpie Marker]" totalPrice=35000 purchaseDate=”2020-10-22”	The application now has an order in the name of "Alejandro".
ShopApp	registerOrder && getotalPriceProductQuantity	SetupStage3	buyerName="Alexander" productList="[Water 500ml","Polo T-shirt", "Sharpie Marker]", totalPrice=35000 purchaseDate=”2020-10-22”	The products of "Water 500 ml", "Polo T-shirt" and "Sharpie marker" have one less unit.
ShopApp	registerOrder	SetupStage4	buyerName=”Samuel”, productList=“[PC Gamer, Soccer Ball]”, totalPrice=2660000, purchaseDate=”2014-09-16”	The number of orders remains the same and the program throws an exception saying that an ordered product is not found.

*The “getProductQuantity” method is tested with any of the three products sold to verify that their number of available units has actually decreased.

Requirement 3: Delete product

Test Objective: Verify that the "deleteProduct" method deletes a product successfully. In case the name of the product to be removed does not exist, the program will throw an exception.

Class	Method	Scenario	Input Values	Expected result
ShopApp	deleteProduct	SetupStage1	productName=”Leche”	The number of products remains the same and the program throws an exception saying that the product is not found.
ShopApp	deleteProduct	SetupStage2	productName=”HIT mango pet 500ml”	The number of products available is reduced to one.

Requirement 4: Increase the quantity of an already registered product.

Test Objective: Verify that the "increaseProductQuantity" method satisfactorily increases the quantity of a product. If the quantity of product to be removed is negative or if the product does not exist, the program will throw an exception.

Class	Method	Scenario	Input Values	Expected result
ShopApp	IncreaseProductQuantity	SetupStage2	productName=”Teclado Gamer” quantityToIncrease=20	Now the product " Gamer Keyboard" has 40 units available.
ShopApp	IncreaseProductQuantity	SetupStage2	productName=”HIT mango pet 500ml” quantityToIncrease=-5	The product " Gamer Keyboard" is left with the same units (20) and the program launches an exceptionsaying that they cannot add negative quantities.
ShopApp	IncreaseProductQuantity	SetupStage2	productName=”Crema dental Colgate” quantityToIncrease=10	The program will launch saying that the product to increase quantity is not registered.

Requirement 5: Search Product

Test Objective: Verify that the searchProductByName, searchProductByPrize, searchProductByCategory and search ProductByNumberOfTimesPurchase methods allow you to search for one or more products. The program will throw an exception in case an unexpected entry is entered in a field or if the product is not found.

Class	Method	Scenario	Input Values	Expected result
ShopApp	searchProductByName	SetupStage2	name=”HIT mango pet 500 ml”	The product is found.
ShopApp	searchProductByName	SetupStage2	name="Chicharrón dulce"	The program throws an exception saying that the product is not found.
ShopApp	searchProductByPrice	SetupStage3	price=2500	There are 2 products ("Sharpie Marker" and "Water 500 ml").
ShopApp	SearchProductByPrice	SetupStage3	price=-2000	The program throws an exception saying that there are no products with negative price.
ShopApp	searchProductByCategordy	SetupStage3	category=CLOTHES_AND_ACCESORIES	The product is found ("Polo T-shirt").
ShopApp	searchProductByNumberOfTimesPurchase	SetupStage3	numberOfTimesPurchase= 14	The product is found ("Water 500 ml").

Requirement 6: Search Order

Test Objective: Verify that the searchOrderByBuyerName, searchOrderByTotalPriceOfOrder and searchOrderByPurchaseDate methods allow you to search for one or more orders. In case of not finding it, the program will throw an exception.				
Class	**Method**	**Scenario**	**Input Values**	**Expected result**
ShopApp	SearchOrderByBuyerName	SetupStage4	name=”Luna”	The order was found.
ShopApp	SearchOrderByBuyerName	SetupStage4	name=”Francisco”	The program will throw an exception saying that the product was not found.
ShopApp	SearchOrderByTotalPriceOfOrder	SetupStage4	totalPrice=25000	The order was found (in the name of "Mariana")
ShopApp	SearchOrderByPurchaseDate	SetupStage4	purchaseDate=”2023-08-02”	There are two orders (one in the name of "Luna" and the other in the name of "Mariana")

Requirement 7: Filter numeric values

Test Objective: Verify that the filterProductByPrices, filterProductByQuantity, filterProductByPurchase, and filterOrdersByTotalPrice methods are working correctly. The program will throw exceptions in case negative values are entered, no product is found in the filter range or if the maximum value is less than the minimum.				
Class	**Method**	**Scenario**	**Input Values**	**Expected result**

ShopApp	filterProductByPrice	SetupStage3	minimum = 10000
			maximum = 50000
				The method is expected to find the product "Polo T-shirt".

ShopApp	filterProductByPrice	SetupStage3	minimum = 10000
			maximum = 2500
				The program is expected to throw an exception that says the maximum range is less than the minimum.

ShopApp	filterProductByQuantity	SetupStage3	minimum = 15
			maximum = 25
				There are two products: "Water 500 ml" and " Polo Shirt".

ShopApp	filterProductByQuantity	SetupStage3	minimum = 30
			maximum = 50
				The program is expected to throw an exception and say that no product was found.

ShopApp	filterProductByPurchase	SetupStage3	minimum = 10
			maximum = 15
				There are two products: "Water 500 ml" and " Polo Shirt".

ShopApp	filterProductByPurchase	SetupStage3	minimum = 15
			maximum = 15
				The program is expected to throw an exception saying no product was found.

ShopApp	filterOrderBy TotalPrice	SetupStage4	minimum = 15000
			maximum = 30000
				An order is found (in the name of "Mariana").

ShopApp	filterOrderBy TotalPrice	SetupStage4	minimum = -2000
			maximum = 0
				The program is expected to throw an exception saying it can't be filtered into negative ranges.

Requirement 8: Filter Strings

Test Objective: Verify that the filterProductByName, filterProductByDescription, filterProductByCategory, filterOrderByBuyerName methods work correctly. The program will throw exceptions in case values other than chars are entered, no product is found in the filter range or if the initial letter is greater in ASCII value than the final letter.

Class	**Method**	**Scenario**	**Input Values**	**Expected result**

ShopApp	filterProduct ByName	SetupStage3	startingLetter='a'
			endingLetter = 'd'
				The method is expected to find the product "Polo T-shirt".

ShopApp	filterProduct ByName	SetupStage3	startingLetter='z'
			endingLettter='z'
				The program is expected to throw an exception and say that no product was found.

ShopApp	filterProduct ByDescriptio n	SetupStage2	startingLetter='c'
			endingLetter='l'
				You will find a product: "HIT mango pet 500ml"

ShopApp	filterProduct ByCategory	SetupStage2	startingLetter='e'
			endingLetter='f'
				A "Gamer Keyboard" product is found

ShopApp	filterProduct ByCategory	SetupStage2	startingLetter="12"
			endingLetter="#k"
				The program is expected to throw an exception saying that the input is invalid.

ShopApp	filterOrderBy BuyerName	SetupStage4	startingLetter='l'
			endingLetter='m'
				There are two orders, one in the name of "Luna" and the other in the name of "Mariana".

ShopApp	filterOrderBy BuyerName	SetupStage4	startingLetter = 'z'
			endingLetter = 'c'
				The program is expected to throw an exception saying that the final letter has a higher ASCII value than the initial letter.

Requirement 9: Order results.

Test Objective: Verify that the sortAscending and sortDescending methods sort the searches correctly.

For this requirement, it must be taken into account that the entry of its operation is conditioned to the filters that the user selected for the search ofhis object (Product / Order). That is, the method cannot work without having received values filtered with previously used methods.

Class	Method	Scenario	Input Values	Expected result

ShopApp	filterProduct ByPrice && sortAscending	Setup Stage1	**Previous filterProductByPrice** entries
			Milk = 5000
			Rice = 10000
			Beans = 2500
			Lentils = 500
			
			Order selection:
			sortAscending = 1 (Ascendingly)
				It is expected that when receiving the filters made byrevised, and that the user has selected the Ascending Sort option, the program will display the following:
				
				Lentils = 500
				Beans = 2500
				Milk = 5000
				Rice = 10000
				
				Organizing the products in an ascending manner according to the price registered in the system.

ShopApp	filterProduct ByName && sortDescending	Setup Stage 2	**Previous filterProductByName** entries Milk Rice Beans Lentils **Order selection:** sortDescending = 2 (Descending)	It is expected that upon receiving the filters previously made, and that the user has selected the Descending Sort option, the program will display the following: Lentils Milk Beans Rice Organizing the products in a descending way according to the name registered in the system.

ShopApp	filterProduct ByQuantity && sortAscending	Setup Stage 3	**Previous filterProductByQuantity** entries
			Milk = 3
			Rice = 8
			Beans = 1
			Lentils = 6
			
			Order selection:
			sortAscending = 1 (Ascendingly)
				It is expected that upon receiving the filters previously made, and that the user has selected the Ascending Sort option, the program will display the following:
				
				Beans = 1
				Milk = 3
				Lentils = 6
				Rice = 8
				Organizing the products in an ascending manner according to the quantity registered in the system.

ShopApp	filterProduct ByPrice && sortRandom	Setup Stage 4	**Previous filterProductByPrice** entries
			Milk = 5000
			Rice = 10000
			Beans = 2500
			Lentils = 500
			
			Order selection:
			sortRandom = 3
			(It is an option not available in the system and the user enters it)
				Inthis case, an exception is expected to jump since the system does not have such an option available, so a message will jump to the user, warning about the error, and giving him the opportunity to choose a valid option in the program.

Requirement 10: Save information.

Objective of the Test: Verify that the storageSystem object of the JSON class correctly stores all the information of products and orders registered by the user of the system; in addition, that it is updated every time the user increases, deletes the quantities of a product.				
Class	**Method**	**Scenario**	**Input Values**	**Expected result**

ShopApp	saveProduct Lists	Setup Stage1	ArrayList <productList> = productList
			
			Product 1:
			name = Coca-Cola
			price = 3000
			quantity = 5
			category = FOOD_AND_DRINK
			
			Product 2:
			name = Pearl necklace
			price = 10000
			quantity = 2
			category = CLOTHES_AND_ACCESORIES
			
			Product 3:
			Name: Blender
			price = 50000
			quantity = 1
			category = ELECTRONIC
				The method is expected to save the 3 products within the product list, and to store them in a JSON file to achieve persistence within the program
				
				No exception will be thrown as all recorded data runsto its data type, and all information entered is allowed by the JSON object.

ShopApp	saveOrders	Setup Stage2	ArrayList <orderList> = orderList Buyer1: name = Matias	The method is expected to save the 3 buyers within the list of buyers, and to store them in a JSON file to achieve persistence within the

			productList = List001
			totalPrice = 50000
			purchaseDate = 15/03/2022
			
			Buyer2:
			name = Laura
			productList = List002
			totalPrice = 20000
			purchaseDate = 07/12/2022
			
			Buyer3:
			name = Tomas
			productList = List003
			totalPrice = 750000
			purchaseDate = 22/07/2022
				program
				
				No exception will be thrown as all recorded data corresponds to your data type, and all information entered is allowed by the JSON object.

ShopApp	saveProducts	Setup Stage 3	name = Dragon handle
			price = Of the thousand
			quantity = 1
			category = CLOTHES_AND_ACCESORIES
				An exception is expected to be thrown indicating to the user that the price entered is not in the correct format, and asks him to enter again an allowed value, i.e. numeric. In this way it is verified that all the data entered is valid for the JSON object and can be saved correctly.

ShopApp	saveOrders	Setup Stage 4	name = Paula
			productList = List007
			totalPrice = 1500000
			purchaseDate = May 22 , 2022
				An exception is expected to be thrown indicating to the user that the returned dateis not in the correct format, and prompts him to re-enter an allowed format, i.e. DD/MM/YY type. In this way it is verified that all the data entered is valid for the JSON object and can be saved correctly.