

2020

TAREA INTEGRADORA I

DOCUMENTACIÓN
ALGORITMOS Y PROGRAMACIÓN II

JUAN ESTEBAN CAICEDO ALZATE | A00365977

Requerimientos funcionales

A parte de desplegar un menú de opciones con todas las funcionalidades no automáticas que implementa el programa, el sistema debe estar en la capacidad de:

RF1: Permitir el registro de restaurantes, pidiendo para cada uno su nombre, NIT, nombre del administrador, y nombre del operario del restaurante encargado de utilizar el sistema. No obstante, para añadir un nuevo restaurante, se verifica antes si la lista está vacía para añadirlo directamente, o en el caso contrario, a través de su NIT que no exista ya un restaurante en el sistema con ese mismo NIT.

RF2: Permitir el registro de productos, pidiendo para cada uno su código, nombre, descripción, costo, NIT del restaurante que lo ofrece y contenido neto (en gramos, litros, etc.). Sin embargo, para añadir un nuevo producto, se verifican antes tres condiciones: que el restaurante que lo ofrece exista (a través del NIT mencionado), que la lista esté vacía para añadirlo directamente, o en el caso contrario, que no exista ya un producto en el sistema con el mismo código del que se quiere añadir (y ofrecido por el mismo restaurante. O sea, que ya exista ese mismo producto para ese restaurante).

RF3: Permitir el registro de clientes, pidiendo para cada uno su tipo de documento, número de documento, apellido, nombre, teléfono y dirección. No obstante, para añadir un nuevo cliente, se verifica antes lo siguiente: que la lista esté vacía para añadirlo directamente, sino, a través de su número de documento, que no exista ya un cliente en el sistema con ese mismo número de identificación. Pasando el filtro precedente, se añade el cliente insertándolo ordenadamente por apellido descendente y nombre descendente en la lista de clientes. Cabe recalcar que el sistema debe mantener esta lista siempre ordenada de esta forma.

RF4: Permitir el registro de pedidos, pidiendo para cada uno el número de documento del cliente y el NIT del restaurante en el que tiene interés en hacerle un pedido. El código del pedido se genera aleatoriamente. Sin embargo, para añadir un nuevo pedido, se verifica antes

lo siguiente: que exista por lo menos un restaurante, un producto y un cliente en sus respectivas listas del sistema. Si pasa este filtro, se busca que el cliente especificado exista y que el restaurante especificado exista. Si pasa este filtro, el usuario irá añadiendo a una lista los productos que desea pedirle al restaurante, siempre el sistema verificando con el código del producto que éste exista en el sistema y que lo ofrezca dicho restaurante. Por último, si el usuario confirma el pedido, se le asigna por defecto la palabra “REQUESTED” al estado del pedido y se toma la fecha y la hora en la que se creó el pedido, para así añadirlo a la lista de pedidos del sistema.

RF5: Actualizar los datos de un restaurante dado su NIT. Para esto, primero se verifica que hayan restaurantes registrados en la lista de restaurantes. Si es así, el sistema debe verificar que exista el restaurante al cual se le quiere acualizar su información. Si se encuentra, entonces el sistema debe preguntarle al usuario qué atributo del restaurante en cuestión es el que quiere actualizar. El sistema debe dar la opción de actualizar cualquiera de los atributos del restaurante. Finalmente, después de ingresar el nuevo valor del atributo, el sistema lo actualiza, le imprime por consola al usuario un mensaje de éxito de realizar la operación, y serializa la nueva información.

RF6: Actualizar los datos de un producto dado su código de producto. Para esto, primero se verifica que hayan productos registrados en la lista de productos. Si es así, el sistema debe verificar que exista el producto al cual se le quiere acualizar su información. Si se encuentra, entonces el sistema debe preguntarle al usuario qué atributo del producto en cuestión es el que quiere actualizar. El sistema debe dar la opción de actualizar cualquiera de los atributos del producto. Finalmente, después de ingresar el nuevo valor del atributo, el sistema lo actualiza, le imprime por consola al usuario un mensaje de éxito de realizar la operación, y serializa la nueva información actualizada del producto.

RF7: Actualizar los datos de un cliente dado su número de documento. Para esto, primero se verifica que hayan clientes registrados en la lista de clientes. Si es así, el sistema debe verificar que exista el producto al cual se le quiere acualizar su información. Si se encuentra, entonces el sistema debe preguntarle al usuario qué atributo del cliente en

cuestión es el que quiere actualizar. El sistema debe dar la opción de actualizar cualquiera de los atributos del cliente. Finalmente, después de ingresar el nuevo valor del atributo, el sistema lo actualiza, le imprime por consola al usuario un mensaje de éxito de realizar la operación, y serializa la nueva información actualizada del cliente.

RF8: Actualizar los datos de un pedido dado su código de pedido. Para esto, primero se verifica que hayan pedidos registrados en la lista de pedidos. Si es así, el sistema debe verificar que exista el pedido al cual se le quiere acualizar su información. Si se encuentra, entonces el sistema debe preguntarle al usuario qué atributo del pedido en cuestión es el que quiere actualizar. El sistema debe dar únicamente la opción de actualizar:

- El ID del cliente que hizo el pedido.
- El NIT del restaurante al que se le hizo el pedido.
- El estado del pedido. El programa debe permitir cambiar el estado de un pedido entre REQUESTED, IN PROCESS, SENT y DELIVERED. También es importante tener en cuenta que se puede cambiar el estado del pedido hacia adelante (por ejemplo de REQUESTED a IN PROCESS, o de REQUESTED a SENT) pero no hacia atrás.
- La lista de productos del pedido. El sistema debe dar la opción de:
 - o Añadir un producto a la lista de productos del pedido. Si este ya existía, se le suma al número de unidades pedidas del producto las unidades que el cliente quiere ahora sí.
 - o Eliminar:
 - Un producto como tal (todo el objeto) de la lista de productos del pedido.
 - Eliminar una cantidad especificada de unidades pedidas del producto.

(Los atributos como el código del pedido [autogenerado] y la fecha y hora de creación de este no se podrán actualizar)

Finalmente, después de escoger e ingresar el nuevo valor del atributo que se quiere actualizar, el sistema lo actualiza, le imprime por consola al usuario un mensaje de éxito de realizar la operación, y serializa la nueva información actualizada del pedido.

RF9: Desplegar el contenido de la lista de restaurantes, ordenados ascendentemente por su nombre. Para esto, primero se verifica que existan restaurantes registrados en el sistema. Si es así, se procede a ordenar la lista de restaurantes por nombre ascendente, y finalmente el sistema debe poder imprimir por consola la información de todos los restaurantes registrados en la lista.

RF10: Desplegar el contenido de la lista de clientes, ordenados descendente por número de teléfono. Para esto, primero se verifica si existen clientes registrados en el sistema. Si es así, se crea una copia de la lista de clientes del sistema se procede a ordenarla por número de teléfono descendente, y finalmente el sistema debe poder imprimir por consola la información de todos los clientes registrados en esta copia de la lista.

RF11: Buscar eficientemente un cliente dado su nombre. Para esto, primero se verifica si existen clientes registrados en la lista de clientes del sistema. Si es así, se hace una copia de esta lista y se ordena por nombre del cliente ascendente. Luego, utilizando búsqueda binaria se intenta encontrar este cliente en la lista con su nombre. Finalmente, el sistema debe indicar el tiempo que tardó la búsqueda hasta encontrarlo.

RF12: Importar datos de un archivo csv con información de restaurantes, otro con información de clientes, otro con información de productos y otro con información de pedidos. Para esto, se le pide al usuario el nombre del archivo con la información que desea importar y qué es lo que quiere importar. Luego, el sistema debe verificar al ser:

- Restaurantes: que el o los restaurantes que se quieren importar no existan ya en el sistema para que de esta forma puedan ser añadidos correctamente a la lista de restaurantes.

- Clientes: que el o los clientes que se quieren importar no existan ya en el sistema para que de esta forma puedan ser añadidos correctamente a la lista de clientes.
- Productos: que el restaurante que ofrece tal(es) producto(s) exista en el sistema (sino se añade primero este y después su(s) producto(s)), y si es así, que el producto o los productos que se quieren importar no existan ya en el sistema para dicho restaurante, para que de esta forma puedan ser añadidos correctamente a la lista de productos.
- Pedidos: que el restaurante asociado a tal pedido exista en el sistema (sino se añade primero este para poder continuar), luego que los productos contenidos en la lista de productos de cada pedido existan en el sistema (sino se añaden primero estos para poder continuar), que los clientes asociados a sus respectivos pedidos existan en el sistema (sino se añaden primero estos para poder continuar), y que los pedidos que se quieren importar no existan ya en el sistema. Finalmente, habiendo puesto todo en orden, el sistema debería poder importar correctamente todos los pedidos del archivo csv que hayan pasado por los filtros anteriores.

RF13: Exportar datos en un archivo csv con información de pedidos, incluyendo para cada registro de producto solicitado, los datos del restaurante, del cliente y del producto pedido. Además, el sistema debe exportarlo ordenadamente por los siguientes criterios en este orden: nit del restaurante ascendente, documento del cliente descendente, fecha del pedido ascendente y código del producto ascendente. Por último, se usa un separador que el usuario especifique para delimitar los atributos entre ellos.

RF14: Guardar toda su información a través de la serialización de sus objetos en archivos binarios, cada vez que se registre o se actualice información sea de restaurantes, productos, clientes o pedidos (guardado automático). Tiene que existir un path file que especifique el nombre del archivo en el que se almacenará la información guardada de los objetos.

RF15: Cargar toda su información a través de la deserialización de los objetos guardados en archivos binarios (sean de restaurantes, productos, clientes o pedidos), cada vez que se ejecuta el programa (carga automática). Cabe recalcar que el sistema verifica primero que

existan los archivos binarios de restaurantes, productos, clientes y pedidos, para ser deserializados. Si uno de ellos no existe, no se carga la información de éste, y sólo se carga entonces la información de los archvios que existan.

Adicionalmente, aunque no lo pidan, se le dio la posibilidad al sistema de:

RF16 extra: Desplegar el contenido de la lista de clientes, ordenados descendentemente por apellido y después por nombre. Para esto, primero se verifica que existan clientes registrados en el sistema. Si es así, el sistema debe poder imprimir directamente por consola la información de todos los clientes registrados en la lista (estos cuando se añaden ya se insertan ordenadamente por apellido descendente y por nombre descendente).

RF17 extra: Desplegar el contenido de la lista de productos. Para esto, primero se verifica que existan restaurantes registrados en el sistema para que pueda haber la posibilidad de que exista al menos un producto a nombre de uno de estos restaurantes. Si es así, se verifica ahora que existan productos en el sistema. Si es así, el sistema debe poder imprimir por consola la información de todos los productos registrados en la lista de productos del sistema.

RF18 extra: Desplegar el contenido de la lista de pedidos. Para esto, primero se verifica que existan restaurantes registrados en el sistema para que pueda haber la posibilidad de que exista al menos un producto a nombre de uno de estos restaurantes, que se haya pedido. Si es así, se verifica ahora que existan productos en el sistema, que hayan podido ser pedidos. Si es así, se verifica ahora que existan clientes en el sistema para que uno haya podido hacer al menos un pedido de un producto. Finalmente, si todos los anteriores filtros son pasados, el sistema debe poder imprimir por consola la información de todos los pedidos registrados en la lista de pedidos del sistema.

Diseño de los escenarios y casos de prueba

Configuración de los Escenarios

Nombre	Clase	Escenario
setupScenary1	Association Restaurant	<p>Un objeto de la clase AssociationRestaurant con nameAssociation = "Restaurant Association®" es creado y se añade un nuevo restaurante a la lista de restaurantes del sistema con nameRestaurant = "El Corral", nitRestaurant = "109238-32111", nameCeo = "Fernando" y operator = "Manuel".</p> <p>Posibilidad de excepciones manejadas: ClassNotFoundException y IOException.</p>
setupScenary2	Association Restaurant	<p>Un objeto de la clase AssociationRestaurant con nameAssociation = "Restaurant Association®" es creado. También se añade un nuevo restaurante a la lista de restaurantes del sistema con nameRestaurant = "Karens", nitRestaurant = "109238-32111", nameCeo = "Fernando" y operator = "Manuel".</p> <p>Por último, se añade un nuevo producto a la lista de productos del sistema con codeProduct = "67v", nameProduct = "Pizza", description = "Good", cost = 20000, nitRestaurant = "109238-32111", content = 10 y amountOrdered = 0.</p> <p>Posibilidad de excepciones manejadas: ClassNotFoundException y IOException.</p>
setupScenary1	Order	Una lista de productos de la clase Order es inicializada (vacía).

Diseño de Casos de Prueba

Objetivo de la Prueba: Verificar que la búsqueda en la lista de restaurantes del sistema retorna los restaurantes buscados cuando están presentes en la lista y null cuando no se encuentran.				
Clase	Método	Escenario	Valores de Entrada	Resultado
AssociationRestaurant	searchRestaurant	setupScenary1	nitRestaurant = "109238-32111"	Se ha encontrado correctamente un objeto de la clase

				Restaurant con el valor pasado por parámetro al método de búsqueda.
AssociationRestaurant	searchRestaurant	setupScenary1	nitRestaurant = "1881881818818"	Null No se ha encontrado correctamente un objeto de la clase Restaurant con el valor pasado por parámetro al método de búsqueda.

Objetivo de la Prueba: Verificar que el método constructor de la clase Restaurant funcione correctamente, asignando apropiadamente los valores de los parámetros a sus respectivos atributos. La prueba involucra, y en este caso también está probando, tanto el método constructor Restaurant(String, String, String, String) como los métodos getters getNameRestaurant(), getNitRestaurant(), getNameCeo() y getOperator().

Clase	Método	Escenario	Valores de Entrada	Resultado
Restaurant	Restaurant	ninguno	nameRestaurant = "CarbsPlus" nitRestaurant = "76767667676" nameCeo = "Yuluka" operator = "Sasuke"	Se ha creado correctamente un objeto de la clase Restaurant con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que se despliega correctamente toda la información de un objeto de la clase Restaurant con todo el contenido y separador definidos en el método `toString()` de esta clase.

Clase	Método	Escenario	Valores de Entrada	Resultado
Restaurant	<code>toString()</code>	ninguno	<code>nameRestaurant = "CarbsPlus"</code> <code>nitRestaurant = "76767667676"</code> <code>nameCeo = "Yuluka"</code> <code>operator = "Sasuke"</code>	Se muestran todos los datos del objeto de la clase Restaurant que se creó con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que el método constructor de la clase Product funcione correctamente, asignando apropiadamente los valores de los parámetros a sus respectivos atributos. La prueba involucra, y en este caso también está probando, tanto el método constructor `Product(String, String, String, double, String, double, int)` como los métodos getters `getCodeProduct()`, `getNameProduct()`, `getDescription()`, `getCost()`, `getNitRestaurant()`, `getContent()` y `getAmountOrdered()`.

Clase	Método	Escenario	Valores de Entrada	Resultado
Product	<code>Product</code>	ninguno	<code>codeProduct = "101010101010"</code> <code>nameProduct = "Edna"</code> <code>description = "Moda"</code> <code>cost = 2918</code> <code>nitRestaurant = "90909090"</code> <code>content = 121</code> <code>amountOrdered = 0</code>	Se ha creado correctamente un objeto de la clase Product con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que se despliega correctamente toda la información de un objeto de la clase Product con todo el contenido y separador definidos en el método `toString()` de esta clase.

Clase	Método	Escenario	Valores de Entrada	Resultado
Product	<code>toString()</code>	ninguno	<code>codeProduct = "101010101010"</code> <code>nameProduct = "Edna"</code> <code>description = "Moda"</code> <code>cost = 2918</code> <code>nitRestaurant = "90909090"</code> <code>content = 121</code> <code>amountOrdered = 0</code>	Se muestran todos los datos del objeto de la clase Product que se creó con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que el método constructor de la clase Client funcione correctamente, asignando apropiadamente los valores de los parámetros a sus respectivos atributos. La prueba involucra, y en este caso también está probando, tanto el método constructor `Client(String, String, String, String, String, String)` como los métodos getters `getTypeDocument()`, `getIdClient()`, `getLastNameClient()`, `getNameClient()`, `getPhone()` y `getAddress()`.

Clase	Método	Escenario	Valores de Entrada	Resultado
Client	<code>Client</code>	ninguno	<code>typeDocument = CITIZENSHIP_CARD</code> <code>idClient = "677676111"</code> <code>lastNameClient = "lalalalala"</code> <code>nameClient = "ioioioioioio"</code> <code>phone = "233287-8710"</code> <code>address = "mnghfdrwejkjkjk";</code>	Se ha creado correctamente un objeto de la clase Client con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que se despliega correctamente toda la información de un objeto de la clase Client con todo el contenido y separador definidos en el método `toString()` de esta clase.

Clase	Método	Escenario	Valores de Entrada	Resultado
Client	<code>toString()</code>	ninguno	<code>typeDocument = CITIZENSHIP_CARD</code> <code>idClient = "677676111"</code> <code>lastNameClient = "lalalalala"</code> <code>nameClient = "ioioioioioio"</code> <code>phone = "233287-8710"</code> <code>address = "mnghfdrwejkjkjk";</code>	Se muestran todos los datos del objeto de la clase Client que se creó con los valores pasados por parámetro al constructor.

Objetivo de la Prueba: Verificar que el método constructor de la clase Order funcione correctamente, asignando apropiadamente los valores de los parámetros a sus respectivos atributos. La prueba involucra, y en este caso también está probando, tanto el método constructor `Order (int, Date, String, String, String)` como los métodos getters `getCodeOrder()`, `getDateTime()`, `getIdClient()`, `getNitRestaurant()`, `getStatus()` y `getProducts()`.

Clase	Método	Escenario	Valores de Entrada	Resultado
Order	<code>Order</code>	<code>setupScenary1</code>	<code>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario. Atributos: codeProduct = " 576438211-66", nameProduct = " Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 104433, amountOrdered = 0.</code> <code>-----</code>	<code>Se ha creado correctamente un objeto de la clase Order con los atributos del pedido pasados por parámetro al constructor.</code>

			<p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p>	
--	--	--	---	--

Objetivo de la Prueba: Verificar que al comparar criterios (atributos) de tipo String de un objeto de la clase Order, devuelve correctamente un número positivo si es mayor lexicográficamente, negativo si es menor lexicográficamente, o igual a 0 si son iguales lexicográficamente. En el método únicamente se comparan el NIT del restaurante, el número del ID del cliente, la fecha y hora del pedido y el código de un producto del pedido. Para esto, visto que para probar esta acción da lo mismo usar cualquiera de estos atributos mencionados para verificar que devuelva el entero adecuado, sólo se usó el **NIT del restaurante** como ejemplo (para objetivo de ordenamiento ascendente y descendente).

Clase	Método	Escenario	Valores de Entrada	Resultado
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento ascendente)</p> <p>Número positivo</p> <p>El resultado de exampleCurrentNit comparado al atributo nitRestaurant del objeto de tipo Order dio correctamente un entero positivo al ser lexicográficamente el atributo actual (exampleCurrentNit = "43871892") mayor al atributo de nitRestaurant ("19191928") del objeto Order.</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "43871892"</p>	
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento ascendente) Número negativo</p> <p>El resultado de exampleCurrentNit comparado al atributo nitRestaurant del objeto de tipo Order dio correctamente un entero negativo al ser lexicográficamente el atributo actual (exampleCurrentNit = " 13765489 ") menor al atributo de nitRestaurant ("19191928") del objeto Order.</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "13765489 "</p>	
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento ascendente) Número 0</p> <p>El resultado de exampleCurrentNit comparado al atributo nitRestaurant del objeto de tipo Order dio correctamente 0 al ser lexicográficamente el atributo actual (exampleCurrentNit = "19191928") igual al atributo de nitRestaurant ("19191928") del objeto Order.</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "19191928"</p>	
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento descendente)</p> <p>Número positivo</p> <p>El resultado del atributo nitRestaurant del objeto de tipo Order comparado al atributo actual exampleCurrentNit dio correctamente un número positivo al ser lexicográficamente el atributo nitRestaurant ("19191928") del objeto Order mayor al atributo actual (exampleCurrentNit = "13765489").</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "13765489"</p>	
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento descendente)</p> <p>Número negativo</p> <p>El resultado del atributo nitRestaurant del objeto de tipo Order comparado al atributo actual exampleCurrentNit dio correctamente un número negativo al ser lexicográficamente el atributo nitRestaurant ("19191928") del objeto Order menor al atributo actual (exampleCurrentNit = "43871892").</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "43871892"</p>	
Order	compareTo()	setupScenary1	<p>Se crea y se añade un objeto de la clase Product a la lista de productos inicializada en el escenario.</p> <p>Atributos:</p> <p>codeProduct = "576438211-66", nameProduct = "Kilko", description = "Nice taste", cost = 9909, nitRestaurant = "19191928", content = 4433, amountOrdered = 0.</p>	<p>(Para ordenamiento descendente)</p> <p>Número 0</p> <p>El resultado del atributo nitRestaurant del objeto de tipo Order comparado al atributo actual exampleCurrentNit dio correctamente 0 al ser lexicográficamente el atributo nitRestaurant ("19191928") del objeto Order igual al atributo actual (exampleCurrentNit = "19191928").</p>

			<p>-----</p> <p>-----</p> <p><u>Atributos del pedido:</u></p> <p>codeOrder = <i>Número entero aleatorio.</i></p> <p>dateTime = <i>fecha y hora actual de tipo Date</i></p> <p>idClient = "655656222"</p> <p>nitRestaurant = "19191928"</p> <p>status = REQUESTED</p> <p>products = <i>lista de productos con el anterior objeto de la clase Product añadido</i></p> <p>-----</p> <p>-----</p> <p>exampleCurrentNit = "19191928"</p>	
--	--	--	---	--

