TRABAJO PRÁCTICO FINAL UTN 2E

ALUMNO: BRUNO DE RENZIS

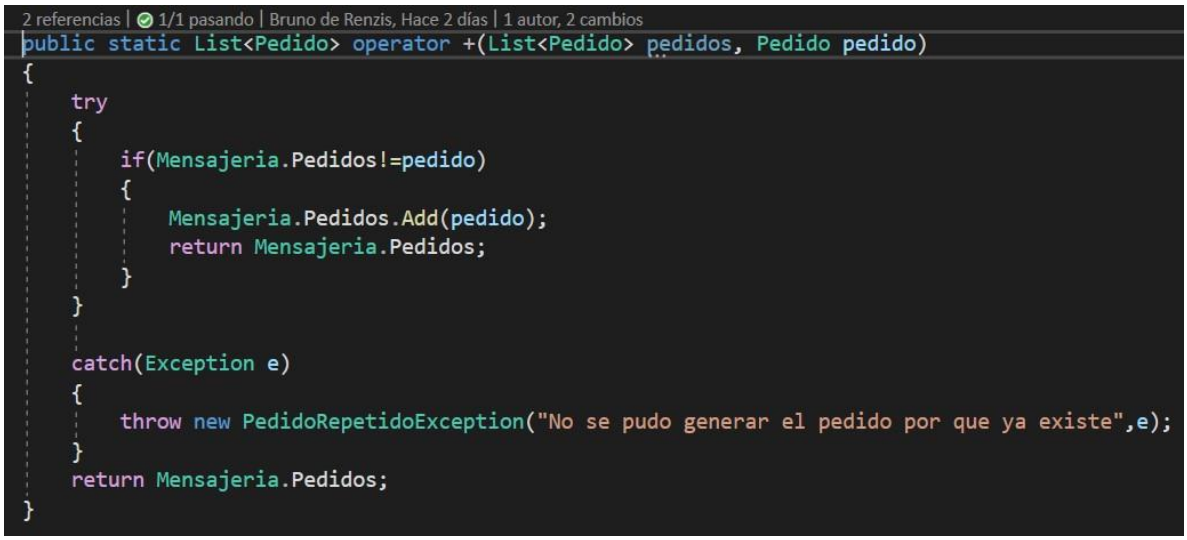
PROFESOR: FEDERICO DÁVILA

La funcionalidad del trabajo práctico es la de simular el proceso de pedidos de una empresa de mensajería. El programa consta de envíos de productos hacia los clientes. Un pedido consta de una lista de productos y el cliente destinatario. El pedido tiene estado de generado en el momento que se crea, y posteriormente al despacharse podrá cambiarse a entregado o no entregado.

Los temas utilizados en este trabajo práctico han sido:

Clase 15: Excepciones

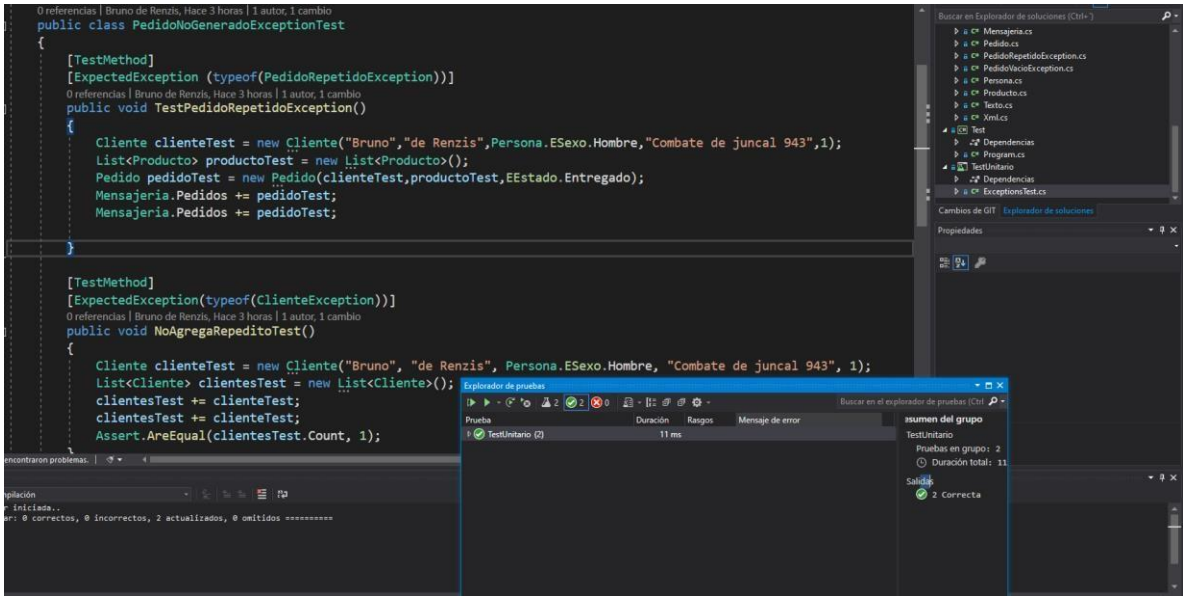
Clase FrmPrincipal



Las excepciones están implementadas en el código para que al momento de estar frente a algún posible error que detenga la ejecución del programa, esta pueda informar de tal error y seguir con el flujo normal del programa, se las utilizará en gran parte del código, se ve en el ejemplo como se utiliza para que al momento de agregar un elemento a la lista, no haya error alguno en su carga.

Clase 16: Test Unitarios

Clase ExceptionsTest



Los test unitarios son útiles al momento de probar aisladamente alguna funcionalidad para comprobar su correcta ejecución. Se los utilizará para probar la ejecución de excepciones al momento de agregar un pedido ya existente o un cliente ya existente.

Clase 17: Tipos genéricos

Clase Xml

```
public class Xml<T> : IArchivo<T>
{
    /// <summary>
    /// Guarda un archivo en formato xml
    /// </summary>
    /// <param name="archivo"></param>
    /// <param name="datos"></param>
    /// <returns>true si se guardo correctamente, false caso contrario</returns>

    public bool Guardar(string archivo, T datos)
    {
        try
        {
            if (archivo != null)
            {
                using (XmlTextWriter writer = new XmlTextWriter(archivo, Encoding.UTF8))
                {
                    XmlSerializer ser = new XmlSerializer(typeof(T));

                    ser.Serialize(writer, datos);

                    return true;
                }
            }
        }
        catch (Exception e)
        {
```

Los tipos genéricos se implementan con la finalidad de facilitar la escritura del código ante métodos listas atributos o interfaces que permitirán una fácil sobreescritura con el desarrollo del método ya establecido. Se lo utilizará para escribir y leer las distintas listas generadas en xml.

Clase 18: Interfaces

IArchivo

```
namespace Entidades
{
    interface IArchivo<T>
    {
        /// <summary>
        /// Guarda los datos de un archivo.
        /// </summary>
        /// <param name="archivo"></param>
        /// <param name="datos"></param>
        /// <returns>True si pudo guardar cambios, false si no guardó</returns>
        bool Guardar(string archivo, T datos);

        /// <summary>
        /// Lee los datos de un archivo
        /// </summary>
        /// <param name="archivo"></param>
        /// <param name="datos"></param>
        /// <returns>true si se pudo leer el archivo, false si no se pudo leer el archivo.</returns>
        bool Leer(string archivo, out T datos);
    }
}
```

Las interfaces se utilizan para desarrollar métodos en la clase que implemente dicha interfaz de manera obligatoria para la correcta compilación del programa. Se lo utilizará para que la clase serializadora Xml desarrolle de manera obligatoria ambos métodos y asegurarse así de la correcta carga y lectura de los archivos.

Clase 19: Archivos

```
static FormPrincipal()
{
    try
    {
        List<Cliente> clientesLeer = new List<Cliente>();
        List<Producto> productosLeer = new List<Producto>();
        List<Pedido> pedidosLeer = new List<Pedido>();

        Xml<List<Cliente>> clientes = new Xml<List<Cliente>>();
        Xml<List<Producto>> productos = new Xml<List<Producto>>();
        Xml<List<Pedido>> pedidos = new Xml<List<Pedido>>();

        string pathClientes = String.Concat(AppDomain.CurrentDomain.BaseDirectory, "Clientes.xml");
        string pathProductos = String.Concat(AppDomain.CurrentDomain.BaseDirectory, "Productos.xml");
        string pathPedidos = String.Concat(AppDomain.CurrentDomain.BaseDirectory, "Pedidos.xml");
        clientes.Leer(pathClientes, out clientesLeer);
        productos.Leer(pathProductos, out productosLeer);
        pedidos.Leer(pathPedidos, out pedidosLeer);
        Mensajeria.Clientes = clientesLeer;
        Mensajeria.Productos = productosLeer;
        Mensajeria.Pedidos = pedidosLeer;


    }
```

Los archivos se utilizarán para serializar y poder leer los pedidos realizados al iniciar el programa, cumpliendo la función de un archivo de recupero de datos. Para que esta funcionalidad sea correctamente implementada el usuario deberá guardar la nueva lista de pedidos antes de cerrar la aplicación. Estos archivos se leen y se guardan en la carpeta bin del proyecto de formulario.

Test (Aplicación de consola):

El test de consola es un proyecto creado con la finalidad de testear las funcionalidades del programa. En este se observa como al agregar un producto o pedido se corroborará previamente que este no exista de otro modo no lo agregará, cumpliendo de manera correcta la funcionalidad de la sobrecarga de operadores, y se crearán pedidos para testear los informes de la aplicación como en este caso cual es el sexo que realizó más pedidos.