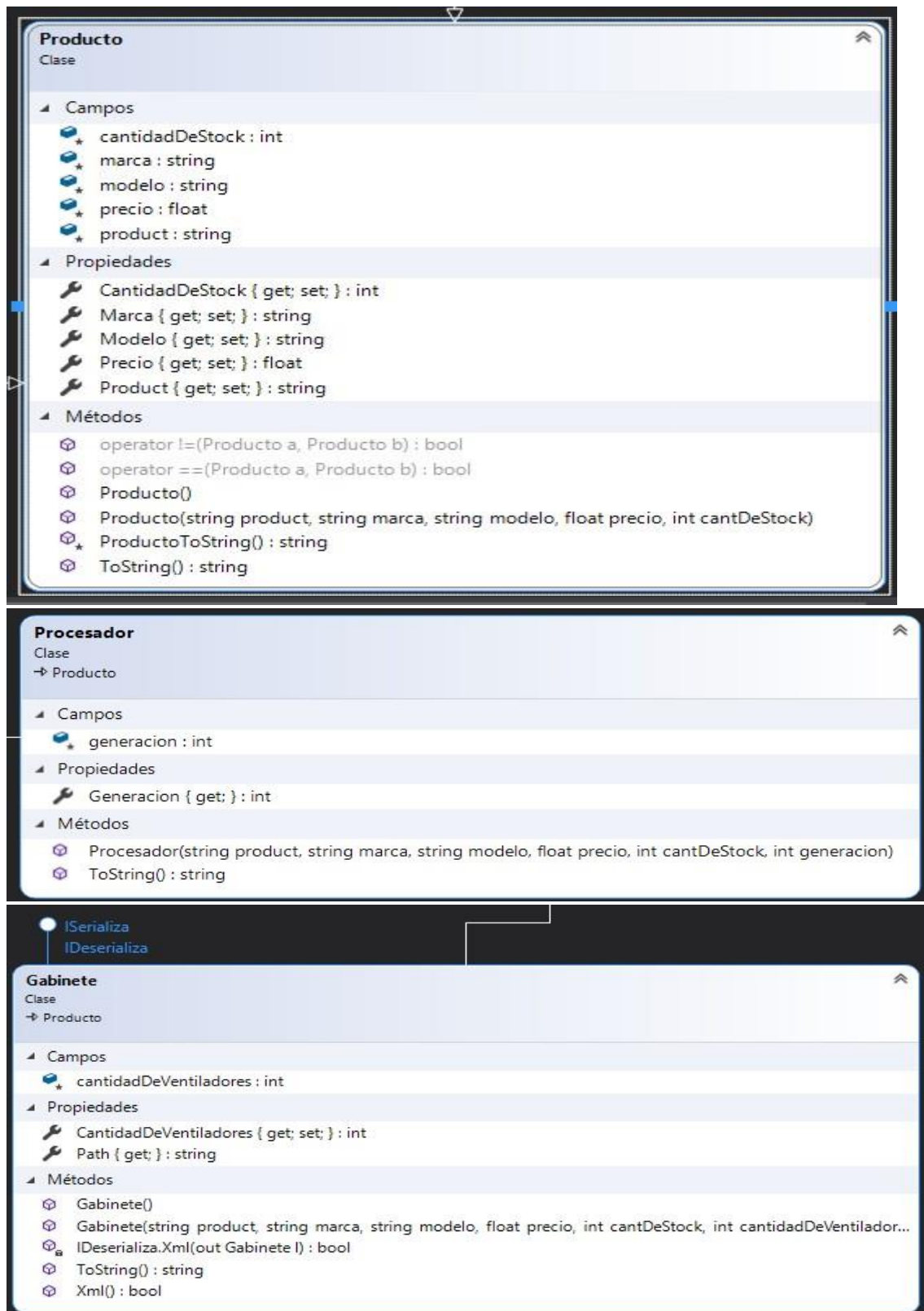


Condiciones de corrección y aprobación

1. Cumplir con las reglas de estilo.
2. Cumplir con lo visto en clase.
3. Se deberán manejar posibles Excepciones.
4. Deberá tener al menos un formulario.
5. Deberá tener un proyecto llamado "Test" de **consola** probando todas las funcionalidades principales que se creen (todo hardcodeado, sin menú ni nada complejo, simplemente para repasar las funcionalidades). El código debe mantenerse simple y corto, como los dados para probar alguna aplicación durante la cursada.
6. Deberá tener otro proyecto de Test Unitarios probando al menos 2 funcionalidades creadas.
7. Deberá utilizar todos los temas vistos entre la clase 15 y 25 (se tomará como referencia el Classroom, no importa cómo se vieron en el aula).
8. Desarrollar funcionalidades para un sistema de venta.
 - a. Se deberá poder agregar productos (al menos 2 diferentes).
 - b. Se deberá poder generar una venta.
 - c. Darle el enfoque que crean correcto.
9. Debe estar bien documentado y acompañado por un documento PDF que explique brevemente la funcionalidad pretendida.
10. En dicho PDF deberá indicarse por cada tema utilizado del Punto 7 de este documento, donde se encuentra su implementación, a fines de facilitar la corrección. Realizar el mismo comentario en el Summary del código entregado.
11. Si 2 proyectos se parecen en lo más mínimo, serán desaprobados.
12. Quienes no cumplan TODAS estas condiciones, no serán evaluados.

Generar una Solución nombrada como: *Apellido.Nombre.Division.TP4*

El proyecto consta de la siguiente jerarquía de clases:





La clase Producto es la clase padre, de la cual derivan Procesador, PlacaDeVideo y Gabinete.

Todos sus atributos son protected, posee propiedad de lectura y escritura en todos sus atributos.

El constructor por default lo único que hace es settear la cantidad de stock en "1".

El constructor parametrizado inicializa todos los parámetros de la clase.

Dos productos son iguales si tienen la misma Marca y Modelo.

ProductoToString: es un método privado el cual se encarga de mostrar toda la información del producto.

ToString: Se sobrecarga al ToString, llamando al método privado ProductoToString, para mostrar la información completa del producto.

La clase Procesador posee un atributo protected "generacion". A su vez posee una propiedad de solo lectura, la cual devuelve la generación del mismo.

Tiene un único constructor parametrizado el cual settea los atributos del :base, y su atributo único.

ToString: Devuelve toda la información del procesador, llama al ToString del base para reutilizar código.

La clase Gabinete posee un único atributo protected "cantidadDeVentiladores". Este posee dos propiedades, una de lectura y

escritura de la cantidad de ventiladores que tenga el objeto. La otra devuelve el path de la serialización realizada en el form.

El constructor por default, crea un gabinete vacío, sin datos.

El constructor parametrizado settea toda la data del gabinete y de la clase producto.

ToString: Devuelve toda la información del gabinete, llama al ToString del base para reutilizar código.

IDeserialize.Xml: lee el archivo de serialización del gabinete. El cual en el form está hardcoded.

Xml: Guarda un objeto gabinete como xml en el escritorio (Este objeto esta hardcoded en el frm.).

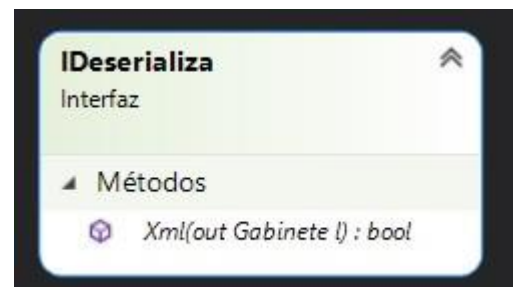
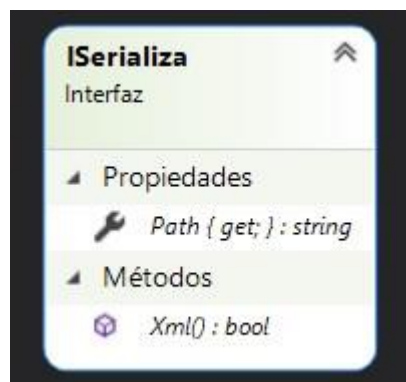
A su vez, implementa las interfaces IDeserialize y ISerialize.

La clase PlacaDeVideo posee un único atributo protected "longitud".

Posee una propiedad de solo lectura de este mismo.

El constructor parametrizado, instancia el objeto con todos los atributos de la clase base y también su propio atributo.

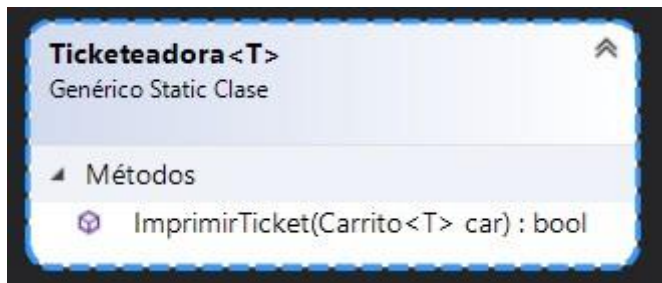
ToString: Devuelve toda la información de la placa de video, llama al ToString del base para reutilizar código.



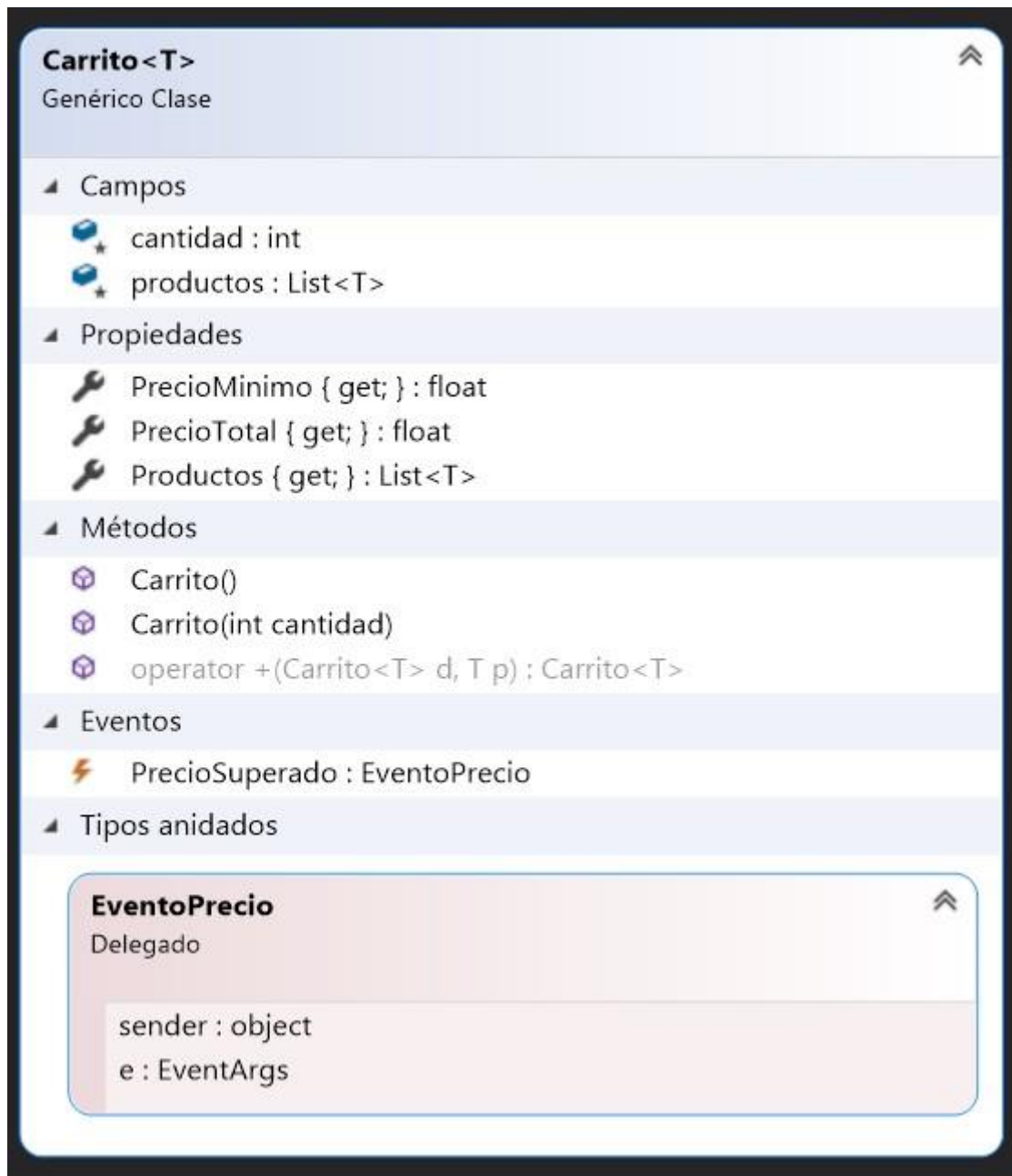
Las interfaces son las encargadas de serializar a XML un objeto Gabinete. ISerializa posee la propiedad de lectura Path la cual recibe la ruta de donde se va a crear el archivo XML, en el caso de este proyecto es el escritorio con el nombre “Nahuel.Cisa.gabinete”.

El Metodo XML es el en cargado de realizar la serialización del objeto, devuelve true si pudo realizarse con éxito, false en caso contrario.

La clase IDeserealiza posee un único método, es el encargado de leer el archivo XML creado en el escritorio, devuelve true si pudo deserializar, false en caso contrario, el único parámetro que recibe es el out de donde se va a guardar el objeto parametrizado.



La clase ticketadora es una clase estatica genérica, la cual posee un único método, este se encarga de a la hora de realizar la venta, crea el ticket de la compra, el ticket se guarda en “documentos”, cada ticket es independiente de cada compra. Recibe como parámetro un carrito, y devuelve true si realizo su tarea exitosamente, o false si hubo algún inconveniente.



La clase genérica Carrito es la encargada de almacenar toda la compra que realiza el cliente, T tiene que es **ESTRICTAMENTE** de tipo Producto, esta posee dos atributos privados, uno es una lista genérica de productos, y el otro la cantidad de productos que puede almacenar dicho carrito, dicha cantidad no puede ser superior a 3 productos.

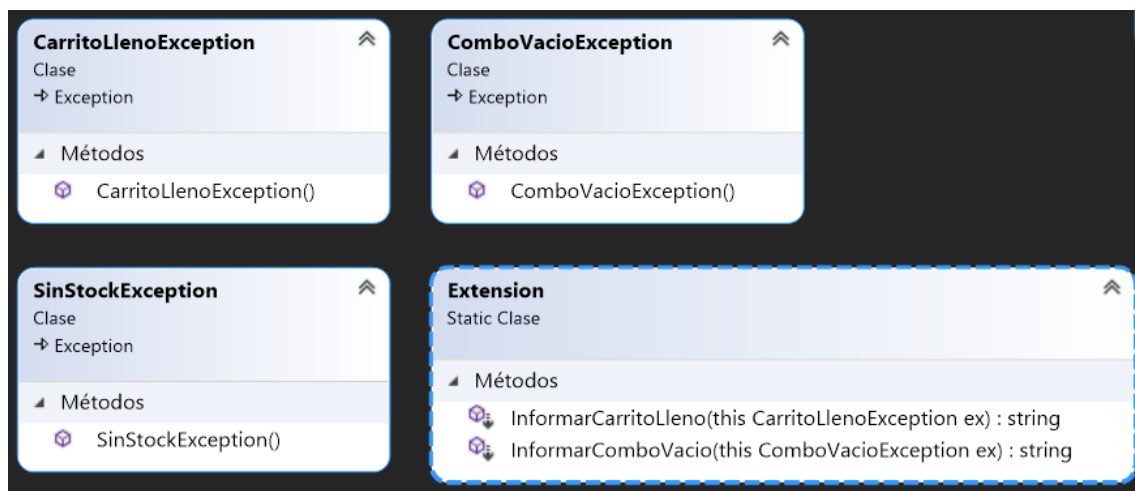
Posee tres propiedades de solo lectura, PrecioMinimo, la cual es la encargada de acumular el precio mínimo del carrito para poder lanzar el EventoPrecio, el cual nos muestra un mensaje que si nuestra compra es mayor a \$500, podemos pagar en cuotas. La propiedad PrecioTotal, es un

acumulador de todos los precios de los elementos que compremos, esta es utilizada en la ticketadora para poner el valor total de la compra. Por último, la propiedad Productos, devuelve la lista de productos del objeto en cuestión.

Posee dos constructores, el constructor por default es el encargado de inicializar la lista y de setear la cantidad de productos a una cantidad por default.

El constructor parametrizado lo único que hace es crear la lista con la cantidad deseada.

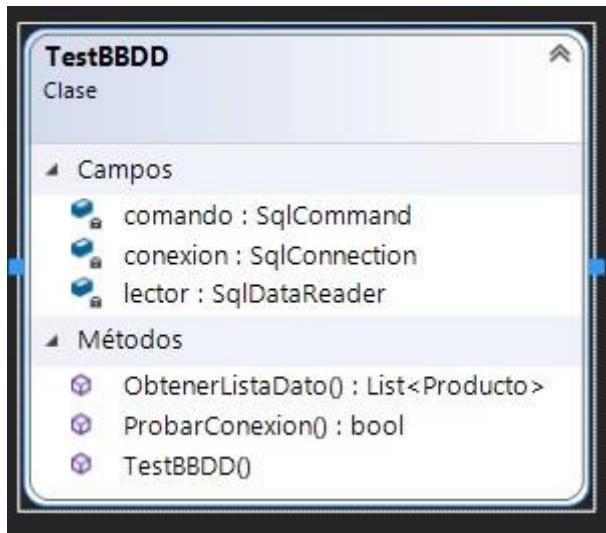
El operador + añade un item a la lista de productos, verifica que no sea null y que la lista tenga espacio, de no tenerlo lanza la excepción CarritoLlenoException. También verifica el precio del producto para ver si este cumple la condición de realizar el invoke al evento de poder pagarlo en cuotas.



CarritoLlenoException: Esta se lanza cuando se desee agregar un producto mas a la lista, y esta no tenga mas espacio. Posee una extensión, la cual es un mensaje informando lo acontecido.

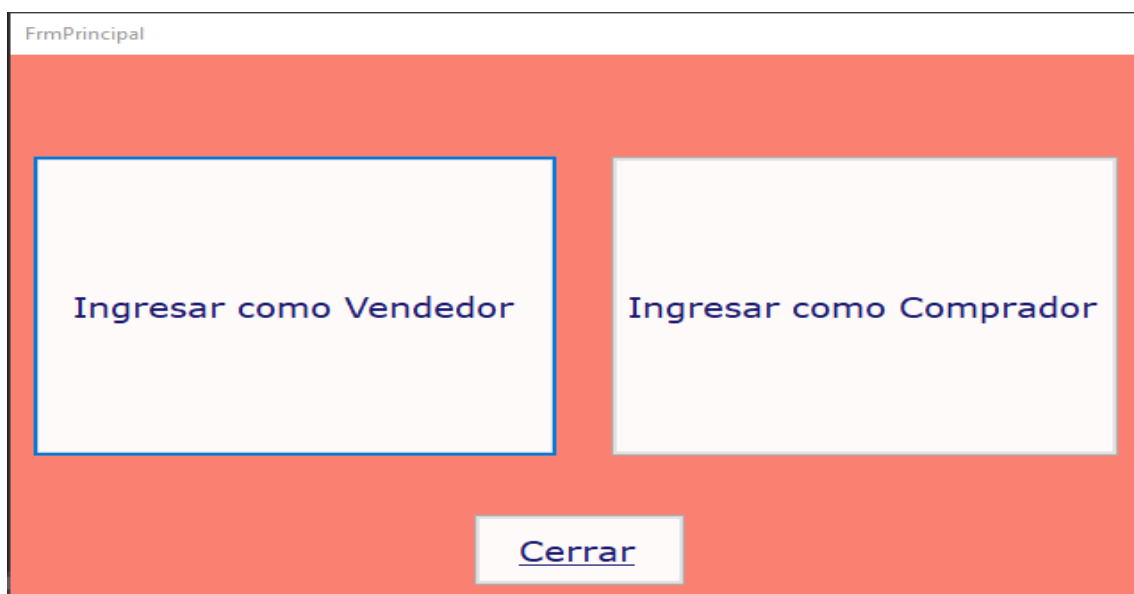
SinStockException: Se lanza cuando se quiera agregar un producto que en la BBDD figure como con Cantidad_de_stock "0".

ComboVacioExeption: Se lanza cuando se quiere crear un nuevo objeto o compra y el combobox encargado de pasarle los parámetros al constructor, esta vacío. Posee una extensión, la cual es un mensaje informando lo acontecido.



La clase TestBBDD fue creada con fines de testear la BBDD en el proyecto consola, prueba la conexión y trae el listado de productos en la bbdd. El constructor settea la conexión a la base de datos.

Al ejecutar el programa nos encontraremos con lo siguiente:



Si tocamos el botón Ingresar como Vendedor, nos abre el siguiente frm:

The screenshot shows a Windows form titled 'FrmVendedor'. It features a table with 6 columns: 'id', 'producto', 'marca', 'modelo', 'precio', and 'cantidad_de_stock'. The table contains 9 rows of data. To the right of the table is a button labeled 'Serializar Gabinete'. Below the table is a large empty rectangular area. At the bottom of the form is a button labeled 'Agregar producto' and a 'Salir' button in the bottom right corner.

id	producto	marca	modelo	precio	cantidad_de_stock
11	Gabinete	Corsair	A70	2500	53
12	Procesador	Sentey	400	34,5	32
13	PlacaDeVideo	Nvidia	2060	95000	45
14	Procesador	Intel	i5 6400	32000	16
15	Gabinete	Cougar	ENG	10500	11
16	PlacaDeVideo	AMD	RX580	19990,99	50
17	PlacaDeVideo	NZXT	596	325	51
18	Gabinete	pirulin	pirulan	22	72
19	Procesador	intel	i5	24189	20

El cual muestra la bbdd, posee dos botones, uno para agregar productos, el cual añade un producto a la bbdd, el frm es de la siguiente forma:

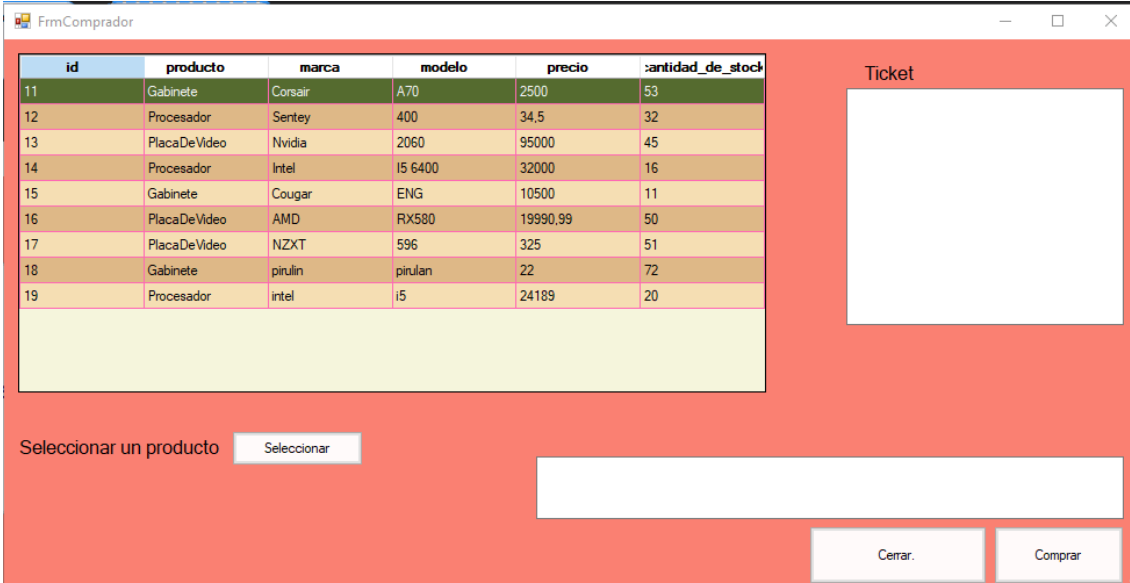
The screenshot shows a Windows form titled 'FrmAdd'. It contains several input fields for adding a new product: 'Producto:' with a dropdown menu, 'Marca:' with a text box, 'Modelo:' with a text box, 'Precio:' with a text box, and 'Cantidad de stock' with a text box. At the bottom are two buttons: 'Aceptar' and 'Cancelar'.

Se eligen los datos, y al darle aceptar, se carga el producto en el data table, el botón serializar, serializa y deserializa un objeto gabinete

hardcodeado y lo lee también, al salir del frm, se actualiza la base de datos a los cambios realizados.

Nos pregunta si estamos seguros de que queremos salir, si le damos que no, continuamos en el frm, caso contrario, nos avisa que la bbdd fue actualizada y cierra el form.

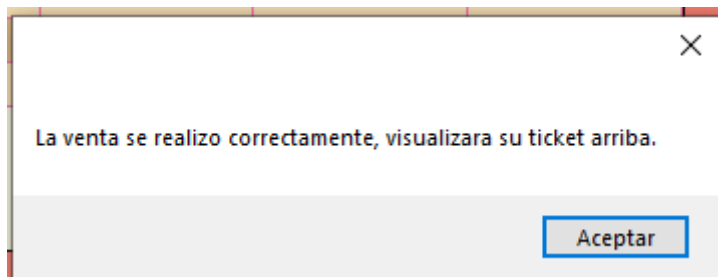
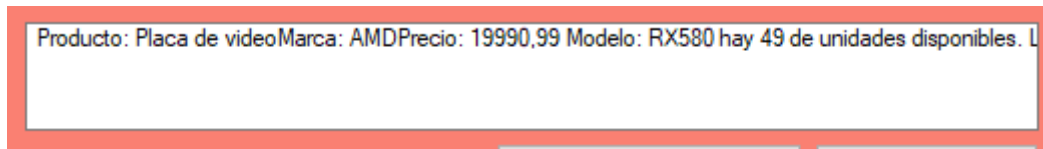
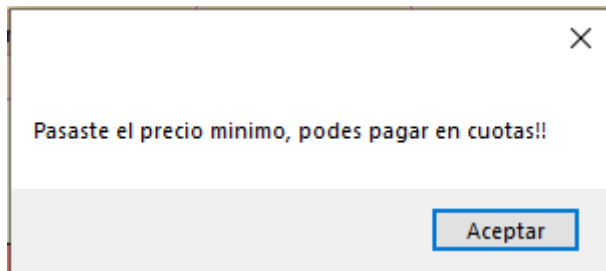
El otro botón nos muestra el siguiente formulario:



The screenshot shows a Windows form titled "FrmComprador". It contains a table with 6 columns: id, producto, marca, modelo, precio, and cantidad_de_stock. The table lists 19 items. To the right of the table is a "Ticket" label and a large empty text box. Below the table is a label "Seleccionar un producto" and a "Seleccionar" button. At the bottom right are "Cerrar" and "Comprar" buttons.

id	producto	marca	modelo	precio	cantidad_de_stock
11	Gabinete	Corsair	A70	2500	53
12	Procesador	Sentey	400	34,5	32
13	PlacaDeVideo	Nvidia	2060	95000	45
14	Procesador	Intel	i5 6400	32000	16
15	Gabinete	Cougar	ENG	10500	11
16	PlacaDeVideo	AMD	RX580	19990,99	50
17	PlacaDeVideo	NZXT	596	325	51
18	Gabinete	pirulin	pirulan	22	72
19	Procesador	intel	i5	24189	20

Nos muestra la bbdd actualizada, esto quiere decir que si salimos o si anteriormente realizamos cambios en ella, se verán reflejados sin tener que cerrar el programa y volverlo a correr. El botón seleccionar nos añade un producto al carrito, antes de añadirlo nos pide que elijamos la característica de dicho producto, en caso de procesador elegir la generación, placa de video la longitud y gabinete la cantidad de ventiladores. Una vez seleccionado el producto, si este vale mas de \$500 se nos muestra un messagebox que dice que superamos el precio mínimo para pagarlo en cuotas (se lanza el evento). Una vez que el producto fue seleccionado, se visualiza en el ListBox de abajo a modo de visualizar los objetos que tenemos en el carrito. Cuando le damos al botón comprar, este genera la compra, actualiza el stock de la bbdd a la cantidad correcta, ya que si compramos, el stock se baja en 1, si se agrega el producto a la bbdd, el stock ya se reduce en uno, pero si no se toca el botón comprar, este no surge ningún efecto, se reinicia al stock normal si la compra es cancelada, en caso de realizar la compra, veremos el siguiente mensaje:



NO SE PUEDE REALIZAR UNA COMPRA SI EL CARRITO ESTA VACIO, nos muestra un mensaje que dice que primero debemos agregar un producto.

A la hora de cerrar el frm de comprador, se nos muestra en un hilo secundario un mensaje el cual nos agradece por comprar en nuestro negocio.

Lugar de implementación de los temas clase 15 a 25:

Clase 15: Excepciones, a la hora de querer añadir un producto al carrito lleno, pasar un combo vacío o querer comprar algo que no tiene stock.

Clase 16: Test unitarios, la solución posee un proyecto llamado Test Unitarios, el cual cumple con las funcionalidades requeridas en el

enunciado, a su vez, se ve complementado con la consola Test para testear funcionalidades principales.

Clase 17: Tipos Genéricos, estos se usan en la clase carrito y en la clase ticketadora.

Clase 18: Interfaces, estos se usaron para la serialización y deserialización xml en la clase gabinete.

Clase 19: Archivos y serialización, archivos se utilizo a modo de “ticket” de compra, y la serialización para serializar el objeto gabinete hardcodeado.

Clase 21 y 22: bbdd y sql, se usaron a modo de “deposito” en los frm comprador y vendedor para exponer los productos.

Clase 23: Hilos, se uso para mostrar un mensaje a la hora de salir del frm de comprador.

Clase 24: Eventos, se uso cuando un producto pasa X precio, lanza el evento que nos avisa que podemos pagarlo en cuotas.

Clase 25: Metodos de extensión, se utilizo para crear un mensaje en las excepciones para mostrarlos por un messagebox.

```
Es el carrito de compra, ahora tiene 0 elementos
Agrego los elementos al carrito, maximo entran 3 productos, al agregar otro, hay una excepcion, carrito lleno exception.
El carrito tiene 3 elementos

El Carrito esta lleno.
-----
testing del toString
-----
Producto: Gabinete
Marca: Corsair
Precio: 46
Modelo: Strix
hay 4 de unidades disponibles.
Cantidad de Ventiladores: 7

Producto: Procesador
Marca: AMD
Precio: 89
Modelo: Ryzen
hay 3 de unidades disponibles. Generacion: 1

Producto: Placa de video
Marca: NVIDIA
Precio: 100
Modelo: Geforce
hay 8 de unidades disponibles. Longitud: 240cm
-----
TEST CONEXION BASE DE DATOS
-----
se conectó
Marca: Corsair
Precio: 2500
Modelo: A70
hay 53 de unidades disponibles.
Marca: Sentey
Precio: 34,5
Modelo: 400
hay 32 de unidades disponibles.
Marca: Nvidia
Precio: 95000
Modelo: 2060
hay 45 de unidades disponibles.
Marca: Intel
Precio: 32000
Modelo: I5 6400
hay 16 de unidades disponibles.
Marca: Cougar
Precio: 10500
Modelo: ENG
hay 11 de unidades disponibles.
Marca: AMD
Precio: 19999,99
Modelo: RX580
hay 49 de unidades disponibles.
Marca: NZXT
Precio: 325
Modelo: 596
hay 51 de unidades disponibles.
Marca: pirulin
Precio: 22
Modelo: pirulan
hay 72 de unidades disponibles.
```

Salida del proyecto TEST de consola.

Alumno: Cisa Nahuel 2A