Ejercicios

Archivos y serialización

Ejercicios de archivos y serialización

Antes de empezar

- Deberá asignarle a la propiedad 'Title' de la clase Console, el número de ejercicio.

  Por ejemplo:
  Console.Title = “Ejercicio Nro ##” donde ## será el número del ejercicio.

- Del mismo modo se deberán nombrar los proyectos de consola.

  Por ejemplo:
  Ejercicio_##.

- Para visualizar los valores decimales de los ejercicios, deberá dar el siguiente formato de salida al método Write/WriteLine: “#,###.00”.

I01. Siempre quise tener un notepad

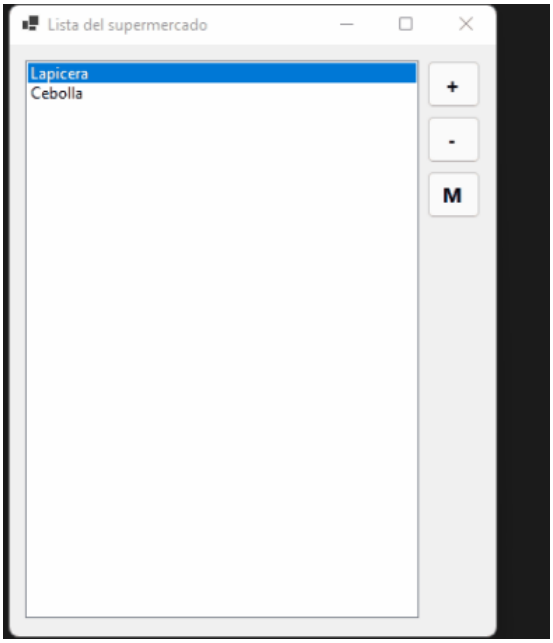
Crear un proyecto de Windows Forms App capaz de abrir, editar y guardar archivos de texto, tal como se puede hacer en un simple editor de texto como puede ser el notepad de Windows.

1. Agregar una barra de menú superior (`MenuStrip`) con las siguientes opciones del menú "Archivo":
   - Archivo -> Abrir
   - Archivo -> Guardar
   - Archivo -> Guardar como...
2. Usando la propiedad `ShortcutKeys`, asociar los siguientes shortcuts (atajos) a las opciones del menú:
   - Abrir: Ctrl + A
   - Guardar: Ctrl + S
   - Guardar como...: Ctrl + Shift + S

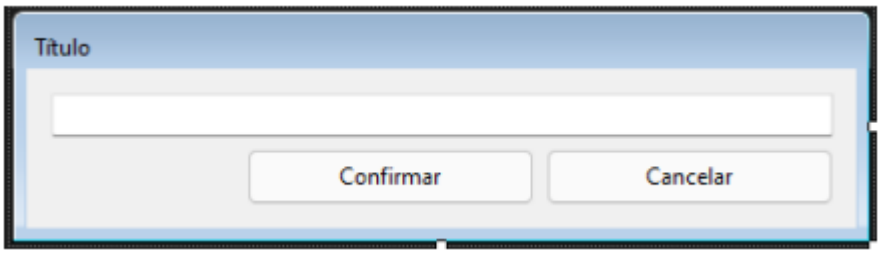
3. Agregar una barra de estado en la parte inferior (`StatusStrip`) que contenga un `ToolStripStatusLabel`.
4. Agregar un `RichTextBox` que deberá estar acoplado al centro del formulario (propiedad Dock).
5. En el `StatusStrip`, informar la cantidad total de caracteres del texto en el `RichTextBox`.
6. Utilizar la propiedad Dock para enlazar el `MenuStrip` al borde superior de la ventana y el `StatusStrip` al borde inferior.
7. Al pulsar el menú "Abrir" se deberá abrir una ventana para seleccionar el archivo a abrir.
   - Utilizar la clase `OpenFileDialog`.
   - Mostrar el contenido del archivo en el `RichTextBox`.
8. Al pulsar el menú "Guardar como..." se deberá abrir una ventana para seleccionar la ubicación donde se guardará el archivo y cómo se llamará.
   - Utilizar la clase `SaveFileDialog`.
   - La propiedad `Filter` de SaveFileDialog deberá tener el valor "*Archivo de texto|.txt*"*.
   - Tomar el contenido a guardar del texto en el `RichTextBox`.
9. Al hacer click sobre "*Guardar*", se deberá guardar en el último archivo guardado o abierto desde la interfaz. En el caso que no haya ningún "último archivo", se comportará igual que la opción "*Guardar como...*". Reutilizar código.
10. En caso de error en cualquiera de las operaciones se deberá mostrar una ventana de error conteniendo el mensaje de la excepción y su stack trace.

I02. La lista del super

El objetivo será crear una aplicación que maneje una lista de supermercado. Se podrá agregar, eliminar y modificar los elementos de la lista.



Crear un proyecto de Windows Forms con un formulario llamado `FrmAltaModificacion` que se vea como el siguiente:



- Deberá iniciar centrado en la pantalla (propiedad `StartPosition`).
- No deberá tener menú de control (propiedad `ControlBox`).
- No deberá poder cambiar de tamaño (propiedad `FormBorderStyle`).
- No tendrá ícono (propiedad `ShowIcon`).
- Tendrá un `TextBox` llamado `txtObjeto`, un `Button` llamado `btnConfirmar` y un `Button` llamado `btnCancelar`.
- El `txtObjeto` no deberá poder contener un texto de más de 50 caracteres (propiedad `MaxLength`).
- Tendrá una propiedad Objeto pública y de sólo lectura que retornará el contenido del `txtObjeto`.
- Al instanciarse deberá configurarse de acuerdo a la siguiente información que recibirá como argumentos en su constructor:
  - El título del formulario.
  - El texto que contendrá el `txtObjeto`.
  - El texto del `btnConfirmar`.
- Si se presiona el `btnConfirmar` o la tecla enter (posicionados dentro del `TextBox`):
  - Validar que el `txtObjeto` no se encuentre vacío.
  - Si se encuentra vacío mostrar una advertencia y no avanzar.
  - Si es válido, cargar la propiedad `DialogResult` con el valor `DialogResult.OK` y cerrar el formulario.

TIP

El evento `KeyPress` se acciona cuando el usuario está haciendo foco en el control que es dueño del evento y presiona una tecla.

Uno de los parámetros de entrada del manejador de dicho evento es de tipo `KeyPressEventArgs` y contiene una propiedad `KeyChar` que tendrá como valor el carácter correspondiente a la tecla presionada por el usuario.

Por ejemplo, para saber si el usuario presionó la tecla enter podemos hacer lo siguiente:

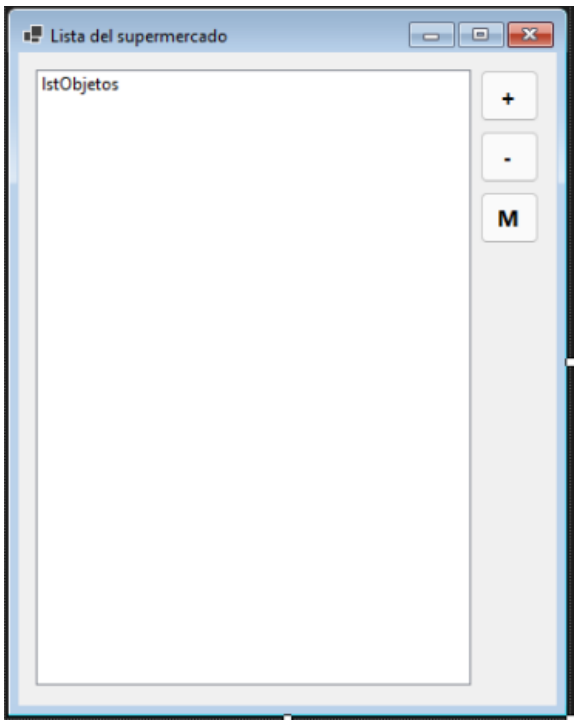
```
if (e.KeyChar == (char)13) // 13 es el código ASCII que representa
a ENTER.
{
    // ...
}
```

Si queremos saber si el usuario presionó s o S (shift + s) podemos hacer lo siguiente:

```
if (e.KeyChar == 's' || e.KeyChar == 'S')
{
    // ...
}
```

- Si se presiona el `btnCancelar` o la tecla escape (posicionados dentro del `TextBox`), se deberá cargar la propiedad `DialogResult` con el valor `DialogResult.Cancel` y cerrar el formulario.

Agregar otro formulario llamado `FrmListaSuper` que se vea como el siguiente:



- Deberá ser el formulario que se muestra al iniciar la aplicación.
- Tendrá un atributo `listaSupermercado` de tipo `List<string>` que contendrá los elementos de la lista del supermercado.
- Deberá iniciar centrado en la pantalla (propiedad `StartPosition`).
- Tendrá un `ListBox` llamado `lstObjetos`.
    - Mostrará la lista de objetos pendientes de comprar en el supermercado.
    - Estará anclado a todos los bordes del formulario (propiedad Anchor).
- Tendrá un `Button` llamado `btnAgregar` cuyo texto será "+" y al pasar por encima deberá mostrar un tooltip con el texto "Agregar objeto".
    - Deberá estar anclado arriba y a la derecha (propiedad Anchor).
    - Al accionarlo instanciará un `FrmAltaModificacion` y lo mostrará de forma modal.
        - El título será "Agregar objeto".
        - El contenido del `txtObjeto` será un texto vacío.
        - El texto del `btnConfirmar` será "Agregar".

    - Si la propiedad `DialogResult` de la instancia de `FrmAltaModificacion` vale `DialogResult.OK`, agregar el elemento retornado por la propiedad Objeto a la lista del supermercado. Caso contrario, no hacer nada.
- Tendrá un `Button` llamado `btnEliminar` cuyo texto será "-" y al pasar por encima deberá mostrar un tooltip con el texto "Eliminar objeto".
    - Deberá estar anclado arriba y a la derecha (propiedad Anchor).
    - Al accionarlo deberá borrar el objeto seleccionado en la lista. Si no hay nada seleccionado, no hacer nada y mostrar un cartel informando que se debe seleccionar un elemento de la lista.
- Tendrá un `Button` llamado `btnModificar` cuyo texto será "M" y al pasar por encima deberá mostrar un tooltip con el texto "Modificar objeto".
    - Deberá estar anclado arriba y a la derecha (propiedad Anchor).
    - Al accionarlo instanciará un FrmAltaModificacion y lo mostrará de forma modal.
        - El título será "Modificar objeto".
        - El contenido del txtObjeto será el elemento seleccionado en lstObjetos. Si no hay nada seleccionado, no hacer nada y mostrar un cartel informando que se debe seleccionar un elemento de la lista.
        - El texto del btnConfirmar será "Modificar".
    - Si la propiedad DialogResult de la instancia de FrmAltaModificacion vale DialogResult.OK, modificar el objeto en la lista del supermercado asignándole el valor de la propiedad Objeto. Caso contrario, no hacer nada.
- En el manejador del evento Load se deberá buscar el archivo listaSupermercado.xml en la carpeta de datos de aplicaciones (Environment.SpecialFolder.ApplicationData) y, si existe, deserializarla desde formato XML como una lista de string.
    - Cargar lstObjetos con los elementos de la lista.
    - Si el archivo no existe, no hacer nada.
- Ante cualquier acción se deberá actualizar el contenido de lstObjetos y el archivo listaSupermercado.xml, que contendrá la lista de objetos serializada a formato XML y se encontrará en la ubicación antes nombrada.

TIP

Para actualizar el contenido de un ListBox se debe cargar la propiedad DataSource como null y luego asignarle a la misma la colección de elementos:

```
listBox.DataSource = null;
listBox.DataSource = listaSupermercado;
```

IMPORTANTE

Cualquier excepción deberá ser manejada mostrando un MessageBox con el mensaje y el stack trace.

Reutilizar código en todo el proyecto siguiendo el principio DRY.

I03. La siempre clásica y eficaz receta

Crear un proyecto de consola. Seguir esta receta al pie de la letra sin agregar nada:

1. Crear una clase `Persona` con dos atributos privados de tipo `string`, nombre y apellido.
    a. Agregarle un constructor que reciba ambos parámetros.
    b. Crear un método estático llamado `Guardar` que reciba un objeto de tipo `Persona`, la serialice en formato XML y la almacene en un archivo con nombre inválido (por ejemplo: "").
    c. Crear un método estático llamado `Leer` que deserialice desde un archivo con nombre inválido (por ejemplo: "") y retorne un objeto de tipo `Persona`.
    d. Sobrecargar el método `ToString` para mostrar los datos de la persona.
2. En el método `Main`, instanciar un objeto del tipo `Persona` e intentar serializarlo.
3. Luego intentar leer ese objeto serializado en una nueva instancia de Persona y mostrarlo por pantalla.
4. Repetir los métodos Guardar y Leer, pero con formato JSON.