Programación Visual en C#.NET

Guion de la práctica 3

OBJETIVOS

- Utilización de cajas de texto con formato
- Aprender a diseñar y utilizar cajas de diálogo
- Aprender a diseñar y utilizar menús emergentes y dinámicos

TEMPORIZACIÓN

Recogida del enunciado:	Semana del 4 de octubre
Desarrollo de la práctica:	**2 semanas**
Fecha de entrega:	Semana del 25 de octubre junto con las prácticas 1, 2 y 4
Fecha límite de entrega:	Semana del 1 de noviembre

PRÁCTICA 3

Cajas de texto con formato, diálogos y menús emergentes y dinámicos

TABLA DE CONTENIDOS:

Introducción

Continuaremos trabajando con el programa desarrollado anteriormente. En esta ocasión dotaremos a nuestro reloj de una alarma. Se le permitirá al usuario introducir la hora de la alarma a través del teclado, aceptándose únicamente los caracteres numéricos y el carácter dos puntos ':'. Para ello crearemos una clase derivada de la clase `MaskedTextBox`, y la personalizaremos para que facilite la edición de horas. La alarma sonará emitiendo un pitido cada segundo durante cinco minutos a partir de la hora establecida por el usuario.

En el punto 6.3 se creará un nuevo formulario que permitirá al usuario definir una zona horaria, probando su buen funcionamiento y visualizando la correcta obtención de los datos requeridos. Para el desarrollo de este formulario reutilizaremos nuestra clase de edición de horas.

En el punto 7.1 se añadirán al programa opciones en un menú para que el usuario pueda activar y desactivar el despertador. Esto se hará de tres maneras, para practicar con las distintas posibilidades existentes: cambiando el texto de la opción del menú, poniéndole una marca (*check*), y habilitándola o inhabilitándola.

En el punto 7.2 se añadirá al programa un menú de zonas horarias. Las opciones de este menú se añadirán y eliminarán de acuerdo a las acciones del usuario. Cuando el usuario seleccione una zona horaria, el programa se limitará a mostrar un mensaje que refleje la selección realizada. La funcionalidad del menú de zonas horarias se completará en una práctica posterior.

En el punto 7.3 se añadirán aceleradores para todas las acciones implementadas anteriormente (activar/desactivar despertador, añadir zona, eliminar zona, etc.)

En el punto 7.4 se añadirán menús emergentes a la aplicación, llamados también menús contextuales.

6. Cajas de texto con formato y diálogos

6.1 Un nuevo control: clase EditHora

Añadiremos a nuestro formulario `RelojDigital` una caja de texto en la que el usuario podrá introducir la hora a la que suene una alarma. El objetivo de este apartado es crear un nuevo tipo de control que filtre los datos introducidos por el usuario en esta caja de texto, de forma que únicamente se le permita introducir caracteres numéricos y el carácter ':'.

Para ello, cree en su proyecto una nueva clase que derive de la clase "caja de texto con máscara" (`MaskedTextBox`). En la ventana *Explorador de soluciones*, haga clic con el botón derecho del ratón sobre el nombre de su proyecto y seleccione *Agregar→ Nuevo elemento*. Seleccione *Clase* como tipo de elemento y póngale como nombre `EditHora`.

A continuación, haremos que nuestra clase derive de la clase que representa una caja de texto con máscara (`MaskedTextBox`). Modifique el comienzo de la definición de la clase `EditHora` así:

```
class EditHora : MaskedTextBox
```

Para que este código compile tendrá que hacer visible el espacio de nombres `System.Windows.Forms`.

Añada un constructor público sin parámetros a `EditHora` (puede usar la herramienta *Editar→IntelliSense→Insertar fragmento de código→ctor*), e introduzca en él el siguiente código, que sirve para iniciar algunas propiedades de nuestro nuevo control:

```
this.Mask = "90:00:99";
this.TextAlign = HorizontalAlignment.Center;
this.BeepOnError = true;
this.PromptChar = '-';
```

En la propiedad **Mask**, un '9' indica una posición para un dígito o espacio opcional, mientras que '0' indica que ahí tiene que escribirse obligatoriamente un dígito.

Cuando el usuario escriba en un control de la clase `EditHora`, la hora introducida la almacenaremos en su atributo `m_Hora` de tipo `DateTime` (puede declarar propiedades con mayor sencillez usando la herramienta *Editar→IntelliSense→Insertar fragmento de código→prop*):

```
private DateTime m_Hora;

public DateTime Hora
{
  ...

}
```

Los clientes de nuestro control accederán a la hora introducida por el usuario a través de esta propiedad, pero antes deberán comprobar que se ha introducido una hora correctamente usando la siguiente propiedad de sólo lectura:

```
public bool HoraValida
{
   get
   {
      if (!this.MaskCompleted)
         return false;

      string aux = this.Text.Replace(this.PromptChar, '0');
      DateTime hora = new DateTime();
      if(!DateTime.TryParse(aux, out hora))
         return false; // hora no correcta

      this.Hora = hora;
      return true; // hora correcta
   }
}
```

La idea es reemplazar el prompt (-) por 0 en la cadena *Text* del objeto *EditHora*, de forma que, por ejemplo, "-5:24:--" se convierta en "05:24:00", cadena que sí se puede convertir en un objeto **DateTime**.

Compilar y ejecutar para ver el resultado.

Una vez compilado el proyecto, el control `EditHora` debería estar disponible en el cuadro de herramientas de Visual Studio. Si no lo está, seleccione *Hermanientas→Opciones…* en el menú principal y en la ventana que aparece, active la opción **AutoToolBoxPopulate** de la pestaña *Diseñador de Windows Forms*. Hecho esto, tal vez tenga que recompilar el proyecto para que se actualice el contenido del cuadro de herramientas.

Usando el cuadro de herramientas, añada un control de tipo `EditHora` al formulario `RelojDigital`, y asigne a su propiedad **Name** el valor "ct_Alarma". Observe cómo se reflejan en el panel de propiedades los valores de las propiedades que iniciamos en el constructor de la clase (por ejemplo, **Mask**).

Agregue una etiqueta encima de la caja de texto que acaba de añadir, con el texto "Alarma:". Opcionalmente, puede cambiar la fuente y el color de fondo de la caja de texto para diferenciarla de la que muestra la hora.

Compilar y ejecutar para ver el resultado.

6.2 Utilización de la nueva clase EditHora. Alarma

Queremos conseguir que nuestra alarma suene, cada segundo, durante cinco minutos a partir de la hora establecida por el usuario. Además, queremos que con sólo establecer el número de horas y de minutos se produzca el sonido de la alarma, independientemente del valor de los segundos (que por defecto será 0).

Para conseguir hacer sonar la alarma en las condiciones anteriormente descritas, modifique el manejador del temporizador que se añadió en la práctica 1 tal y como se indica a continuación:

```
private void timer_Tick(object sender, EventArgs e)
{
  MostrarHoraActual();

  if (!ct_Alarma.HoraValida)
  {
    System.Diagnostics.Debug.WriteLine("Hora alarma nó válida");
    return;
  }
  DateTime horaActual = DateTime.Now + m_DesfaseHorario;
  DateTime horaAlarma = ct_Alarma.Hora;

  int segundosActual = horaActual.Second
                       + horaActual.Minute * 60
                       + horaActual.Hour * 3600;

  // Segundos alarma:
  ...
```

```
if (segundosActual >= segundosAlarma)
{
  if (segundosActual - segundosAlarma < 300)
    Console.Beep(); // o bien System.Media.SystemSounds.Beep.Play();
}
else if (segundosAlarma - segundosActual > 86400 - 300)
    Console.Beep(); // o bien System.Media.SystemSounds.Beep.Play();
```

Compilar y ejecutar en modo depuración (F5) para ver el resultado.

Pruebe a introducir el valor "-5:24:--". Observará que en la ventana de resultados se muestra un mensaje indicando que la hora de la alarma no es válida. ¿Por qué sucede? Verifique el valor de la propiedad **Text**. La solución pasa por establecer adecuadamente la propiedad **TextMaskFormat** en el constructor de la clase del control:

```
public EditHora()
{
  ...
  this.TextMaskFormat = MaskFormat.IncludePromptAndLiterals;
}
```

Después de compilar la aplicación, verifique en la ventana de propiedades del control ya incluido en el formulario `RelojDigital` que su propiedad **TextMaskFormat** tiene el valor establecido, si no cámbielo manualmente.

6.3 Reutilización. Un nuevo diálogo con un control EditHora

A continuación añadiremos al programa la funcionalidad necesaria para mostrar al usuario la hora en diferentes zonas horarias. El usuario podrá agregar zonas horarias introduciendo sus datos en una ventana.

Añada un nuevo formulario (menú *Proyecto→Agregar Windows Form…→Windows Form* con el nombre `DlgDatosZona`. Añada los controles que se muestran a continuación, teniendo en cuenta que la caja de texto "Diferencia horaria" debe ser del tipo `EditHora` creado con anterioridad.

Asigne nombres descriptivos, como "ct_Nombre", "bt_Aceptar" o "rb_Positivo", a estos controles (salvo a las etiquetas, que pueden quedarse con el nombre que toman por defecto). Asigne a la propiedad **FormBorderStyle** del formulario el valor **FixedToolWindow**. Las propiedades **AcceptButton** y **CancelButton** del formulario deberán vincularse con los botones "Aceptar" y "Cancelar", respectivamente. Además,

la propiedad **DialogResult** del botón "Aceptar" tiene que valer **None** y la de "Cancelar", **Cancel**. Active la propiedad **Checked** del botón de opción "Positivo".

Habrá que tener especial cuidado en el orden de los controles (el orden en el que van siendo recorridos por el foco cuando el usuario pulsa la tecla *Tab*). Este orden puede ser modificado mediante la propiedad **TabIndex** de cada objeto, o bien ejecutando la orden *Orden Tab* del menú *Ver*.

Añada a la clase `DlgDatosZona` tres campos privados `m_Nombre`, `m_Diferencia` y `m_Positivo` para guardar en ellos los datos del formulario cuando el usuario pulse el botón "Aceptar". Para acceder a estos campos desde fuera de la clase, agregue propiedades públicas, `Nombre`, `Diferencia` y `Positivo`, que se limiten a devolver/asignar el campo correspondiente.

Añada un manejador para el botón "Aceptar" del formulario `DlgDatosZona` y edítelo como se muestra a continuación:

```
private void bt_Aceptar_Click(object sender, EventArgs e)
{
   if (ct_Nombre.Text != "")
      this.Nombre = ct_Nombre.Text;
   else
   {
     ...
   }
   if (ct_Diferencia.HoraValida)
   {
      DateTime hora = ct_Diferencia.Hora;
      this.Diferencia = new TimeSpan(hora.Hour, hora.Minute,
                                     hora.Second);
   }
   else
   {
      MessageBox.Show("Introduzca una diferencia horaria válida",
                      "Error en los datos",
                      MessageBoxButtons.OK, MessageBoxIcon.Error);
      ct_Diferencia.Focus();
      ct_Diferencia.SelectAll();
      return;
   }
   this.Positivo = rb_Positivo.Checked;

   // Como los datos son válidos el resultado devuelto por el
   // diálogo debe ser OK
   ...
}
```

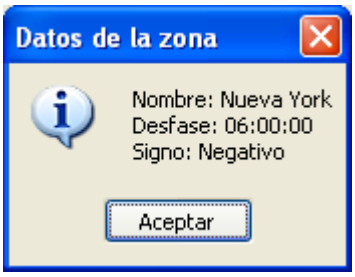
Para probar el formulario recién creado, añada un botón titulado "Diálogo zona" al formulario `RelojDigital`. A continuación, agregue el siguiente manejador para el evento **Click** de este botón:

```
private void bt_ZonaHoraria_Click(object sender, EventArgs e)
{
   DlgDatosZona dlg = new DlgDatosZona();

   if( ... )
   {
```

```
            string desfase = dlg.Diferencia.ToString();
            string signo = dlg.Positivo ? "Positivo" : "Negativo";

            // Visualizar una caja de diálogo que muestre los datos
            // de la zona recogidos en el diálogo
            ...
        }
    }
```



Compilar y ejecutar para ver el resultado.

7. Menús emergentes y dinámicos

7.1 Estado de los elementos del menú

Añada a la clase `RelojDigital` la siguiente propiedad de tipo `bool`:

```
private bool m_DespertadorActivado = false;

public bool DespertadorActivado
{
   get  {   return m_DespertadorActivado;  }
   set  {   m_DespertadorActivado = value; }
}
```

Modifique el método que responde al evento **Tick** del temporizador, para que sólo funcione la alarma cuando `DespertadorActivado` valga *true.*

Añada a la barra de menús un nuevo menú “Despertador”, entre los menús “Opciones” y “Ayuda”, formado por una única orden “Activar”. Asigne a la propiedad **Name** de esta orden el identificador “DespertadorActivar*”*. Añada el siguiente manejador para el evento **Click** de la orden:

```
private void DespertadorActivar_Click(object sender,  EventArgs e)
{
  this.DespertadorActivado = !this.DespertadorActivado;
}
```

Compilar y ejecutar para ver el resultado.

¿Cómo puede el usuario saber si el despertador está activado? Según el desarrollo realizado hasta ahora, sólo puede saberlo esperando a que llegue la hora a la que tiene que sonar. Para solucionar esto, modifique el método `set` de la propiedad `DespertadorActivado` como se muestra a continuación:

```
set
{
   m_DespertadorActivado = value;
   DespertadorActivar.Text =
             m_DespertadorActivado ? "Desacti&var" : "Acti&var";
}
```

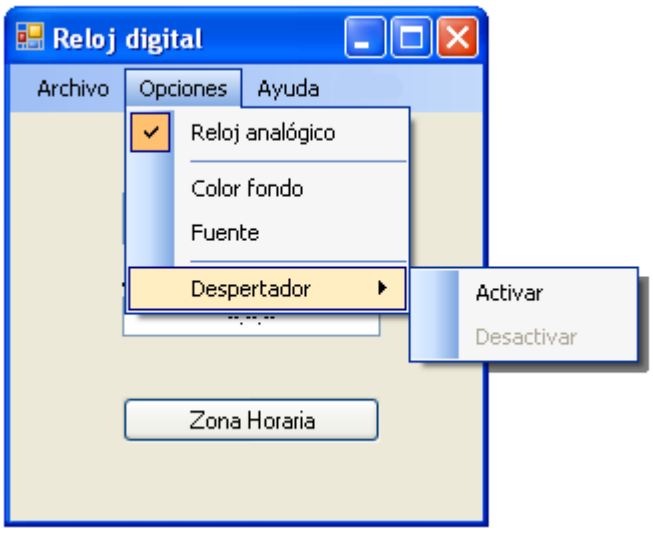
Compilar y ejecutar para ver el resultado.

Pruebe también el siguiente código:

```
set
{
   m_DespertadorActivado = value;
   DespertadorActivar.Checked = m_DespertadorActivado;
}
```

Compilar y ejecutar para ver el resultado.

Elimine el menú "Despertador" y cree uno nuevo como submenú de "Opciones", tal y como se indica en la siguiente figura (la orden "Desactivar" debe comenzar deshabilitada):



Añada manejadores para el evento **Click** de las opciones "Activar" y "Desactivar". En cada momento sólo una de las dos opciones deberá estar habilitada. Para conseguir este efecto, modifique el método `set` de la propiedad `DespertadorActivado` como se muestra a continuación:

```
set
{
   m_DespertadorActivado = value;

   OpcionesDespertadorActivar.Enabled = !m_DespertadorActivado;
   OpcionesDespertadorDesactivar.Enabled = m_DespertadorActivado;
}
```

Compilar y ejecutar para ver el resultado.

7.2 Menús dinámicos: añadir/eliminar elementos durante la ejecución

Añada el siguiente campo privado a la clase `RelojDigital`:

```
const int MaxZonas = 4;
```

Añada, a la derecha de "Opciones", un nuevo menú "Zona" formado por dos órdenes, etiquetadas "Añadir" y "Eliminar", y un separador.

Asigne a la propiedad **Name** de "Zona", de "Añadir", de "Eliminar" y del separador los valores `menúZona`, `zonaAñadir`, `zonaEliminar` y `zonaSeparador`, respectivamente.

Agregue ahora la siguiente propiedad, que permite conocer el número de zonas que ha añadido el usuario hasta el momento:

```
public int NúmeroZonas
{
   get { return menúZona.DropDownItems.Count - 3; }
}
```

Añada un manejador para el evento **DropDownOpened** del menú "Zona". Edite este manejador para que oculte la orden "Añadir" cuando se haya alcanzado el límite de zonas, y para que oculte la orden "Eliminar" y el separador cuando no exista ninguna zona (haga uso de la propiedad `NúmeroZonas`).

Compilar y ejecutar para ver el resultado.

Añada el siguiente manejador para el evento **Click** de la orden "Añadir" del menú "Zona":

```
private void zonaAñadir_Click(object sender, EventArgs e)
{
   ToolStripMenuItem zonaNueva = new ToolStripMenuItem();
   zonaNueva.Text = "Zona " + (this.NúmeroZonas + 1);

   // Especifique el manejador del evento Click de zonaNueva; la
   // respuesta a este evento vendrá dada por el método zona_Click.
   // Añada el elemento zonaNueva al menú menúZona.
}
```

Añada ahora el método que deberá ejecutarse en respuesta a la orden "Eliminar" del menú "Zona". Este método deberá quitar la última zona del menú.

Por último añada el método `zona_Click` que se ejecutará en respuesta a las órdenes "Zona n" del menú "Zona". Su función, por el momento, será simplemente mostrar mediante un mensaje el nombre de la zona seleccionada por el usuario (pista: el manejador tiene un parámetro `sender` que puede utilizar para obtener el nombre de la orden seleccionada del menú).

Compilar y ejecutar para ver el resultado.

7.3 Atajos de teclado

Añada los atajos de teclado (mediante la propiedad **ShortcutKeys**) correspondientes a las órdenes "Despertador > Activar" (Ctrl+T), "Despertador > Desactivar" (Ctrl+V), "Zona > Añadir" (Ctrl+Shift+R) y "Zona > Eliminar" (Ctrl+Shift+L). Opcionalmente puede añadir otros atajos que considere necesarios, como el de la orden "Archivo > Salir".

Compilar y ejecutar para ver el resultado.

¿Qué pasa si pulsa el atajo de la orden "Zona > Añadir" muchas veces seguidas? Añada el siguiente código al comienzo del manejador `zonaAñadir_Click` para solucionar este problema:

```
if (this.NúmeroZonas == MaxZonas)
{
   Console.Beep(); // o bien System.Media.SystemSounds.Beep.Play();
   return;
}
```

El manejador de la orden "Eliminar" puede dar lugar a un problema similar. Soluciónelo.

Compilar y ejecutar para ver el resultado.

7.4 Menús Contextuales

Añada un menú contextual (objeto **ContextMenuStrip** del cuadro de herramientas) denominado "menúContextoAlarma" con dos órdenes "Activar" y "Desactivar". Asócielo con la caja de texto que visualiza la hora de la alarma.

Compilar y ejecutar para ver el resultado.

Seleccione el elemento "Activar" del menú contextual y en el editor de propiedades seleccione el evento **Click**. Haciendo uso del menú desplegable seleccione como manejador para este evento el que implementó en el apartado 7.1 para la orden "Opciones > Despertador > Activar". Haga lo propio con la orden "Desactivar".

Compilar y ejecutar para ver el resultado.

Elimine los elementos "Activar" y "Desactivar" que cuelgan del submenú "Opciones > Despertador". Asocie a la propiedad **DropDown** de este submenú el menú contextual "menúContextoAlarma". Cambie la propiedad **Name** de los elementos "Activar" y "Desactivar" del menú contextual por los valores "OpcionesDespertadorActivar" y "OpcionesDespertadorDesactivar", respectivamente. Este último deberá aparecer inhabilitado inicialmente (**Enabled** = *false*).

Compilar y ejecutar para ver el resultado.

Crearemos ahora un menú contextual "Zona" procediendo de forma análoga.

Elimine todos los elementos que cuelgan del menú "Zona". Añada otro menú contextual denominado "menúContextoZona" compuesto por dos órdenes, "Añadir" y "Eliminar", y por un separador. En el editor de propiedades póngales el mismo nombre que tenían los elementos que colgaban del menú "Zona". Asocie los manejadores ya creados para el menú "Zona" con sus correspondientes elementos en el nuevo menú contextual (incluido el manejador `menúZona_DropDownOpened`, que ahora corresponderá al evento **Opened**).

En el código del archivo "RelojDigital.cs", cambie todas las referencias al objeto "menúZona" por referencias al nuevo menú "menúContextoZona". Si intenta compilar ahora el proyecto, verá que no es posible ya que la clase `ContextMenuStrip` no contiene una propiedad **DropDownItems**. Cambie todos los accesos a esta propiedad por accesos a la propiedad correspondiente de la clase `ContextMenuStrip`.

En el editor de propiedades del menú "Zona" modifique la propiedad **DropDown** seleccionando `menúContextoZona` en la lista desplegable.

Compilar y ejecutar para ver el resultado.

Asocie el menú contextual `menúContextoZona` a la propiedad **ContextMenuStrip** del formulario `RelojDigital`.

Añada un control **ToolTip** a este formulario y establezca para la barra de menús el mensaje abreviado "Añada/elimine una zona pulsando el botón derecho del ratón" (esto debe hacerse a través de la propiedad **ToolTip** de la propia barra de menús).

Compilar y ejecutar para ver el resultado.

Modifique la última acción realizada y haga que el mensaje abreviado vinculado con la barra de menús se establezca desde el recurso `mensajeBarraMenus` del proyecto en el instante en el que se carga el formulario.

Compilar y ejecutar para ver el resultado.