Tarea Integradora

Juan Camilo Zorrilla Calvache

Documentos del código de la tarea integradora 1

Juan Manuel Reyes

Universidad ICESI

Facultad de Ingeniería

Ingeniería de Sistemas

Santiago de Cali

Septiembre 26 del 2020

Requerimientos Funcionales

RF1 *Agregar un nuevo restaurante*, debe permitir el ingreso de un nuevo restaurante con los atributos correspondientes los cuales son nombre del restaurante, nit del restaurante y nombre administrador del restaurante.

RF2 *Agregar un nuevo producto*, debe permitir el ingreso de un nuevo producto con los atributos correspondientes los cuales son código del producto, nombre del producto, descripción del producto, costo del producto, nit del restaurante que posee el producto.

RF3 *Agregar un nuevo cliente*, debe permitir el ingreso de un nuevo cliente con los atributos correspondientes los cuales son tipo de identificación, número de identificación, nombre del cliente, apellido del cliente, teléfono del cliente y dirección del cliente. Al introducirlos estos se deben agregar de manera ordenada al arreglo.

RF4 *Agregar una nueva orden*, debe permitir el ingreso de una nueva orden con los atributos correspondientes los cuales son código de orden, fecha de orden, código del cliente, nit del restaurante, una lista de productos a llevar la cual posee el nombre del producto y la cantidad a llevar y estado del pedido.

RF5 *Actualizar datos de un restaurante*, debe permitir la actualización de los datos de un restaurante según el nit que ingrese el usuario, permitiría actualizar sus atributos.

RF6 *Actualizar datos de un producto*, debe permitir la actualización de los datos de un producto según el código del producto que ingrese el usuario, permitiría actualizar sus atributos.

RF7 *Actualizar datos de un cliente*, debe permitir la actualización de los datos de un cliente según el número de documento del cliente que ingrese el usuario, permitiría actualizar sus atributos.

RF8 *Actualizar datos de una orden*, debe permitir la actualización de los datos de una orden según el código de orden que ingrese el usuario, permitiría actualizar sus atributos.

RF9 *Actualizar el estado del pedido*, debe permitir la actualización del estado de un pedido de manera ascendente y no de manera descendente, los posibles estados de pedidos “SOLICITADO, EN PROCESO, ENVIADO y ENTREGADO”.

RF10 *Comprobar productos de orden,* se debe comprobar que los productos de un pedido pertenezcan al restaurante que el cliente eligió para hacer el pedido.

RF11 *Guardar información del programa a través de la serialización,* el programa debe permitir guardar toda su información (restaurantes, productos, clientes, ordenes) a través de la serialización.

RF12 *Generar archivo csv que contenga la información de las ordenes*, el programa debe permitir generar un archivo de las ordenes para cada registro de productos solicitado, es decir, por producto pedido generar una línea. Se debe generar ordenadamente con los siguientes criterios de orden: nit del restaurante ascendente, documento del cliente descendente, fecha del pedido

ascendente y código del producto ascendente. Además, debe preguntar que separador se utilizara.

RF13 *Mostrar los restaurantes en orden alfabético de forma ascendente,* el programa debe permitir mostrar en pantalla todos los restaurantes en orden alfabético ascendentemente.

RF14 *Mostrar los clientes en orden descendente con respecto a su número telefónico,* el programa debe permitir mostrar en pantalla todos los clientes en orden descendente según du número de teléfono.

RF15 *Buscar cliente y mostrar tiempo que tarda,* el programa debe permitir buscar un cliente según su nombre y mostrar su información. Además, debe mostrar el tiempo que tardo la búsqueda.

RF16 *Importar datos de restaurantes,* el programa debe permitir importar datos de restaurantes desde un archivo csv y guardarlos en la lista de restaurantes.

RF17 *Importar datos de productos,* el programa debe permitir importar datos de los productos desde un archivo csv y guardarlos en la lista de productos.

RF18 *Importar datos de clientes,* el programa debe permitir importar datos de los clientes desde un archivo csv y guardarlos en la lista de clientes.

RF19 *Importar datos de las ordenes,* el programa debe permitir importar datos de ordenes desde un archivo csv y guardarlos en la lista de ordenes.

Formato Pruebas Unitarias

Nombre	Clase	Escenario
setupStage1	Restaurant	vacío
setupStage1	Product	vacío
setupStage1	Client	vacío
setupStage1	Order	Vacío
setupStage1	MyFastFood	Un objeto de MyFastFood creado, Lista de restaurantes con 1 restaurante ingresado con nombre=“ Burguer Stack” , nit=” 812394”, nombre administrador=”Jose”
setupStage2	MyFastFood	Un objeto de MyFastFood creado, lista de restaurantes vacía

Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Restaurant	Restaurant	setupStage1	name = "Burguer Stack"; nit = "812394"; nameAdmin = "Jose";	El método constructor funciona y los geters realizan su trabajo correctamente.
Restaurant	Seters de Restaurant	setupStage1	name = "Burguer Stack"; nit = "812394"; nameAdmin = "Jose";	Los seters funcionan correctamente.

Clase	**Método**	**Escenario**	**Valores de Entrada**	**Resultado**
Product	Product	setupStage1	code = "789235"; name = "Blue Mike"; description = "La Blue Mike es generosa en carne: 200 gramos de la marca registrada Certified Angus Beef, salsa de queso azul, cebolla morada y tocineta caramelizada. Esto, dentro de un pan artesanal. ";	El método constructor funciona y los geters realizan su trabajo correctamente.

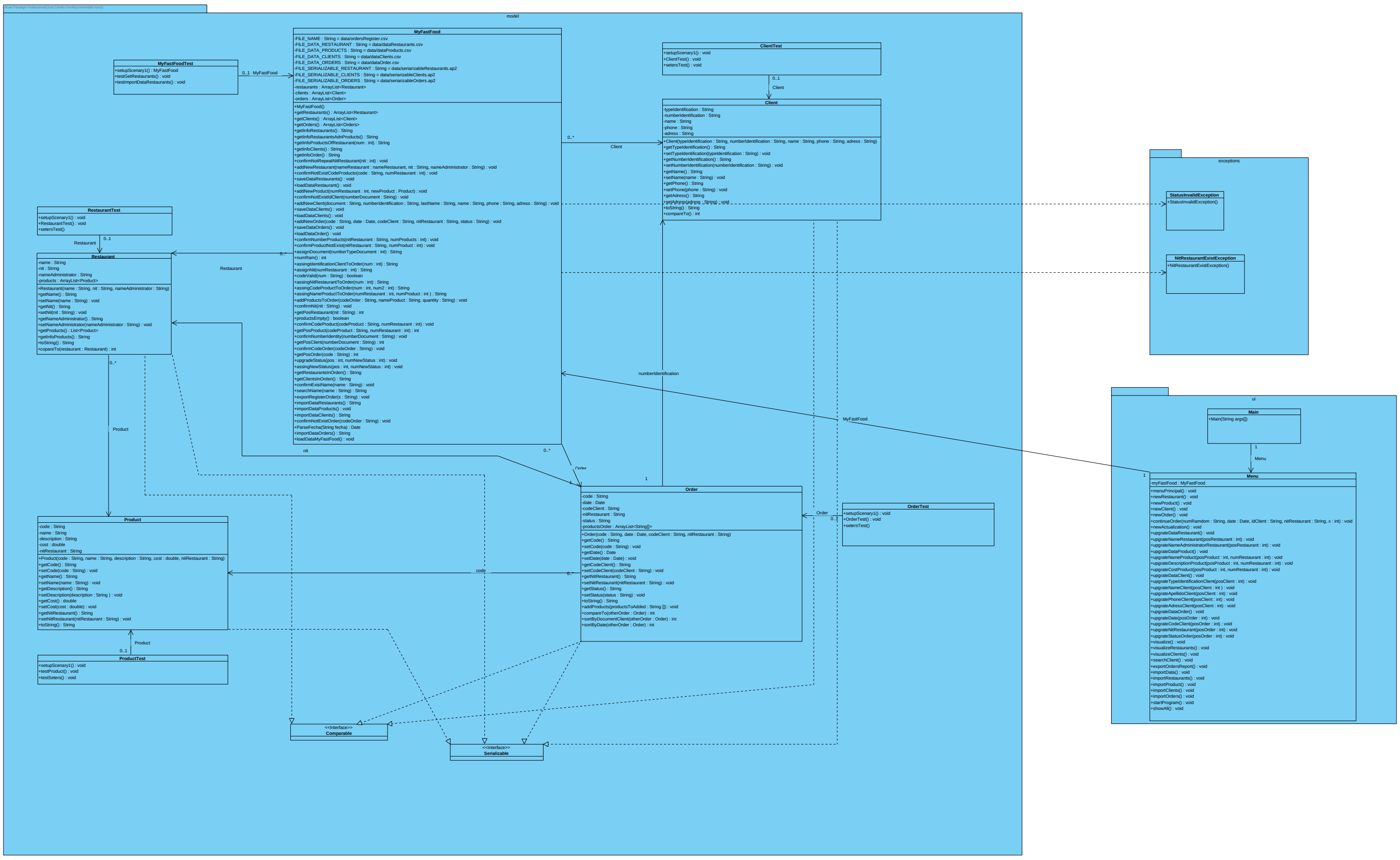
			cost = 17900.0; nitRestaurant = "812394";	
Product	Seters de Product	setupStage1	code = "789235"; name = "Blue Mike"; description = "La Blue Mike es generosa en carne: 200 gramos de la marca registrada Certified Angus Beef, salsa de queso azul, cebolla morada y tocineta caramelizada. Esto, dentro de un pan artesanal. "; cost = 17900.0; nitRestaurant = "812394";	Los seters de Product funciona correctamente.

Clase	Método	Escenario	Valores de Entrada	Resultado
Client	Client y geters	setupStage1	typeIdentification = "Cedula Ciudadania"; numberIdentification = "1007462934"; name = "Adolfo"; lastName = "Restrepo"; phone = "3185471356"; adress = "Cra 1k #48-56";	El método constructor funciona y los geters realizan su trabajo correctamente.
Client	Seters de Client	setupStage1	typeIdentification = "Cedula Ciudadania"; numberIdentification = "1007462934"; name = "Adolfo"; lastName = "Restrepo"; phone = "3185471356"; adress = "Cra 1k #48-56";	Los seters funcionan correctamente.

Clase	Método	Escenario	Valores de Entrada	Resultado
Order	Ordery geters	setupStage1	code = "928374"; date = new java.util.Date();; codeClient = "1007462934"; nitRestaurant = "812394"; [] productsOrder = {"Blue Mike","2"}; status = "SOLICITADO";	El método constructor funciona y los geters realizan su trabajo correctamente.
Order	Seters de Order	setupStage1	code = "928374"; date = new java.util.Date();; codeClient = "1007462934"; nitRestaurant = "812394"; [] productsOrder = {"Blue Mike","2"}; status = "SOLICITADO";	Los seters funcionan correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
MyFastFood	getRestaurants	setupStage1	Ninguno	El metodo get obtiene correctamente el objeto de tipo Restaurante.
MyFastFood	importData getInfoData	setupStage2	Ninguno	Importa los datos exitosamente

Diagrama de Clases


Visual Paradigm Professional(Juan Camilo Zorrilla(Universidad Icesi))
model
MyFastFoodTest
+setupScenary1() : MyFastFood
+testGetRestaurants() : void
+testImportDataRestaurants() : void
0..1 MyFastFood
MyFastFood
-FILE_NAME : String = data/ordersRegister.csv
-FILE_DATA_RESTAURANT : String = data/dataRestaurants.csv
-FILE_DATA_PRODUCTS : String = data/dataProducts.csv
-FILE_DATA_CLIENTS : String = data/dataClients.csv
-FILE_DATA_ORDERS : String = data/dataOrder.csv
-FILE_SERIALIZABLE_RESTAURANT : String = data/seriarizableRestaurants.ap2
-FILE_SERIALIZABLE_CLIENTS : String = data/seriarizableClients.ap2
-FILE_SERIALIZABLE_ORDERS : String = data/seriarizableOrders.ap2
-restaurants : ArrayList<Restaurant>
-clients : ArrayList<Client>
-orders : ArrayList<Order>
+MyFastFood()
+getRestaurants() : ArrayList<Restaurant>
+getClients() : ArrayList<Client>
+getOrders() : ArrayList<Orders>
+getInfoRestaurants() : String
+getInfoRestaurantsAdnProducts() : String
+getInfoProductsOfRestaurant(num : int) : String
+getInfoClients() : String
+getInfoOrder() : String
+confirmNotRepeatNitRestaurant(nit : int) : void
+addNewRestaurant(nameRestaurant : nameRestaurant, nit : String, nameAdministrator : String) : void
+confirmNotExistCodeProducto(code : String, numRestaurant : int) : void
+saveDataRestaurants() : void
+loadDataRestaurant() : void
+addNewProduct(numRestaurant : int, newProduct : Product) : void
+confirmNotExistIdClient(numberDocument : String) : void
+addNewClient(document : String, numberIdentification : String, lastName : String, name : String, phone : String, adress : String) : void
+saveDataClients() : void
+loadDataClients() : void
+addNewOrder(code : String, date : Date, codeClient : String, nitRestaurant : String, status : String) : void
+saveDataOrders() : void
+loadDataOrder() : void
+confirmNumberProducts(nitRestaurant : String, numProducts : int) : void
+confirmProductNotExist(nitRestaurant : String, numProduct : int) : void
+assignDocument(numberTypeDocument : int) : String
+numRam() : int
+assingIdentificationClientToOrder(num : int) : String
+assignNit(numRestaurant : int) : String
+codeValid(num : String) : boolean
+assingNitRestaurantToOrder(num : int) : String
+assingCodeProductToOrder(num : int, num2 : int) : String
+assingNameProductToOrder(numRestaurant : int, numProduct : int) : String
+addProductsToOrder(codeOrder : String, nameProduct : String, quantity : String) : void
+confirmNit(nit : String) : void
+getPosRestaurant(nit : String) : int
+productsEmpty() : boolean
+confirmCodeProduct(codeProduct : String, numRestaurant : int) : void
+getPosProduct(codeProduct : String, numRestaurant : int) : int
+confirmNumberIdentity(numberDocument : String) : void
+getPosClient(numberDocument : String) : int
+confirmCodeOrder(codeOrder : String) : void
+getPosOrder(code : String) : int
+upgradeStatus(pos : int, numNewStatus : int) : void
+assingNewStatus(pos : int, numNewStatus : int) : void
+getRestaurantsInOrden() : String
+getClientsInOrden() : String
+confirmExistName(name : String) : void
+searchName(name : String) : String
+exportRegisterOrder(s : String) : void
+importDataRestaurants() : String
+importDataProducts() : void
+importDataClients() : String
+confirmNotExistOrder(codeOrder : String) : void
+ParseFecha(String fecha) : Date
+importDataOrders() : String
+loadDataMyFastFood() : void
ClientTest
+setupScenary1() : void
+ClientTest() : void
+setersTest() : void
0..1
Client
0..*
Client
Client
-typeIdentification : String
-numberIdentification : String
-name : String
-phone : String
-adress : String
+Client(typeIdentification : String, numberIdentification : String, name : String, phone : String, adress : String)
+getTypeIdentification() : String
+setTypeIdentification(typeIdentification : String) : void
+getNumberIdentification() : String
+setNumberIdentification(numberIdentification : String) : void
+getName() : String
+setName(name : String) : void
+getPhone() : String
+setPhone(phone : String) : void
+getAdress() : String
+setAdress(adress : String) : void
+toString() : String
+compareTo() : int
RestaurantTest
+setupScenary1() : void
+RestaurantTest() : void
+setersTest()
0..1
Restaurant
0..*
Restaurant
Restaurant
-name : String
-nit : String
-nameAdministrator : String
-products : ArrayList<Product>
+Restaurant(name : String, nit : String, nameAdministrator : String)
+getName() : String
+setName(name : String) : void
+getNit() : String
+setNit(nit : String) : void
+getNameAdministrator() : String
+setNameAdministrator(nameAdministrator : String) : void
+getProducts() : List<Product>
+getInfoProducts() : String
+toString() : String
+copareTo(restaurant : Restaurant) : int
0..*
Product
numberIdentification
MyFastFood
nit
0..*
Order
1
1
Order
-code : String
-date : Date
-codeClient : String
-nitRestaurant : String
-status : String
-productsOrder : ArrayList<String[]>
+Order(code : String, date : Date, codeClient : String, nitRestaurant : String)
+getCode() : String
+setCode(code : String) : void
+getDate() : Date
+setDate(date : Date) : void
+getCodeClient() : String
+setCodeClient(codeClient : String) : void
+getNitRestaurant() : String
+setNitRestaurant(nitRestaurant : String) : void
+getStatus() : String
+setStatus(status : String) : void
+toString() : String
+addProducts(productsToAdded : String []) : void
+compareTo(otherOrder : Order) : int
+sortByDocumentClient(otherOrder : Order) : int
+sortByDate(otherOrder : Order) : int
Order
0..1
OrderTest
+setupScenary1() : void
+OrderTest() : void
+setersTest()
Product
-code : String
-name : String
-description : String
-cost : double
-nitRestaurant : String
+Product(code : String, name : String, description : String, cost : double, nitRestaurant : String)
+getCode() : String
+setCode(code : String) : void
+getName() : String
+setName(name : String) : void
+getDescription() : String
+setDescription(description : String) : void
+getCost() : double
+setCost(cost : double) : void
+getNitRestaurant() : String
+setNitRestaurant(nitRestaurant : String) : void
+toString() : String
code
0..*
Product
0..1
ProductTest
+setupScenary1() : void
+testProduct() : void
+testSeters() : void
<<Interface>>
Comparable
<<Interface>>
Serializable
exceptions
StatusInvalidException
+StatusInvalidException()
NitRestaurantExistException
+NitRestaurantExistException()
ui
Main
+Main(String args[])
1
Menu
1
Menu
-myFastFood : MyFastFood
+menuPrincipal() : void
+newRestaurant() : void
+newProduct() : void
+newClient() : void
+newOrder() : void
+continueOrder(numRamdom : String, date : Date, idClient : String, nitRestaurant : String, x : int) : void
+newActualization() : void
+upgrateDataRestaurant() : void
+upgrateNameRestaurant(posRestaurant : int) : void
+upgrateNameAdministratorRestaurant(posRestaurant : int) : void
+upgrateDataProduct() : void
+upgrateNameProduct(posProduct : int, numRestaurant : int) : void
+upgrateDescriptionProduct(posProduct : int, numRestaurant : int) : void
+upgrateCostProduct(posProduct : int, numRestaurant : int) : void
+upgrateDataClient() : void
+upgrateTypeIdentificationClient(posClient : int) : void
+upgrateNameClient(posClient : int) : void
+upgrateApellidoClient(posClient : int) : void
+upgratePhoneClient(posClient : int) : void
+upgrateAdressClient(posClient : int) : void
+upgrateDataOrder() : void
+upgrateDate(posOrder : int) : void
+upgrateCodeClient(posOrder : int) : void
+upgrateNitRestaurant(posOrder : int) : void
+upgrateStatusOrder(posOrder : int) : void
+visualize() : void
+visualizeRestaurants() : void
+visualizeClients() : void
+searchClient() : void
+exportOrdersReport() : void
+importData() : void
+importRestaurants() : void
+importProduct() : void
+importClients() : void
+importOrders() : void
+startProgram() : void
+showAll() : void