| Universidad Tecnológica Nacional<br>Facultad Regional Avellaneda | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos | | | | | | | | | | |
| Materia: Laboratorio de Programación II | | | | | | | | | | |
| Apellido: | | | Fecha: | | 23/05/2019 | | | | | |
| Nombre: | | | Docente(2): | | F. Dávila / M. Cerizza | | | | | |
| División: | 2°C | | Nota(2): | | | | | | | |
| Legajo: | | | Firma(2): | | | | | | | |
| Instancia(1): | **PP** | | **RPP** | X | **SP** | | **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.

**IMPORTANTE:**

- **2 (dos) errores en el mismo tema anulan su puntaje.**
- La correcta documentación y reglas de estilo de la cátedra serán evaluadas. Comentar el código.
- 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.
- No se corregirán exámenes que no compilen.
- **Reutilizar** tanto código como crean necesario.
- Colocar nombre de la clase (en estáticos), **this** o **base** en todos los casos que corresponda.
- Aplicar los principios de la encapsulación. Los datos sólo deben ser accesibles mediante propiedades y métodos cuando sea necesario y realizando las validaciones y cálculos correspondientes. Se indicará cómo y cuándo hacerlo en las siguientes instrucciones.
- 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.

El cliente pidió ampliar la funcionalidad de la aplicación para la comiquería para incluir un sistema de generación de comprobantes y la posibilidad de modificar el precio de un producto.

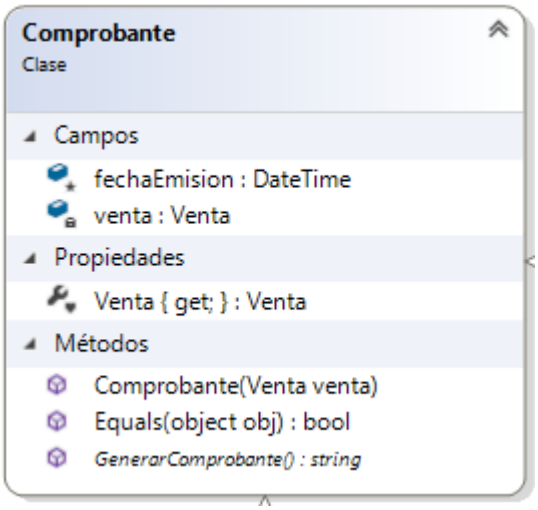
1. La resolución y aprobación de este examen recuperatorio requiere trabajar sobre la solución “ComiqueriaApp”. Es responsabilidad del alumno haber traído este material resuelto por el mismo, cualquier indicio de que no sea de su autoría implicará la desaprobación de ambos alumnos. La solución debe encontrarse funcionando de manera correcta y tal como indican los enunciados del primer parcial.

2. **Agregar las nuevas clases en un NUEVO NAMESPACE** llamado “ComprobantesLogic” (mismo proyecto). Modificar las ya existentes según se indica.

3. **Clase Producto**:
   a. Modificar la propiedad Precio para que sea de lectura-escritura. Permitir que se pueda cambiar el precio validando que el nuevo valor sea mayor o igual a 1, si es menor no hacer nada.

4. **Clase Venta**:
    a. Agregar un método de conversión EXPLICITO de Venta a Producto que devolverá el producto correspondiente a esa venta.
    b. Agregar un campo “cantidad” de tipo int, inicializarlo con el parámetro que ya recibe el constructor y agregar una propiedad de solo lectura que exponga el dato.

5. **Clase Comprobante**:
    a. Crear la clase en un nuevo namespace como se indica en el punto 2.
    b. El constructor inicializará el campo fechaEmision con la fecha de la venta. Inicializar el resto de los campos con los parámetros de entrada.
    c. Tendrá una propiedad de solo lectura que publicará el campo venta. Sólo debe ser accesible desde el mismo ensamblado (assembly, .dll).
    d. El campo fechaEmision debe ser accedido sólo por las clases derivadas y la misma clase.
    e. Tendrá un método “GenerarComprobante” que deberá ser implementado obligatoriamente por las clases derivadas. Haga las modificaciones necesarias a la clase.
    f. Sobrescribir el método Equals. Devolverá true si el objeto a comparar es de tipo Comprobante y la fecha de emisión es la misma.



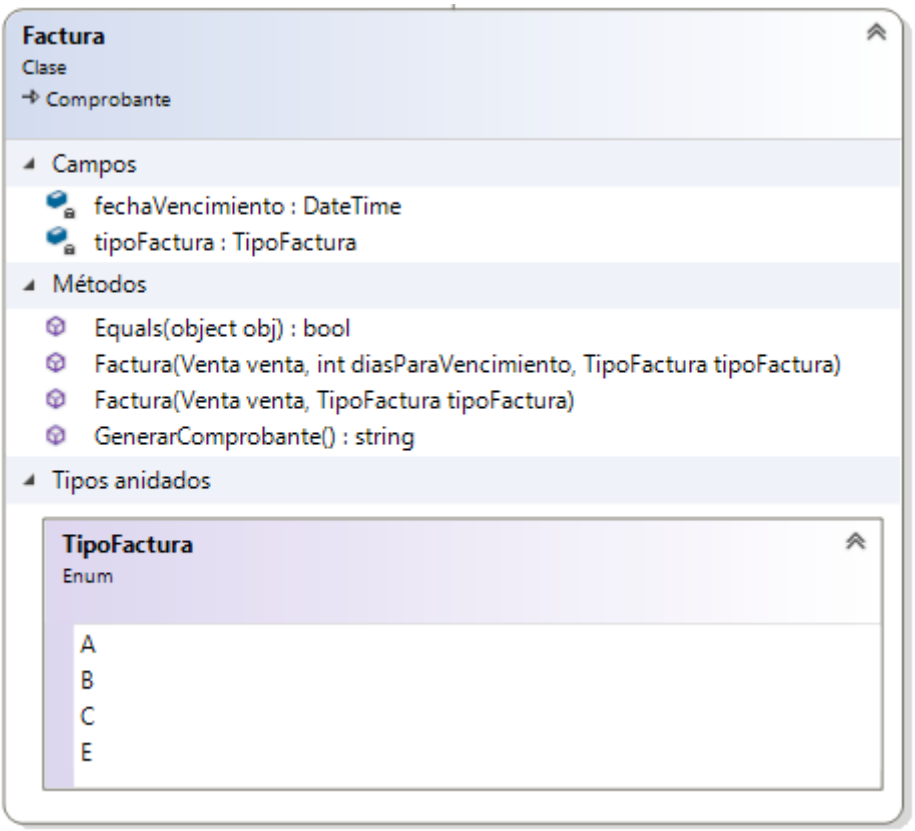

6. **Clase Factura**:
    a. Crear la clase en un nuevo namespace como se indica en el punto 2.
    b. Hereda de Comprobante.
    c. Tiene un enumerado TipoFactura con cuatro posibles valores: A, B, C, E.
    d. El constructor deberá inicializar el campo fechaVencimiento con la fecha actual más la cantidad de días que recibe en el parámetro de tipo int “diasParaVencimiento”. Inicializar el resto de los campos con los parámetros de entrada.
    e. La sobrecarga del constructor que no recibe “diasParaVencimiento” cargará este dato con el valor por defecto de 15 días. Reutilizar código, no repetir.
    f. Sobrescribir el método Equals. Devolverá true si el objeto a comparar es de tipo Factura, la fecha de emisión es la misma (reutilizar código) y el tipo de factura es el mismo (campo tipoFactura).
    g. Implementar “GenerarComprobante”. Deberá devolver un string con los datos de la factura. **Se deberá utilizar StringBuilder y/o métodos de la clase String, no utilizar operador + y derivados (no concatenar).** El texto devuelto debe tener el siguiente formato (reemplazar lo que está entre asteriscos por el valor correspondiente):

*FACTURA *tipoFactura**

*Fecha Emisión: *fecha de emisión**
*Fecha Vencimiento: *fecha de vencimiento**

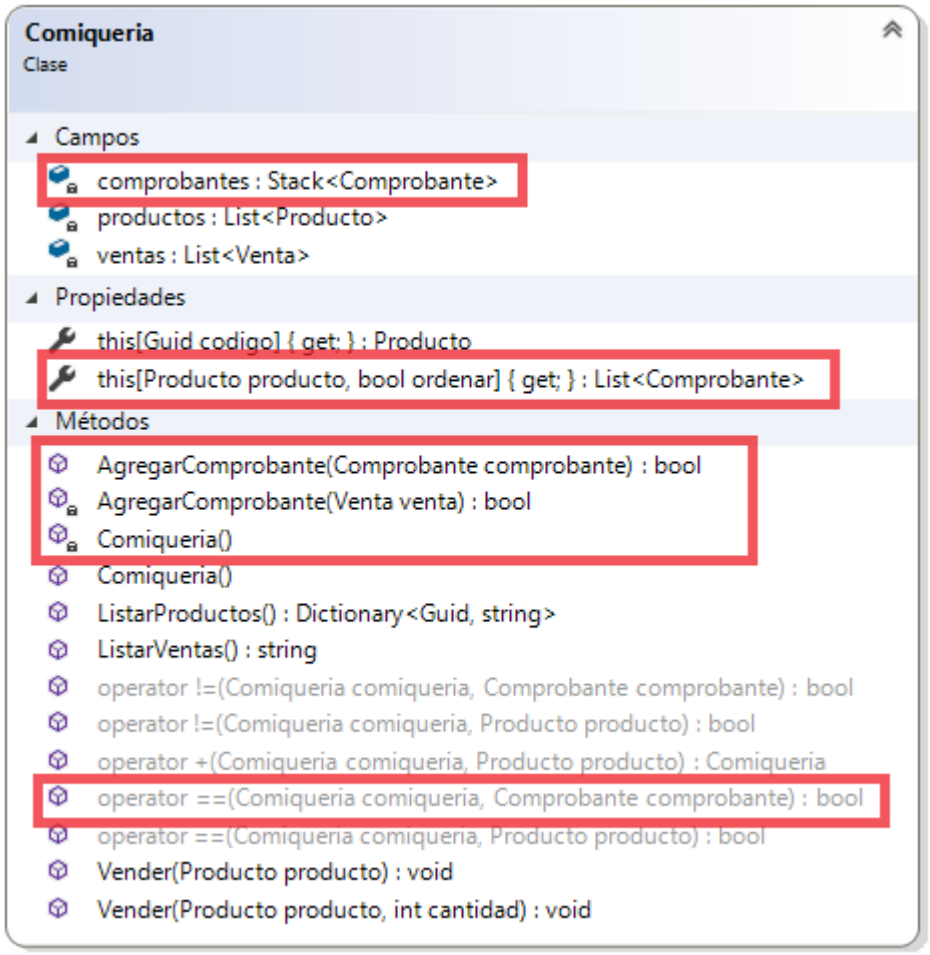
*Producto: *descripción del producto**
*Cantidad: *cantidad de unidades vendidas**
*Precio Unitario: $*Precio unitario del producto formateado con 2 decimales**

*Subtotal: $*Precio unitario por la cantidad**
*Importe IVA: $*Valor agregado por el IVA. Calcular con los otros valores.**
*Importe Total: $*Precio final con IVA**




7. **PrincipalForm**:
   a. Agregar al ***groupBoxAcciones*** un control de tipo button llamado ***btnModificar*** con un texto que diga “Modificar” (Propiedad Text).
   b. El evento click del ***btnModificar*** deberá abrir una ventana del ModificarProductoForm de forma NO MODAL.

8. **Clase Comiqueria**:
   a. Agregar un nuevo campo estático de tipo Stack<Comprobante> con el identificador “comprobantes”.
   b. Agregar un constructor estático que instancie el campo “comprobantes”.
   c. Tendrá un indexador de solo lectura que recibirá como parámetro un Producto y un booleano. Devolverá una List<Comprobante> con los comprobantes correspondientes a ese producto (comparar los códigos de los productos, el que recibe como parámetro y el de la venta del comprobante). Si el booleano es true la lista deberá estar ordenada por fecha de emisión (misma fecha de la venta) de la más antigua a la más reciente, sino deberá mostrarse tal como lo devuelve la pila.
   d. Agregar una sobrecarga del operador == entre Comiqueria y Comprobante. Devuelve true si el comprobante ya se encuentra en la pila de comprobantes, sino false. Reutilizar código y agregar lo que se deba.
   e. Agregar un método de instancia “AgregarComprobante” que reciba un objeto de tipo Comprobante y lo agregue a la colección de “comprobantes”, verificando previamente que no haya un comprobante igual en la pila. Devuelve true si pudo agregarlo, sino false. Reutilizar código.
   f. Agregar una sobrecarga privada al método “AgregarComprobante”, que reciba sólo una venta. Con ese dato instanciar una nueva factura tipo B y agregarla a la lista de comprobantes. Reutilizar código.




9. **ModificarProductoForm**:
    a. Crear un nuevo formulario llamado “ModificarProductoForm” en el proyecto de ComiqueriaApp. ***Recuerde que la clase del formulario sigue las mismas reglas que el resto de las clases. Modifique el constructor, agregue campos, utilice sus eventos según crea necesario. Va a necesitar acceso a la instancia del producto seleccionado en el principalForm.***
    b. Cambiar el título a “Modificar Producto” (Propiedad Text).
    c. Agregar los siguientes controles:
        i. ***lblDescripcion:*** Tipo Label que deberá mostrar la descripción del producto que se está modificando **en negrita** (Investigar propiedad Font) y con la propiedad AutoSize configurada en “true”.
        ii. ***lblPrecioActual:*** Tipo Label con un texto fijo que diga “Precio Actual:”.
        iii. ***txtPrecioActual:*** Tipo TextBox. Propiedad ReadOnly configurada en “true”.
        iv. ***lblNuevoPrecio:*** Tipo Label con un texto fijo que diga “Nuevo Precio:”.
        v. ***txtNuevoPrecio:*** Tipo TextBox.
        vi. ***btnModificar:*** Agregar un control de tipo button con un texto que diga “Modificar” (Propiedad Text).
        vii. ***btnCancelar:*** Agregar otro button de “*Cancelar*”.
        viii. ***lblError:*** Tipo Label. Sin texto por defecto. Propiedad ForeColor configurada en “Red”.
    d. Todos los controles deben tener nombres descriptivos (Propiedad Name) tal como indica el punto anterior.
    e. El ***txtPrecioActual*** debe estar cargado inicialmente con el precio del producto seleccionado formateado para mostrarse con dos decimales.
    f. Al presionar el ***btnCancelar*** se debe cerrar el formulario.
    g. Al presionar el ***btnModificar***:
        i. Se validará con el método **TryParse** de la clase **Double** el nuevo precio ingresado.
        ii. Si no es válido, se cargará el ***lblError*** con el siguiente texto: *“Error. Debe ingresar un precio válido.”*.

iii. Si es válido se deberá mostrar un MessageBox pidiendo confirmación al usuario para realizar la modificación. **El mensaje debe tener un título, un ícono de tipo "Warning" y botones de sí y no "YesNo".** Si el usuario presiona Sí, actualizar el precio del producto y cerrar la ventana. Si presiona No, no hacer nada.

h. Asociar el evento TextChanged del ***txtNuevoPrecio***. En el manejador del evento (método al que llama), borrar el texto del ***lblError*** (cargarlo con un string vacío).

i. El aspecto del formulario debe ser similar al de la imagen: