TP4 Laboratorio 2, Sagnella Franco Ezequiel

La funcionalidad que buscaba con este proyecto era realizar un sistema de ventas de electrodomésticos, los electrodomésticos que se venden pueden ser cafeteras y televisores.

Funciona de la siguiente manera:

- Primero se abre un formulario que representa al comercio, con una tabla que contiene el stock de productos
- Dicho stock de productos se carga desde una base de datos, y también se dispone de un ABM para poder modificarlo en tiempo real (Al cerrarse el formulario se sincroniza con la base de datos para guardar los cambios).
- El formulario permite crear un carrito de compra.
- Dicho carrito es un nuevo formulario con su propia tabla. El usuario puede seleccionar productos del stock del Comercio y agregarlos al carrito, se pueden agregar y eliminar del carrito tantos productos como se necesite.
- Desde el carrito se puede cancelar la compra, lo que cierra el formulario, o confirmar, haciendo que se imprima en pantalla el resumen de dicha compra y además lo imprima en un archivo que guarda el historial de ventas .
- Por último, desde el formulario del Comercio, con un botón se puede abrir el archivo que guarda el historial de las ventas.

TEMAS UTILIZADOS:

Clase 15:

El proyecto tiene una biblioteca de clases llamada Excepciones, donde están definidas todas las clases de excepciones que cree
las excepciones creadas son:

```
/// <summary>
/// Clase archivos exception, se lanzaría en caso de que se genere algún
/// error leyendo o escribiendo en archivos
/// </summary>
public class ArchivosException : Exception

/// <summary>
/// Excepción que se lanzará en caso de que se intente hacer alguna acción sobre
/// el carrito mientras este no este abierto, o si intentan abrir dos al mismo tiempo
/// </summary>
public class CarritoAbiertoException : Exception

/// <summary>
/// Excepción que se lanza en caso de que se intente asignar un modelo
/// erróneo a un objeto Electrodoméstico
/// (por ejemplo, si se crea una cafetera y se le asigna un modelo propio de televisores)
/// </summary>
public class ModeloException : Exception
```

```
/// <summary>
/// Excepción que se lanza si se intenta agregar al carrito un elemento que ya fue
/// agregado anteriormente
/// </summary>
public class ProductoRepetidoException : Exception

/// <summary>
/// Excepción para cuando se ingrese un precio con un formato invalido
/// a un producto
/// </summary>
public class PrecioInvalidoException : Exception

/// <summary>
/// Excepción que se lanza cuando no se especifica el tipo
/// de producto al crearlo
/// </summary>
public class ProductoInvalidoException : Exception
```

Clase 16:

La solución tiene un proyecto llamado TestsUnitarios donde se prueban las siguientes funcionalidades:

```
/// <summary>
/// Método Test que valida que se instancie bien un objeto Tv
/// </summary>
[TestMethod]
public void Validar_Instancia_Tv()

/// <summary>
/// Metodo test prueba que se instancien objetos Cafetera
/// </summary>
[TestMethod]
public void Validar_Instancia_Cafetera()

/// <summary>
/// Prueba que se lance correctamente la excepción
/// ModeloException al crear un objeto con un modelo válido
/// </summary>
[TestMethod]
[ExpectedException(typeof(ModeloException))]
public void ModeloException_Prueba_Tv()

/// <summary>
/// Prueba que se lance correctamente la excepción
/// ModeloException al crear un objeto con un modelo válido
/// </summary>
```

```
[TestMethod]
[ExpectedException(typeof(ModeloException))]
public void ModeloException_Prueba_Cafetera()

/// <summary>
/// Prueba que se imprima correctamente un archivo de texto
/// </summary>
[TestMethod]
public void Prueba_ImprimirTicket()

/// <summary>
/// Prueba que se lea correctamente de un archivo de texto
/// </summary>
[TestMethod]
public void Prueba_LeerTicket()
```

Clase 17 y 18:

Cree una interfaz genérica, que define firmas de métodos para manejar archivos de texto Esta interfaz se llama IArchivos, está en el proyecto Archivos, es implementada por la clase ArchivoTexto (Que se encuentra en el mismo proyecto).

Ademas, la clase Ticketeadora, dentro del proyecto Archivos, tambien es generica, restringida para solamente ser utilizada con tipos Electrodomestico (O clases hijas de Electrodomestico).

Clase 19:

La lectura y escritura de archivos son manejadas por la clase Ticketeadora, se encuentra en el proyecto Archivos: posee los métodos :

```
/// <summary>
/// Permite imprimir los datos de una venta en un archivo de texto
/// </summary>
/// <param name="obj"></param>
/// <param name="path"></param>
/// <returns></returns>
public static bool imprimirHistorialVentas(object obj, string path)

/// <summary>
/// Permite leer de un archivo de texto los datos de las ventas
/// </summary>
/// <param name="path"></param>
/// <returns></returns>
public static string Leer(string path)
```

que internamente utilizan los métodos de la interfaz IArchivos que fueron implementados por la clase ArchivoTexto

Clase 21 - 22:

Cree una clase llamada ManejadorSql, en el proyecto EntidadesInstanciables, que tiene como atributo un SqlDataAdapter, posee un método ConfigurarDataAdapter donde se configura la conexión y los comandos.

Al iniciar la aplicación, en el evento FormLoad se ejecuta el método Fill de DataAdapter para cargar el stock con los elementos de la base de datos, y al cerrarla, en el evento FormClosing se ejecuta el método Update del DataAdapter, para guardar en la base de datos todos los cambios que se hayan producido en el stock.

Clase 23:

Para utilizar hilos desarrolle un método, se encuentra en la clase FrmComercio, dentro del proyecto Formularios:

```
/// <summary>
/// Método que se ejecuta en el hilo secundario para mostrar
/// un array con imágenes que va cambiando en el tiempo
/// </summary>
/// <param name="param"></param>
private void MostrarPublicidad(object param)
```

la idea es que el método se ejecute en un hilo secundario y que vaya mostrando diferentes imágenes a modo de publicidad, mientras el usuario puede seguir comprando en el hilo principal.

Clase 24:

Tengo una clase Delegados donde está definido:

```
/// <summary>
/// Delegado para manejar un evento que permite añadir items al carrito de compra
/// </summary>
/// <param name="fila"></param>
public delegate void AgregarAlCarritoDelegado(DataRow fila);
```

La clase FrmComercio define un evento con este delegado. El evento es lanzado en el manejador de eventos del form **btnAgregarCarrito,** al tocar este botón, el evento agregarAlCarritoEvento se asocia con el metodo **AgregarProducto** definido en la clase FrmCarrito, después el evento se invoca:

```
/// <summary>
/// Manejador de eventos para el evento agregarAlCarritoEvento del formComercio,
/// crea una nueva fila con los datos de la fila que recibe como parámetro
/// y lo guarda en la lista de electrodomésticos
/// </summary>
```

/// <param name="fila"></param>
public void AgregarProdutco(DataRow fila)

De esta forma logré comunicar un Form con el otro, le paso desde el formulario del comercio una fila al carrito para que la agregue.

clase 25:

Defino métodos de extensión en la clase MetodosExtension, que se encuentra en el proyecto MetodosExtension. En ella se implementa el siguiente método:

/// <summary>
/// Metodo de extension de la clase Ticket, devuelve una cadena con el
/// resumen de una venta
/// </summary>
/// <param name="t"></param>
/// <param name="obj"></param>
/// <returns></returns>
public static string ObtenerResumenVenta(this Ticketeadora t, object obj)

La idea es agregarle a la clase ticketeadora un método que devuelva una cadena con el resumen de la compra, para mostrarla en el momento en que se realiza, entonces la clase ticketeadora sirve tanto para escribir y leer los archivos con el historial de ventas, como para imprimir en el momento el resumen de la compra