Universidad Tecnológica Nacional
Facultad Regional Avellaneda

Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos

Materia: Laboratorio de Programación II			
Apellido: Zafferano		Fecha:	06-06-2022
Nombre: Gonzalo Fabian		Docente:	Davila, Federico
División: 2° C	2°C	Nota:	
Legajo:		Firma:	
Instancia:	**TP 3**		

Consigna:

Desarrollar funcionalidades para un ***sistema de atención de público***:

-Deberá proveerse un contexto y tipo de empresa, así como una dinámica realista.

Se deberá poder cargar gente desde distintas fuentes (archivos, base de datos, a mano) y registrar datos de su atención.

-Se deberá poder importar y exportar información, o consultar alguna fuente, o dar datos de entrada, o similar.

-También se deberá poder cargar y exportar esa información a mano.

-Darle el enfoque que crean correcto, teniendo en cuenta que el sistema tiene que permitir el ingreso y atención del público.

-Debe estar bien documentado y acompañado por un documento PDF que explique brevemente la funcionalidad pretendida.

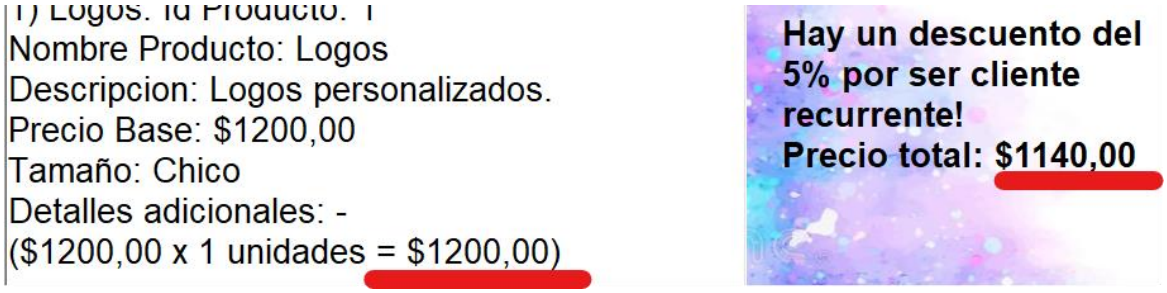
En dicho PDF deberá indicarse por cada tema utilizado del Punto 6 o 7 de este documento (dependiendo la instancia en la que se encuentren), y marcar donde se encuentra su implementación, a fines de facilitar la corrección. Realizar el mismo comentario en el Summary del código entregado.

Resolución:
Para empezar, podemos ***iniciar sesión*** ingresando el **usuario** y **contraseña.**
Sin embargo, para ***agilizar*** las pruebas, podemos ***AUTOMATIZAR*** el ingreso, al presionar sobre el botón ***MAS>>,*** que nos habilita la opción de ingresar como un **Usuario, Administrador o Jefe.**
-El ***usuario*** podrá atender al público.
-El ***Administrador***, además de atender, podrá gestionar clientes, productos y otros empleados (*NO otros administradores*). Por gestión se entiende alta/baja/modificación.
El ***Jefe***, además de atender y gestionar clientes, productos y otros empleados (*incluidos administradores*), puede agregar un nuevo dueño, alta y baja de administradores, y revisar ***todos los tickets*** *que se van generando por compra*.

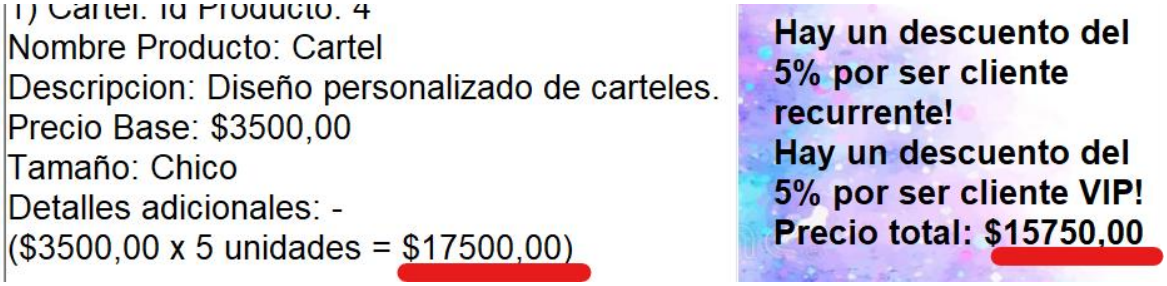
¡Importante!
TODOS LOS ARCHIVOS SE GENERAN EN LA RUTA:
Zafferano.Gonzalo.2C.TP3\Vista\bin\Debug\net 5.0-windows\RespaldoArchivos

Desarrollo:

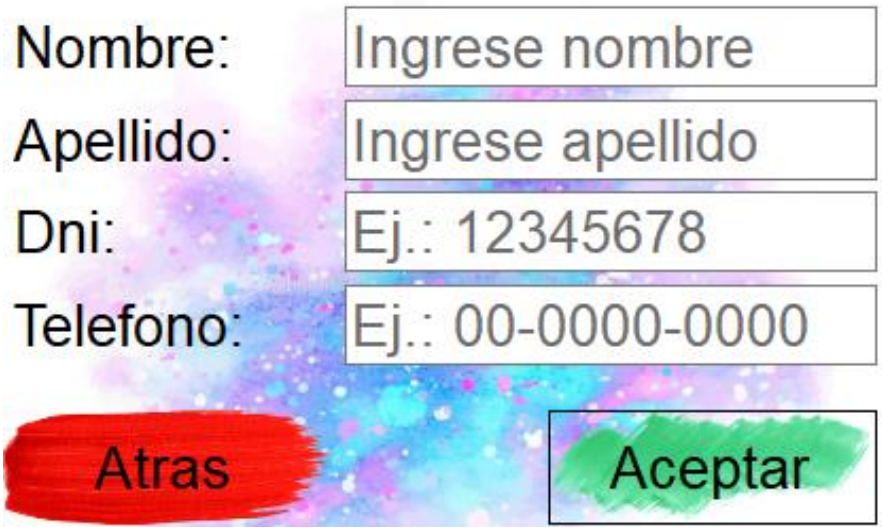
Una empresa de diseño gráfico que se encarga de proveer **Diseños** e **Impresiones** nos solicitó un sistema de atención al público, que permita atender tanto a **clientes registrados** como **clientes eventuales**.
La ventaja de aquellos clientes que ***decidan registrarse***, es que, *si realizan 7 o más compras en los últimos 30 días, entraran en la categoría de* ***Cliente Recurrente****, obteniendo asi un 5% de descuento en sus futuras compras.* Este descuento se mantendrá activo mientras el cliente siga siendo ***Cliente Recurrente***.



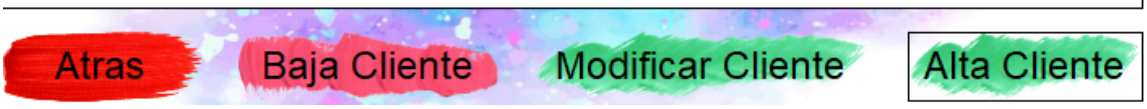
Además, los clientes registrados pueden gozar de otro *descuento del 5% si, en los últimos 90 días han invertido en el local un total de $25.000, entrando a la categoría de* ***Cliente VIP****.* Este descuento se mantendrá activo mientras el cliente siga siendo ***Cliente VIP.***
Ambos descuentos pueden acumularse para un total de 10% de descuento en cada compra, siempre y cuando el cliente mantenga ambas categorías.



El sistema permite dar de alta clientes, con solo nombre completo, Dni y teléfono. De todos estos datos, solo puede variar el número de teléfono, por lo que el sistema permite modificar dicho campo. Por otra parte, el sistema permite dar de baja clientes que, por alguna razón, deban borrarse.



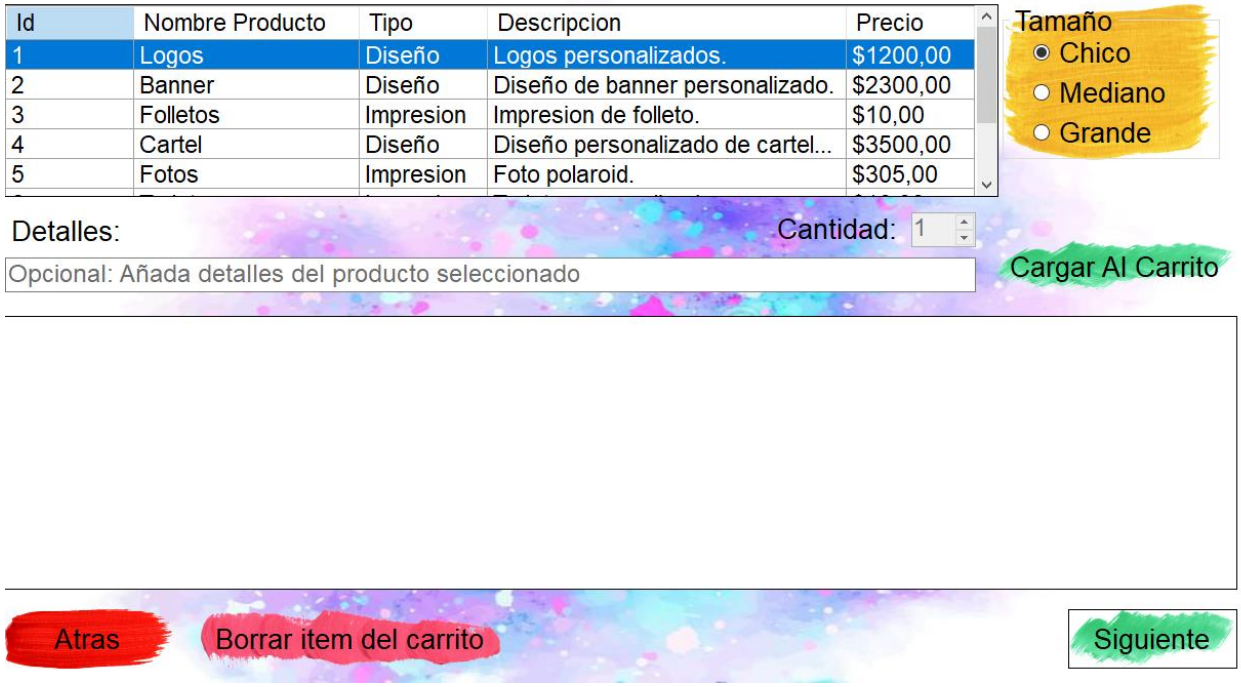
Dni	Nombre	Telefono	Inversion Anual
32165498	Lopez, Maria La...	15-2598-1234	$2000,00
36987654	Gimenez, Juan	15-4986-4879	$2750,00
11951357	Perez, Julian	11-9876-5432	$13800,00
30987654	Faisiste, Marcos	11-7896-5432	$12000,00
27987456	Quilmes, Floren...	11-9873-1597	$2000,00
11234987	Miguel, Luis	11-1235-6124	$68263,50
13245422	Vila, Daiana	11-1598-3169	$4248,00



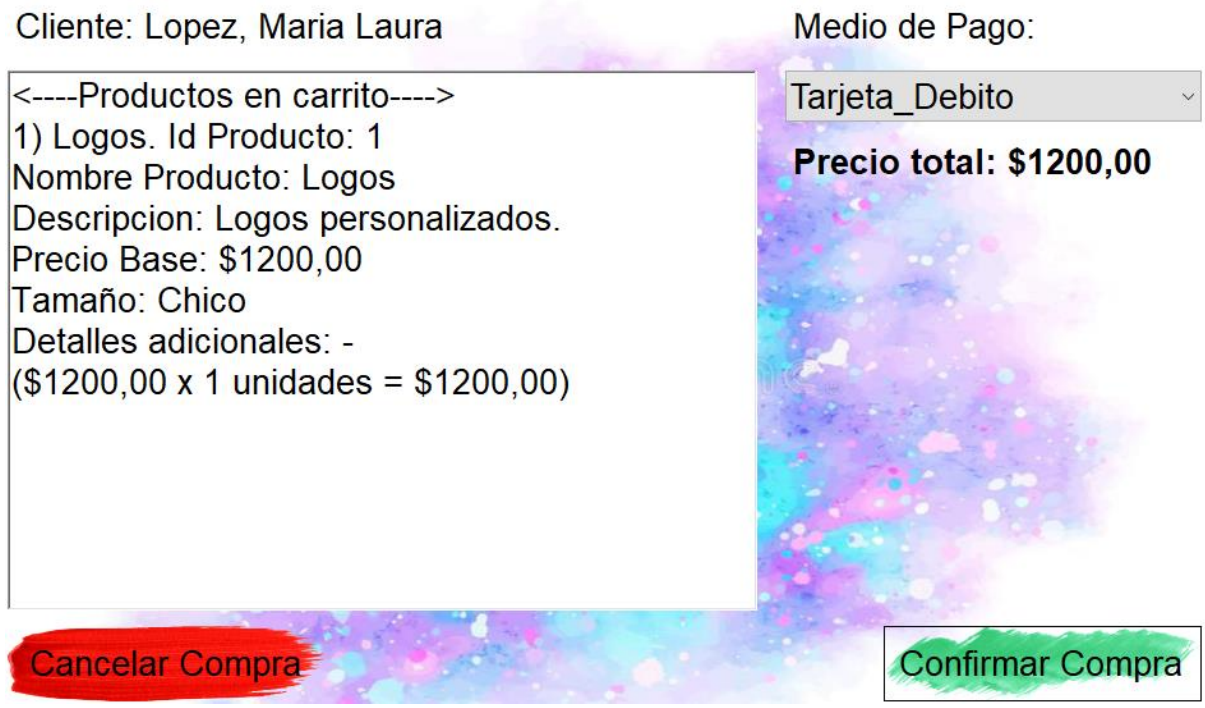
El sistema de atención permite al empleado, elegir cualquiera de los productos cargados en el sistema, y ***añadirlos al carrito de compras del cliente***.
En el caso de **Impresiones**, el sistema permite establecer la cantidad de unidades que el cliente llevara, y si es ***Sin color o Con color (el color duplica el precio del producto).***
Por otro lado, los **Diseños** se venden por unidad únicamente, ya que se está vendiendo el diseño de un logo, un banner o un cartel, por ejemplo. Los diseños vienen en 3 tamaños: ***Chico, mediano (duplica el costo) y grande (triplica el costo).*** Los diseños siempre vienen con color.

Adicionalmente, se permite agregar detalles (opcionales) acerca del producto que el cliente llevara, por ejemplo: “El cliente solicito que el logo se realice con colores claros”, “El cliente solicito que el banner este con colores vibrantes”.

En caso de ser necesario, el empleado puede remover ítems del carrito.

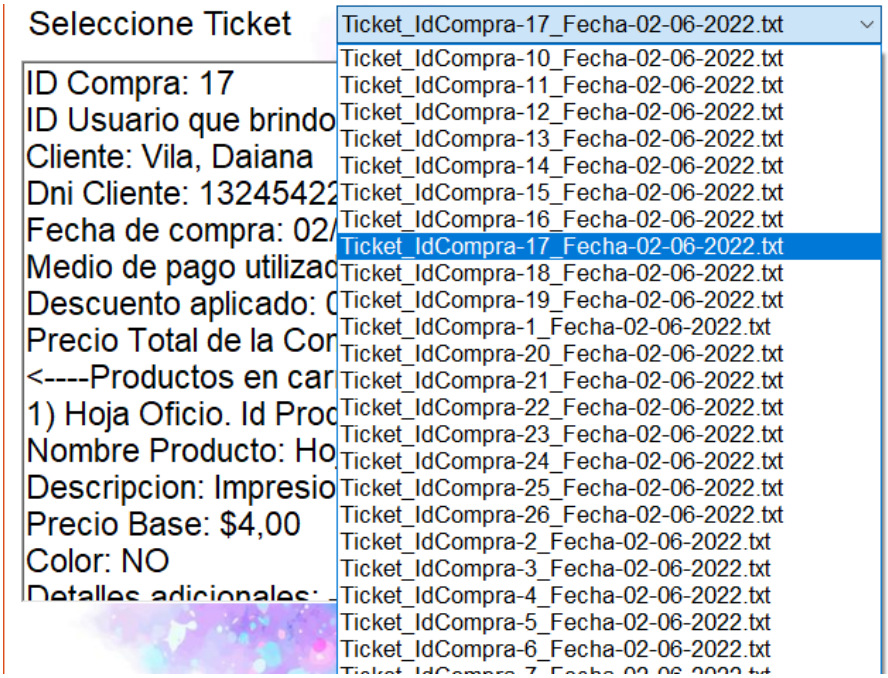


Una vez terminada la atención, se abrirá una nueva ventana de confirmación de compra, con toda la información de los productos en el carrito, así como los posibles descuentos que le corresponden al cliente registrado y el medio de pago a utilizar.



Al confirmar la compra, saldrá un mensaje informando que la compra fue exitosa, y ***se generara un archivo .txt con la información del ticket.***

Por último, el jefe de la empresa tiene la posibilidad de acceder a todos los tickets que se hayan generado:



Al sistema de atención solicitado, se **sumó** la posibilidad de **agregar, modificar y borrar** empleados, existiendo 3 categorías: Jefe, Administrador y Empleado. También se pueden **agregar, modificar y eliminar** productos correspondientes al área de Diseño e Impresiones.

Temas implementados

Clase 10 – excepciones:

Se utiliza en varias partes del código, principalmente se LANZAN en las propiedades que cargan datos de empleados, clientes y productos, y en los métodos de ***Leer*** y ***Guardar*** de Archivos, ***Xml*** y ***Json***. Son capturadas en la vista. Se crean 2 tipos de excepciones propias: ***ArchivoException*** y ***CargaDeDatosInvalidosException***.

Ejemplo:
Clase ***CargaDeDatosInvalidosException***:

```
namespace Entidades
{
    73 referencias
    public class CargaDeDatosInvalidosException : Exception
    {
        /// <summary>
        /// Representa una excepcion ocurrida durante la carga de datos.
        /// </summary>
        /// <param name="message">Mensaje de la excepcion.</param>
        20 referencias
        public CargaDeDatosInvalidosException(string message) : base(message)
        {
        }

        /// <summary>
        /// Representa una excepcion ocurrida durante la carga de datos.
        /// </summary>
        /// <param name="message">Mensaje de la excepcion.</param>
        /// <param name="innerException">Excepcion interna.</param>
        0 referencias
        public CargaDeDatosInvalidosException(string message, Exception innerException) : base(message, innerException)
        {
        }
    }
```

Implementado en Clase ***Empleado***, propiedad ***Salario***:

```
/// <summary>
/// Obtiene y setea (serializacion) el salario del empleado, previa validacion.
/// </summary>
/// <exception cref="CargaDeDatosInvalidosException">Datos invalidos</exception>
15 referencias | 6/6 pasando
public double Salario
{
    get
    {
        return this.salario;
    }
    set
    {
        if(value > 0)
        {
            this.salario = value;
        }
        else
        {
            throw new CargaDeDatosInvalidosException("El salario de un empleado no puede ser 0 o negativo.");
        }
    }
}
```

Se captura en el ***FrmModificarUsuario*** (Vista):

```
try
{
    this.empleado.Salario = salario;

    if(this.cBoxEsAdmin.Checked != this.empleado.EsAdministrador)
    {
        nuevoEmpleado = Administrador.CambiarPuestoDeEmpleado(this.empleado);

        if(nuevoEmpleado is not null && this.administrador.EliminarUnEmpleadoDelSistema(this.empleado))
        {
            this.administrador.CargarUnEmpleadoAlSistema(nuevoEmpleado);
        }
    }

    if(this.administrador.CargarModificacionDeEmpleado())
    {
        this.DialogResult = DialogResult.OK;
    }
}
catch(CargaDeDatosInvalidosException ex)
{
    MessageBox.Show(ex.Message, "Aviso: Carga de datos invalidos", MessageBoxButtons.OK, MessageBoxIcon.Hand);
}
```

Otro ejemplo de implementación lo encontramos en la clase ***Empleado***, en la propiedad ***NombreUsuario***:

```
    {
        return this.nombreUsuario;
    }
    set
    {
        if (!string.IsNullOrWhiteSpace(value))
        {
            if(value.Length >= 4 && value.Length <= 8)
            {
                if (value.EsCadenaAlfanumerica())
                {
                    this.nombreUsuario = value.DarFormatoDeNombre();
                }
                else
                {
                    throw new CargaDeDatosInvalidosException("El nombre de usuario solo puede contener letras y numeros.");
                }
            }
            else
            {
                throw new CargaDeDatosInvalidosException("El nombre de usuario debe tener entre 4 y 8 caracteres.");
            }
        }
        else
        {
            throw new NullReferenceException("El nombre de usuario no puede ser NULL o Vacio o Espacios en blanco");
        }
```

Métodos de extensión:

Hay una clase específica, ***ExtensionString***, donde hay métodos de extensión, por ejemplo:

```
/// <summary>
/// Evalua si una cadena respeta el formato telefonico.
/// </summary>
/// <param name="cadena">Cadena que se evaluara.</param>
/// <returns>True si la cadena respeta el formato telefonico (00-0000-0000), caso contrario, False.</returns>
3 referencias | 2/2 pasando
public static bool EsFormatoTelefonico(this string cadena)
{
    return ExtensionString.EsCadenaValida(cadena, "^[0-9]{2}-[0-9]{4}-[0-9]{4}$");
}
```

```
/// <summary>
/// Da el formato de nombre (primera letra de cada palabra en Mayuscula, resto de la palabra en minusculas).
/// Tambien se encarga de borrar espacios EXTRAS, dejando, de ser necesario, un unico espacio en blanco entre palabr
/// </summary>
/// <param name="cadena">Cadena que se evaluara.</param>
/// <returns>Una copia de la cadena con la primera letra de cada palabra convertida a mayusculas, el resto en minusc
5 referencias | 1/1 pasando
public static string DarFormatoDeNombre(this string cadena)
{
    string cadenaAuxiliar = string.Empty;

    if (!string.IsNullOrWhiteSpace(cadena))
    {
        cadenaAuxiliar = ExtensionString.ConvertirCaracterInicialDeCadaPalabraAMayuscula(cadena);
    }

    return cadenaAuxiliar;
}
```

Implementación en clase Cliente (entre otras clases que también lo implementan):

```
5 referencias
public string Telefono
{
    get
    {
        return this.telefono;
    }
    set
    {
        if (!string.IsNullOrWhiteSpace(value))
        {
            if(value.EsFormatoTelefonico())
            {
                this.telefono = value;
            }
            else
            {
                throw new CargaDeDatosInvalidosException("Te
            }
        }
```

La clase Persona, en su propiedad Nombre implementa 2 métodos de extensión: ***EsCadenaAlfabeticaConEspacios***() y ***DarFormatoDeNombre***()

```
public string Nombre
{
    get
    {
        return this.nombre;
    }
    set
    {
        if (!string.IsNullOrWhiteSpace(value))
        {
            if (value.EsCadenaAlfabeticaConEspacios())
            {
                this.nombre = value.DarFormatoDeNombre();
            }
            else
            {
                throw new CargaDeDatosInvalidosException("El
```

Clase 11 – Test Unitarios:

Hay un proyecto ***Tests***, con 3 clases en donde se testean algunos métodos de la clase ***Administrador***, ***ListaGenerica*** y ***ExtensionString***.

En el ejemplo se evalúa que, al cambiar el puesto de un empleado, efectivamente se retorne el nuevo tipo (clase) de empleado.

```
public class TestAdministrador
{
    [TestMethod]
    | 0 referencias
    public void CambiarPuestoDeEmpleado_CuandoElEmpleadoEsEmpleado_DeberiaRetornarUnAdministrador()
    {
        //ARRANGE
        Empleado empleadoA = new Empleado()
        {
            Apellido = "Z",
            Nombre = "G",
            Dni = 11111111,
            Id = 987,
            NombreUsuario = "abcd",
            Puesto = Empleado.Rol.Empleado,
            Salario = 1
        };

        //ACT
        Empleado actual = Administrador.CambiarPuestoDeEmpleado(empleadoA);

        //ASSERT
        Assert.IsInstanceOfType(actual, typeof(Administrador));
    }
```

Clase 12 - Generics:

Se utiliza ***Generics*** en ***Serializacion*** (XML y JSON) y en una ***ListaGenerica***. Esta lista genérica internamente está encapsulando una List<>, pero agrega métodos genéricos cuyo código, de otra forma, se estaría repitiendo en todas las clases.

Es implementada por la clase ***Cliente***, ***Producto***, ***Administrador***, ***Compra***.

Clase genérica: ***ListaGenerica***

```
19 referencias
public class ListaGenerica<T> where T : class, IObtenerIgualdad
{
    private List<T> elementos;

    /// <summary>
    /// Contructor de la clase. Instancia una lista Encapsulada de
    /// </summary>
    9 referencias | 4/4 pasando
    public ListaGenerica()
    {
        this.elementos = new List<T>();
    }

    /// <summary>
    /// Retorna la cantidad de elementos.
    /// </summary>
    9 referencias
    public int Count
    {
        get
ontraron problemas.
```

Implementación:

```
54 referencias
public class Compra : IObtenerIgualdad, IArchivo
{
    6 referencias
    public enum MedioDePago { Tarjeta_Debito, Tarjeta_Credito, Efectivo }

    private const string rutaRelativaArchivo = "Compras\\compras";
    private const string rutaRelativaCarpetaTickets = "Tickets\\";

    private static ListaGenerica<Compra> compras;
```

```
public class Cliente : Persona, IArchivo
{
    private const string rutaRelativaArchivo = "Clientes\\clientes";

    private static ListaGenerica<Cliente> clientes;
```

Esta lista genérica permite borrar un elemento, pero no por referencia, sino que por algún dato especifico del elemento. Ese dato especifico lo determinara CADA CLASE que implemente la lista genérica.

```
5 referencias
public bool EliminarElementoDelSistema(T elemento)
{
    if (elemento is not null)
    {
        for (int i = 0; i < this.Count; i++)
        {
            if (this[i].EsMismoElemento<T>(elemento))
            {
                this.elementos.RemoveAt(i);
                return true;
            }
        }
    }
    return false;
}
```

Lo mismo sucede para cargar elementos, la ***ListaGenerica*** chequea 1) El elemento no sea NULL y 2) Que el elemento no esté en la lista. En caso de cargar, retorna True, caso contrario False.
De esta forma, la ***ListaGenerica*** me garantiza que Ninguno de sus elementos es NULL y que NO hay repetidos.

```
15 referencias | 3/3 pasando
public bool CargarElementoAlSistema(T elemento)
{
    bool retorno = false;

    if(elemento is not null && !this.ExisteElementoEnSistema(elemento))
    {
        this.elementos.Add(elemento);
        retorno = true;
    }
    return retorno;
}
```

Esta ***ListaGenerica*** al tener encapsulada a la List<T>, solo me permite agregar y borrar a través de estos métodos mencionados.
La ***ListaGenerica*** también retorna elementos por Índice, evalúa si un elemento existe en la lista o los retorna por Identificador (el identificador que hace que 2 elementos sean iguales, se determina en cada clase que implemente la ***ListaGenerica***).
Ejemplos de implementación en clase ***Administrador***, de los métodos de ***CargarElementoAlSistema*** y ***EliminarElementoDelSistema***:

```
5 referencias
public virtual bool CargarUnEmpleadoAlSistema(Empleado empleado)
{
    if (!empleado.EsAdministrador)
    {
        if(Administrador.empleados.CargarElementoAlSistema(empleado))
        {
            return ((IArchivo)this).GuardarArchivo();
        }
    }
    return false;
}
```

```
/// <summary> Elimina un empleado del sistema. IMPLEMENTACION GENERICS.
5 referencias
public virtual bool EliminarUnEmpleadoDelSistema(Empleado empleado)
{
    if(!empleado.EsAdministrador)
    {
        if(Administrador.empleados.EliminarElementoDelSistema(empleado))
        {
            return ((IArchivo)this).GuardarArchivo();
        }
    }
    return false;
}
```

Clase 13 - Interfaces:

Hay 2 interfaces en el proyecto: ***IArchivo***, que me obliga a implementar los métodos ***GuardarArchivo*** y ***LeerArchivo***:

```
30 referencias
public interface IArchivo
{
    20 referencias
    bool GuardarArchivo();
    5 referencias
    void LeerArchivo();
}
```

Esta interfaz es implementada en las clases ***Cliente***, ***Administrador***, ***IdentificadorUnico***, ***Productos*** y ***Compras***. Ejemplo Clase ***Administrador***:

```
16 referencias
bool IArchivo.GuardarArchivo()...

/// <summary>
/// Lee la lista de empleados que esta respaldada
/// </summary>
1 referencia
void IArchivo.LeerArchivo()
{
    Administrador.LeerArchivoDeEmpleados();
}
```

Si bien se podría usar esta interface como restricción de la clase genérica ***SerializacionXml***<T> y ***SerializacionJSON***<T>, no me permitiría serializar List<T> porque las listas no implementan la interfaz ***IArchivo***.
Por otro lado, la interfaz ***IObtenerIgualdad***, tiene 2 métodos:

```
public interface IObtenerIgualdad
{
    6 referencias
    bool EsMismoElemento<T>(T elemento) where T : class, IObtenerIgualdad;
    6 referencias
    bool EsMismoIdentificador(int identificador);
}
```

Esta interfaz SI es utilizada como restricción de la clase ***ListaGenerica***:

```
19 referencias
public class ListaGenerica<T> where T : class, IObtenerIgualdad
{
```

Por lo que, cada clase que quiera utilizar la ***ListaGenerica*** debe implementar la interfaz ***IObtenerIgualdad***, y en sus métodos definirá en base a que se determina la igualdad de 2 elementos.
Ejemplo de implementación de la interfaz ***IObtenerIgualdad*** en clase ***Compra***:

```
/// <summary> Evalua si 2 elementos son iguales. IMPLEMENTA INTERFACES.
3 referencias
bool IObtenerIgualdad.EsMismoElemento<T>(T elemento)
{
    bool retorno = false;

    if(elemento is Compra compraAuxiliar)
    {
        retorno = this.IdCompra == compraAuxiliar.IdCompra;
    }
    return retorno;
}

/// <summary> Evalua si 2 elementos son iguales segun el identificador (dni). IM ...
3 referencias
bool IObtenerIgualdad.EsMismoIdentificador(int identificador)
{
    return this.IdCompra == identificador;
}
```

Clase 14 - Archivos:

¡Importante!

TODOS LOS ARCHIVOS SE GENERAN EN LA RUTA:

Zafferano.Gonzalo.2C.TP3\Vista\bin\Debug\net5.0-windows\RespaldoArchivos

Hay una clase ***Archivo***, en la cual se desarrollan los métodos ***Leer***, ***Guardar***, ***GuardarTxt*** y ***LeerArchivosDeUnDirectorio***.

```
2 referencias
public static string Leer(string rutaRelativaDelArchivo)...

/// <summary> Guarda un archivo .Txt en la ruta especificada.
1 referencia
public static bool GuardarTxt(string rutaRelativaArchivo, string textoAGuardarEnArchivo)...

/// <summary> Realiza una busqueda en el directorio BASE del sistema para obtene ...
1 referencia
public static Dictionary<string, string> LeerArchivosDeUnDirectorio(string rutaRelativaDelDir
```

Estos métodos son implementados en la clase ***Compra***, cada vez que se genera un Ticket o cuando se quiere leer un ticket, y en la clase ***SerializacionJSON*** (reutilizando los métodos de ***guardar*** y ***leer***).
Ejemplo de implementación en clase ***Compra***:

```
2 referencias
public bool GenerarTicketDeCompra()
{
    try
    {
        string nombreTicket = string.Format("Ticket_IdCompra-{0}_Fecha-{1}", this.IdCompra,

        return Archivo.GuardarTxt($"{Compra.rutaRelativaCarpetaTickets}{nombreTicket}", this
    }
    catch (ArchivoException)
    {
/// <exception cref="Exception">Error externo.</exception>
1 referencia
public static Dictionary<string,string> LeerTicketsDeCompra()
{
    return Archivo.LeerArchivosDeUnDirectorio(Compra.rutaRelativaCarpetaTickets, "Ticket_IdCor
}
```

Clase 15 - Serializacion:

Hay 2 clases de serializacion en el proyecto: ***SerializacionXml*** y ***SerializacionJSON***.
La clase ***SerializacionJSON*** reutiliza los métodos de ***Guardar*** y ***Leer*** de la clase Archivo.
SerializacionJSON es implementado en la clase ***IdentificadorUnico*** y ***Compra***, para guardar y leer las listas de compras que se hayan realizado. Ejemplo de clase ***Compra***:

```
bool IArchivo.GuardarArchivo()
{
    try
    {
        List<Compra> comprasASerializar = new List<Compra>();

        for (int i = 0; i < Compra.Count; i++)
        {
            comprasASerializar.Add(Compra.compras[i]);
        }

        return SerializadorJSON<List<Compra>>.GuardarJSON(Compra.rutaRelativaArchivo, comp
    }
    catch (ArchivoException)
    {
        throw;
    }
    catch(ArgumentNullException)
```

Mientras que la serializacion XML es utilizada por las clases ***Administrador***, ***Cliente***, ***Producto***.
Ejemplo de implementación en clase ***Cliente***:

```
bool IArchivo.GuardarArchivo()
{
    try
    {
        List<Cliente> clientesASerializar = new List<Cliente>();

        for (int i = 0; i < Cliente.Count; i++)
        {
            clientesASerializar.Add(Cliente.clientes[i]);
        }

        return SerializadorXml<List<Cliente>>.GuardarXml(Cliente.rutaRelativaArchivo, client
    }
```