Universidad Tecnológica Nacional Facultad Regional Avellaneda										
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos										
Materia: Laboratorio de Programación II										
Apellido:			Fecha:		27/06/2019					
Nombre:			Docente(2):		F. Dávila / M. Cerizza					
División:	2°C		Nota(2):							
Legajo:			Firma(2):							
Instancia(1):	**PP**		**RPP**		**SP**	X	**RSP**		**FIN**	

(1) Las instancias validas son: 1er Parcial (**PP**), Recuperatorio 1er Parcial (**RPP**), 2do Parcial (**SP**), Recuperatorio 2do Parcial (**RSP**), Final (**FIN**). Marque con una cruz.

(2) Campos a ser completados por el docente.

Instrucciones:

- Colocar sus datos personales en el nombre del proyecto principal, colocando: Apellido.Nombre.Division. Ej: Pérez.Juan.2D. No sé corregirán proyectos que no sea identificable su autor.
- Al finalizar, colocar la carpeta de la Solución completa en un archivo ZIP que deberá tener como nombre Apellido.Nombre.division.zip y dejar este último en el Escritorio de la máquina.

  Luego presionar el botón  de la barra superior, colocar un mensaje y apretar Aceptar. Finalmente retirarse del aula y aguardar por la corrección.

Criterios de corrección:

- La correcta documentación y reglas de estilo de la cátedra serán evaluadas.
- No se corregirán exámenes que no compilen.
- **Reutilizar** tanto código como crean necesario. No reutilizar código se considera un error.
- Colocar nombre de la clase (en estáticos), **this** o **base** en todos los casos que corresponda.

IMPORTANTE:

- Se debe utilizar y **trabajar sobre el código fuente que acompaña al parcial** (Solución ComiqueriaApp).
- En el proyecto de Windows Forms (la vista) sólo va el código de interacción con el usuario, la lógica del negocio va en las bibliotecas de clase.
- De un nombre descriptivo a todos los métodos y clases que cree.
- Aplique abstracción y encapsulamiento:
  a. Encapsule (agrupe) los datos y los métodos que los operan en clases diferentes según la relación entre ellos y la entidad u objeto que representan.
  b. Oculte los detalles de la implementación siempre que sea posible (modificadores de acceso).

1. Crear en una nueva clase un método de extensión "***FormatearPrecio***" que extienda el tipo Double y devuelva un string con el valor formateado con 2 decimales y el signo $ por delante.
   a. Utilice ***FormatearPrecio*** en el método ***ObtenerDescripcionBreve*** de la clase ***Venta*** para que se devuelva el precio final formateado.

2. Crear una nueva clase "***ComiqueriaException***" que funcionará como un nuevo tipo de excepción del sistema. Tendrá un constructor que recibirá un mensaje y una excepción (innerException). Pasarle los argumentos al constructor de la clase base.

✓ *Ayuda:*
- *Si no sabe crear nuevos tipos de excepción, cuando en un enunciado se pida que se lance una ComiqueriaException utilice la clase Exception (Perderá puntos en tema Excepciones).*

3. En SQLServer crear una nueva base de datos usando el script "***ComiqueriaDB.sql***" que se encuentra en la carpeta "***script_base_datos***" dentro del comprimido entregado al inicio del parcial.

4. Agregue el código necesario para que:
   a. Cuando se presione el botón ***btnAgregar*** en el ***AgregarProductoForm*** y se pasen todas las validaciones existentes, se inserten los datos (descripción, precio y stock) en la tabla de productos.
   b. Al presionar el botón ***btnEliminar*** en el ***PrincipalForm*** y se pasen todas las validaciones existentes, se realice una baja física del producto en la tabla de productos.
   c. Cree también un método que retorne la lista de productos (List<Producto>) almacenada en la tabla de productos. Utilice este método para cargar la lista de productos en la clase ***Comiqueria*** cuando se instancie una nueva comiquería.
   d. Recuerde cerrar siempre las conexiones y evitar repetir código.

✓ *Ayuda:*
- *Si no pudo crear la base de datos o la tabla igual desarrolle el código para la conexión y el ABML.*
- *Recuerde que la conexión e interacción con la base de datos, si bien surge de las peticiones y datos ingresados por el usuario, NO va en la capa de la vista (no va en los formularios).*
- *Estructura básica de las sentencias SQL de manipulación de datos (DML):*
  i. ***SELECT*** *[campo1], [campo2], […]* ***FROM*** *[tabla]* ***WHERE*** *[condicion];*
  ii. ***UPDATE*** *[tabla]* ***SET*** *[campo1] = [valor1], [campo2] = [valor2], […]* ***WHERE*** *[condicion];*
  iii. ***DELETE FROM*** *[tabla]* ***WHERE*** *[condicion];*
  iv. ***INSERT INTO*** *[tabla]* ***(*** *[campo1, campo2, …]* ***) VALUES (*** *[valor1, valor2, …]* ***)****;*
- *La condición (el WHERE) es opcional en todos los casos, se utiliza si se necesita.*
- *El campo* ***codigo*** *de la tabla* ***productos*** *es identidad / autoincremental.*

5. En la clase donde se hace el ABML de productos en la base de datos:
   a. Declare un delegado para métodos que no retornen nada y reciban un parámetro de tipo ***AccionesDB*** (un enumerado que ya se encuentra declarado en el namespace ***ComiqueriaLogic***).
   b. Declare un evento estático del tipo del delegado declarado en el punto anterior.
   c. Cada vez que se realice un insert, delete o update de forma exitosa, emita el evento.
   d. En la clase ***Comiqueria***, en el constructor de instancia, asocie un manejador del evento (declarado también en ***Comiqueria***) que al ejecutarse llame al método que recupera la lista de productos de la base de datos (el que hace el SELECT) y actualice la propiedad ***Productos***.

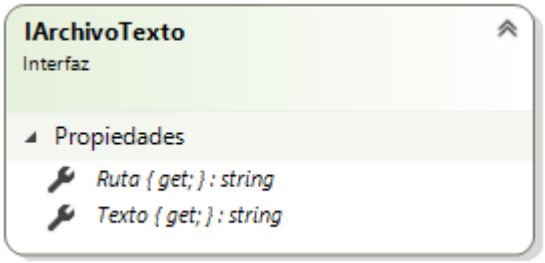
✓ *Ayuda:*
- *Si no supo implementar el código de bases de datos, no deje de realizar estos puntos, si es necesario en los métodos sólo lance el evento y asocie el manejador (de forma de no perder puntos en el tema eventos por no saber bases de datos).*

6. Crear una nueva clase estática parametrizada "***Serializador***" que reciba un sólo tipo genérico:
   a. El tipo genérico deberá ser una clase y tener un constructor sin parámetros público.
   b. Crear un método que permita serializar el tipo genérico de la clase a formato binario.
   c. Crear un método que permita serializar el tipo genérico de la clase a formato xml.
   d. Crear un método que permita deserializar de xml al tipo genérico de la clase.
   e. Crear un método que permita deserializar de binario al tipo genérico de la clase.
   f. Los 4 métodos deben recibir la ruta donde escribir o desde donde leer, además de los otros parámetros (si los tuviera).

g. En al menos uno de los métodos de serialización:
    i. Capturar las excepciones de tipo ***ArgumentException*** y relanzarlas sin perder la pila de llamadas.
    ii. Capturar las excepciones de tipo ***DirectoryNotFoundException*** y lanzar una ***ComiqueriaException*** con el mensaje "Error: Directorio no encontrado.". Guardar la excepción capturada como InnerException.
    iii. Capturar cualquier otra excepción que se produzca y lanzarla con el mensaje "Ocurrió un error, contacte al administrador"
h. Cada vez que se realice una venta (método ***Vender*** de ***Comiqueria***), serializar a binario y a xml la nueva venta.
    i. Recuerde qué condiciones debe cumplir la clase para que sea serializable en cada caso. Se deben serializar todos los atributos de la clase. Modifique lo necesario.
i. Recuerde cerrar siempre las conexiones.

✓ *Ayuda:*
- *Si no sabe trabajar con clases genéricas cree una clase regular y/o trabaje con el tipo Producto (Perderá puntos en tema Generics).*
- *Si no sabe trabajar con serialización, cree y utilice la clase y los métodos tal como se indica, pero no implemente la parte de serialización. No deje de incorporar todo el código específico de generics (Perderá puntos en tema Serialización).*
- *Mismo criterio para los enunciados relacionados a excepciones.*

7. Crear la siguiente interfaz e implementarla en la clase ***ComiqueriaException***:



a. La propiedad Texto debe retornar la fecha y la hora actual y el mensaje de la excepción concatenado con todos los mensajes de las InnerExceptions almacenadas en la cadena de excepciones.
b. La propiedad Ruta contendrá la ubicación donde se guardará el archivo de la excepción, la misma debe ser el directorio correspondiente al escritorio de Windows y tendrá como nombre del archivo "log.txt".

✓ *Ayuda:*
- *Para la ruta pueden ayudarse con la clase Enviroment.*

8. Crear una nueva clase estática "***ArchivoTexto***":
    a. Agregarle un método estático "***Escribir***":
        i. Que reciba cualquier objeto que implemente ***IArchivoTexto*** (NO usar generics).
        ii. Que también reciba un parámetro "append" de tipo bool que indicará: true para anexar datos al archivo, false para sobrescribir el archivo.
        iii. Deberá guardar el texto devuelto por la propiedad Texto en un archivo (de texto) en el path especificado por la propiedad Ruta, que el texto se anexe o sobrescriba según lo indicando por el parámetro "append", el encoding deberá ser UTF8.
    b. Agregarle un método estático "***Leer***" que reciba la ruta a un archivo de texto y devuelva todo su contenido como tipo string.
    c. Utilizar el método ***Escribir*** cuando se genere una nueva ***ComiqueriaException*** de forma que se vaya anexando el Texto de cada excepción en el archivo de log.
    d. Recuerde cerrar siempre las conexiones.

9. En el método "***InicializarFechaHora***" del "***PrincipalForm***" instanciar y correr un nuevo hilo que ejecute un método que actualice el "***lblFechaHora***" cada 1 segundo con la fecha y la hora actual.

- ✓ *Ayuda:*
  - *Recordar que en los formularios no se puede invocar a los controles desde un hilo diferente al principal, para esto deberá usarse el código visto en clase.*

10. Crear un test unitario que verifique alguna de las funcionalidades que programó durante este parcial.
    a. Recuerde que los tests unitarios son un código aislado que se crea con la misión de comprobar otro código muy concreto y no debería llevarle demasiado tiempo crearlo.
    b. Se valorará que use el patrón AAA (Arrange, Act, Assert).