Minorist Store Manager (MinSMa)

LAURA DANIELA MARTÍNEZ ORTÍZ - A00365187

ARIEL EDUARDO PABÓN BOLAÑOS - A00368734

30 DE MAYO DE 2021

ALGORITMOS Y PROGRAMACIÓN II

UNIVERSIDAD ICESI

1. REQUERIMIENTOS FUNCIONALES.

RF 1. Crear una cuenta. El programa debe permitir a los usuarios la creación de una cuenta de tipo administrador, vendedor o cliente. Todas las cuentas tienen un nombre de usuario, una contraseña y una imagen de perfil.

- ***RF 1.1 Crear cuenta del administrador.*** Se debe permitir la creación de una cuenta de tipo administrador que va a preguntar por los nombres y apellidos del usuario.
- ***RF 1.2 Crear cuenta del vendedor.*** Se debe permitir la creación de una cuenta de tipo vendedor que va a preguntar por el nombre comercial del usuario. Al crearla, este usuario tendrá una lista de productos y de pedidos.
- ***RF 1.3 Crear cuenta del cliente.*** Se debe permitir la creación de una cuenta de tipo cliente que va a preguntar por los nombres y apellidos, el teléfono y la dirección del usuario. Al crearla, este usuario tendrá una lista de pedidos.

RF 2. Implementar las funciones del administrador.

- ***RF 2.1. Gestionar la información de la cuenta.*** El administrador debe poder actualizar la información de su cuenta, como su contraseña, su imagen de perfil, sus nombres y sus apellidos.
- ***RF 2.2. Gestionar el estado del pedido.*** El administrador puede cambiar el estado de un pedido entre SOLICITADO, EN PROCESO, ENVIADO, y ENTREGADO. Este estado solamente puede cambiar hacia adelante dentro de esta lista.
- ***RF 2.3. Gestionar las solicitudes de los vendedores.*** El administrador debe poder aceptar o rechazar las solicitudes que tienen los vendedores para crear, actualizar, eliminar o deshabilitar un producto.
- ***RF 2.4. Gestionar las categorías del sistema.*** El administrador debe poder crear, actualizar, eliminar o deshabilitar categorías del sistema.

- ***RF 2.5. Gestionar los métodos de pago del sistema.*** El administrador debe poder crear, actualizar, eliminar o deshabilitar los métodos de pago del sistema.
- ***RF 2.6. Exportar un reporte de productos y pedidos.*** El administrador debe poder generar un archivo .csv de productos y pedidos.
    - ***RF 2.6.1. Exportar un reporte de productos.*** El archivo de productos debe contener los códigos, los nombres, las categorías, las marcas, los precios, la cantidad, el número de ventas y las ganancias totales generadas con las ventas de los productos separados por filas, y deben estar ordenados por el número de ventas ascendente.
    - ***RF 2.6.2. Exportar un reporte de pedidos.*** El archivo de pedidos debe contener los códigos de orden, la fecha y la hora de los pedidos, los datos del cliente que los solicita, la lista de productos y sus cantidades, el precio, el estado de la orden y la información de pago separados por filas, y deben estar ordenados por fecha y hora en orden ascendente.
        - ***RF 2.6.2.1. Organizar por rango de horas.*** El reporte de pedidos debe poder generarse en un rango de fechas y hora, es decir, al momento de generarlo se pregunta la fecha y hora inicial, y la fecha y hora final del reporte. Por defecto, el valor en la fecha y hora inicial son las 00:00 del día actual, y la fecha final debe ser por defecto 23:59 del día actual. Este reporte debe estar ordenado por fecha y hora del pedido ascendente.
    - ***RF 2.6.3. Preguntar por separador.*** En el momento en que el usuario elija la opción de exportar cualquiera de los reportes, debe preguntársele también cuál es el separador que se utilizará, aunque por defecto debe tener puesto en ese campo el valor punto y coma (;).

RF 3. Implementar las funciones del vendedor.

- ***RF 3.1. Gestionar la información de la cuenta.*** El vendedor debe poder actualizar la información de su cuenta, como su contraseña, su imagen de perfil y su nombre comercial.
- ***RF 3.2. Hacer una solicitud para gestionar los productos.*** El vendedor puede hacer una solicitud para crear, actualizar, eliminar y deshabilitar un producto dentro de la plataforma, la cual va a ser revisada por un administrador.
- ***RF 3.4. Exportar un reporte de productos.*** El vendedor debe poder generar un archivo .csv para los productos que vende en la plataforma. El archivo de productos debe contener los códigos, los nombres, las categorías, las marcas, los precios, la cantidad, el número de ventas y las ganancias totales generadas con las ventas de los productos separados por filas, y deben estar ordenados por el número de ventas ascendente.
    - ***RF 3.4.1. Preguntar por separador.*** En el momento en que el usuario elija la opción de exportar cualquiera de los reportes, debe preguntársele también cuál es el separador que se utilizará, aunque por defecto debe tener puesto en ese campo el valor punto y coma (;).
- ***RF 3.5. Importar datos de un archivo.*** El programa le debe permitir al vendedor importar datos de un archivo .csv para solicitar la creación de varios productos.

RF 4. Implementar las funciones del cliente.

- ***RF 4.1. Gestionar la información de la cuenta.*** El cliente debe poder ver y actualizar la información de su cuenta, como sus nombres y apellidos, su teléfono y su dirección.
- ***RF 4.2. Realizar un pedido.*** El cliente debe poder realizar un pedido, el cual tiene un código (autogenerado), una fecha y hora (la cual es tomada del

sistema), el cliente que hace el pedido, una lista de productos y sus cantidades, un precio, un estado y la información de pago.

- ***RF 4.3. Gestionar sus pedidos.*** El cliente debe poder ver y cancelar sus pedidos. En el caso que un cliente cancele su pedido, el estado cambiará a **CANCELADO** y podrá solicitar que se elimine del sistema.

RF 5. Visualizar los productos. Después del inicio de sesión, estos se presentan en un listado por casillas que muestran las imágenes de los productos, sus nombres y sus marcas. Este listado muestra los productos de acuerdo al vendedor, a la marca o a su categoría, y puede ser filtrado por precio y por artículos más vendidos. El programa le debe permitir al vendedor modificar los datos de los productos a través de una opción de edición, la cual está disponible al inspeccionar uno de los productos. Además, ellos deben poder revisar los productos que venden dentro de la plataforma.

RF 6. Visualizar las categorías. Después de mostrar la información del usuario, se le debe permitir al administrador visualizar las categorías que hay dentro del programa.

RF 7. Visualizar los métodos de pago. Después de mostrar la información del usuario, se le debe permitir al administrador visualizar los métodos de pago que hay dentro del programa.

RF 8. Visualizar las solicitudes. Después de mostrar la información del usuario, se le debe permitir al administrador visualizar las solicitudes que hay dentro del programa.

RF 9. Visualizar las órdenes. Después de mostrar la información del usuario, se le debe permitir al administrador y al cliente visualizar las órdenes que tienen dentro del programa.

RF 10. Guardar toda la información. Cada vez que se registre o actualice la información, el programa lo guardará en archivos serializados.

2. REQUERIMIENTOS NO FUNCIONALES.

RNF 1. Archivos serializados. Implementar archivos serializados para guardar la información del programa.

RNF 2. Excepciones. Implementar al menos 4 excepciones con el API de Java y 3 excepciones propias.

RNF 3. Pruebas unitarias. Implementar pruebas unitarias para garantizar el buen funcionamiento del programa.

RNF 4. Búsqueda binaria. Codificar al menos 2 implementaciones propias de búsqueda binaria.

RNF 5. Listas enlazadas. Implementar al menos 2 listas enlazadas dentro del programa.

RNF 6. Herencias e interfaces. Implementar al menos 4 relaciones de herencia y 3 casos de desacoplamiento por interfaces.

RNF 7. Árboles binarios. Implementar 2 árboles binarios dentro del programa.

RNF 8. Recursión. Implementar al menos 4 métodos recursivos dentro del programa.

RNF 9. Concurrencia. Implementar 3 hilos para ejecutar métodos dentro del programa (además del hilo principal).

RNF 10. Búsqueda interna. Implementar métodos de búsqueda para implementar los métodos funcionales de gestionar cuentas, productos, categorías, información y métodos de pago, y órdenes.

RNF 11. Algoritmos de ordenamiento. Implementar los algoritmos de burbuja, inserción y selección. También, se deben implementar ordenamientos utilizando Comparator y Comparable.

3. REQUERIMIENTOS FUNCIONALES IMPLEMENTADOS.

RF 1. Crear una cuenta. El programa debe permitir a los usuarios la creación de una cuenta de tipo administrador, vendedor o cliente. Todas las cuentas tienen un nombre de usuario, una contraseña y una imagen de perfil.

RF 1.1 Crear cuenta del administrador. Se debe permitir la creación de una cuenta de tipo administrador que va a preguntar por los nombres y apellidos del usuario.

RF 1.2 Crear cuenta del vendedor. Se debe permitir la creación de una cuenta de tipo vendedor que va a preguntar por el nombre comercial del usuario. Al crearla, este usuario tendrá una lista de productos y de pedidos.

RF 1.3 Crear cuenta del cliente. Se debe permitir la creación de una cuenta de tipo cliente que va a preguntar por los nombres y apellidos, el teléfono y la dirección del usuario. Al crearla, este usuario tendrá una lista de pedidos.

4. DISEÑO DE LAS PRUEBAS UNITARIAS.

Configuración de los Escenarios

Nombre	Clase	Escenario
setupScenary 1	StoreTest	**Se ha registrado una persona como vendedor:** username= “apple” password= “12345” tradeName= “Apple Inc.” productList= {iPhone 11} **Se ha agregado una categoría:** name = “Electronic Devices” **Se ha agregado 1 producto:** ID=82498429 name= “iPhone 11” category= Electronic Devices brand= Apple price= $2.300.000 stock= 10 description= “This is a phone” sellerList= {Apple}

setupScenary 2	StoreTest	**Se han registrado 2 personas como vendedores:** username= “apple” password= “12345” tradeName= “Apple Inc.” productList= {iPhone 11} username= “official-LG” password= “2468” tradeName= “LG” productList= {WashTowers} **Se han agregado 2 categorías:** name = “Electronic Devices” name = “Home Appliances” **Se han agregado 2 productos:** ID=82498429 name= “iPhone 11” category= Electronic Devices brand= Apple price= $2.300.000 stock= 10 description= “This is a phone” sellerList= {Apple Inc.} ID=56789012 name= “WashTowers” category= Home Appliances brand= LG price= $1.890.000 stock= 4 description= “This is a washing machine” sellerList= {LG} **Se ha registrado una orden:** ID=1234567 date=12/06/2021 client={Ariel Pabón} productList= {WashTowers} productQuantity= {1} price= $1.890.000 orderState= REQUESTED paymentInformation= {Card}

Diseño de Casos de Prueba

Métodos para agregar:

Objetivo de la Prueba: Validar que se agregue correctamente un producto.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	addProduct	setupScenary 1	ID=234576 name= “iMac” category= Electronic Devices brand= Apple price= $7.300.000 stock= 5 description= “This is a desktop computer” sellerList= {Apple}	true Se agregó correctamente un nuevo producto cuya categoría, marca y lista de vendedores que lo ofrecen corresponden a la información disponible en este escenario. La lista de productos del vendedor ahora está compuesta de 2 productos, el iPhone 11 y el iMac.
MinoristStore	addProduct	setupScenary 2	ID=135790 name= “LG OLED UHD” category= Electronic Devices brand= LG price= $6.400.000 stock= 3 description= “This is a TV” sellerList= {LG}	true Se agregó correctamente un nuevo producto. La lista de productos del vendedor “LG” ahora está compuesta de 2 productos, la lavadora “WashTowers” y el televisor “LG OLED UHD”.

Objetivo de la Prueba: Validar que se agregue correctamente una cuenta de usuario.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	addAccount	setupScenary 1	username= “Admin11” password= “54321” names= “Laura” surnames=”Martínez”	true Se agregó correctamente una nueva cuenta de usuario, en este caso un administrador. La lista de cuentas de usuario ahora está compuesta por un vendedor y un administrador.
MinoristStore	addAccount	setupScenary 2	username= “Juan01” password= “0000” names= “Juan” surnames=”Ramos”	true Se agregó correctamente una nueva cuenta de usuario, en este caso un administrador. La lista de cuentas de usuario ahora está compuesta por 2 vendedores, un

				administrador y el cliente que hizo la orden.

Objetivo de la Prueba: Validar que **NO** se agregue una cuenta de usuario.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	addAccount	setupScenary 1	username= “apple” password= “54321” tradeName= “Apple1.0” productList= {iPhone XR}	false No se agregó esta cuenta de usuario, en este caso un vendedor, porque usó el mismo username que el usuario ya creado en este escenario. La lista de cuentas de usuario se mantiene compuesta por un solo vendedor.
MinoristStore	addAccount	setupScenary 2	username= “official-LG” password= “0000” tradeName= “LG11” productList= {LG OLED UHD}	false No se agregó esta cuenta de usuario, en este caso un vendedor, porque usó el mismo username que uno de los usuarios ya creados en este escenario. La lista de cuentas de usuario se mantiene compuesta por 2 vendedores.

Objetivo de la Prueba: Validar que se agregue correctamente una orden.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	addOrder	setupScenary 1	ID=2467893 date=1/07/2021 client={José Peréz} productList={iPhone 11} productQuantity={1} price= $2.300.000 orderState=SENT paymentInformation= {OnlineSystem}	true Se agregó correctamente una orden cuyas características corresponden a la información disponible en este escenario.
MinoristStore	addOrder	setupScenary 2	ID=2467893 date=1/07/2021 client={Luis Reyes} productList={WashTowers}	true Se agregó correctamente una orden cuyas características corresponden a la información disponible en este escenario.

			productQuantity={2} price= $3.780.000 orderState=IN_PROCESS paymentInformation= {OnlineSystem}	Ahora hay 2 órdenes con sus respectivos clientes, productos y cantidades.

Objetivo de la Prueba: Validar que se agregue correctamente una categoría.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	addCategory	setupScenary 1	name = “Home products”	true Se agregó correctamente una categoría. Ahora se manejan 2 categorías, “Electronic Devices” y “Home products”
MinoristStore	addCategory	setupScenary 2	name = “Sports”	true Se agregó correctamente una categoría. Ahora se manejan 3 categorías, “Electronic Devices”, “Home Appliances” y “Sports”

Métodos para eliminar:

Objetivo de la Prueba: Validar que se elimine correctamente un producto.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	removeProduct	setupScenary 1	productList={iPhone 11}	true Como no hay ordenes en este escenario, se puede eliminar. Al eliminarse, no quedan productos disponibles.
MinoristStore	removeProduct	setupScenary 2	productList={iPhone 11}	true Se puede eliminar ya que no está solicitado en la orden. Al eliminarse, queda disponible un solo producto, la lavadora “WashTowers”.

Objetivo de la Prueba: Validar que **NO** se elimine una cuenta de usuario.

Clase	Método	Escenario	Valores de Entrada	Resultado

MinoristStore	removeAccount	setupScenary 1	sellerList= {Apple}	false Al estar instanciado en el producto, no se puede eliminar.
MinoristStore	removeAccount	setupScenary 2	sellerList= {LG}	false Al estar instanciado en el producto y, por lo tanto, en la orden, no se puede eliminar.

Objetivo de la Prueba: Validar que **NO** se elimine una categoría.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	removeCategory	setupScenary 1	name = “Electronic Devices”	false Al estar instanciada en el producto no se puede eliminar.
MinoristStore	removeCategory	setupScenary 2	name = “Electronic Devices”	true Al estar instanciada en uno de los productos no se puede eliminar.

Métodos para actualizar:

Objetivo de la Prueba: Validar que se actualice correctamente un producto.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	updateProduct	setupScenary 1	ID=82498429 name=“iPhone 11” category=Electronic Devices brand= Apple price= $1.950.000 stock= 7 description= “This is a phone” sellerList= {Apple}	true Se actualizó correctamente un producto, en este caso se modificó el precio y la cantidad en stock del iPhone 11.
MinoristStore	updateProduct	setupScenary 2	ID=56789012 name= “WashTowers” category= Home Appliances brand= LG price= $2.300.000 stock= 4	true Se actualizó correctamente un producto, en este caso se modificó el precio de la lavadora WashTowers.

			description= “This is a washing machine” sellerList= {LG}	

Objetivo de la Prueba: Validar que se actualice correctamente una cuenta de usuario.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	updateAccount	setupScenary 1	username= “apple” password= “apple00” tradeName=“Apple Inc.” productList={iPhone 11}	true Se actualizó correctamente la contraseña de la cuenta del vendedor.
MinoristStore	updateAccount	setupScenary 2	username=“official-LG” password= “1356” tradeName= “LG” productList={WashTowers, LG OLED UHD}	true Se actualizó correctamente la contraseña de la cuenta de uno de los vendedores y su lista de productos ofrecidos.

Objetivo de la Prueba: Validar que se actualice correctamente una orden.

Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	updateOrder	setupScenary1	-	En este escenario no hay ninguna orden, por lo tanto no se puede actualizar.
MinoristStore	updateOrder	setupScenary2	ID=1234567 date=12/06/2021 client={Ariel Pabón} productList= {iPhone 11} productQuantity= {1} price= $2.300.000 orderState= REQUESTED paymentInformation= {Card}	true Se actualizó correctamente una orden. Ahora se solicita otro producto, por lo que el costo final también se modifica.

Objetivo de la Prueba: Validar que se actualice correctamente una categoría.

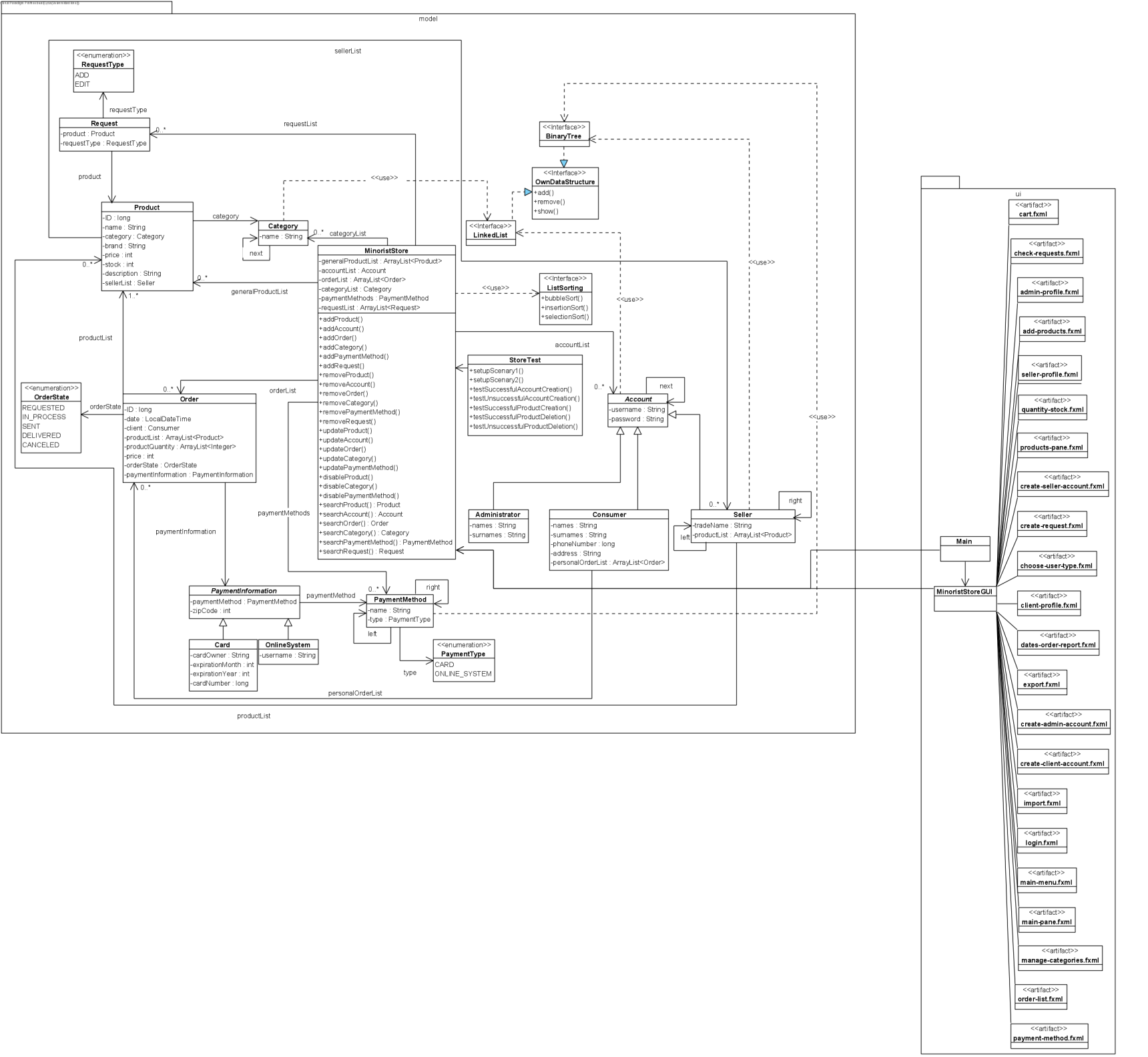
Clase	Método	Escenario	Valores de Entrada	Resultado
MinoristStore	updateCategory	setupScenary1	name = “Cellphones”	true Se actualizó correctamente una

				categoría. Ahora el producto pertenece a la categoría “Cellphones”.
MinoristStore	updateCategory	setupScenary2	name = “Washing Machines”	true Se actualizó correctamente una categoría. Ahora el producto pertenece a la categoría “Washing Machines”.

Métodos para buscar:

Objetivo de la Prueba: Validar que se pueda encontrar un producto.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
MinoristStore	searchProduct	setupScenary 1	name= “iPhone 11”	true El valor ingresado corresponde al producto de este escenario, por lo tanto se retorna lo solicitado.
MinoristStore	searchProduct	setupScenary 2	name= “WashTowers”	true El valor ingresado corresponde a uno de los productos de este escenario, por lo tanto se retorna lo solicitado.

Objetivo de la Prueba: Validar que se pueda encontrar una cuenta de usuario.				
Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
MinoristStore	searchAccount	setupScenary 1	username= “apple”	true El nombre de usuario a buscar corresponde al vendedor en este escenario, por lo tanto se retorna el usuario solicitado.
MinoristStore	searchAccount	setupScenary 2	username=“official-LG”	true El nombre de usuario a buscar corresponde a uno de los vendedores en este escenario, por lo tanto se retorna el usuario solicitado.


model
sellerList
<<enumeration>>
RequestType
ADD
EDIT
requestType
Request
-product : Product
-requestType : RequestType
0..*
requestList
product
<<Interface>>
BinaryTree
<<Interface>>
OwnDataStructure
+add()
+remove()
+show()
<<use>>
Product
-ID : long
-name : String
-category : Category
-brand : String
-price : int
-stock : int
-description : String
-sellerList : Seller
category
Category
-name : String
next
0..*
categoryList
<<Interface>>
LinkedList
MinoristStore
-generalProductList : ArrayList<Product>
-accountList : Account
-orderList : ArrayList<Order>
-categoryList : Category
-paymentMethods : PaymentMethod
-requestList : ArrayList<Request>
+addProduct()
+addAccount()
+addOrder()
+addCategory()
+addPaymentMethod()
+addRequest()
+removeProduct()
+removeAccount()
+removeOrder()
+removeCategory()
+removePaymentMethod()
+removeRequest()
+updateProduct()
+updateAccount()
+updateOrder()
+updateCategory()
+updatePaymentMethod()
+disableProduct()
+disableCategory()
+disablePaymentMethod()
+searchProduct() : Product
+searchAccount() : Account
+searchOrder() : Order
+searchCategory() : Category
+searchPaymentMethod() : PaymentMethod
+searchRequest() : Request
generalProductList
<<Interface>>
ListSorting
+bubbleSort()
+insertionSort()
+selectionSort()
productList
accountList
StoreTest
+setupScenary1()
+setupScenary2()
+testSuccessfulAccountCreation()
+testUnsuccessfulAccountCreation()
+testSuccessfulProductCreation()
+testSuccessfulProductDeletion()
+testUnsuccessfulProductDeletion()
<<enumeration>>
OrderState
REQUESTED
IN_PROCESS
SENT
DELIVERED
CANCELED
orderState
Order
-ID : long
-date : LocalDateTime
-client : Consumer
-productList : ArrayList<Product>
-productQuantity : ArrayList<Integer>
-price : int
-orderState : OrderState
-paymentInformation : PaymentInformation
orderList
Account
-username : String
-password : String
Administrator
-names : String
-surnames : String
Consumer
-names : String
-surnames : String
-phoneNumber : long
-address : String
-personalOrderList : ArrayList<Order>
Seller
-tradeName : String
-productList : ArrayList<Product>
left
right
paymentMethods
paymentInformation
PaymentInformation
-paymentMethod : PaymentMethod
-zipCode : int
paymentMethod
PaymentMethod
-name : String
-type : PaymentType
Card
-cardOwner : String
-expirationMonth : int
-expirationYear : int
-cardNumber : long
OnlineSystem
-username : String
<<enumeration>>
PaymentType
CARD
ONLINE_SYSTEM
type
personalOrderList
productList
ui
<<artifact>>
cart.fxml
<<artifact>>
check-requests.fxml
<<artifact>>
admin-profile.fxml
<<artifact>>
add-products.fxml
<<artifact>>
seller-profile.fxml
<<artifact>>
quantity-stock.fxml
<<artifact>>
products-pane.fxml
<<artifact>>
create-seller-account.fxml
<<artifact>>
create-request.fxml
<<artifact>>
choose-user-type.fxml
<<artifact>>
client-profile.fxml
<<artifact>>
dates-order-report.fxml
<<artifact>>
export.fxml
<<artifact>>
create-admin-account.fxml
<<artifact>>
create-client-account.fxml
<<artifact>>
import.fxml
<<artifact>>
login.fxml
<<artifact>>
main-menu.fxml
<<artifact>>
main-pane.fxml
<<artifact>>
manage-categories.fxml
<<artifact>>
order-list.fxml
<<artifact>>
payment-method.fxml
Main
MinoristStoreGUI