2020

# Documentación

## DOCUMENTACIÓN DE LA TAREA INTEGRADORA 1

JOSE ALEJANDRO GARCIA MARCOS

# Documentación

## Requerimientos Funcionales

**RF1**: Registrar restaurantes: El programa pide y recibe los datos de un restaurante para agregarlo: Su nombre, NIT y el nombre del administrador del restaurante. Una vez ingresados los datos, el programa guarda el restaurante con sus datos (siempre y cuando no esté ya agregado)

**RF2**: Comprobar que el restaurante a agregar no esté registrado ya: Para esto, utiliza el NIT proporcionado en el momento del registro y revisa que no haya un restaurante ya registrado con ese NIT; si no encuentra un restaurante con ese NIT, permite el registro del nuevo restaurante, si lo encuentra, muestra en pantalla que ya hay un restaurante con ese NIT y no agrega un nuevo restaurante.

**RF3**: Registrar productos: El programa pide y recibe los datos de un producto para agregarlo. Primero pide el NIT del restaurante al que va a agregar el producto; si no hay un restaurante registrado con ese NIT en la lista de restaurantes, muestra en pantalla que no existe un restaurante con ese NIT registrado y no agrega el nuevo producto; si hay un restaurante registrado con ese NIT continúa el registro del nuevo producto y pide los demás datos: código, nombre, descripción y costo. Una vez ingresados los datos, el programa guarda el restaurante con sus datos (siempre y cuando no esté ya agregado)

**RF4**: Comprobar que el producto a agregar no esté registrado ya: al momento de registrar un nuevo producto, se utiliza el código de producto para revisar que no haya un producto con ese código registrado ya en ningún restaurante; si lo hay, sale en pantalla que existe un producto con ese código registrado, muestra los datos del producto y no agrega un nuevo producto; si no hay un producto con ese código registrado, permite el registro del nuevo producto.

**RF5**: Registrar clientes: El programa pide y recibe los datos de un cliente para agregarlo: Tipo de identificación, número de identificación, nombre completo, teléfono y dirección. Una vez ingresados los datos del cliente, el programa guarda el cliente con sus datos (siempre y cuando no esté ya agregado).

**RF6**: Comprobar que el cliente a agregar no esté registrado ya: al momento de registrar un nuevo cliente, se utiliza el número de identificación para revisar que no haya un cliente con ese número de identificación registrado ya; si lo hay, sale en pantalla que un cliente con ese número de identificación ya está registrado y no agrega un nuevo cliente; si no hay un cliente con ese número de identificación registrado, permite el registro del nuevo cliente.

**RF7**: Registrar un pedido: El programa pide y recibe los datos de un pedido para agregarlo: Primero pide el número de identificación del cliente que está haciendo el pedido; si está registrado, continúa pidiendo los datos. A continuación, el programa pide el NIT del restaurante al que se le hace el pedido; si está registrado, despliega en pantalla la lista de productos del restaurante y pide al usuario que especifique qué producto va a registrar en el pedido, una vez elegido el producto, se pregunta la cantidad del producto para el pedido, después pregunta si se van a añadir más productos al pedido y repite el proceso. Cuando el usuario decida que no va a agregar más productos al pedido, el pedido queda guardado con un código de pedido autogenerado, la hora y la fecha de cuando se registró el pedido y se guarda con el estado “SOLICITADO”.

**RF8**: Comprobar que un cliente exista: Al momento de registrar un pedido, se utiliza el número de identificación provisto, si no hay un cliente registrado con ese número de identificación en la lista de clientes, muestra en pantalla que no existe un cliente con ese número de identificación registrado y no agrega el nuevo pedido; si hay un cliente registrado con ese número de identificación, permite la continuación del registro del nuevo pedido.

**RF9**: Comprobar que un restaurante exista: Al momento de registrar un pedido, se utiliza el NIT provisto, si no hay un restaurante registrado con ese NIT en la lista de restaurantes, muestra en pantalla que no existe un restaurante con ese NIT registrado y no agrega el nuevo pedido; si hay un restaurante registrado con ese NIT, permite la continuación del registro del nuevo pedido.

**RF10**: Mostrar la lista de productos de un restaurante dado su NIT: Al momento de registrar un pedido, se utiliza el NIT provisto para identificar el restaurante, desplegar su lista de productos y que así se registren los productos del pedido.

**RF11**: Comprobar un mismo producto dentro de un pedido: Al momento de registrar un pedido, cuando se estén agregando los productos del pedido, si se agrega un producto que ya fue agregado al pedido, simplemente se aumenta la cantidad de la primera agregación del producto.

**RF12**: Leer la hora del equipo. Al momento de guardar un producto, este se guarda con la fecha y hora del registro pedido, el programa toma esta hora directamente del equipo.

**RF13**: Autogenerar códigos de pedidos: Al momento de guardar un producto, este se guarda con un código de producto autogenerado, el programa genera este código.

**RF14**: Actualizar los datos de un restaurante: Cuando el usuario pida al programa actualizar los datos de un restaurante, el programa le pide al usuario el NIT del restaurante del cual desea cambiar los datos. Si el programa no encuentra un restaurante registrado con ese NIT, muestra en pantalla que no encontró un restaurante con ese NIT registrado y no actualiza los datos de ningún restaurante. Si el programa encuentra el restaurante, le pide al usuario que especifique qué dato desea actualizar, después le pide el nuevo valor de ese dato; luego pregunta si desea actualizar algún otro dato del restaurante; si sí, repite el proceso, si no, guarda el restaurante con los nuevos valores de sus datos.

**RF15**: Actualizar los datos de un producto: Cuando el usuario pida al programa actualizar los datos de un producto, el programa le pide al usuario el código del producto del cual desea cambiar los datos. Si el programa no encuentra un producto registrado con ese código, muestra en pantalla que no encontró un producto con ese código registrado y no actualiza los datos de ningún producto. Si el programa encuentra el producto, le pide al usuario que especifique qué dato desea actualizar, después le pide el nuevo valor de ese dato; luego pregunta si desea actualizar algún otro dato del producto; si sí, repite el proceso, si no, guarda el producto con los nuevos valores de sus datos.

**RF16**: Actualizar los datos de un cliente: Cuando el usuario pida al programa actualizar los datos de un cliente, el programa le pide al usuario el número de identificación del cliente del cual desea cambiar los datos. Si el programa no encuentra un cliente registrado con ese número de identificación, muestra en pantalla que no encontró un cliente con ese número de identificación registrado y no actualiza los datos de ningún cliente. Si el programa encuentra el cliente, le pide al usuario que especifique qué dato desea actualizar, después le pide el nuevo valor de ese dato; luego pregunta si desea actualizar algún otro dato del cliente; si sí, repite el proceso, si no, guarda el cliente con los nuevos valores de sus datos.

**RF17**: Actualizar los datos de un pedido: Cuando el usuario pida al programa actualizar los datos de un pedido, el programa le pide al usuario el código del pedido del cual desea cambiar los datos. Si el programa no encuentra un pedido registrado con ese código, muestra en pantalla que no encontró un pedido con ese código registrado y no actualiza los datos de ningún pedido. Si el programa encuentra el pedido, le pide al usuario que especifique qué dato desea actualizar, después le pide el nuevo valor de ese dato; luego pregunta si desea actualizar algún otro dato del cliente; si sí, repite el proceso, si no, guarda el pedido con los nuevos valores de sus datos. Si el usuario desea cambiar los productos del pedido, debe ingresar los nuevos datos de los productos del pedido uno por uno. Si el usuario pide cambiar el NIT del restaurante del pedido, se borra la lista de productos del pedido. Si el usuario pide cambiar el estado del pedido, sólo lo puede actualizar a estados posteriores, no anteriores. El programa no actualiza la hora, fecha ni código del pedido.

**RF18**: Ordenar los clientes en una lista: Cada vez que un cliente es registrado, este se agrega a la lista de clientes del programa. El cliente se agrega a la lista en el lugar donde debe ir, la lista de clientes está organizada en orden alfabético de apellido y nombre.

**RF19**: Verificar que los productos del pedido pertenezcan al mismo restaurante: Cada vez que se registre un pedido o se actualicen sus datos, el programa revisa que todos los productos del pedido pertenezcan al mismo restaurante, que es el restaurante al que se le hizo el pedido. Si los productos no coinciden con el restaurante del pedido registrado, se muestra en pantalla que no coinciden el restaurante de los productos del pedido y el restaurante al que se le hizo el pedido y no se hace el registro o actualización del pedido.

**RF20**: Guardar la información del programa: Cada que se produce un cambio en el programa, se guarda el estado de este en archivos por método de serialización. Esto no requiere ninguna acción del usuario.

**RF21**: Generar un reporte de los pedidos: El usuario puede pedir generar un reporte de los pedidos. Si lo hace, el programa le pide que especifique el separador que se va a usar para separar los datos. Una vez el usuario provea el separador, el programa genera un archivo .csv con la información de los pedidos, cada producto en el pedido incluye los datos de su restaurante, el cliente que hizo el pedido y del producto mismo. La lista de pedidos está organizada de acuerdo con los siguientes 4 criterios (si hay una coincidencia en el criterio “actual”, pasa al siguiente): 1: NIT del restaurante, ascendente. 2: Documento del cliente, descendente. 3. Fecha del pedido, ascendente. 4. Código del producto, ascendente. La primera línea del documento .csv contiene los nombres de las columnas separadas por el separador especificado por el usuario.

**RF22**: Mostrar los restaurantes: El usuario puede pedirle al programa que muestre los restaurantes en pantalla. El programa mostrará en pantalla la lista de los restaurantes en orden alfabético ascendente de nombre de restaurante.

**RF23**: Mostrar los clientes: El usuario puede pedirle al programa que muestre los clientes en pantalla. El programa mostrará en pantalla la lista de los clientes en orden descendente de su número de teléfono.

**RF24**: Buscar cliente: El usuario puede pedirle al programa que busque un cliente, si lo hace el programa le pedirá que provea su nombre. El programa después buscará eficientemente al cliente en la lista de clientes. Se muestra en pantalla el tiempo que demoró en encontrarlo o encontrarlo.

**RF25**: Importar información de restaurantes. El programa permite importar información de restaurantes, para esto, pide la dirección del archivo .csv que contenga dicha información, el nombre del archivo y el separador que se usó para separar la información de los restaurantes. Imprime el resultado (éxito o fallo en la importación) en pantalla.

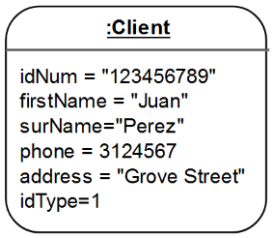
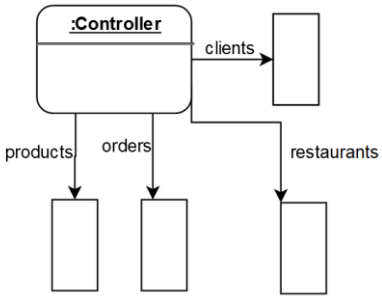
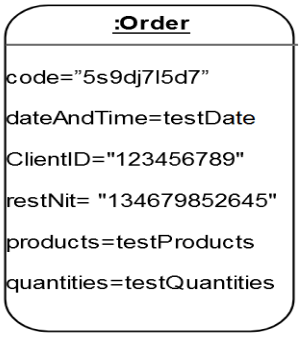
**RF26**: Importar información de clientes. El programa permite importar información de clientes, para esto, pide la dirección del archivo .csv que contenga dicha información, el nombre del archivo y el separador que se usó para separar la información de los clientes. Imprime el resultado (éxito o fallo en la importación) en pantalla.

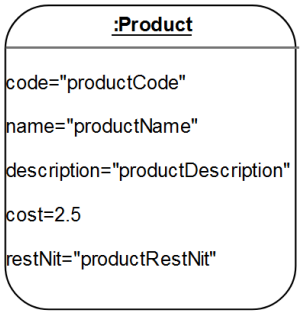
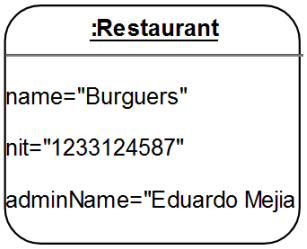
**RF27**: Importar información de productos. El programa permite importar información de productos, para esto, pide la dirección del archivo .csv que contenga dicha información, el nombre del archivo y el separador que se usó para separar la información de los productos. Imprime el resultado (éxito o fallo en la importación) en pantalla.

**RF28**: Importar información de pedidos. El programa permite importar información de pedidos, para esto, pide la dirección del archivo .csv que contenga dicha información, el nombre del archivo y el separador que se usó para separar la información de los pedidos. Imprime el resultado (éxito o fallo en la importación) en pantalla.

## Diseño de pruebas

**Configuración de los Escenarios**

| Nombre | Clase | Escenario |
|---|---|---|
| setupScenario1 | ClientTest | Vacío |
| setupScenario2 | ClientTest |  |
| setupScenario1 | ControllerTest | Vacío |
| setupScenario2 | ControllerTest |  |
| setupScenario1 | OrderTest | Vacío |
| setupScenario1 | OrderTest |  |
| setupScenario1 | ProductTest | Vacío |

| | | |
|---|---|---|
| setupScenario2 | ProductTest |  |
| setupScenario1 | RestaurantTest | Vacío |
| setupScenario2 | RestaurantTest |  |

**Diseño de Casos de Prueba**

| **Objetivo de la Prueba:** Probar el método constructor de la clase Client | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ClientTest | Client | setupScenario1 | idNum = "123456789"<br>firstName = "Juan"<br>surName="Perez"<br>phone = 3124567<br>address = "Grove Street"<br>idType=1 | Se crea un Client con esos valores en sus atributos |

| **Objetivo de la Prueba:** Probar sobreescritura del tipo de documento | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ClientTest | setIdType | setupScenario2 | 2 | Queda guardado el ID type del cliente como 2 |

| **Objetivo de la Prueba:** Probar el método constructor de la clase Controller | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ControllerTest | Controller | setupScenario1 | | Se crea un Controller con sus Arrays Iniciados pero vacíos |

| **Objetivo de la Prueba:** Probar que se añaden y guardan los datos de un restaurante | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ControllerTest | registrarRestaurante | setupScenario2 | name="Burguers" nit="1233124587" adminName="Eduardo Mejia" | Se guarda un Restaurant con esos valores en sus atributos en restaurants de Controller |

| **Objetivo de la Prueba:** Probar el método constructor de la clase Order | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| OrderTest | Order | setupScenario1 | code=”5s9dj7l5d7” dateAndTime=testDate ClientID="123456789" restNit= "134679852645" products=testProducts quantities=testQuantities | Se crea un Order con esos valores en sus atributos |

| **Objetivo de la Prueba:** Probar sobreescritura del nit de restaurante de un Order | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| OrderTest | setRestNit | setupScenario2 | “newRestNit” | Queda guardado el Nit de restaurante del Order como “newRestNit” |

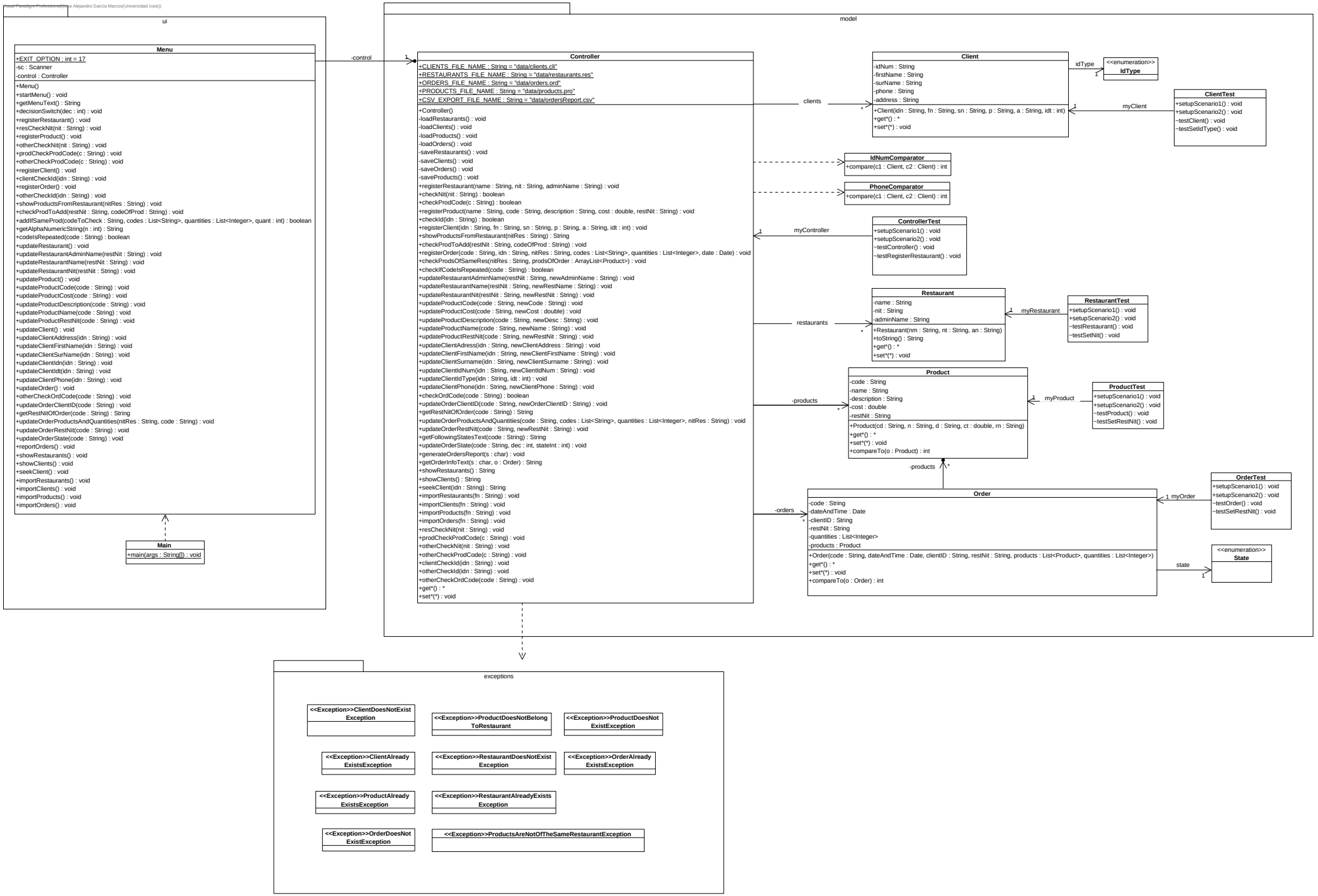
| **Objetivo de la Prueba:** Probar el método constructor de la clase Product | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ProductTest | Product | setupScenario1 | code="productCode"<br>name="productName"<br>description="productDescription"<br>cost=2.5<br>restNit="productRestNit" | Se crea un Product con esos valores en sus atributos |

| **Objetivo de la Prueba:** Probar sobreescritura del nit de restaurante de un Product | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| ProductTest | setRestNit | setupScenario2 | “newRestNit” | Queda guardado el Nit de restaurante del Product como “newRestNit” |

| **Objetivo de la Prueba:** Probar el método constructor de la clase Restaurant | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| RestaurantTest | Restaurant | setupScenario1 | name="Burguers"<br>nit="1233124587"<br>adminName="Eduardo Mejia" | Se crea un Restaurant con esos valores en sus atributos |

| **Objetivo de la Prueba:** Probar sobreescritura del nit de restaurante | | | | |
|---|---|---|---|---|
| **Clase** | **Método** | **Escenario** | **Valores de Entrada** | **Resultado** |
| RestaurantTest | setNit | setupScenario2 | “newRestNit” | Queda guardado el Nit del Restaurant como “newRestNit” |

## Class Diagram


Visual Paradigm Professional(Diana Alejandro García Marcos(Universidad Iceo))
ui
Menu
+EXIT_OPTION : int = 17
-sc : Scanner
-control : Controller
+Menu()
+startMenu() : void
+getMenuText() : String
+decisionSwitch(dec : int) : void
+registerRestaurant() : void
+resCheckNit(nit : String) : void
+registerProduct() : void
+otherCheckNit(nit : String) : void
+prodCheckProdCode(c : String) : void
+otherCheckProdCode(c : String) : void
+registerClient() : void
+clientCheckId(idn : String) : void
+registerOrder() : void
+otherCheckId(idn : String) : void
+showProductsFromRestaurant(nitRes : String) : void
+checkProdToAdd(restNit : String, codeOfProd : String) : void
+addIfSameProd(codeToCheck : String, codes : List<String>, quantities : List<Integer>, quant : int) : boolean
+getAlphaNumericString(n : int) : String
+codeIsRepeated(code : String) : boolean
+updateRestaurant() : void
+updateRestaurantAdminName(restNit : String) : void
+updateRestaurantName(restNit : String) : void
+updateRestaurantNit(restNit : String) : void
+updateProduct() : void
+updateProductCode(code : String) : void
+updateProductCost(code : String) : void
+updateProductDescription(code : String) : void
+updateProductName(code : String) : void
+updateProductRestNit(code : String) : void
+updateClient() : void
+updateClientAddress(idn : String) : void
+updateClientFirstName(idn : String) : void
+updateClientSurName(idn : String) : void
+updateClientIdn(idn : String) : void
+updateClientIdt(idn : String) : void
+updateClientPhone(idn : String) : void
+updateOrder() : void
+otherCheckOrdCode(code : String) : void
+updateOrderClientID(code : String) : void
+getRestNitOfOrder(code : String) : String
+updateOrderProductsAndQuantities(nitRes : String, code : String) : void
+updateOrderRestNit(code : String) : void
+updateOrderState(code : String) : void
+reportOrders() : void
+showRestaurants() : void
+showClients() : void
+seekClient() : void
+importRestaurants() : void
+importClients() : void
+importProducts() : void
+importOrders() : void
Main
+main(args : String[]) : void
-control
1
model
Controller
+CLIENTS_FILE_NAME : String = "data/clients.cli"
+RESTAURANTS_FILE_NAME : String = "data/restaurants.res"
+ORDERS_FILE_NAME : String = "data/orders.ord"
+PRODUCTS_FILE_NAME : String = "data/products.pro"
+CSV_EXPORT_FILE_NAME : String = "data/ordersReport.csv"
+Controller()
-loadRestaurants() : void
-loadClients() : void
-loadProducts() : void
-loadOrders() : void
-saveRestaurants() : void
-saveClients() : void
-saveOrders() : void
-saveProducts() : void
+registerRestaurant(name : String, nit : String, adminName : String) : void
+checkNit(nit : String) : boolean
+checkProdCode(c : String) : boolean
+registerProduct(name : String, code : String, description : String, cost : double, restNit : String) : void
+checkId(idn : String) : boolean
+registerClient(idn : String, fn : String, sn : String, p : String, a : String, idt : int) : void
+showProductsFromRestaurant(nitRes : String) : String
+checkProdToAdd(restNit : String, codeOfProd : String) : void
+registerOrder(code : String, idn : String, nitRes : String, codes : List<String>, quantities : List<Integer>, date : Date) : void
+checkProdsOfSameRes(nitRes : String, prodsOfOrder : ArrayList<Product>) : void
+checkIfCodeIsRepeated(code : String) : boolean
+updateRestaurantAdminName(restNit : String, newAdminName : String) : void
+updateRestaurantName(restNit : String, newRestName : String) : void
+updateRestaurantNit(restNit : String, newRestNit : String) : void
+updateProductCode(code : String, newCode : String) : void
+updateProductCost(code : String, newCost : double) : void
+updateProductDescription(code : String, newDesc : String) : void
+updateProductName(code : String, newName : String) : void
+updateProductRestNit(code : String, newRestNit : String) : void
+updateClientAdress(idn : String, newClientAddress : String) : void
+updateClientFirstName(idn : String, newClientFirstName : String) : void
+updateClientSurname(idn : String, newClientSurname : String) : void
+updateClientIdNum(idn : String, newClientIdNum : String) : void
+updateClientIdType(idn : String, idt : int) : void
+updateClientPhone(idn : String, newClientPhone : String) : void
+checkOrdCode(code : String) : boolean
+updateOrderClientID(code : String, newOrderClientID : String) : void
+getRestNitOfOrder(code : String) : String
+updateOrderProductsAndQuantities(code : String, codes : List<String>, quantities : List<Integer>, nitRes : String) : void
+updateOrderRestNit(code : String, newRestNit : String) : void
+getFollowingStatesText(code : String) : String
+updateOrderState(code : String, dec : int, stateInt : int) : void
+generateOrdersReport(s : char) : void
+getOrderInfoText(s : char, o : Order) : String
+showRestaurants() : String
+showClients() : String
+seekClient(idn : String) : String
+importRestaurants(fn : String) : void
+importClients(fn : String) : void
+importProducts(fn : String) : void
+importOrders(fn : String) : void
+resCheckNit(nit : String) : void
+prodCheckProdCode(c : String) : void
+otherCheckNit(nit : String) : void
+otherCheckProdCode(c : String) : void
+clientCheckId(idn : String) : void
+otherCheckId(idn : String) : void
+otherCheckOrdCode(code : String) : void
+get*() : *
+set*(*) : void
clients
*
Client
-idNum : String
-firstName : String
-surName : String
-phone : String
-address : String
+Client(idn : String, fn : String, sn : String, p : String, a : String, idt : int)
+get*() : *
+set*(*) : void
idType
1
<<enumeration>>
IdType
1
myClient
ClientTest
+setupScenario1() : void
+setupScenario2() : void
~testClient() : void
~testSetIdType() : void
IdNumComparator
+compare(c1 : Client, c2 : Client) : int
PhoneComparator
+compare(c1 : Client, c2 : Client) : int
1
myController
ControllerTest
+setupScenario1() : void
+setupScenario2() : void
~testController() : void
~testRegisterRestaurant() : void
restaurants
*
Restaurant
-name : String
-nit : String
-adminName : String
+Restaurant(nm : String, nt : String, an : String)
+toString() : String
+get*() : *
+set*(*) : void
1
myRestaurant
RestaurantTest
+setupScenario1() : void
+setupScenario2() : void
~testRestaurant() : void
~testSetNit() : void
-products
*
Product
-code : String
-name : String
-description : String
-cost : double
-restNit : String
+Product(cd : String, n : String, d : String, ct : double, rn : String)
+get*() : *
+set*(*) : void
+compareTo(o : Product) : int
1
myProduct
ProductTest
+setupScenario1() : void
+setupScenario2() : void
~testProduct() : void
~testSetRestNit() : void
-products
*
-orders
*
Order
-code : String
-dateAndTime : Date
-clientID : String
-restNit : String
-quantities : List<Integer>
-products : Product
+Order(code : String, dateAndTime : Date, clientID : String, restNit : String, products : List<Product>, quantities : List<Integer>)
+get*() : *
+set*(*) : void
+compareTo(o : Order) : int
1
myOrder
OrderTest
+setupScenario1() : void
+setupScenario2() : void
~testOrder() : void
~testSetRestNit() : void
state
1
<<enumeration>>
State
exceptions
<<Exception>>ClientDoesNotExist Exception
<<Exception>>ProductDoesNotBelong ToRestaurant
<<Exception>>ProductDoesNot ExistException
<<Exception>>ClientAlready ExistsException
<<Exception>>RestaurantDoesNotExist Exception
<<Exception>>OrderAlready ExistsException
<<Exception>>ProductAlready ExistsException
<<Exception>>RestaurantAlreadyExists Exception
<<Exception>>OrderDoesNot ExistException
<<Exception>>ProductsAreNotOfTheSameRestaurantException