Trabajo Practico N.4

Juan Gabriel Palermo – 2do “A”

Profesor: Maximiliano Neiner

Objetivo del programa:

El programa funciona como un punto de venta, desde la perspectiva del vendedor.

A través del mismo, se puede:

- Agregar productos al deposito
- Seleccionarlos y acumularlos dentro de una venta
- Imprimir los tickets de las ventas
- Ver la información detallada de los productos
- Exportar los datos del depósito a través de un archivo XML.

Además, la información almacenada en el DataTable se actualiza automáticamente durante toda la ejecución del programa.

Por otro lado, si se cumple el requisito de monto de la compra, se lanzará un evento que le notificará al cajero que debe darle un váucher de descuento para su próxima compra al cliente.

Temas a utilizar en la realización del Trabajo Practico:

Excepciones

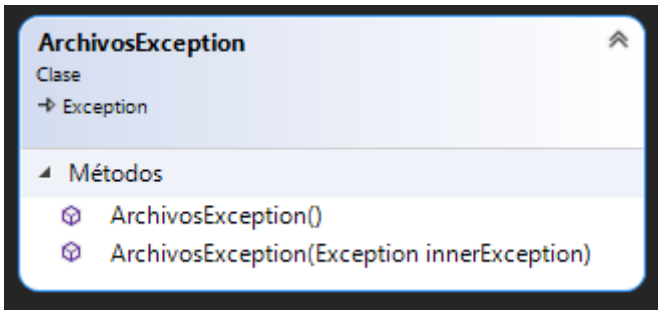
Se crean estructuras try-catch para que controlen las posibles excepciones que puedan surgir a lo largo de la ejecución del programa. Por ejemplo, se adjunta la generada en uno de los eventos del Form:

Ruta de la imagen: Proyecto: PuntoDeVenta // Form: PuntoDeVenta // Región: Eventos – Ciclo de Vida

```
private void PuntoDeVenta_Load(object sender, EventArgs e)
{

    try
    {
        CrearDataTable();
        ConfigurarGrilla();
        ConfigurarDataAdapter();
        CargarGrilla();
        compra = new Compra<Periferico>();

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

    try
    {
        hiloDataTable = new Thread(this.CargarGrillaHilo);
        hiloDataTable.Start();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
```

Además, también se agrega una excepción personalizada para Archivo.

Ruta de la imagen: Proyecto: Excepciones // Clase: ArchivosException.cs



Test Unitarios

Se crea un proyecto de Test Unitarios, validando algunas de las funcionalidades creadas en el Trabajo Practico. Particularmente, se crea una ejecutando un Assert (validando que no sea null la compra creada), y también otra que espera que se lance una excepción personalizada al momento de pasarle un directorio inexistente a la creación de un archivo txt.

Se adjunta la imagen:

Ruta de la imagen: Proyecto: TestUnitarios // Clase: TestUnitarios.cs

```
namespace TestUnitarios
{
    [TestClass]
    0 referencias
    public class TestUnitarios
    {
        [TestMethod]
        0 referencias
        public void TestColeccionCompra()
        {
            Compra<Periferico> compra = new Compra<Periferico>();

            Assert.IsNotNull(compra.Elementos);
        }

        [TestMethod]
        [ExpectedException(typeof(ArchivosException))]
        0 referencias
        public void TestArchivosTexto()
        {
            Teclado teclado = new Teclado(Periferico.Marca.Gigabyte, 123, "Prueba Teclado");

            Texto txtaux = new Texto();

            txtaux.Guardar(@"ABC:\DirectorioErroneo.abc", teclado.ToString());

        }

    }
}
```

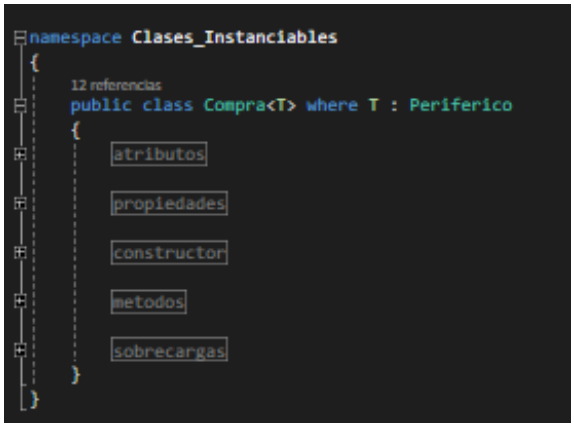
Generics e Interfaces

Se crea una clase genérica, que permitirá almacenar los productos que se están almacenando en la compra, a fin de poder calcular el valor total del ticket, y también de poder imprimirlo con el detalle de los productos.

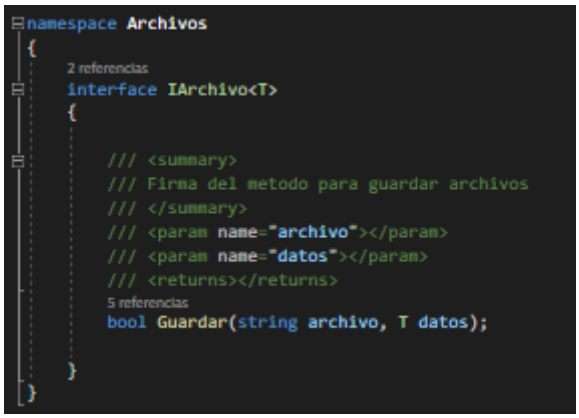
Además, también se crea una interfaz que es genérica, la cual contendrá la firma del método Guardar para los archivos.

Se adjuntan las imágenes:

Ruta de la imagen: Proyecto: Clases Instanciables // Clase: Compra.cs

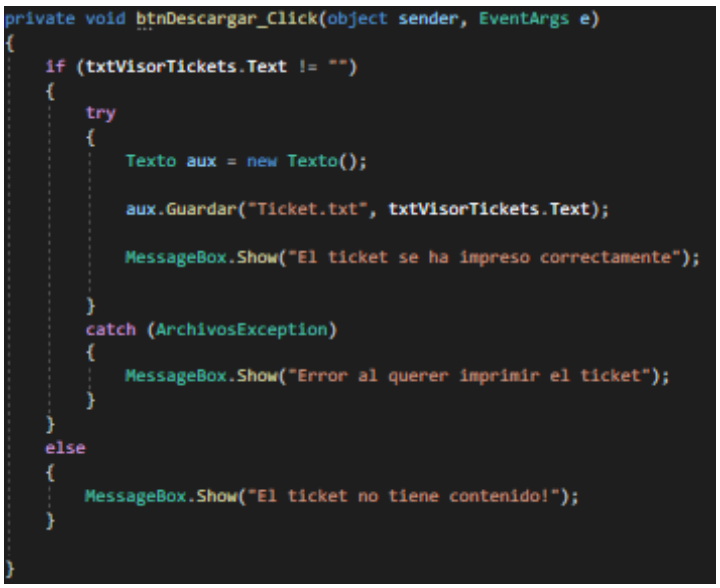


Ruta de la imagen: Proyecto: Archivos // Interfaz: IArchivo.cs



Archivos y Serialización

Se crea archivo de tipo txt, el cual contendrá el valor de la última venta realizada. El mismo se guardará en el directorio por defecto.
Carpeta: tp_laboratorio_2\TPS_ProgYLabo2\TP4\Palermo.Juan.2A.TP4\PuntoDeVenta\bin\Debug
La creación de este archivo será controlada por el botón que dice “Descargar ticket de compra”.
Se adjunta imagen:
Ruta de la imagen: Proyecto: PuntoDeVenta // Form: PuntoDeVenta // Región: Eventos – Botones



La serialización se hace a XML, y la misma es controlada por el evento FormClosing del Form principal (PuntoDeVenta), que llama al método GuardarXml. Esta serialización hará que se almacene en un archivo XML tanto el contenido del DataTable, como el Schema del mismo.

Se adjunta imagen:
Ruta de la imagen: Proyecto: PuntoDeVenta // Form: PuntoDeVenta // Región: Métodos – Ciclo de Vida

```
1 referencia
private bool GuardarXml()
{
    bool res = true;

    try
    {
        this.dt.WriteXmlSchema(PATH_XML_PRODUCTOS_SCHEMA);
        this.dt.WriteXml(PATH_XML_PRODUCTOS);
    }
    catch (Exception e)
    {
        res = false;
        MessageBox.Show(e.Message);
    }

    return res;
}
```

SQL

Se traen los productos del deposito (BBDD SQL) hacia el DataGridView del Form principal.
A través del form, se pueden agregar nuevos productos, que se sincronizarán con la base de datos, y los productos que estén, se podrán seleccionar para agregar a la venta que se esta realizando.
Esto se configura a través de diferentes métodos dentro de la región Métodos del Form principal, que son llamados en el evento Load del mismo, y también del proyecto Bases de Datos, en el archivo ConexionBD.cs.

Se adjuntan imágenes:

Ruta de la imagen 1: Proyecto: PuntoDeVenta // Form: PuntoDeVenta // Región: Eventos – Ciclo de Vida

Ruta de la imagen 2: Proyecto: Bases de Datos // Archivo: ConexionBD.cs

```
private void PuntoDeVenta_Load(object sender, EventArgs e)
{

    try
    {
        CrearDataTable();
        ConfigurarGrilla();
        ConfigurarDataAdapter();
        CargarGrilla();
        compra = new Compra<Periferico>();

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

```
namespace Bases_de_Datos
{
    4 referencias
    public static class ConexionBD<T> where T : Periferico
    {
        Atributos

        Constructor

        Metodos
    }
}
```

Hilos

Se crea un hilo que permita actualizar el Data Table automáticamente, cada vez que se agregue un nuevo producto.
El mismo comienza en el evento Load del form principal; y se aborta en el evento FormClosing del mismo.
Se adjuntan imágenes:

```
try
{
    hiloDataTable = new Thread(this.CargarGrillaHilo);
    hiloDataTable.Start();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

```
try
{
    hiloDataTable.Abort();
}
catch (ThreadAbortException)
{

}
```

Eventos y Delegados

Se crea el evento compra_EventoPrecio, que hace que se le lance una notificación al vendedor en el caso de que el precio total de la compra del cliente sea mayor a $20000, indicándole que debe darle un váucher de descuento para su próxima compra al cliente.
La notificación se enviara al momento de confirmar la compra.

Se adjunta imagen:

Ruta de la imagen: Proyecto: PuntoDeVenta // Form: PuntoDeVenta // Región: Eventos – Evento Precio

```
1 referencia
private void compra_EventoPrecio(object sender, EventArgs e)
{
    MessageBox.Show("El cliente ha alcanzado el premio, entregarle cupon de descuento para su proxima compra.");
    this.compra.PremioAlcanzado = null;
}
```

Paralelo al evento, se crea el delegado EventoDescuento, el cual funcionara para poder administrar dicho evento.

Se adjunta imagen:

Ruta de la imagen: Proyecto: Clases Instanciables // Archivo: Compra.cs

```
public delegate void EventoDescuento(object sender, EventArgs e);

public EventoDescuento PremioAlcanzado;
```

Métodos de extensión

Se crea un método de extensión que permita obtener la información detallada del Row seleccionado en el DataTable, en este caso, además de la información que muestra la sobreescritura del ToString() del mismo, le agrega la información sobre la Garantía.

La utilización de este método esta controlada por el botón que dice “Mostrar info del producto”.

Se adjunta imagen:

Ruta de la imagen: Proyecto: Extension // Archivo: Extension.cs

```
0 referencias
public static class Extension
{
    /// <summary>
    /// Metodo de extension que me permitira traer los datos que devuelve el ToString del objeto que le paso como parametro.
    /// Ademas, le agrega la informacion de la garantia.
    /// </summary>
    /// <param name="p">Objeto del que se obtendran los datos</param>
    /// <returns>string con los datos del objeto, y la Garantia</returns>
    3 referencias
    public static string ObtenerInfoExtendida(this Periferico p)
    {

        StringBuilder sb = new StringBuilder();

        sb.AppendLine(p.ToString());
        sb.AppendLine("Cuenta con garantia? -> " + p.Garantia.ToString());

        return sb.ToString();
    }

}
```