BLOCKBUSTER IS BACK!!

Alumno: Ignacio Martín Smirlian

Materia: Laboratorio de Programación II

Instancia: Entrega Trabajo Práctico #4

Bienvenidos a mi TP4!! La continuación de mi Blockbuster 2.0

En este caso, para facilitar la revisión voy a poner en **NEGRITA y AZUL** todas las cosas nuevas del flujo del programa.

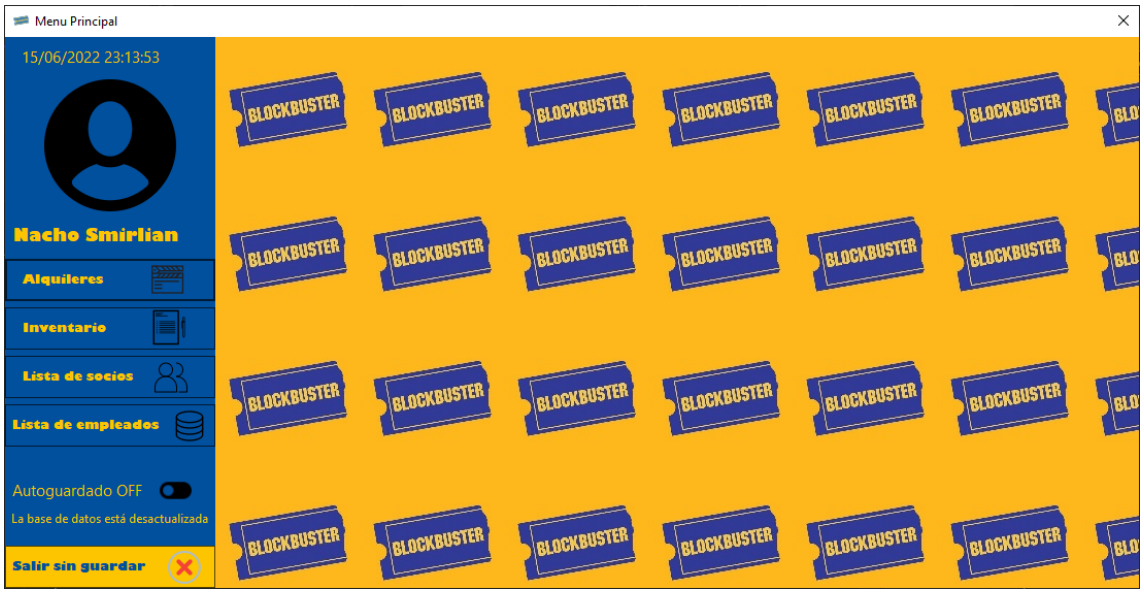
Toda la información necesaria para crear y cargar la base de datos en SQL se encuentra en el archivo script BlockbusterSQLScript

Flujo del programa:

El flujo del programa es muy simple ya que en este caso decidí incursionar en manejar todos los formularios (o al menos los principales) desde el menú principal.

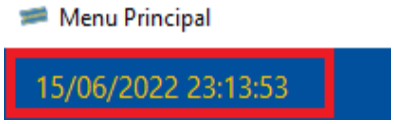
Al iniciar, nos vamos a encontrar con la clásica pantalla de Log In. Si presionamos el botón de autocompletar van a ingresar con “mi usuario” pero pueden probar cualquier de los otros usuarios. **La lectura de datos se hace desde SQL. Todos los empleados se encuentran en la tabla del script adjunto. En caso de haber un error al cargar los datos, el programa lo informará y pedirá al usuario reiniciar.**

Una vez logueados, podemos acceder al menú principal desde donde podremos acceder a los 4 “módulos” de este programa.



En este panel principal hay 3 nuevas funcionalidades que no se encontraban presentes en mi anterior entrega:

Reloj:

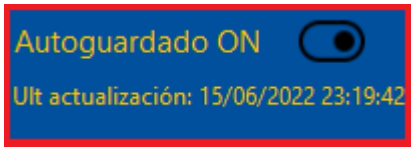


Este reloj, gracias a la programación multihilos, se irá actualizando de manera constante hasta que se apague la aplicación (cumple la misma función que un elemento Timer, pero está codeado enteramente por mí)

Botón lista de empleados:

En la entrega anterior, la información de los empleados no se mostraba, ya que no existía la posibilidad de modificar, agregar ni eliminar empleados.

Botón para activar autoguardado:



Este botón nos permite activar y desactivar la actualización automática de datos (tanto los archivos JSON y XML como la base de datos SQL)

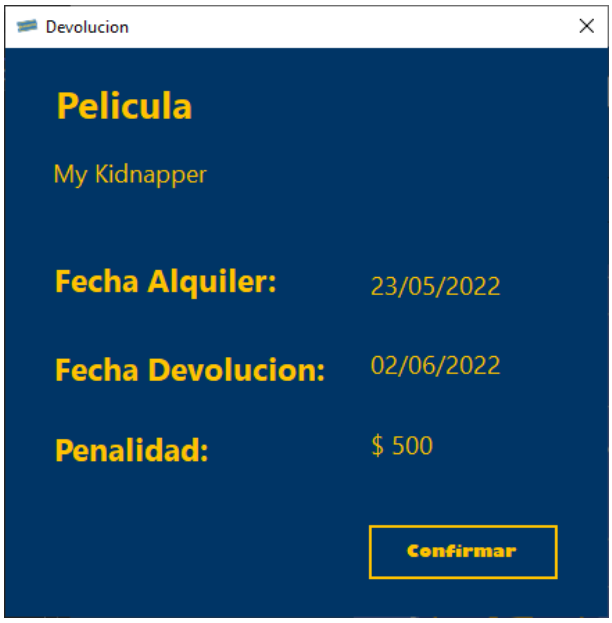
Accediendo al primer botón, entramos en la **sección alquileres**. Aquí podremos buscar a cada socio por su número de ID y traer toda su información (nombre, apellido, límite de películas, alquileres activos, etc).

Si el usuario presenta alquileres prontos a vencerse (pero aún no vencidos) la fila de ese alquiler se coloreará de **NARANJA** y si el alquiler ya está vencido, el color será **ROJO**. Si ninguno de estos escenarios se da, el color ser el clásico azul oscuro.

En este menú nosotros tenemos 3 acciones posibles:

>>> Apretando este botón nos traerá el usuario cuyo ID ingresemos en el textbox.

DEVOLUCION >>> Esto se da apretando la fila del alquiler que deseamos devolver.



En este pequeño formulario nos muestra la información de la película a devolver y su penalidad (existe un cálculo interno donde la penalidad es igual a $150 por día de demora, pero, además, si el usuario el premium, pagará solo un 30% de la penalidad).

Si presionamos Confirmar, la película se borra de la lista de alquileres de socio, y vuelve a la lista de películas de Blockbuster (aumentando en uno el stock de la película). Si, por el contrario, cerramos este form, la devolución se anula.

>>> Si presionamos este botón, abriremos el formulario para agregar un alquiler a este Socio

*Si el usuario ya llegó a su límite de películas (5 socios clásicos y 20 socios premium) este botón estará deshabilitado.

Aquí encontramos 2 datagrid. El primero contiene todas las películas disponibles (aquellas que no haya stock no aparecerán) con un buscador de titulo dinámico.

Por debajo, está el datagrid con toda la información del “CandyShop”. Aquí el socio puede elegir snacks o golosinas para comer. Estos se registrarán en la compra actual, pero no quedarán guardados en el historial del socio.

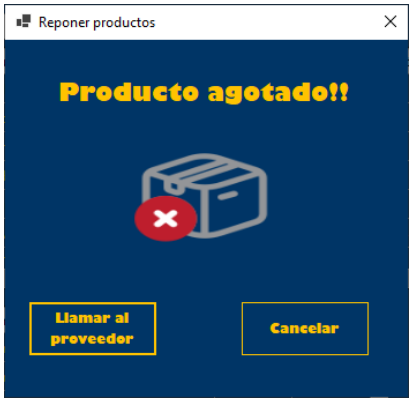
Para elegir tanto las películas como los productos, simplemente hay que hacer click en la fila que deseamos agregar y ya aparecerá en el richtextbox de la derecha. Si hubo algún error, y deseamos sacar algún producto, simplemente haciendo click en el botón editar convertiremos el richtextbox en un checkboxlist donde podemos seleccionar los items a borrar.

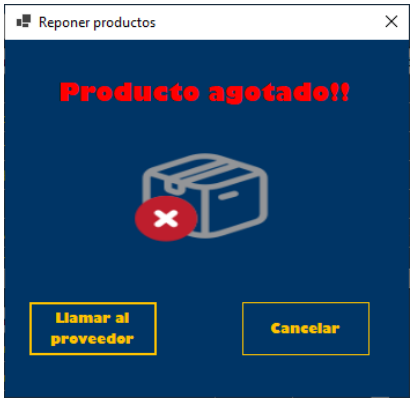
Cualquiera de estos movimientos que describí, impactan directamente en el inventario de los productos en tiempo real.

Para finalizar, simplemente Aceptar o cerramos el form, anulando toda la operación

La funcionalidades de este formulario no cambiaron, pero ahora se manejan a través de delegados o eventos. El método agregar alquiler de la clase socio recibe una pelicula y un delegado. Dentro del método agregar, si se llega al límite de peliculas, se invocará al delegado.

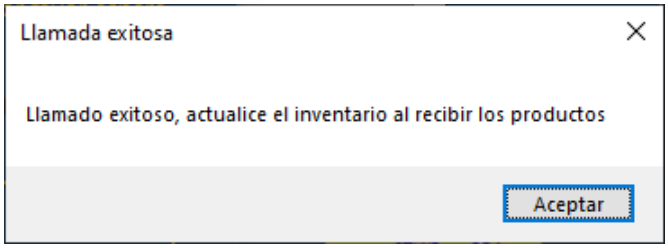
En el caso de los productos, al hacer click en una celda del dataGrid, asociamos el evento del producto InformarNoHayStock al método LlamarAlProveedor (verificando que ya no esté asignado). Cuando el producto llega a cantidad 0, se llama al metodo LlamarAlProveedor que abre un formulario nuevo:



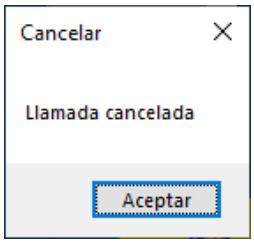


Este formulario, al abrirse nos muestra una pequeña animación con multihilos donde cambia el color del titulo entre amarrilo y rojo hasta que se cierre el form.

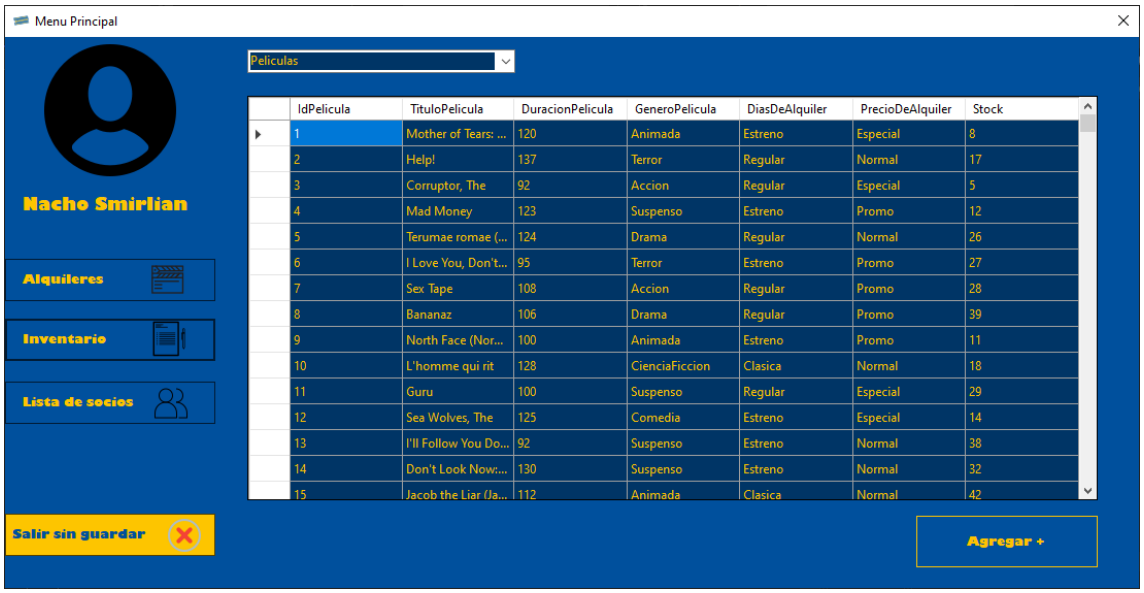
Si presionamos cancelar, el formulario se cerrará, pero si en cambio presionamos Llamar al proveedor simularemos una llamada para encargar la reposición del producto (a través de programación multi hilos la llamada toma 5 segundos), luego se nos informa con un MessageBox que la llamada fue exitosa.



Si por el contrario, cerramos o cancelamos el form antes que pase el tiempo necesario, se nos informará que se canceló la llamada:



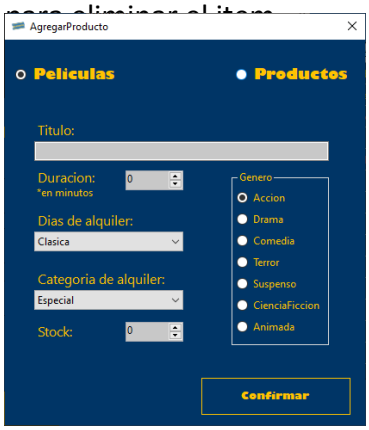
Segundo módulo, el inventario.



Aquí tenemos un formulario que consta casi únicamente de un datagrid con toda la información de los diferentes productos y películas. Para intercambiar entre películas y productos solo tenemos que seleccionar la opción correcta del combobox.

Este datagrid tiene un comportamiento similar al que encontramos en la pantalla anterior, pero se le agregó un pequeño efecto que resalta toda la fila en la que estamos parados, sin necesidad de seleccionarla (efecto hover).

En esta pantalla tanto si apretamos el boton AGREGAR + como si hacemos click en alguna fila del datagrid, el formulario que va a aparecer es exactamente el mismo. La única diferencia es que al presionar AGREGAR +, el formulario está vacío y nos permite elegir cargar un producto o una película. En cambio, al hacer click en una celda, el formulario aparecerá con toda la información ya cargada del item para que podamos editarlo, con el radio botón del otro elemento deshabilitado, y un botón de borrar, para eliminar el item.



Luego llegamos al **módulo de socios**. Aquí podremos realizar búsquedas rápidas de socios por ID, Nombre, Apellido o incluso e-mail.

El funcionamiento del datagrid es idéntico al formulario anterior. Al hacer click en una celda, nos abrirá un formulario con todos los datos de ese socio para que podamos modificarlo y actualizarlo. Si presionamos AGREGAR + abriremos un formulario en blanco.

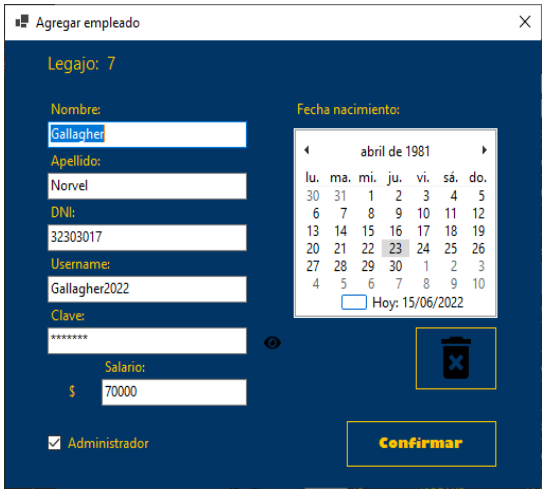
Importante destacar que existen validaciones para cada campo (Nombre, Apellido, Email, Teléfono y Tarjeta) de acuerdo a los criterios de formato más comunes en Argentina (muchos socios ya cargados no respetan ese formato, principalmente teléfono, porque fueron traídos de Mookaro, por lo que al guardar una modificación, el programa va a pedirnos que ingresemos un formato válido)

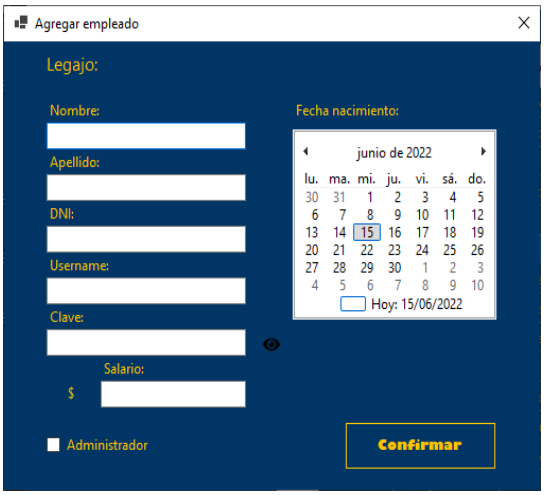
Información: Sólo se pide tarjeta de garantía para los socios clásicos ya que, en mi lógica, los usuarios premium ya tienen una suscripción mensual, por lo que no necesitan garantía.

Finalmente, llegamos al último módulo, el de lista de empleados.

Legajo	Nombre	Apellido	DNI	Usuario	Salario	Ingreso	Fecha de nacimiento
1	Britney	Stanlake	45345123	Brittne2022	$80000	01/09/2019	21/05/1965
2	Paxton	Dunnico	48151205	Paxton2022	$100000	13/02/2022	02/04/1988
4	Viva	Grint	44543743	Viva2022	$110000	04/08/2020	10/02/1988
5	Ambros	Hackford	45763019	Ambros2022	$70000	15/10/2021	29/08/1998
6	Lexine	Jermin	40913598	Lexine2022	$120000	27/02/2018	27/09/1962
7	Gallagher	Norvel	32303017	Gallagher2022	$70000	21/06/2019	23/04/1981
8	Carleton	Bradberry	44833629	Carleton2022	$200000	28/01/2020	24/09/1971
9	Tuckie	Lilleman	49774791	Tuckie2022	$80000	23/01/2019	14/11/1962
10	Charlotte	Goldney	41130397	Charlotte2022	$120000	16/05/2021	15/03/1969
12	Tammy	Sims	40175070	Tammy2022	$90000	22/07/2018	19/12/1961
13	Vida	Korfmann	41094759	Vida2022	$200000	31/05/2020	01/07/1997
14	Freda	De Zamudio	48433090	Freda2022	$110000	17/01/2022	05/06/1998
15	Jobyna	Plastow	34928650	Jobyna2022	$80000	15/12/2018	12/10/1972
16	Smitty	Lias	34951517	Smitty2022	$80000	01/04/2022	09/03/1995
17	Dorette	Heisham	34552263	Dorette2022	$120000	09/04/2018	08/12/1979

Al igual que los anteriores, cuenta con una datagrid en donde podemos cliquear para editar un empleado, o apretar agregar para cargar uno nuevo.





Extra: al apretar el icono del ojo, la contranseña se ocultará o se mostrará, me pareció un lindo detalle.

Fuera de eso, este módulo se comporta igual que los anteriores. La información actualizada se guarda de manera local en la clase estática Blockbuster, y luego se actualiza con la base de datos SQL (cada 5 segundos si está activado el auto-guardado o al cerrar la app)

Llegando al final de la revisión del flujo, es importante destacar 2 momentos que suceden “por detrás” de la aplicación. En primer lugar, la CARGA DE DATOS.

Esta se realiza en 2 partes. Al abrir el formulario de Login se cargan al sistema toda la información de los usuarios. Luego, al abrir el menú principal, se leen toda la información de las películas, productos, y socios.

En segundo lugar, el GUARDADO DE LOS DATOS. Esto se hace de manera automática al cerrar el programa (informándonos si se pudo guardar todo con éxito o no) **o a través del método de autoguardado si está activado**. En este punto se actualizan todas las bases de datos y se guarda la facturación diaria **(la facturación diaria solo se actualiza al cerrar la app)** que se va guardando a lo largo de todo el programa a través de cada transacción de compra o devolución. (el nombre del archivo donde se guarda la facturación diaria lleva la fecha del día, por lo tanto, si ya existe un archivo con la fecha de hoy, se agregará la información a ese archivo. De lo contrario se creará un nuevo archivo). Sin embargo, hay casos en que el usuario puede querer salir sin guardar sus cambios, y es por eso que incluí un botón de Salir sin guardar en el panel principal **(obviamente que si el autoguardado está activado, este botón no tiene mucha utilidad)**

Por último, me parece importante destacar que en esta instancia yo elegí hacer la clase Blockbuster estática ya que en mi parcial yo había utilizado una clase restaurante instanciable y me pareció interesante probar el funcionamiento con una clase estática.

Disclaimer: La forma en que yo asigno los ID's/legajos al crear una nueva entrada es buscando el mayor número de cada lista a través de un método (clase estática blockbuster). Eso me garantiza que nunca se van a repetir los ID's ya que cada vez que cargo uno, busco el valor máximo y mi nuevo ID va a ser el número siguiente. Pero también trae un problema, si yo cargo un usuario y me da ID 1000, y luego lo borro, si cargo otra persona, va a asignarle nuevamente el ID 1000, ya que va a buscar la lista y va a encontrar que el número máximo es 999. Esto podría solucionarse guardando el ID máximo en un archivo, para leerlo y sobreescribirlo cada vez que sea crea un nuevo registro, pero por cuestiones de tiempo, y por miedo a romper todo lo que ya hice, lo dejé como está.

Temas:

- ### Excepciones:
  A lo largo de todo el programa fui evaluando los escenarios donde podía llegar a darse una excepción y se agregar los try/catch necesarios. A su vez, se han creado 4 excepciones específicas para la validación de Email, Nombre o Apellido, Tarjeta de crédito y Teléfono.
  - Archivo >> Biblioteca de clases >>> Excepciones
  - Aplicado en >>> AgregarSocio
- ### Pruebas unitarias:
  En el proyecto "TestUnitarios" podemos encontrar diversos test unitarios para los métodos de validación previamente mencionados y los métodos de búsqueda.
  - Archivo >>> TestUnitarios

- Aplicado en >>> BuscarIndiceProducto,BuscarPeliculasPorId,BuscarProductoPorId,BuscarUsuarioPorLegajo,CheckLognIn,ValidacionMail,ValidacionTarjeta,ValidacionTelefono

- # Tipos genéricos:

  Puede verse en 2 partes en el TP. En primer lugar, en todos los métodos de serialización de la ClaseSerializadora que leen o escriben archivos de tipo T. También puede verse que la clase Alquiler es genérica. Esto es así porque mi idea era expandir el sistema incluyendo videojuegos y series, por lo que los alquileres puedan no ser necesariamente una película. Lamentablemente, por falta de tiempo no logré implementarlo.
  - Archivo >>> TestUnitarios >>> Excepciones
  - Aplicado en >>> AgregarSocio

- # Interfaces:

  En el TP se usa la interfaz IModificarse, que incluye la firma de 2 métodos que son utilizados tanto por el formulario AgregarProducto, AgregarSocio y CargarDatosForm.
  - Archivo >>> IModificarse
  - Aplicado en >>> AgregarProducto, AgregarSocio y CargarDatosForm

- # Archivos y serialización:

  Esto se puede ver en toda la información que el programa maneja, ya que todos los datos de usuarios, películas, socios, etc, son cargados a través de archivos JSON o XML, y luego, todas las modificaciones son serializadas nuevamente y guardadas en los formatos indicados.
  - Archivo >>> ClaseSerializadora
  - Aplicado en >>>Método autoguardado**,** MenuPrincipal load event, MenuPrincipal formClosing event.

- # SQL y Base de datos:

  Toda la lógica de la conexión y la actualización con la base de datos a través de SQL se hace en la clase MetodosSQL, que cuenta con un constructor estático con todos los datos necesarios para establecer los datos de la conexión, y luego los métodos GuardarUsuario, ModificarUsuario,GuardarListaUsuarios,LeerListasUsuarios.
  - Archivo >>> MetodosSQL
  - Aplicado en >>> Método autoguardado, Login load event, MenuPrincipal formClosing event.

- # Delegados y expresiones lambda:

  Estos temas pueden encontrarse en varias partes dentro del programa
  <u>Archivo Producto:</u>

```
namespace BibliotecaDeClases
{
    public delegate void DelegadoStock();
    35 referencias
    public class Producto
    {
        public event DelegadoStock InformarNoHayStock; //Se crea este evento para que ser llamado al acabarse el stock
```

  Aquí declaramos un delegado y luego lo asociamos con el evento InformarNoHayStock

  <u>Archivo Socio:</u>

```
1 referencia
public void AgregarAlquiler(List<Alquiler<Pelicula>> peliculasElegidas, Pelicula pelicula, Action delegadoInformacion)
{                                   //El delegado se le pasa para ser invocado cuando el socio llegue al limite de peliculas
    if((peliculasElegidas.Count + this.listaDeAlquileres.Count) < this.LimitePeliculas)
    {
        peliculasElegidas.Add(new Alquiler<Pelicula>(pelicula));
        Blockbuster.BuscarPelicula(pelicula.IdPelicula).Stock--;
    }
    else
    {
        delegadoInformacion.Invoke();
    }
}
```

Acá estamos declarando un método que recibe un delegado como parametro, que será invocado al cumplirse la condición establecida.

El delegado que se le pasa, podemos encontrarlo en el form AgregarAlquiler, el método es MostrarLimitePeliculas

```
1 referencia
public void MostrarLimitePeliculas()
{
    MessageBox.Show($"Error, este usuario ya llegó a su límite de alquileres", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

```
private void dGridPeliculas_CellClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        if (e.RowIndex >= 0)
        {
            DataGridViewTextBoxCell cell = (DataGridViewTextBoxCell)dGridPeliculas.Rows[e.RowIndex].Cells[0];
            socioAtendido.AgregarAlquiler(listaAlquilerAux, Blockbuster.BuscarPelicula((int)cell.Value), MostrarLimitePeliculas);
            dGridPeliculas.Rows.Clear();
            CargarPeliculas();
            ActualizarCuenta();
        }
    }
    catch (Exception ex)
```

Y acá pasamos dicho método.

En la siguiente imagen, podemos ver instrucciones lambda en el form MenuPrincipal

```
129         private void MostrarHora()
130         {
131             Task task = Task.Run(() =>
132             {
133                 do
134                 {
135                     ActualizarReloj();
136                     Thread.Sleep(1000);
137                 } while (true);
138             });
139         }
```

```
1 referencia
154         private void AutoGuardado(CancellationToken cancelacion)
155         {
156             Task task = Task.Run(() =>
157             {
158                 while(!cancelacion.IsCancellationRequested)
159                 {
160                     ActualizarBaseDeDatos();
161                     MostrarInfoActualizacion();
162                     Thread.Sleep(5000);
163                 }
164             });
165         }
```

También acá podemos encontrar 2 delegados del tipo Action en los métodos MostrarInfoActualizacion y ActualizarReloj

```
141         private void ActualizarReloj()
142         {
143             if(lblReloj.InvokeRequired)
144             {
145                 Action delegadoActualizarHora = ActualizarReloj;
146                 lblReloj.Invoke(delegadoActualizarHora);
147             }
148             else
149             {
150                 lblReloj.Text = DateTime.Now.ToString("dd/MM/yyy HH:mm:ss");
151             }
```

```
2 referencias
176         private void MostrarInfoActualizacion()
177         {
178             if (lblActualizacionInfo.InvokeRequired)
179             {
180                 Action delegadoMostrarInfo = MostrarInfoActualizacion;
181                 lblActualizacionInfo.Invoke(delegadoMostrarInfo);
182             }
183             else
184             {
185                 lblActualizacionInfo.Text = "Ult actualización: " + DateTime.Now.ToString();
186             }
187         }
```

Hay otros ejemplos también en el form ReponerProducto similares a los anteriores

- Multi-hilo y concurrencia:

Estos temas pueden verse aplicados principalmente en 4 ocasiones:

- Reloj (form MenuPrincipal):

```
1 referencia
private void MostrarHora()
{
    Task task = Task.Run(() =>
    {
        do
        {
            ActualizarReloj();
            Thread.Sleep(1000);
        } while (true);
    });
}

2 referencias
private void ActualizarReloj()
{
    if(lblReloj.InvokeRequired)
    {
        Action delegadoActualizarHora = ActualizarReloj;
        lblReloj.Invoke(delegadoActualizarHora);
    }
    else
    {
        lblReloj.Text = DateTime.Now.ToString("dd/MM/yyy HH:mm:ss");
    }
}
```

- Autoguardado (form MenuPrincipal):

```
private void AutoGuardado(CancellationToken cancelacion)
{
    Task task = Task.Run(() =>
    {
        while(!cancelacion.IsCancellationRequested)
        {
            ActualizarBaseDeDatos();
            MostrarInfoActualizacion();
            Thread.Sleep(5000);
        }
    });
}
2 referencias
private void ActualizarBaseDeDatos()
{
    ClaseSerializadora<List<Socio>>.EscribirXml(Blockbuster.ListaDeSocios, "baseDatosSocios");
    ClaseSerializadora<List<Producto>>.EscribirJson(Blockbuster.ListaDeProductos, "baseDatosProductos");
    ClaseSerializadora<List<Pelicula>>.EscribirJson(Blockbuster.ListaDePeliculas, "baseDatosPeliculas");
    MetodosSQL.GuardarListaUsuarios(Blockbuster.ListaDeEmpleados);

}
2 referencias
private void MostrarInfoActualizacion()
{
    if (lblActualizacionInfo.InvokeRequired)
    {
        Action delegadoMostrarInfo = MostrarInfoActualizacion;
        lblActualizacionInfo.Invoke(delegadoMostrarInfo);
    }
    else
    {
        lblActualizacionInfo.Text = "Ult actualización: " + DateTime.Now.ToString();
    }
```

- Animación “Producto Agotado!!” (form ReponerProducto):

```
1 referencia
private void AnimarMensaje()
{
    Task task = Task.Run(() =>
    {
        do
        {
            CambiarColor(Color.Red);
            Thread.Sleep(500);
            CambiarColor(ColorTranslator.FromHtml("#ffc300"));
            Thread.Sleep(500);
        } while (true);
    });
}

3 referencias
private void CambiarColor(Color color)
{
    if (lblMensaje.InvokeRequired)
    {
        Action<Color> delegadoCambioColor = color => CambiarColor(color);
        lblMensaje.Invoke(delegadoCambioColor,color);
    }
    else
    {
        lblMensaje.ForeColor = color;
    }
}
```

- Simulación “Llamada al proveedor” (form ReponerProducto):

```
1 referencia
private void btnLlamarProveedor_Click(object sender, EventArgs e)
{
    picNoStock.Visible = false;
    btnLlamarProveedor.Enabled = false;
    Task esperarLlamada = Task.Run(() =>
    {
        int vueltas = 0;
        MostrarPanel();
        while (!cts.IsCancellationRequested && vueltas < 5)
        {
            Thread.Sleep(1000);
            vueltas++;
        }
        if (!cts.IsCancellationRequested)
        {
            MessageBox.Show("Llamado exitoso, actualice el inventario al recibir los productos",
                "Llamada exitosa");
            this.DialogResult = DialogResult.OK;
        }

    }, cts.Token);
}
2 referencias
private void MostrarPanel()
{
    if (panLlamada.InvokeRequired)
    {
        Action mostrarPanelDelegado = MostrarPanel;
        panLlamada.Invoke(mostrarPanelDelegado);
    }
    else
    {
        panLlamada.Visible = true;
```

- # Eventos:

  Este tema podemos encontrarlo solo en el formular AgregarAlquiler y clase Producto

```
public delegate void DelegadoStock();
33 referencias
public class Producto
{
    public event DelegadoStock InformarNoHayStock; //Se crea este evento para que ser llamado al acabarse el stock
    private string nombreProducto;
```

Declaración del delegado y asignación del evento del tipo delegado creado.

```
1 referencia
public void ActualizarStock()
{
    Blockbuster.BuscarProducto(this.IdProducto).StockProducto--;

    if(Blockbuster.BuscarProducto(this.IdProducto).StockProducto < 1)
    {
        if(InformarNoHayStock is not null) //Verificamos que alguien este suscripto al evento
        {
            InformarNoHayStock();
        }
    }
}
```

Acá se llama el evento, quién está escuchando es el método LlamarAlProveedor del form AgregarAlquiler

```
1 referencia
private void dGridProducto_CellClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        if (e.RowIndex >= 0)
        {
            DataGridViewTextBoxCell cell = (DataGridViewTextBoxCell)dGridProducto.Rows[e.RowIndex].Cells[0];
            Producto productoAux = Blockbuster.BuscarProducto((int)cell.Value);
            if (productoAux.ValidarQueElEventoNoEsteAsignado()) //Aqui validamos que el evento no haya sido ya asignado
            {
                productoAux.InformarNoHayStock += LlamarAlProveedor; //Sino, lo asignamos
            }
            listaProductosAux.Add(productoAux);
```

```
public void LlamarAlProveedor()
{
    ReponerProducto frmReposicion = new ReponerProducto();
    frmReposicion.ShowDialog();
}
```

- # Métodos de extensión:

  Este tema puede verse aplicado en el tipo de dato INT y el tipo de dato STRING

  - Clase ExtensiónIntDni (validar valores del DNI):

```
0 referencias
public static class ExtensionIntDni
{
    1 referencia
    public static bool ValidarDni(this int dni)
    {
        return dni > 15000000 && dni < 60000000;
    }
}
```

  - Clase ExtensiónStringUsuarioClave (validar longitud de clave y usuario)

```
0 referencias
public static class ExtensionStringUsuarioClave
{
    1 referencia
    public static bool ValidarUsuario(this string usuario)
    {
        return usuario.Length > 5 && usuario.Length < 17;
    }

    1 referencia
    public static bool ValidarClave(this string clave)
    {
        return clave.Length > 3 && clave.Length < 17;
    }
}
```

Y se implemente en el método Validaciones() del form AgregarEmpleado

```
1 referencia
private void Validaciones()
{
    if (!txtNombreEmpleado.Text.All(char.IsLetter))
    {
        throw new NombreOApellidoInvalido("Favor revisar el campo nombre");
    }

    if (!txtApellidoEmpleado.Text.All(char.IsLetter))
    {
        throw new NombreOApellidoInvalido("Favor revisar el campo apellido");
    }
    ;
    int.TryParse(txtDni.Text, out int dniAux);
    if (!dniAux.ValidarDni())
    {
        throw new DniInvalido("Favor verificar el DNI ingresado (sin puntos ni espacios)");
    }

    if ((!txtUsername.Text.ValidarUsuario()) || (!txtClave.Text.ValidarClave()))
    {
        throw new DatosIncompletos("Favor verificar el nombre de usuario y/o clave ingresados");
    }
```