Method	**#**	**Test Description**	**Sample Input Data**	**Expected Output**	**Actual Output**	**P/F**
Item						
Item()	1	Test the default constructor.	N/A	An Item object with default values for all fields.	An Item object with default values for all fields.	Pass
Item(String name, float calorieCount, float price)	2	Test the parameterized constructor.	("Test Item", 100.0f, 10.5f)	An Item object with the provided name, calorie count, and price; unique set to true, and quantity set to 0.	An Item object with the provided name, calorie count, and price; unique set to true, and quantity set to 0.	Pass
getItemName()	3	Test the getItemName() method to retrieve the item's name.	An Item object with name set to "Test Item".	"Test Item"	"Test Item"	Pass
setItemName(String name)	4	Test the setItemName() method to change the item's name.	An Item object with name set to "Test Item". New name: "Updated Item".	The Item object with name updated to "Updated Item".	The Item object with name updated to "Updated Item".	Pass
getCalorieCount()	5	Test the getCalorieCount() method to retrieve the item's calorie count.	An Item object with calorie count set to 150.0f.	150.0f	150.0f	Pass
setCalorieCount(float calorieCount)	6	Test the setCalorieCount() method to change the item's calorie count.	An Item object with calorie count set to 150.0f. New calorie count: 200.0f.	The Item object with calorie count updated to 200.0f.	The Item object with calorie count updated to 200.0f.	Pass

isUnique()	7	Test the isUnique() method to check if the item is unique.	An Item object with unique set to true.	true	true	Pass
setUnique(boolean unique)	8	Test the setUnique() method to change the uniqueness of the item.	An Item object with unique set to true. New unique value: false.	The Item object with unique updated to false.	The Item object with unique updated to false.	Pass
getPrice()	9	Test the getPrice() method to retrieve the item's price.	An Item object with price set to 5.99f.	5.99f	5.99f	Pass
setPrice(float price)	10	Test the setPrice() method to change the item's price.	An Item object with price set to 5.99f. New price: 7.49f.	The Item object with price updated to 7.49f.	The Item object with price updated to 7.49f.	Pass
getQuantity()	11	Test the getQuantity() method to retrieve the item's quantity.	An Item object with quantity set to 10.	10	10	Pass
setQuantity(int quantity)	12	Test the setQuantity() method to change the item's quantity.	An Item object with quantity set to 10. New quantity: 5.	The Item object with quantity updated to 5.	The Item object with quantity updated to 5.	Pass
equals(Object obj)	13	Test the equals() method to check if two items are equal based on their names.	Item object A with name "Test Item" and Item object B with name "Test Item".	true	true	Pass
hashCode()	14	Test the hashCode() method to generate a hash code based on the item's name.	An Item object with name "Test Item".	An integer hash code.	An integer hash code.	Pass

Maintenance						
Maintenance(RVM vending)	1	Test the constructor.	Create an RVM object.	A Maintenance object with the specified RVM instance.	A Maintenance object with the specified RVM instance.	Pass
restockItems(String itemName, JTextArea displayArea)	2	Test restocking an existing item.	An existing item named "Soda" in the vending machine.	Display shows: "Item restocked: Soda"	Display shows: "Item restocked: Soda"	Pass
restockItems(String itemName, JTextArea displayArea)	3	Test restocking a non-existing item.	A non-existing item named "Chips" in the vending machine.	Display shows: "Item not found!"	Display shows: "Item not found!"	Pass
setPrice(Item item, float price, JTextArea displayArea)	4	Test setting the price of an item.	An existing item named "Soda" with price $1.25. Set price to $1.50.	Display shows: "Item: Soda - Price: 1.50"	Display shows: "Item: Soda - Price: 1.50"	Pass
collectMoney(float amount, JTextArea displayArea)	5	Test collecting money from the vending machine.	Vending machine has $50.00 in total. Collect $10.00.	Display shows: "Money collected: 10.00"	Display shows: "Money collected: 10.00"	Pass
replenishMoney(float amount, JTextArea displayArea)	6	Test replenishing money in the vending machine.	Vending machine has $50.00 in total. Replenish with $20.00.	Display shows: "Money replenished: 20.00"	Display shows: "Money replenished: 20.00"	

RVM						
RVM(int capacity)	1	Test the constructor	Create an RVM Object	An RVM object is now made with the constructor’s specifications	True	Pass
addItem(Item item, int capacity, JTextArea displayArea)	2	Test the addItem() method	Apple with a capacity of 5	Item will be added to the array of items. Shows “Item added”	Item added.	Pass
setItem(Item item, int capacity, JTextArea displayArea)	3	Test the setItem() method	Banana with a capacity of 5	Item will be set to the array of items. Shows “Item set”	Item set.	Pass
deleteItem((Item item, JTextArea displayArea)	4	Test the deleteItem() method	Apple with a capacity of 5	Item will be deleted from the array of items. Shows “Item deleted.” Nullifies its existence and sets it to zero.	Item deleted.	Pass.
displayItem(String itemName, JTextArea displayArea)	5	Test the displayItem() method	ItemName is Orange	Displays the details of the specified item.	ITEM : Orange PRICE : 10 CALORIES : 100 QUANTITY : 5	Pass
displayInventory(JTextArea displayArea)	6	Test the displayInventory() method	N/A		Starting Inventory: Item: Apple, Quantity: 5	Pass

					Ending Inventory: Item: Apple, Quantity: 5	
getItemQuantity(Item item)		Test the getItemQuantity() method	Item: Banana	Correct quantity for the specified item	5	Pass
Slot						
Slot(int capacity)	1	Test the constructor.	Create a Slot object with capacity of 10.	A Slot object with the specified capacity.	A Slot object with the specified capacity.	Pass
setItems(Item item)	2	Test setting a regular item in the slot.	Create a Slot object. Set a regular item "Soda" in the slot.	The Slot object with the regular item "Soda" set.	The Slot object with the regular item "Soda" set.	Pass
getItems()	3	Test getting the regular item from the slot.	Create a Slot object with a regular item "Chips" in the slot.	The regular item "Chips".	The regular item "Chips".	Pass
setSpecialItem(SpecialItem specialItem)	4	Test setting a special item in the slot.	Create a Slot object. Set a special item "DiscountCoupon" in the slot.	The Slot object with the special item "DiscountCoupon" set.	The Slot object with the special item "DiscountCoupon" set.	Pass

getSpecialItem ()	5	Test getting the special item from the slot.	Create a Slot object with a special item "GiftCard" in the slot.	The special item "GiftCard".	The special item "GiftCard".	Pass
getQuantity()	6	Test getting the current quantity of items in the slot.	Create a Slot object with a quantity of 8 items.	The current quantity of items in the slot (8).	The current quantity of items in the slot (8).	Pass
setQuantity(int quantity, JTextArea displayArea)	7	Test setting the quantity of items in the slot within capacity.	Create a Slot object with a capacity of 20. Set quantity to 10 using displayArea JTextArea.	Display shows: No error message.	Display shows: No error message.	Pass
setQuantity(int quantity, JTextArea displayArea)	8	Test setting the quantity of items in the slot exceeding capacity.	Create a Slot object with a capacity of 5. Set quantity to 8 using displayArea JTextArea.	Display shows: "You exceeded the capacity! Try again."	Display shows: "You exceeded the capacity! Try again."	Pass
decrementQuantity(int quantity, JTextArea displayArea)	9	Test decrementing the quantity of items in the slot with available quantity.	Create a Slot object with a quantity of 15 items. Set quantity to decrement to 5 using displayArea JTextArea.	Display shows: "Insufficient quantity available for item: Chips"	Display shows: "Insufficient quantity available for item: Chips"	Pass
decrementQuantity(int quantity, JTextArea displayArea)	10	Test decrementing the quantity of items in the slot without available quantity.	Create a Slot object with a quantity of 5 items. Set quantity to decrement to 8 using displayArea JTextArea.	Display shows: No error message.	Display shows: No error message.	Pass

SpecialItem						
SpecialItem(String name, String code, float calorieCount, float price, Item[] itemList, Item[] specialIngredients, int[] quantities, List<String> cookingSteps)	1	Test the constructor with valid inputs.	Create a SpecialItem object with valid parameters: name, code, calorieCount, price, itemList, specialIngredients, quantities, cookingSteps.	A SpecialItem object with the specified attributes is created.	A SpecialItem object with the specified attributes is created.	Pass
SpecialItem(String name, String code, float calorieCount, float price, Item[] itemList, Item[] specialIngredients, int[] quantities, List<String> cookingSteps)	2	Test the constructor with invalid special ingredients.	Create a SpecialItem object with invalid special ingredients not registered as items in the vending machine.	IllegalArgumentException with appropriate error message is thrown.	IllegalArgumentException with appropriate error message is thrown.	Pass
displayCookingSteps(JTextArea displayArea)	3	Test displaying cooking steps for a SpecialItem.	Create a SpecialItem object with a list of cooking steps.	The cooking steps are displayed in the provided JTextArea.	The cooking steps are displayed in the provided JTextArea.	Pass

addSpecialIngredient(String specialIngredient)	4	Test adding a special ingredient to a SpecialItem.	Create a SpecialItem object. Add a special ingredient "Spices".	The special ingredient "Spices" is added to the SpecialItem.	The special ingredient "Spices" is added to the SpecialItem.	Pass
getSpecialIngredient()	5	Test getting the special ingredient of a SpecialItem.	Create a SpecialItem object with a special ingredient "Herbs".	The special ingredient "Herbs" is retrieved.	The special ingredient "Herbs" is retrieved.	Pass
getIngredients()	6	Test getting the map of ingredients and their quantities for a SpecialItem.	Create a SpecialItem object with a list of special ingredients and their quantities.	The map of ingredients with their quantities is retrieved.	The map of ingredients with their quantities is retrieved.	Pass
getName()	7	Test getting the name of a SpecialItem.	Create a SpecialItem object with the name "Special Burger".	The name "Special Burger" is retrieved.	The name "Special Burger" is retrieved.	Pass
getCode()	8	Test getting the code of a SpecialItem.	Create a SpecialItem object with the code "SPECIAL123".	The code "SPECIAL123" is retrieved.	The code "SPECIAL123" is retrieved.	Pass
setName(String name)	9	Test setting the name of a SpecialItem.	Create a SpecialItem object. Set the name to "New Special Burger".	The name of the SpecialItem is updated to "New Special Burger".	The name of the SpecialItem is updated to "New Special Burger".	Pass
getCalorieCount()	10	Test getting the calorie count of a SpecialItem.	Create a SpecialItem object with a calorie count of 500.	The calorie count 500 is retrieved.	The calorie count 500 is retrieved.	Pass
setCalorieCount(float calorieCount)	11	Test setting the calorie count of a SpecialItem.	Create a SpecialItem object. Set the calorie count to 600.	The calorie count of the SpecialItem is updated to 600.	The calorie count of the SpecialItem is updated to 600.	Pass

getPrice()	12	Test getting the price of a SpecialItem.	Create a SpecialItem object with a price of 12.99.	The price 12.99 is retrieved.	The price 12.99 is retrieved.	Pass
setPrice(float price)	13	Test setting the price of a SpecialItem.	Create a SpecialItem object. Set the price to 15.49.	The price of the SpecialItem is updated to 15.49.	The price of the SpecialItem is updated to 15.49.	Pass
isUnique()	14	Test checking if a SpecialItem is unique.	Create a SpecialItem object with uniqueness set to true.	The SpecialItem is unique.	The SpecialItem is unique.	Pass
setUnique(boolean unique)	15	Test setting the uniqueness of a SpecialItem.	Create a SpecialItem object. Set the uniqueness to false.	The uniqueness of the SpecialItem is updated to false.	The uniqueness of the SpecialItem is updated to false.	Pass
containsIngredients(List<Item> itemList)	16	Test checking if a list of items contains all ingredients of a SpecialItem.	Create a SpecialItem object with a list of special ingredients. Create a list of items containing all the special ingredients and some additional items.	True is returned, indicating all the ingredients are present.	True is returned, indicating all the ingredients are present.	Pass
SpecialMaintenace						
SpecialMaintenance(SVM specialVending)	1	Test the constructor with a valid SVM instance.	Create an SVM object. Create a SpecialMaintenance object with the SVM instance.	A SpecialMaintenance object is created with the specified SVM.	A SpecialMaintenance object is created with the specified SVM.	Pass

restockItems(String itemName, JTextArea displayArea)	2	Test restocking an item that exists in the SVM.	Create an SVM object with some items. Call restockItems("Soda", displayArea) on the SpecialMaintenance object.	Display shows: "Item restocked: Soda"	Display shows: "Item restocked: Soda"	Pass
restockItems(String itemName, JTextArea displayArea)	3	Test restocking an item that does not exist in the SVM.	Create an SVM object with some items. Call restockItems("Juice", displayArea) on the SpecialMaintenance object.	Display shows: "Item not found!"	Display shows: "Item not found!"	Pass
setPrice(SpecialItem item, float price, JTextArea displayArea)	4	Test setting the price for a special item in the SVM.	Create an SVM object with some special items. Call setPrice(item, 10.99, displayArea) on the SpecialMaintenance object.	Display shows: "Item: Special Burger - Price: 10.99"	Display shows: "Item: Special Burger - Price: 10.99"	Pass
collectMoney(float amount, JTextArea displayArea)	5	Test collecting money from the SVM.	Create an SVM object with some money in it. Call collectMoney(50.0, displayArea) on the SpecialMaintenance object.	Display shows: "Money collected: 50.0"	Display shows: "Money collected: 50.0"	Pass
replenishMoney(float amount, JTextArea displayArea)	6	Test replenishing money in the SVM.	Create an SVM object. Call replenishMoney(100.0, displayArea) on the SpecialMaintenance object.	Display shows: "Money replenished: 100.0"	Display shows: "Money replenished: 100.0"	Pass

Transaction						
public int getID()	1	Get ID of a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	1001	1001	P
	2	Get ID of a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	1002	1002	P
	3	Get ID after initializing with invalid ID	Transaction(-1, specialItem2, 1, 50.0f, 0.0f)	-1	-1	P
public Item getItem()	1	Get regular item from a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	item1	item1	P
	2	Get regular item from a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	null	null	P
	3	Get regular item after initializing with null item	Transaction(1003, null, 0, 0.0f, 0.0f)	null	null	P
public SpecialItem getSpecialItem ()	1	Get special item from a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	null	null	P
	2	Get special item from a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	specialItem1	specialItem1	P
	3	Get special item after initializing with null special item	Transaction(1003, null, 0, 0.0f, 0.0f)	null	null	P

public int getQuantity()	1	Get quantity of a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	2	2	P
	2	Get quantity of a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	1	1	P
	3	Get quantity after initializing with zero quantity	Transaction(1003, null, 0, 0.0f, 0.0f)	0	0	P
public float getPayment()	1	Get payment amount of a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	50.0f	50.0f	P
	2	Get payment amount of a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	80.0f	80.0f	P
	3	Get payment amount after initializing with zero payment	Transaction(1003, null, 0, 0.0f, 0.0f)	0.0f	0.0f	P
public float getChange()	1	Get change amount of a regular item purchase	Transaction(1001, item1, 2, 50.0f, 10.0f)	10.0f	10.0f	P
	2	Get change amount of a special item purchase	Transaction(1002, specialItem1, 1, 80.0f, 20.0f)	20.0f	20.0f	P
	3	Get change amount after initializing with zero change	Transaction(1003, null, 0, 0.0f, 0.0f)	0.0f	0.0f	P
public void displayTransactionHistory(JTextArea displayArea)	1	Display transaction history with multiple entries	Transaction(1001, item1, 2, 50.0f, 10.0f) - TransactionHistory with entries	Expected history as StringBuilder	Actual history displayed	P

	2	Display transaction history with no entries	Transaction(1002, item2, 0, 0.0f, 0.0f) - TransactionHistory with no entries	Empty StringBuilder	Empty output displayed	P
	3	Display transaction history after initializing with null JTextArea	Transaction(1003, item3, 1, 100.0f, 0.0f) - TransactionHistory with entries	Expected history as StringBuilder	Nothing displayed (null output)	F