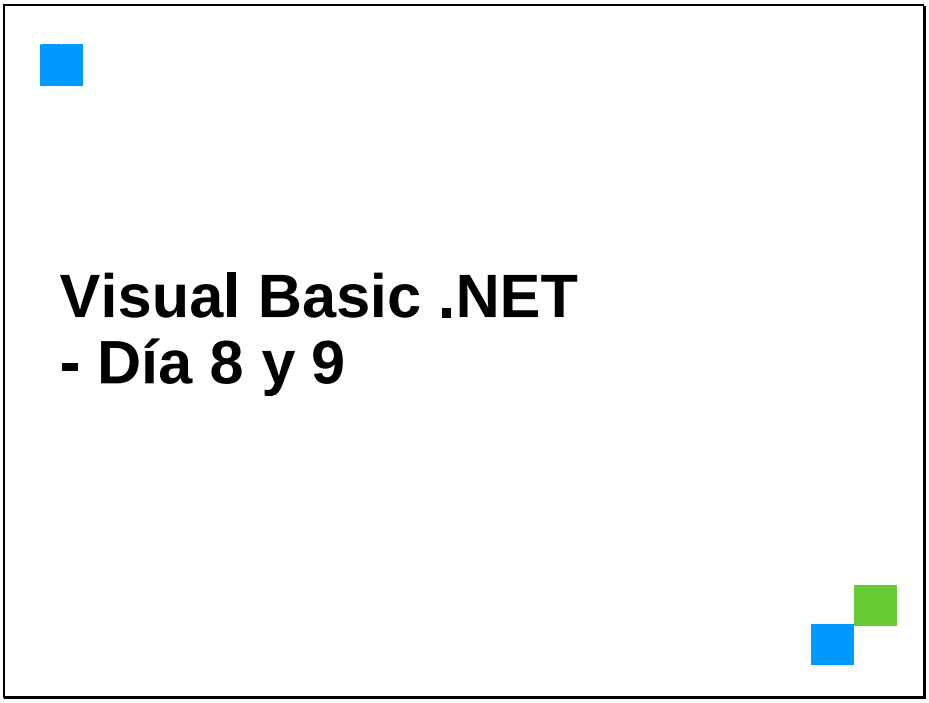
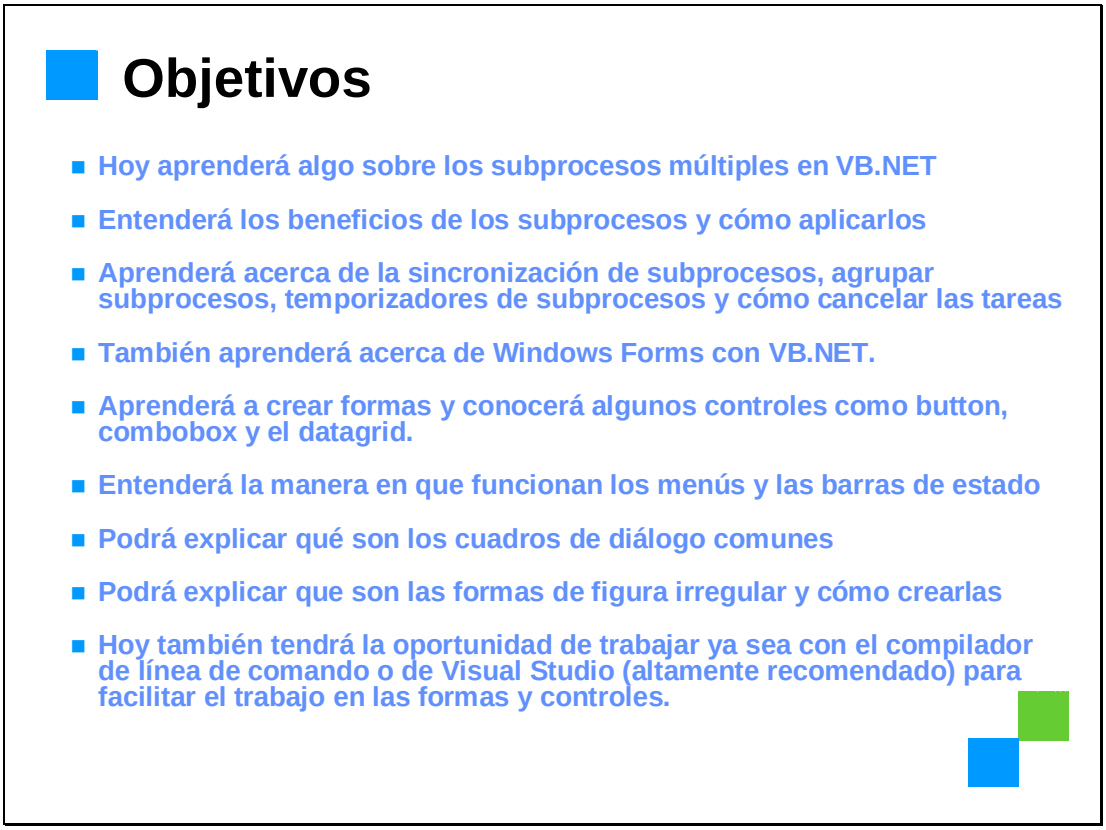
Diapositiva 1



Diapositiva 2



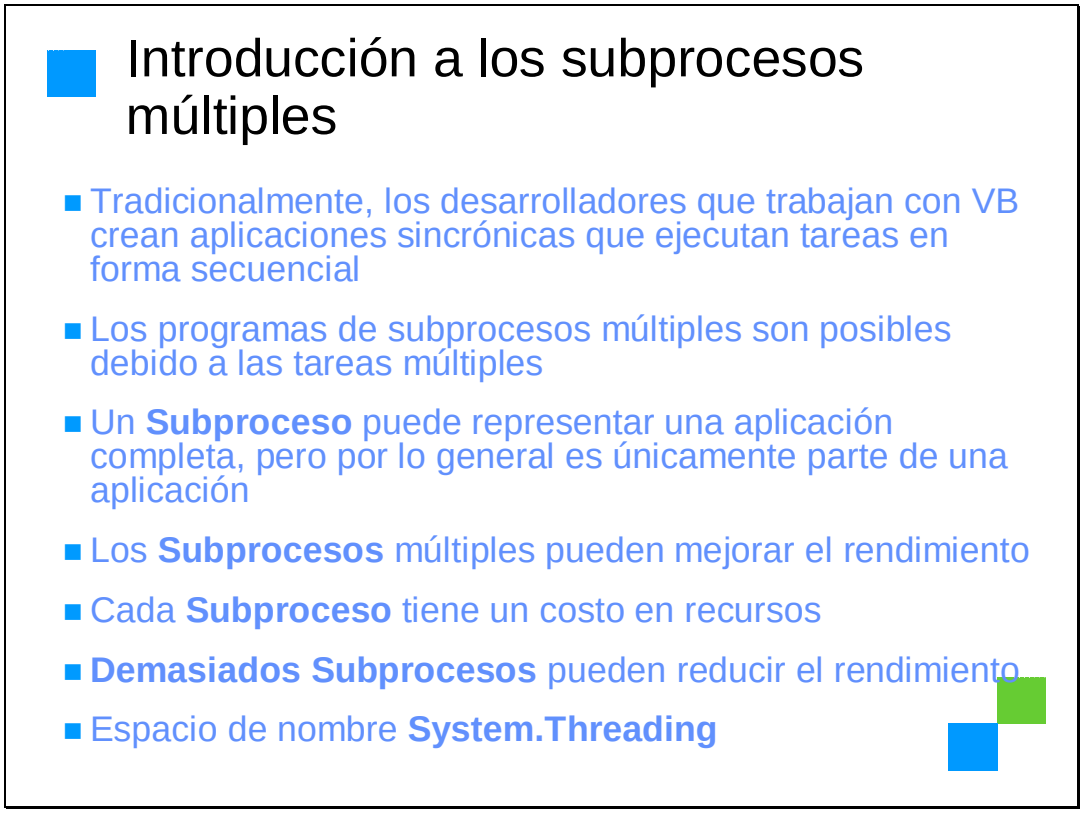
Diapositiva 3

Parte 1/2

Subprocesos múltiples

- Introducción
- Ventajas de los subprocesos
- Sincronización
- Agrupar subprocesos
- Temporizadores de subprocesos
- Cancelar tareas

Diapositiva 4



Tradicionalmente, los desarrolladores Visual Basic han creado aplicaciones sincrónicas en las cuales las tareas del programa se ejecutan en forma secuencial. Aunque las aplicaciones con subprocesos múltiples pueden ser más eficientes debido a que múltiples tareas se ejecutan de una manera más o menos simultánea, era difícil crearlas utilizando las versiones anteriores de Visual Basic.

Los programas con subprocesos múltiples son posibles debido a una función del sistema operativo llamada multitareas que simula la capacidad de ejecutar aplicaciones múltiples al mismo tiempo. Aunque la mayoría de las PCs únicamente tienen un solo procesador, los sistemas operativos modernos proporcionan multitareas al dividir el tiempo del procesador entre múltiples piezas de código ejecutable llamadas subprocesos.

Un subproceso puede representar una aplicación completa, pero con frecuencia representa únicamente una parte de la aplicación que se puede ejecutar de manera separada. El sistema operativo asigna tiempo de procesamiento para cada subproceso con base en factores tales como la prioridad del subproceso y cuánto tiempo ha transcurrido desde que se ejecutó por última vez ese subproceso.

Los subprocesos múltiples pueden mejorar de manera importante el rendimiento cuando se ejecutan tareas que requieren mucho tiempo, tales como la entrada y salida de archivos.

Aquí debemos hacer una aclaración. Aunque los subprocesos múltiples pueden mejorar el rendimiento, cada subproceso tiene un costo asociado en términos de memoria adicional requerida para crear el subproceso y tiempo de procesador requerido para mantenerlo en ejecución. Crear demasiados subprocesos de hecho puede reducir el rendimiento de su aplicación. Al diseñar aplicaciones con subprocesos múltiples deberá ponderar los beneficios de agregar subprocesos adicionales contra su costo.

Las tareas múltiples han sido parte de los sistemas operativos durante algún tiempo. Hasta hace poco tiempo, sin embargo, los programadores de Visual Basic únicamente podían realizar tareas de subprocesos múltiples utilizando funciones no documentadas o de manera indirecta al utilizar los componentes COM o las partes asíncronas del sistema operativo.

.NET Framework ofrece un apoyo completo en el espacio de nombre **System.Threading** para desarrollar aplicaciones de subprocesos múltiples.

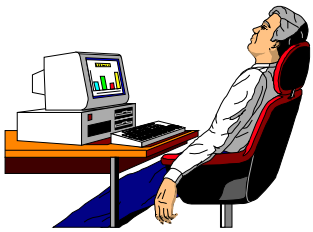
Diapositiva 5

Subprocesos múltiples en .NET

¿Por qué utilizar subprocesos?

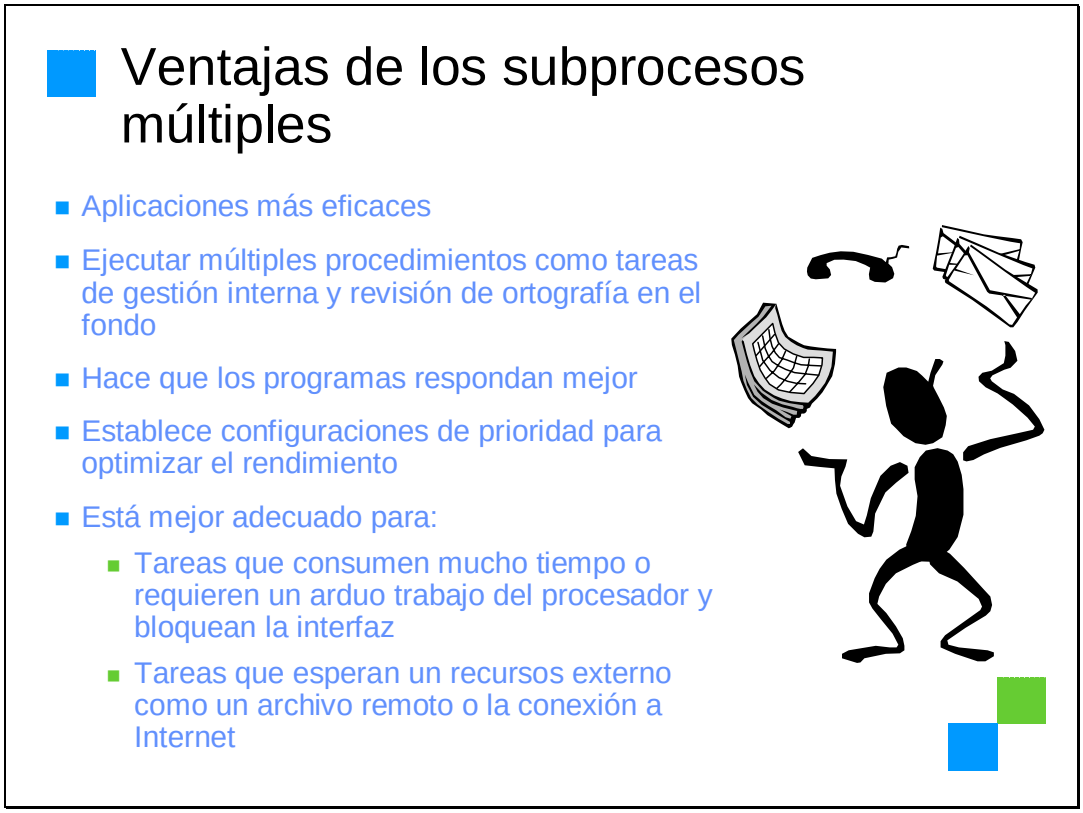
Rendimiento

Rendimiento percibido



Simplificar la codificación

Diapositiva 6



Ventajas del procesamiento con subprocesos múltiples

A pesar de que las aplicaciones sincrónicas son más fáciles de desarrollar, con frecuencia son menos eficaces que las aplicaciones con subprocesos múltiples debido a que una tarea nueva no se puede iniciar hasta que la anterior haya terminado. Si completar una tarea asíncrona requiere más de lo esperado, su aplicación puede parecer que no responde.

El procesamiento con subprocesos múltiples puede ejecutar simultáneamente varios procedimientos. Por ejemplo, una aplicación de procesador de texto puede revisar la ortografía como una tarea por separado mientras que continúa trabajando en el documento. Debido a que las aplicaciones con subprocesos múltiples dividen sus programas en tareas independientes, pueden mejorar de manera sustancial el rendimiento en las siguientes formas:

Las técnicas de subprocesos múltiples pueden hacer que se programa responda mejor debido a que la interfaz puede permanecer activa mientras que se realizan otros trabajos.

Las tareas que no están ocupadas en ese momento pueden ofrecer tiempo de procesador a otras tareas.

Las tareas que utilizan una gran cantidad de tiempo de procesamiento pueden brindarlo periódicamente a otras tareas.

Las tareas se pueden detener en cualquier momento.

Puede establecer la prioridad de las tareas individuales a un nivel mayor o menor para optimizar el rendimiento.

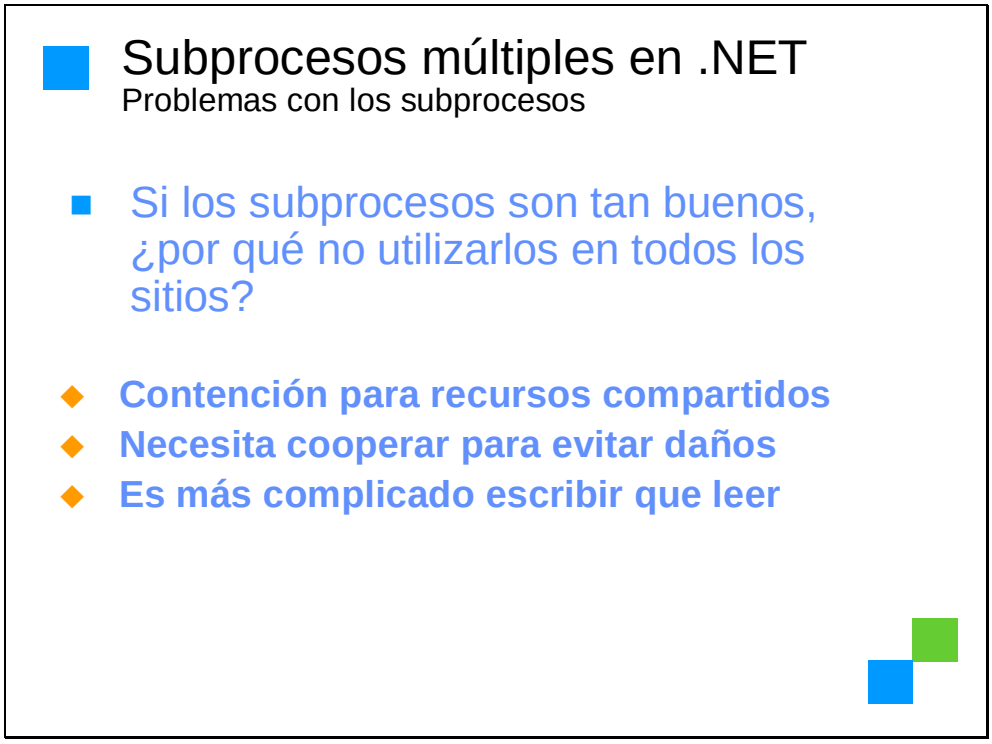
La decisión de crear explícitamente una aplicación con subprocesos múltiples depende de varios factores. Los subprocesos múltiples son mejores en situaciones donde:

Las tareas que requieren mucho tiempo o requieren un uso intensivo del procesador bloquean la interfaz.

Las tareas individuales deben esperar para recibir un recurso externo, como puede ser un archivo remoto o una conexión a Internet.

Por ejemplo, considere un "robot" de Internet, una aplicación que sigue los vínculos en las páginas Web y descarga archivos para satisfacer criterios específicos. Esta aplicación puede descargar de manera asíncrona cada archivo uno tras otro o utilizar los subprocesos varios para descargar múltiples archivos al mismo tiempo. El enfoque de subprocesos múltiples puede ser mucho más eficiente que el enfoque sincrónico debido a que los archivos se descargan incluso cuando algunos subprocesos reciban respuestas lentas de los servidores Web remotos.

Diapositiva 7



Diapositiva 8

Crear nuevos subprocesos

- Subproceso nuevo:

```
Dim Thread1 As New System.Threading.Thread(AddressOf
  SomeTask)
Thread1.Start ' Code here runs immediately.
```

- El método proporcionado como argumento para el método Thread no puede tener un parámetro o valor de retorno. Sin embargo esto se puede lograr al envolverlo en una clase.

```
Class TasksClass
    Friend StrArg As String
    Friend RetVal As Boolean
    Sub SomeTask()
        ' Use the StrArg field as an argument.
        MsgBox("The StrArg contains the string " & StrArg)
        RetVal = True ' Set a return value in the return argument.
    End Sub
End Class
' To use the class, set the properties or fields that store parameters,
' and then asynchronously call the methods as needed.
Sub DoWork()
    Dim Tasks As New TasksClass()
    Dim Thread1 As New System.Threading.Thread( _
        AddressOf Tasks.SomeTask)
    Tasks.StrArg = "Some Arg" ' Set a field that is used as an argument
    Thread1.Start() ' Start the new thread.
    Thread1.Join() ' Wait for thread 1 to finish.
    ' Display the return value.
    MsgBox("Thread 1 returned the value " & Tasks.RetVal)
End Sub
```

Crear nuevos subprocesos

La manera más directa al crear un subproceso es al crear una nueva instancia de la clase Thread y utilizar la instrucción **AddressOf** para pasar un delegado para el procedimiento que desea ejecutar.

Por ejemplo, el siguiente código ejecuta un subproceso llamado SomeTask como un subprocesos separado.

```
Dim Thread1 As New System.Threading.Thread(AddressOf SomeTask)
Thread1.Start ' Code here runs immediately.
```

Eso es todo lo que se necesita para crear e iniciar un subproceso.

Cualquier código que siga una llamada al método Start del subproceso se ejecuta de inmediato sin tener que esperar a que termine el subprocesos anterior.

La siguiente tabla muestra algunos de los métodos que puede utilizar para controlar subprocesos individuales.

Método	**Acción**
Start	Causa que el subproceso se empiece a ejecutar.
Sleep	Pausa un subproceso por un tiempo específico.
Suspend	Pausa un subproceso cuando llega a un punto seguro.
Abort	Detiene un subproceso cuando alcanza un punto seguro.
Resume	Reanuda un subproceso suspendido
Join.	Causa que el subproceso actual espere a que otro subproceso termine. Si se utiliza con un valor de temporizador, este método devuelve True si el subproceso termina dentro del tiempo asignado.

La mayoría de estos métodos son auto explicatorios, pero el concepto de los puntos seguros puede ser nuevo.

Los puntos seguros son lugares en el código en los que es seguro que el tiempo de ejecución de lenguaje común realice una recolección automática de basura, el proceso de liberar variables no utilizadas y reclamar memoria.

Cuando invoca los métodos Abort o Suspend de un subproceso, el tiempo de ejecución de lenguaje común analiza el código y determina la ubicación y el lugar adecuado en donde se debe detener el subproceso.

Los subprocesos también contienen una serie de propiedades útiles, como aparecen en la siguiente tabla:

Propiedad	Valor
IsAlive	Contiene el valor True si el subproceso está activo.
IsBackground	Obtiene o establece un booleano que indica si el subproceso es o debe ser un subproceso de fondo. Los subprocesos de fondo son como los subprocesos de frente pero los de fondo no evitan que un proceso termine. Una vez que todos los subprocesos del frente que pertenecen a un mismo proceso al terminado, el tiempo de ejecución de lenguaje común termina el proceso al invocar el método Abort que aún está vivo.
Name	Obtiene o establece el nombre del subprocesos. Se utiliza con mayor frecuencia para descubrir los subprocesos individuales al depurar.
Priority	Obtiene o establece un valor utilizado por el sistema operativo para priorizar el programa de subprocesos.
ApartmentState	Obtiene o establece el modelo de subproceso utilizado por un subproceso específico. Los modelos de subproceso son importantes cuando un subproceso invoca un código no administrado.
ThreadState	Contiene un valor que describe el estado o estados del subproceso.

Las propiedades y métodos de los subprocesos son útiles al crear y administrar subprocesos. La sección de sincronización de subprocesos en este artículo analiza la manera en que puede utilizar estas propiedades y métodos para controlar y coordinar subprocesos.

Argumentos de subprocesos y valores de retorno

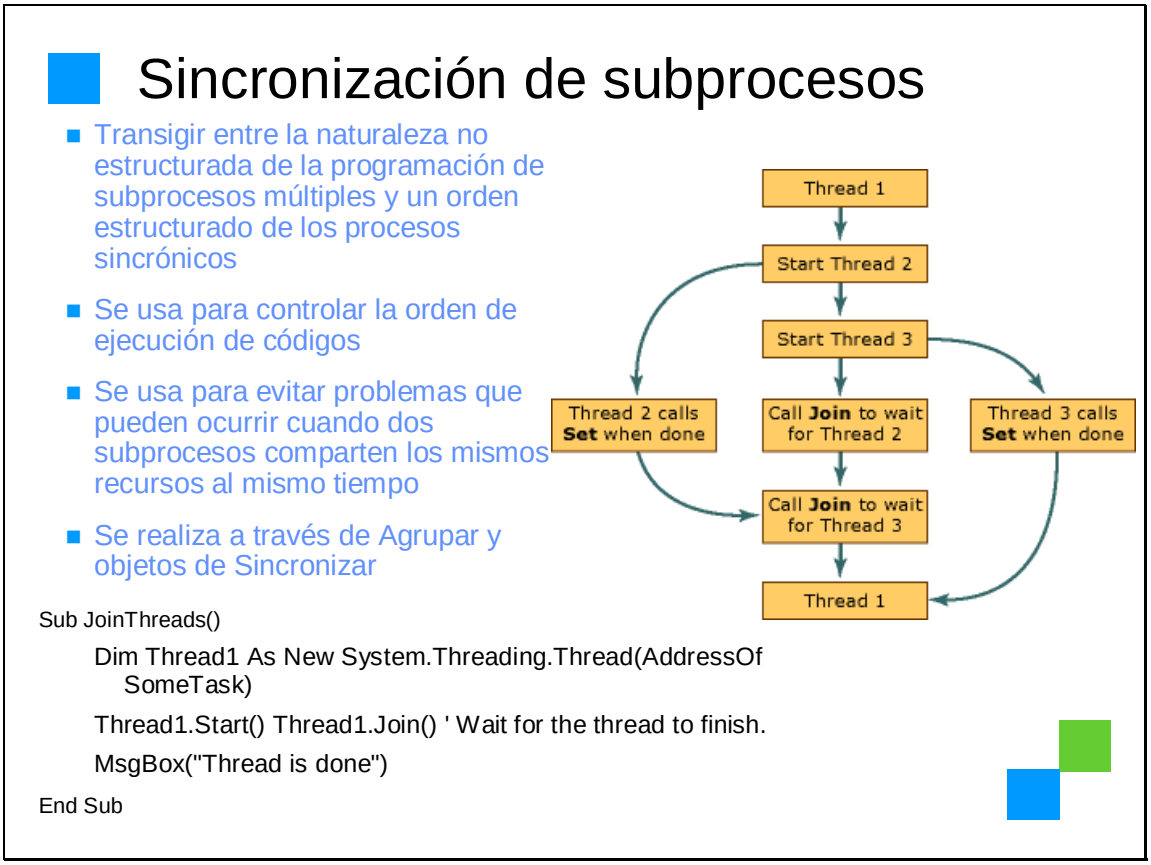
El método Call en el siguiente ejemplo no puede contener ningún parámetro o valor de retorno.

Esta limitación es una de las principales desventajas para crear y ejecutar subprocesos en esta forma. No obstante, puede proporcionar y devolver argumentos para procedimientos que se ejecutan en subprocesos separados al envolverlos ya sea en una clase o una estructura.

```
Class TasksClass
    Friend StrArg As String
    Friend RetVal As Boolean
    Sub SomeTask()
        ' Use the StrArg field as an argument.
        MsgBox("The StrArg contains the string " & StrArg)
        RetVal = True ' Set a return value in the return argument.
    End Sub
End Class
' To use the class, set the properties or fields that store parameters,
' and then asynchronously call the methods as needed.
Sub DoWork()
    Dim Tasks As New TasksClass
    Dim Thread1 As New System.Threading.Thread( _
    AddressOf Tasks.SomeTask)
    Tasks.StrArg = "Some Arg" ' Set a field that is used as an argument
    Thread1.Start() ' Start the new thread.
    Thread1.Join() ' Wait for thread 1 to finish.
    ' Display the return value.
    MsgBox("Thread 1 returned the value " & Tasks.RetVal)
End Sub
```

Crear y administrar manualmente los subprocesos es lo mejor para aplicaciones en las que desea tener un control muy fino de los detalles tales como prioridades del subproceso y modelo de subproceso. Como puede imaginar, puede ser difícil administrar un gran número de subprocesos de esta manera. Considere agrupar subprocesos para reducir la complejidad cuando necesita muchos subprocesos.

Diapositiva 9



Sincronización de subprocesos

La sincronización permite transar entre la naturaleza no estructurada de la programación de subprocesos múltiples y el orden estructurado del procesamiento sincrónico.

Utiliza las técnicas de sincronización para:

- Controlar explícitamente el orden en el que se ejecuta el código siempre que las tareas se deban realizar en una secuencia específica.
- Evitar problemas que puedan ocurrir cuando dos subprocesos comparten los mismos recursos al mismo tiempo.

Por ejemplo, puede utilizar la sincronización para causar que un procedimiento de presentación espere hasta que se complete el procedimiento de recuperación de datos que se ejecuta en otro subproceso.

Existen dos enfoques para la sincronización, agrupar objetos y utilizar la sincronización de objetos.

Agrupar – verifica repetidamente el estado de una llamada asíncrona desde un circuito. Agrupar es la forma menos eficiente de agrupar los subprocesos debido a que desperdicia recursos al verificar repetidamente el estado de las diferentes propiedades del subproceso.

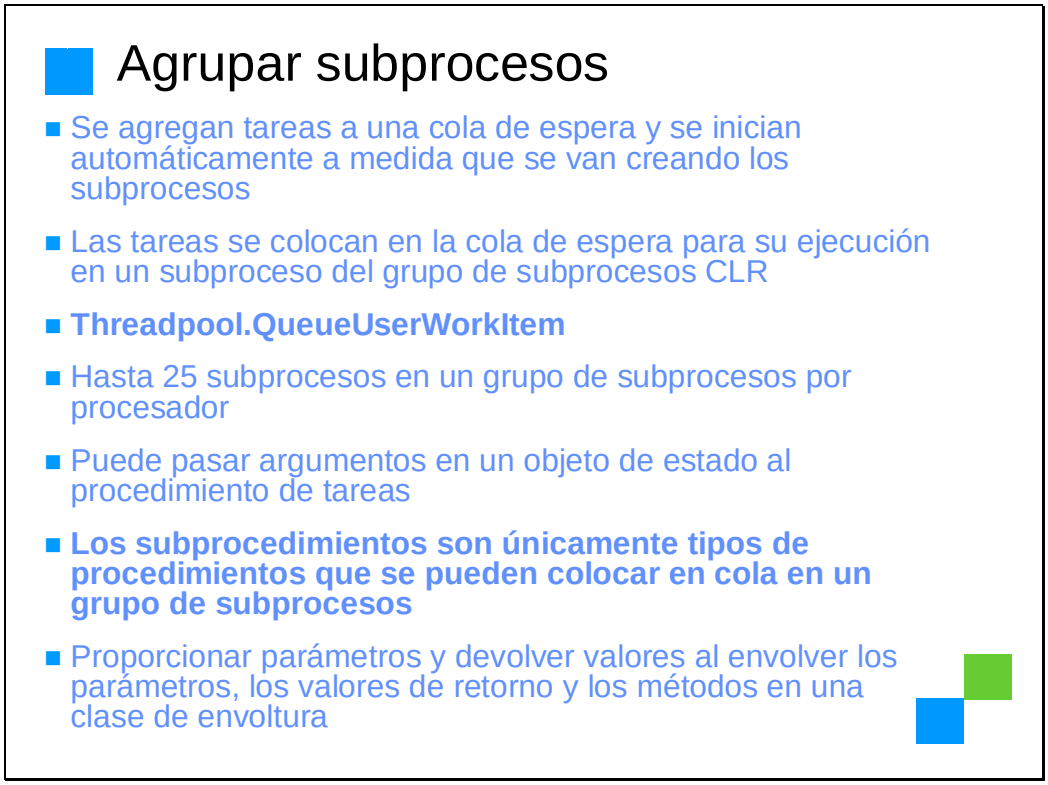
La propiedad IsAlive se puede utilizar al agrupar para ver si ha terminado un subproceso. Utilice está propiedad con precaución debido a que un subproceso que está vivo no necesariamente se está ejecutando. Puede utilizar la propiedad ThreadState para obtener información más detallada acerca del subproceso. Debido a que los subprocesos pueden estar en más de un estado en un momento dado, el valor almacenado en ThreadState puede ser una combinación de los valores en la enumeración System.Threading.Threadstate. Por consiguiente, debe verificar con cuidado todos los estados relevantes de los subprocesos al hacer grupos. Por ejemplo, si el estado de un subproceso indica que no se está Ejecutando, puede haber terminado. Por otro lado, puede estar suspendido o en hibernación.

Sincronización – Como se puede imaginar, agrupar sacrifica algunas ventajas de los subprocesos múltiples a cambio de tener control sobre otros subprocesos que se ejecutan. Un enfoque más eficiente podría ser utilizar el método Join para controlar los subprocesos. Join causa que el procedimiento invocar espere ya sea hasta que se termine el subproceso o hasta que se termine el tiempo de la invocación si se ha especificado un tiempo. El nombre, join, se basa en la idea de crear un nuevo subproceso que genera una nueva vertiente en la ruta de ejecución. Puede utilizar Join para fusionar rutas de ejecución separadas de nuevo en un solo subproceso.
Un punto debe quedar claro: Join es una llamada sincrónica o de *bloqueo*. Una vez que invoca Join, o un método de espera de una manilla de espera, el procedimiento de invocación se detiene y espera hasta que el subproceso señale que ha terminado.

```
Sub JoinThreads()
    Dim Thread1 As New System.Threading.Thread(AddressOf SomeTask)
    Thread1.Start(Thread1.Join())  ' Wait for the thread to finish.
    MsgBox("Thread is done")
End Sub
```

Estas formas sencillas de controlar los subprocesos, las cuales son útiles cuando administra un número pequeño de subprocesos, son difíciles de utilizar en proyectos más grandes. La próxima sección analiza algunas técnicas avanzadas que puede utilizar para sincronizar subprocesos.

Diapositiva 10



Agrupar subprocesos

Agrupar subprocesos es un mecanismo de subprocesos múltiples en la que las tareas se agregan a una cola de espera y se inician automáticamente a medida que se generan los subprocesos. Con el agrupamiento de subprocesos, puede invocar el método Threadpool.QueueUserWorkItem con un delegado para el procedimiento que desea ejecutar, y Visual Basic .NET genera el subproceso y ejecuta su procedimiento. El siguiente ejemplo muestra la manera en que puede utilizar el agrupamiento de subprocesos para iniciar varias tareas.

```
Sub DoWork()
    Dim TPool As System.Threading.ThreadPool ' Queue a task
    TPool.QueueUserWorkItem(New System.Threading.WaitCallback _
     (AddressOf SomeLongTask)) ' Queue another task
    TPool.QueueUserWorkItem(New System.Threading.WaitCallback _
     (AddressOf AnotherLongTask))
End Sub
```

Agrupar subprocesos es útil cuando desea iniciar varias tareas separadas sin configurar individualmente las propiedades de cada subproceso. Cada subproceso se inicia con un tamaño de pila predeterminado y una prioridad. Por predeterminación, se pueden ejecutar hasta 25 subprocesos en un grupo de subprocesos por procesador del sistema. Los subprocesos adicionales que excedan el límite se pueden colocar en cola de espera, pero no se iniciarán hasta que los otros subprocesos hayan terminado.

Una ventaja de agrupar subprocesos es que puede pasar argumentos en un objeto de estado hacia un procedimiento de tarea. Si el procedimiento que invoca requiere más de un argumento, puede difundir una estructura o una instancia de una clase hacia un tipo de datos Object.

Parámetros del grupo de subprocesos y valores de retorno

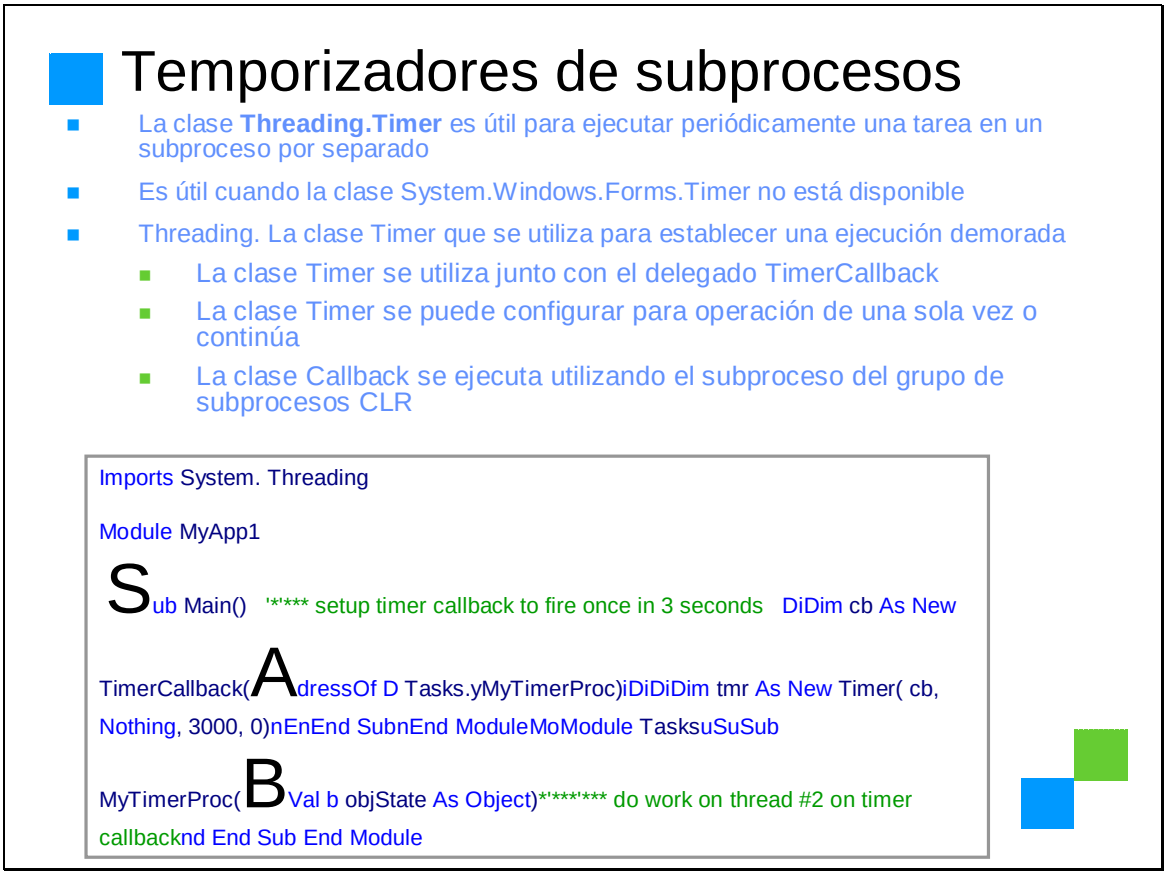
Los valores de retorno de un subproceso en un grupo de subprocesos no son directos. La forma estándar de devolver valores de una invocación de función no se permite debido a que los sub procedimientos son el único tipo de procedimiento que se puede colocar en cola de espera para un grupo de subprocesos. Una manera en que puede proporcionar parámetros y valores de retorno es al envolver los parámetros, los valores de retorno y los métodos en una clase de envoltura. Una forma más fácil de proporcionar parámetros y valores de retorno es utilizando la variable opcional de objeto de estado ByVal del método QueueUserWorkItem. Si utiliza esta variable para pasar una referencia a una instancia de una clase, los miembros de la instancia se pueden modificar por el subproceso del grupo de subprocesos y utilizar como valores de retorno. Al principio pareciera que no se puede modificar un objeto referido por una variable que se pasa por valor. Esto es posible debido a que únicamente la referencia del objeto se pasa por el valor. Cuando realiza cambios al objeto de la referencia, los cambios aplican a la instancia de clase real.

Las estructuras no se pueden utilizar para devolver valores dentro de objetos de estado. Debido a que las estructuras son tipos de valor, los cambios que realiza un proceso asíncrono no cambian los miembros de la estructura original. Utilice las estructuras para proporcionar parámetros cuando no se requieran valores de retorno.

Ejemplo:

```
Friend Class StateObj
    Friend StrArg As String
    Friend IntArg As Integer
    Friend RetVal As String
End Class
Sub ThreadPool Test()
    Dim TPool As System.Threading.ThreadPool
    Dim StObj1 As New StateObj
    Dim StObj2 As New StateObj
    ' Set some fields that act like parameters in the state object.
    StObj1.IntArg = 10
    StObj1.StrArg = "Some string"
    StObj2.IntArg = 100
    StObj2.StrArg = "Some other string"
    ' Queue a task
    TPool.QueueUserWorkItem(New System.Threading.WaitCallback _
     (AddressOf SomeOtherTask), StObj1)
    ' Queue another task
    TPool.QueueUserWorkItem(New System.Threading.WaitCallback _
     (AddressOf AnotherTask), StObj2)
End Sub
Sub SomeOtherTask(ByVal StateObj As Object) ' Use the state object fields as
arguments.
    Dim StObj As StateObj
    StObj = CType(StateObj, StateObj) ' Cast to correct type.
    MsgBox("StrArg contains the string " & StObj.StrArg)
    MsgBox("IntArg contains the number " & CStr(StObj.IntArg))
    ' Use a field as a return value.
    StObj.RetVal = "Return Value from SomeOtherTask"
End Sub
Sub AnotherTask(ByVal StateObj As Object)
    ' Use the state object fields as arguments.
    ' The state object is passed as an Object.
    ' Casting it to its specific type makes it easier to use.
    Dim StObj As StateObj
    StObj = CType(StateObj, StateObj)
    MsgBox("StrArg contains the String " & StObj.StrArg)
    MsgBox("IntArg contains the number " & CStr(StObj.IntArg))
    ' Use a field as a return value.
    StObj.RetVal = "Return Value from AnotherTask"
End Sub
```

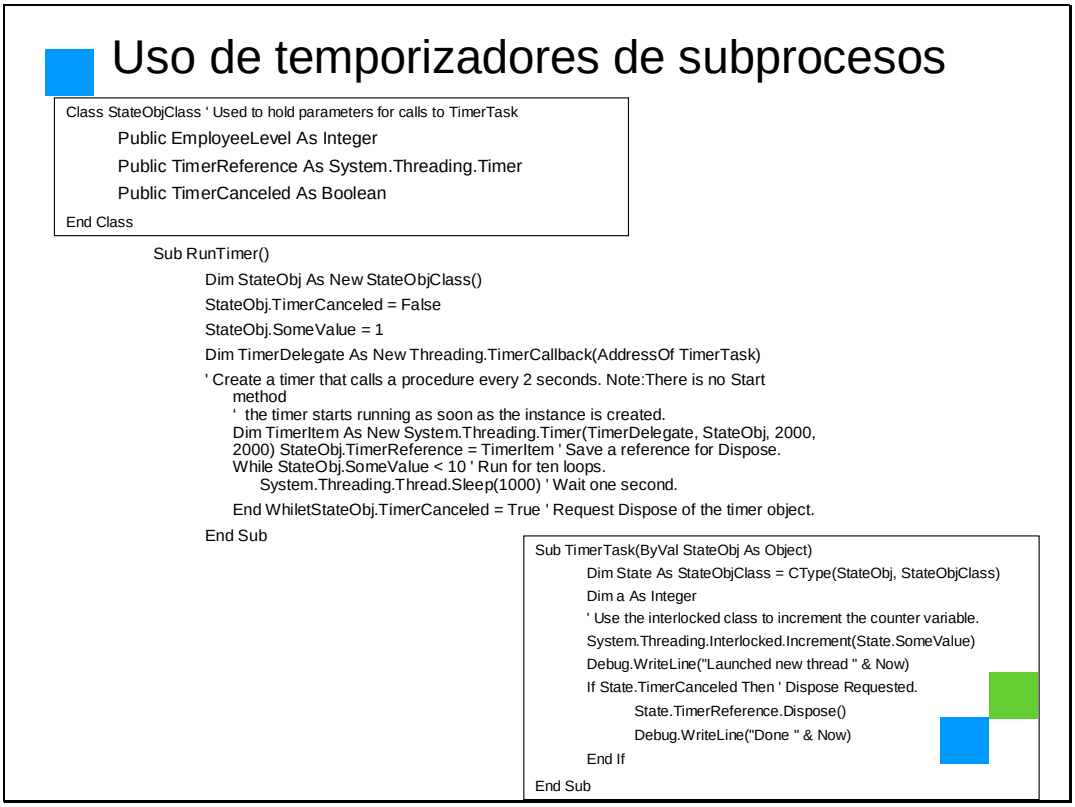
Diapositiva 11



Temporizadores de subprocesos

La clase Threading.Timer es útil para ejecutar periódicamente las tareas en un subproceso separado. Por ejemplo, puede utilizar un temporizador de subproceso para verificar el estado y la integridad de una base de datos o para respaldar archivos críticos.

Diapositiva 12



El siguiente ejemplo inicia una tarea cada dos segundos y utiliza una bandera para iniciar el método **Dispose** que detiene el temporizador.

Este ejemplo coloca un estado en la ventana de salida, de manera que puede hacer esta ventana visible al presionar Control+Alt+O antes de probar su código.

```
Class StateObjClass ' Used to hold parameters for calls to TimerTask
    Public EmployeeLevel As Integer
    Public TimerReference As System.Threading.Timer
    Public TimerCanceled As Boolean
End Class
Sub RunTimer()
    Dim StateObj As New StateObjClass
    StateObj.TimerCanceled = False
    StateObj.SomeValue = 1
    Dim TimerDelegate As New Threading.TimerCallback(AddressOf TimerTask)
    ' Create a timer that calls a procedure every 2 seconds.
Note: There is no Start method; the timer starts running as soon as
    ' the instance is created.
    Dim TimerItem As New System.Threading.Timer(TimerDelegate, StateObj, 2000,
2000)
    StateObj.TimerReference = TimerItem ' Save a reference for Dispose.
    While StateObj.SomeValue < 10 ' Run for ten loops.
        System.Threading.Thread.Sleep(1000) ' Wait one second.
    End While
    StateObj.TimerCanceled = True ' Request Dispose of the timer object.
End Sub
```

```
Sub TimerTask(ByVal StateObj As Object)
   Dim State As StateObjClass = CType(StateObj, StateObjClass)
   Dim x As Integer ' Use the interlocked class to increment the counter variable.
   System.Threading.Interlocked.Increment(State.SomeValue)
   Debug.WriteLine("Launched new thread " & Now)
   If State.TimerCanceled Then ' Dispose Requested.
      State.TimerReference.Dispose()
      Debug.WriteLine("Done " & Now)
   End If
End Sub
```

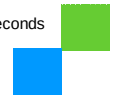
Los temporizadores de subprocesos son particularmente útiles cuando la clase **System.Windows.Forms.Timer** no está disponible, como en los casos en los que desarrolla aplicaciones de consola.

Diapositiva 13

Cancelar tareas

- Debido a los subprocesos múltiples la interfaz de una aplicación continúa respondiendo
- Los eventos de sincronización y los campos que actúan como banderas se utilizan con frecuencia para que otro subproceso sepa que debe detenerse

```
Public CancelThread As New System.Threading.ManualResetEvent(False)
Public ThreadisCanceled As New System.Threading.ManualResetEvent(False)
Private Sub SomeLongTask()
    ...
    While Not CancelThread.WaitOne(0, False) And LoopCount < Loops  'loop for 10
      seconds
        System.Threading.Thread.Sleep(1000) ' Sleep for one second.
        LoopCount += 1
    End While
    If CancelThread.WaitOne(0, False) Then
        'Acknowledge that the ManualResetEvent CancelThread is set.
        ThreadisCanceled.Set()
    ...
End Sub
Public Sub StartTask() ' Starts a new thread.
    Dim th As New System.Threading.Thread(AddressOf SomeLongTask)
    CancelThread.Reset()
    ThreadisCanceled.Reset()
    th.Start()
End Sub
Public Sub CancelTask()
    CancelThread.Set() ' Set CancelThread to ask the thread to stop.
    If ThreadisCanceled.WaitOne(4000, False) Then    ' Wait up to 4 seconds
    ...
End Sub
```



Cancelar tareas

Una ventaja de los subprocesos múltiples es que la interfaz que es parte de una aplicación continúa respondiendo, incluso cuando las tareas se realizan en otros subprocesos.

Los eventos y campos de sincronización que actúan como banderas con frecuencia se utilizan para notificar a otro subprocesos que desea que se detengan.

El siguiente ejemplo utiliza los eventos de sincronización para cancelar una tarea.

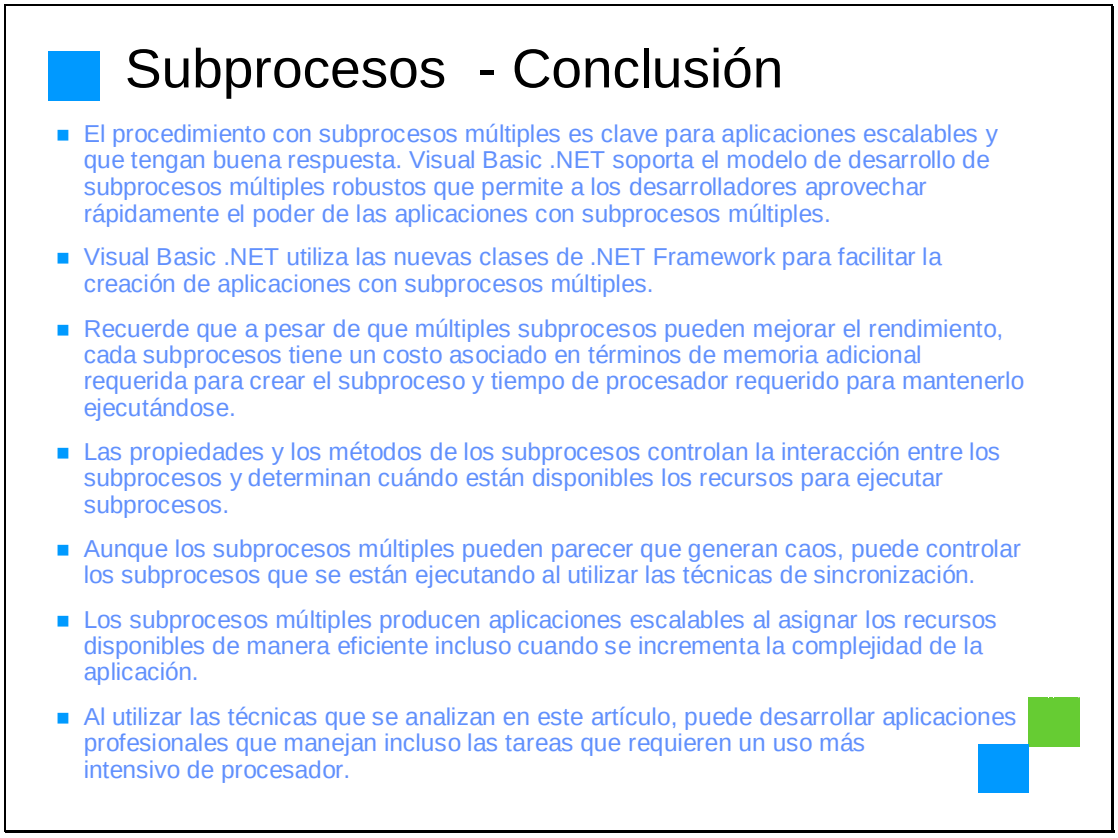
Para utilizar este ejemplo, agregue el siguiente módulo a un proyecto.

Para iniciar un subproceso, invoque el método StartCancel.StartTask().

Para cancelar uno o más subprocesos que se están ejecutando, invoque el método StartCancel.CancelTask().

```
Module StartCancel
    Public CancelThread As New System.Threading.ManualResetEvent(False)
    Public ThreadisCanceled As New System.Threading.ManualResetEvent(False)
    Private Sub SomeLongTask()
        Dim LoopCount As Integer
        Dim Loops As Integer = 10
        ' Run code in a While loop until 10 seconds passes, or
        ' until CancelThread is set.
        While Not CancelThread.WaitOne(0, False) And LoopCount < Loops
            ' Do some kind of task here.
            System.Threading.Thread.Sleep(1000) ' Sleep for one second.
            LoopCount += 1
        End While
        If CancelThread.WaitOne(0, False) Then
            'Acknowledge that the ManualResetEvent CancelThread is set.
            ThreadisCanceled.Set()
            MsgBox("Canceling thread")
        Else
            MsgBox("Thread is done")
        End If
    End Sub
    Public Sub StartTask() ' Starts a new thread.
        Dim th As New System.Threading.Thread(AddressOf SomeLongTask)
        CancelThread.Reset()
        ThreadisCanceled.Reset()
        th.Start()
        MsgBox("Thread Started")
    End Sub
    Public Sub CancelTask()
        ' Stops any threads started by the StartTask procedure.
        ' Notice that this thread both receives and sends ' synchronization events to
coordiante the threads. CancelThread.Set() ' Set CancelThread to ask the thread to
stop.
        If ThreadisCanceled.WaitOne(4000, False) Then
            ' Wait up to 4 seconds for the thread to
            ' acknowledge that it has stopped.
            MsgBox("The thread has stopped.")
        Else
            MsgBox("The thread could not be stopped.")
        End If
    End Sub
End Module
```

Diapositiva 14



- El procedimiento con subprocesos múltiples es clave para aplicaciones escalables y que tengan buena respuesta. Visual Basic .NET soporta el modelo de desarrollo de subprocesos múltiples robustos que permite a los desarrolladores aprovechar rápidamente el poder de las aplicaciones con subprocesos múltiples.
- Visual Basic .NET utiliza las nuevas clases de .NET Framework para facilitar la creación de aplicaciones con subprocesos múltiples.
- Recuerde que a pesar de que múltiples subprocesos pueden mejorar el rendimiento, cada subprocesos tiene un costo asociado en términos de memoria adicional requerida para crear el subproceso y tiempo de procesador requerido para mantenerlo ejecutándose.
- Las propiedades y los métodos de los subprocesos controlan la interacción entre los subprocesos y determinan cuándo están disponibles los recursos para ejecutar subprocesos.
- Aunque los subprocesos múltiples pueden parecer que generan caos, puede controlar los subprocesos que se están ejecutando al utilizar las técnicas de sincronización.

- Los subprocesos múltiples producen aplicaciones escalables al asignar los recursos disponibles de manera eficiente incluso cuando se incrementa la complejidad de la aplicación.
- Al utilizar las técnicas que se analizan en este artículo, puede desarrollar aplicaciones profesionales que manejan incluso las tareas que requieren un uso más intensivo de procesador.

Recursos adicionales

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconasynchronousexecution.asp

Explica el concepto de la programación asíncrona y su soporte en .NET Framework.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconasynchronousprogrammingdesignpattern

Demuestra las prácticas de programación asíncrona mediante ejemplos.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconremotingexampleasynchronousremoting.asp

Incluye un ejemplo de la capacidad remota asíncrona

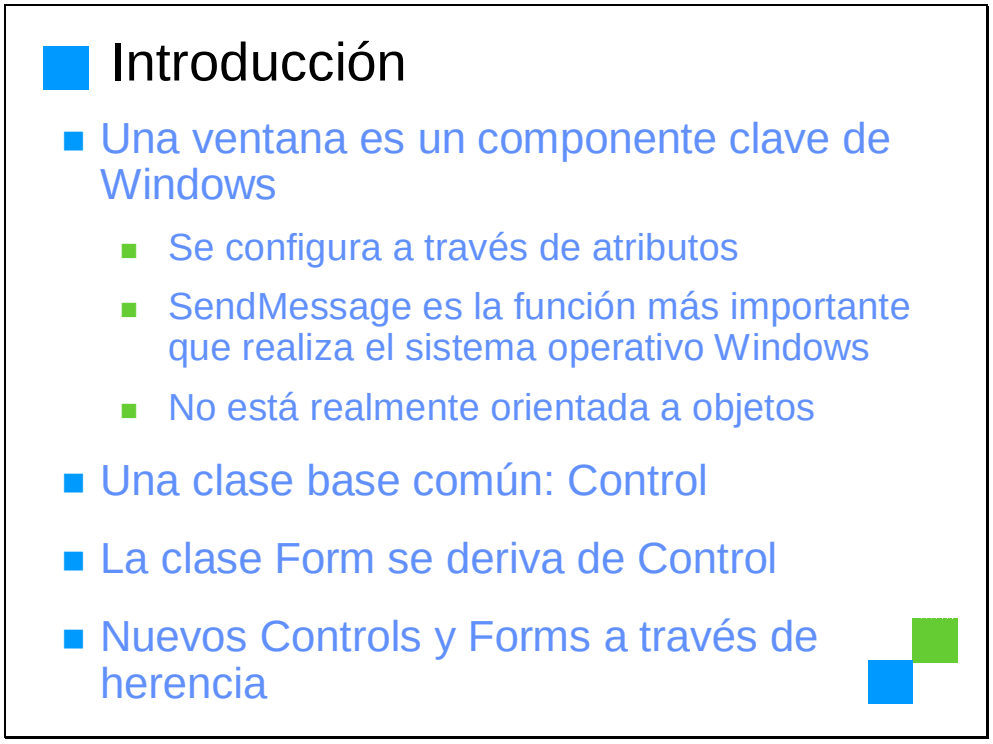
Diapositiva 15

Parte 2/2

Windows Forms

- Crear formularios
  - Controles
  - Menús
  - Barra de estado
- Cuadros de diálogo comunes
- Formularios de figura irregular
  - Figuras
  - Gráficos

Diapositiva 16



Windows Forms es la nueva plataforma para el desarrollo de aplicaciones Microsoft Windows basada en .NET Framework. Este marco proporciona un claro conjunto de clases, orientado a los objetos y ampliable que le permite desarrollar aplicaciones Windows enriquecidas. Además, Windows Forms pueden actuar como la interfaz local en una solución distribuida de multiniveles.

Una ventana es un componente clave de Windows

Una ventana en una Windows Application se representa mediante un objeto Form. A nivel de sistema operativo, las ventanas se comunican utilizando mensajes. Estos mensajes se envían usando el método SendMessag. Este método no es parte de una Windows Application debido a que reside en un componente no administrado que es parte de los componentes del sistema operativo Windows y usted no necesita preocuparse por esta comunicación, excepto que necesite enviar mensajes personalizados entre ventanas.

A nivel de aplicación, usted trabaja únicamente en un ambiente orientado a objetos, instancia objetos Form y trabaja con sus métodos y propiedades, además de que agrega la lógica a su aplicación.

Nuevos Controls y Forms a través de herencia

Un objeto Form y los controles que contiene se heredan de sus clases correspondientes. Un formulario se hereda de System.Windows.Forms.Form y los controles de System.Windows.Forms.Control, aunque cada control tiene su propia

clase que se deriva de System.Windows.Forms.Control, por ejemplo un control TextBox se deriva de la clase System.Windows.Forms.TextBox.

Ejemplo. Crear un objeto formulario.

```
Public Class Form2
   Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

   Public Sub New()
      MyBase.New()

      'El Diseñador de Windows Forms requiere esta llamada.
      InitializeComponent()

      'Agregar cualquier inicialización después de la llamada a InitializeComponent()

   End Sub
```

Ejemplo. Crear un control sobre el formulario.

Este formulario contiene dos controles: un control TextBox y un control Button.

```
…
Public Class Form2
   Inherits System.Windows.Forms.Form

#Region " Código generado por el Diseñador de Windows Forms "

   Public Sub New()
      MyBase.New()

      'El Diseñador de Windows Forms requiere esta llamada.
      InitializeComponent()

      'Agregar cualquier inicialización después de la llamada a InitializeComponent()

   End Sub

   'Form reemplaza a Dispose para limpiar la lista de componentes.
   Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
      If disposing Then
         If Not (components Is Nothing) Then
            components.Dispose()
         End If
      End If
      MyBase.Dispose(disposing)
   End Sub

   'Requerido por el Diseñador de Windows Forms
   Private components As System.ComponentModel.IContainer

   'NOTA: el Diseñador de Windows Forms requiere el siguiente procedimiento
   'Puede modificarse utilizando el Diseñador de Windows Forms.
   'No lo modifique con el editor de código.
   Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
```

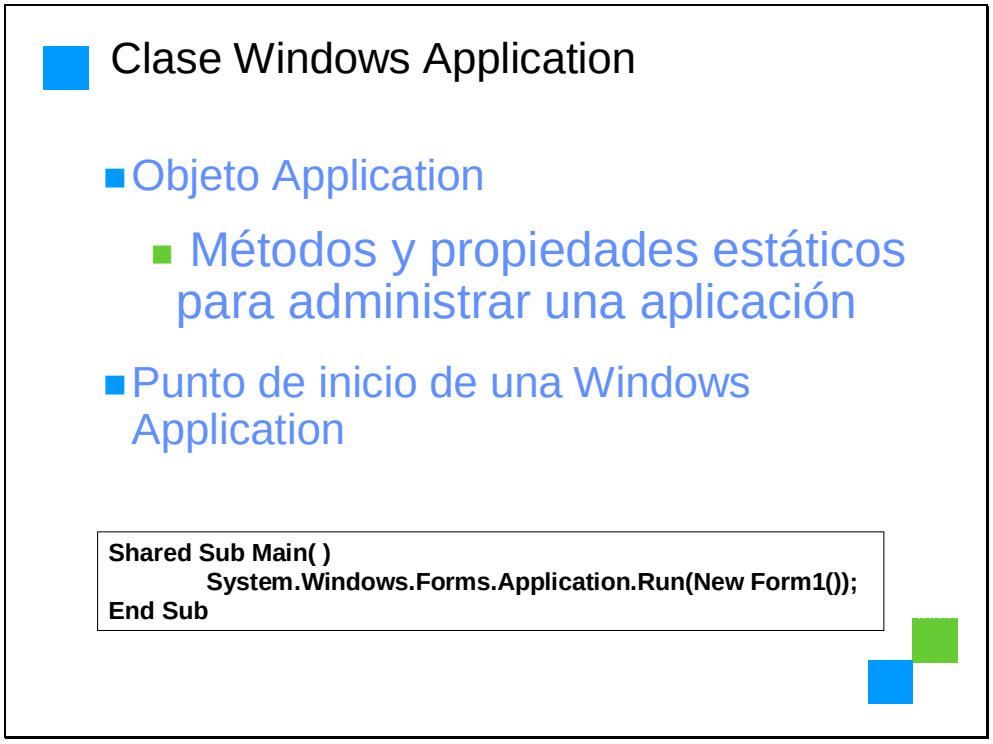
```
    Friend WithEvents Button1 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.TextBox1 = New System.Windows.Forms.TextBox
        Me.Button1 = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'TextBox1
        '
        Me.TextBox1.Location = New System.Drawing.Point(208, 48)
        Me.TextBox1.Name = "TextBox1"
        Me.TextBox1.TabIndex = 0
        Me.TextBox1.Text = "TextBox1"
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(216, 128)
        Me.Button1.Name = "Button1"
        Me.Button1.TabIndex = 1
        Me.Button1.Text = "Button1"
        '
        'Form2
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.Add(Me.Button1)
        Me.Controls.Add(Me.TextBox1)
        Me.Name = "Form2"
        Me.Text = "Form2"
        Me.ResumeLayout(False)

    End Sub

#End Region

End Class
```

Diapositiva 17



Objeto Application

La clase Application tiene métodos para iniciar y detener aplicaciones y subprogramas, y para procesar mensajes Windows. Invoque Run para iniciar el circuito de mensaje de la aplicación en el subproceso actual y, opcionalmente, hacer un formulario visible. Invoque Exit o ExitThread para detener el circuito de un mensaje. Invoque DoEvents para procesar mensajes mientras que su programa está en circuito.

Ejemplo

El método Main invoca Run para iniciar la aplicación, la cual crea el formulario, listBox1 y button1.

Cuando el usuario hace clic en button1, el método button1_Click agrega los números uno a tres al cuadro de lista y muestra un MessageBox.

Si el usuario hace clic en No en MessageBox, el método button1_Click agrega otro número a la lista.

Si el usuario hace clic en Sí, la aplicación invoca Exit para procesar todos los mensajes restantes en la cola de espera y después para salir.

El ejemplo supone que listBox1 y button1 han sido instanciados y colocados en un formulario.

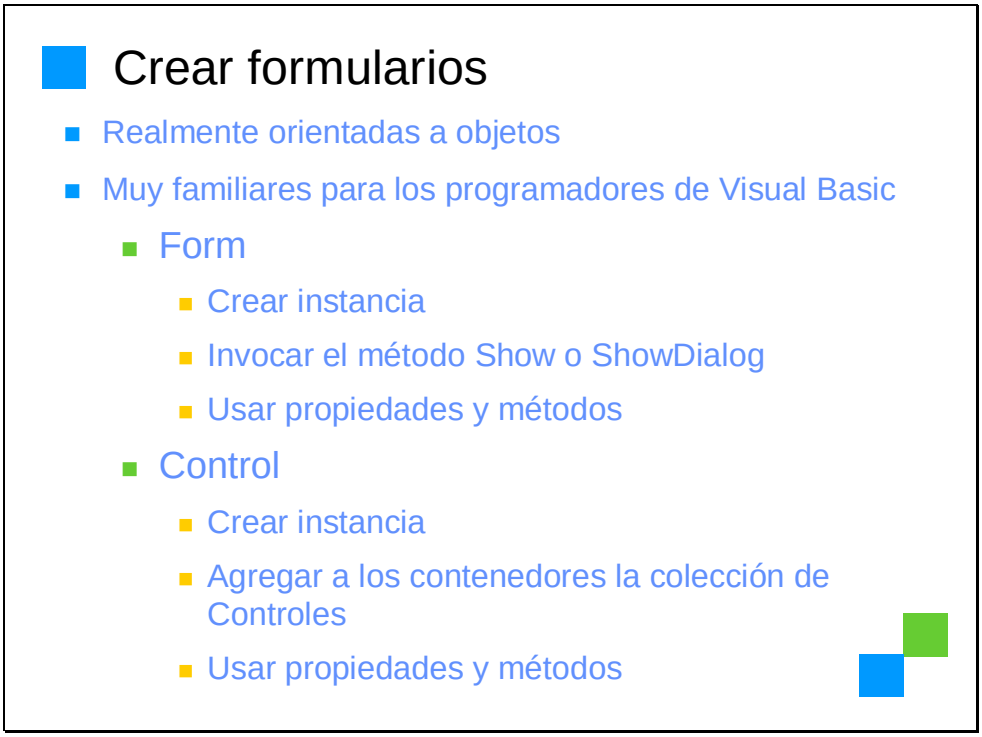
```
    Public Shared Sub Main()
        'Starts the application.
        System.Windows.Forms.Application.Run(New Form1)
    End Sub

    Protected Sub button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
        'Populates a list box with three numbers
        Dim I As Integer = 3
        Dim j As Integer
        For j = 1 To i
            listBox1.Items.Add(j)
        Next J

        'Checks to see whether the user wants to exit the application.
        'If not, adds another number to the list box.
        While (MessageBox.Show("Exit application?", "", MessageBoxButtons.YesNo) =
DialogResult.No)
            'Increment the counter and add the number to the list box.
            i += 1
       listBox1.Items.Add(i);
        End While

        'The user wants to exit the application. Close everything down.
        Application.Exit()
    End Sub
```

Diapositiva 18



¿Qué es una Form?

Un formulario es un espacio de la propiedad en la pantalla, generalmente rectangular, que puede usar para presentar información al usuario o aceptar información de entrada del usuario. Los formularios pueden ser ventanas estándar, ventanas de interfaz múltiple de documento (MDI), cuadros de diálogo o superficies de presentación para rutinas gráficas. Además, los formularios son controles, debido a que se heredan desde la clase Control.

Crear una instancia

Cada formulario en su aplicación es una clase y usted sólo necesita instanciar la clase form requerida y utilizarla.

```
MyForm = new FormClient()
MyForm.ClientId = 23 ‘Public property.
MyForm.Text = "Hello“ ‘Change the title.
MyForm.Refresh() ‘Redraw itself.
```

Mostrar un formulario

Los objetos Form contienen dos método a mostrar: Show y ShowDialog.

Show: muestra un formulario al usuario. Mostrar el control es equivalente a configurar la propiedad Visible en true. Después de invocar el método Show, la propiedad Visible devuelve un valor true hasta que se invoca el método Hide.

```
MyForm = new FormClient()
MyForm.ClientId = 23 ‘Public property.
MyForm.Text = "Hello“ ‘Change the title.
MyForm.Show();
```

ShowDialog: muestra un formulario como un cuadro de diálogo modal. Cuando se muestra un formulario de manera modal, no puede ocurrir ningún ingreso de información (ya sea utilizando el teclado o haciendo clic con el *mouse*) con la excepción de los objetos del formulario modal. El programa debe ocultar o cerrar el formulario modal (con frecuencia en respuesta a alguna acción del usuario) antes de que pueda ocurrir el ingreso a algún otro formulario. Los formularios que aparecen de manera modal generalmente se utilizan como cuadros de diálogo en una aplicación.

```
Public Sub ShowMyDialogBox()
    MyForm = New FormClient
    MyForm.ClientId = 23 'Public property.
    MyForm.Text = "Hello" 'Change the title.
    MyForm.Show()
    'Show FClient as a modal dialog and determine if DialogResult = OK.
    If (MyForm.ShowDialog(this) = DialogResult.OK) Then
        'Do something
    Else
        ' Cancel. Do something
    End If
    MyForm.Dispose()
End Sub
```

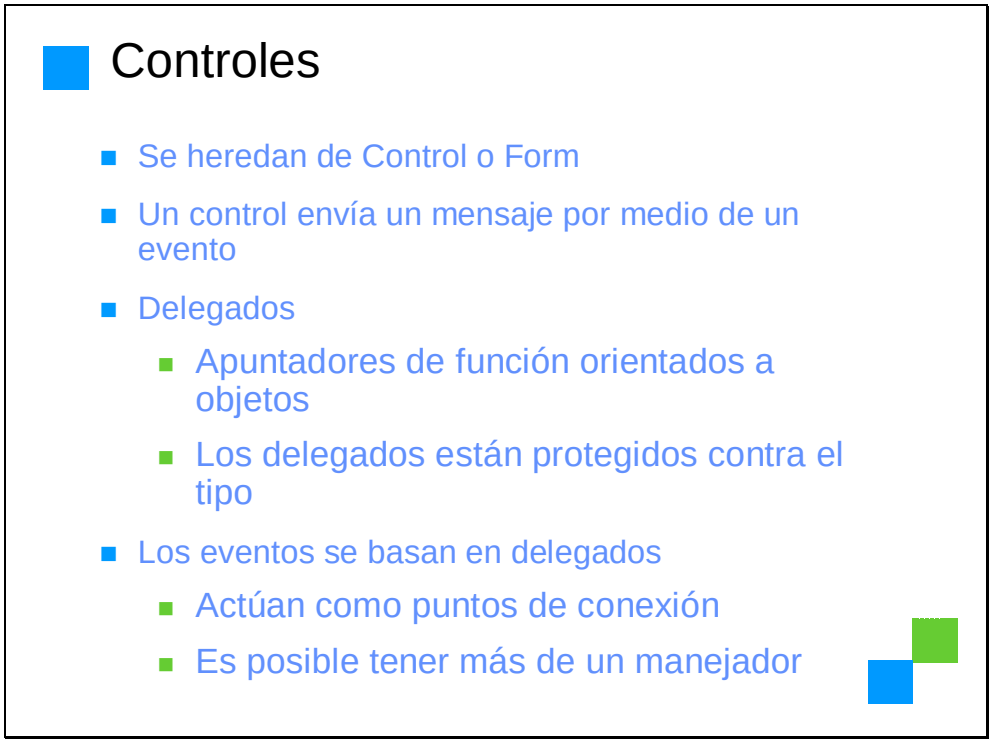
Propiedad DialogResult

El resultado de un diálogo de un formulario es el valor que se devuelve desde el formulario cuando aparece como un cuadro de diálogo modal. Si el formulario aparece como un cuadro de diálogo, al configurar esta propiedad con un valor desde la enumeración System.Windows.Forms.DialogResult (Abort, Cancel, Ignore, No, None, Ok, Retry, Yes) establece el valor del resultado del cuadro de diálogo para el formulario, oculta el cuadro de diálogo modal y devuelve el control al formulario que se invoca.

Esta propiedad por lo general se configura a través de la propiedad DialogResult del control Button en el formulario. Cuando el usuario hace clic en el control Button, el valor asignado a la propiedad DialogResult de Button se asigna a la propiedad DialgoResult del formulario.

Cuando aparece un formulario como un cuadro de diálogo modal, hacer clic en el botón Close (el botón con una X en la esquina superior derecha del formulario) hace que el formulario se oculte y la propiedad DialogResult se configure a DialogResult.Cancel.

Diapositiva 19



Clase Control

Define la clase base para los controles, que son componentes con una representación visual. La clase Control implementa una funcionalidad muy básica requerida por las clases que presentan información al usuario. Maneja el ingreso de información del usuario a través del teclado y de dispositivos de señalamiento. Maneja el enrutamiento y la seguridad de los mensajes. Define los límites de un control (su posición y tamaño), aunque no implementa su dibujo. Proporciona una manija – *handler* - de ventana (hWnd). Los controles Windows Forms utilizan propiedades de ambiente de manera que los controles hijos puedan parecer similares al ambiente que los rodea. Una propiedad de ambiente es una propiedad de control que, si no se configura, se recupera del control padre.

Declaración de evento dentro de una Form

El siguiente ejemplo muestra la manera en que se agrega un manejador de evento a un control Button en un Form.

Cuando se define el control en el constructor de la clase Form, debe asignar el manejador.

```
…
        Me.Button1 = New System.Windows.Forms.Button
        Me.Button1.Location = New System.Drawing.Point(136, 16)
        Me.Button1.Name = "button1"
        Me.Button1.TabIndex = 1
        Me.Button1.Text = "button1"
        AddHandler Me.Button1.Click, AddressOf Me.button1_Click
…
```

La clase Form debe contar con el método button1_Click.

```
    Private Sub button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        // Add code to handle the event.
    End Sub
```

El manejador de evento recibe un argumento del tipo EventArgs que contiene los datos relacionados con este evento.

Delegados

Los delegados son el fundamento de los eventos en .NET Framework.

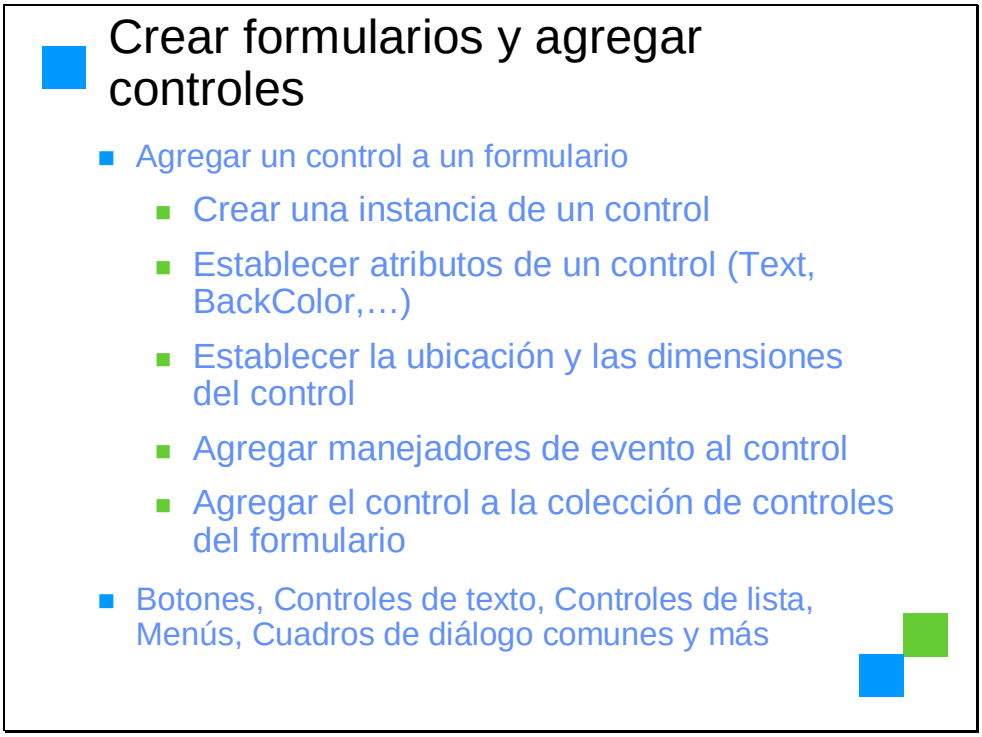
En su forma más sencilla un delegado apunta a una función que se invoca indirectamente a través de su referencia.

En .NET, se han diseñado exclusivamente para manejar eventos.

Ejemplo:

```
Public Class Investor
    Public Delegate Sub Transact(ByVal Quantity As Integer, ByVal StockSymbol As
String)
    Event Buy As Transact
    Event Sell As Transact
End Class
Public Class Broker
    Public WithEvents Client As Investor
    Sub New()
        Client = New Investor
    End Sub
    Private Sub HandleBuy(ByVal Quantity As Integer, ByVal StockSymbol As String)
Handles Client.Buy
        'code
    End Sub
    Private Sub HandleSell(ByVal Quantity As Integer, ByVal StockSymbol As String)
Handles Client.Sell
        'Code
    End Sub
End Class
```

Diapositiva 20



El Diseñador de Windows Forms de Visual Studio .NET proporciona una solución de desarrollo rápida para crear aplicaciones Windows. El primer paso en el diseño de un formulario es establecer sus propiedades. Los menús, cuadros de diálogo, barras de estado y barras de herramientas son los elementos que le permiten exponer la funcionalidad a sus usuarios o alertarlos para información importante en su aplicación. La mayoría de los formularios se diseñan al agregar controles a su superficie, para definir la interfaz.

Un control es un componente en un formulario utilizado para mostrar la información o aceptar el ingreso de datos del usuario.

Para arrastrar un control a un formulario

1. Abra el formulario.
2. En la caja de herramientas haga clic en el control que desea y arrástrelo hasta su formulario. El control se agrega al formulario en la ubicación especificada en su tamaño predeterminado. Puede hacer doble clic en un control en la caja de herramientas para agregarlo en la esquina superior izquierda del formulario en su tamaño predeterminado.

Agregar controles de manera dinámica

Puede agregar controles de manera dinámica a un formulario durante el tiempo de ejecución.

En el siguiente ejemplo, se agregará un control TextBox al formulario cuando se haga clic en el control Button.

El siguiente procedimiento supone la existencia de un formulario con un control Button, Button1, ya instalado.

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    myText = New TextBox
    myText.Location = New Point(25, 25)
    myText.Text = "Hello World!!!"
    this.Controls.Add(myText)  'Add the control to the Controls collection of the form.
End Sub
```

Diapositiva 21

Hay más controles que se pueden incluir en un Form.

Esta no es una lista completa. También, todos estos controles tienen propiedades compartidas de la clase de control y propiedades protegidas de la clase derivada. Refiérase a la documentación en el sitio Web de Visual Studio o MSDN para obtener más información sobre estas propiedades, métodos o eventos.

Sólo abordaremos algunos temas. A continuación encontrará un vínculo para obtener más información sobre estas clases: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/winxpicons.asp.

Edición de texto

TextBox - Muestra texto ingresado durante el diseño que pueden editar los usuarios durante el tiempo de ejecución, o lo pueden cambiar de manera programática.

Ejemplo:

```
Private Sub CreateMyMultilineTextBox()
    ' Create an instance of a TextBox control.
    Dim textBox1 As New TextBox
    ' Set the Multiline property to true.
    textBox1.Multiline = True
    ' Add vertical scroll bars to the TextBox
    control.textBox1.ScrollBars = ScrollBars.Vertical
    ' Allow the RETURN key to be entered in the TextBox
    control.textBox1.AcceptsReturn = True
    ' Allow the TAB key to be entered in the TextBox
    control.textBox1.AcceptsTab = True
    ' Set WordWrap to True to allow text to wrap to the next line.
    textBox1.WordWrap = True
    ' Set the default text of the control.
    textBox1.Text = "Welcome!"
End Sub
```

RichTextBox - Permite que el texto aparezca con formato en texto plano o de texto enriquecido (RTF).

Texto mostrado (sólo lectura)

Label - Muestra texto que los usuarios no pueden editar directamente.

LinklLabel - Muestra texto con un vínculo estilo Web e inicia un evento cuando el usuario hace clic en un texto especial. Por lo general el texto es un vínculo a otra ventana o sitio Web.

StatusBar - Muestra la información acerca del estado actual de la aplicación utilizando una ventana enmarcada, por lo general aparece en la parte inferior del formulario padre.

Ejemplo de uso de StatusBar

Con frecuencia, un programa le pedirá que actualice los contenidos de los paneles de la barra de estado dinámicamente durante el tiempo de ejecución, con base en los cambios al estado de la aplicación u otra interacción del usuario.

Esta es una forma común para indicar a los usuarios que teclas tales como CAPS LOCK, NUM LOCK, o SCROLL LOCK están habilitadas, o para proporcionar la fecha o un reloj como una referencia conveniente.

El área programable dentro del control StatusBar consiste en instancias de la clase StatusBarPanel.

Éstas se agregan durante el periodo del diseño a través del Editor de colección de StatusBarPanel, y durante el tiempo de ejecución a través de la clase StatusBarPanelCollection.

Este método agrega un panel a StatusBar.

```
Public Sub CreateStatusBarPanels()
    ' Create panels and set text property.
    statusBar1.Panels.Add("One")
    statusBar1.Panels.Add("Two")
    statusBar1.Panels.Add("Three")
End Sub
'This method updates the first panel of the status bar with the current date.
Private Sub UpdateStatusBar()
    statusBar1.Panels(0).Text = DateTime.Now.ToShortTimeString()
    statusBar1.Panels(1).Text = "Hello"
    statusBar1.Panels(2).Text = "World"
End Sub
```

Selección de una lista

CheckedListBox - Muestra una lista desplazable de elementos, cada uno acompañado de un cuadro de verificación.

DomainUpDown - Muestra una lista desplegable de elementos, a través de la cual se pueden desplazar los usuarios con los botones hacia arriba y hacia abajo.

ListBox - Muestra una lista de elementos de texto y gráficos (iconos).

Ejemplo:

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ' Create an instance of the ListBox.
    Dim listBox1 As New ListBox   ' Set the size and location of the ListBox.
    listBox1.Size = New System.Drawing.Size(200, 100)
    listBox1.Location = New System.Drawing.Point(10, 10) ' Add the ListBox to the
form.
    Me.Controls.Add(listBox1)
    ' Set the ListBox to display items in multiple columns.
    listBox1.MultiColumn = True
    ' Set the selection mode to multiple and extended.
    listBox1.SelectionMode = SelectionMode.MultiExtended ' Shutdown the painting
of the ListBox as items are added. listBox1.BeginUpdate()
    ' Loop through and add 50 items to the ListBox.
    Dim a As Integer
    For x = 1 To 50
        listBox1.Items.Add("Item " & x.ToString())
    Next x
    ' Allow the ListBox to repaint and display the new items.
    listBox1.EndUpdate()
    ' Select three items from the
    ListBox.listBox1.SetSelected(1, True)
    listBox1.SetSelected(3, True)
    listBox1.SetSelected(5, True)
    ' Display the second selected item in the ListBox to the console.
System.Diagnostics.Debug.WriteLine(listBox1.SelectedItems(1).ToString())
    ' Display the index of the first selected item in the ListBox.
System.Diagnostics.Debug.WriteLine(listBox1.SelectedIndices(0).ToString())
End Sub
```

ListView - Muestra los elementos en una de cuatro formas diferentes. Las vistas incluyen sólo texto, texto con iconos pequeños, texto con iconos grandes y una vista de reporte. (Igual que Windows Explorer)

NumericUpDown - Muestra una lista de números, los usuarios se pueden desplazar con los botones hacia arriba y hacia abajo.

TreeView - Muestra una colección jerárquica de objetos de nodos que puede consistir en texto con cuadros de verificación o iconos opcionales. (Igual que Windows Explorer)

Controles del menú

MainMenu - Proporciona una interfaz durante el tiempo de diseño para crear menús.

ContextMenu - Implementa un menú que aparece cuando el usuario hace clic con el botón alterno (botón derecho) en un objeto.

Ejemplo

- Este método crea un menú principal y lo agrega al objeto form.

```
Public Sub CreateMyMainMenu()
    'Create an empty MainMenu derived of class System.Windows.Forms.MainMenu.
    Dim mainMenu1 As New MainMenu
    Dim menuItem1 As New MenuItem
    Dim menuItem2 As New MenuItem
    menuItem1.Text = "File"
    menuItem2.Text = "Edit"
    'Add two MenuItem objects to the MainMenu.
    mainMenu1.MenuItems.Add(menuItem1)
    mainMenu1.MenuItems.Add(menuItem2)

    'Bind the MainMenu to Form1.
  // The form object have a property to get or set the MainMenu that is displayed in the form.
    Me.Menu = mainMenu1
End Sub
```

- Este método crea un menú de contexto y lo agrega al objeto form.

```
Public Sub AddContextMenuAndItems()
    Dim mnuContextMenu As New ContextMenu

    'Add ContextMenu to the Form
    'The form object have a property to get or set the ContextMenu that is displayed in the form.
    this.ContextMenu = mnuContextMenu
    Dim mnuItemNew As New MenuItem
    Dim mnuItemOpen As New MenuItem

    mnuItemNew.Text = "&New"
    mnuItemOpen.Text = "&Open"
    mnuContextMenu.MenuItems.Add(mnuItemNew)
    mnuContextMenu.MenuItems.Add(mnuItemOpen)
    Dim mnuItemOpenWith As New MenuItem
    mnuItemOpenWith.Text = "Open &With…"
    mnuItemOpen.MenuItems.Add(mnuItemOpenWith)
End Sub
```

Comandos

Button - Se utiliza para iniciar, detener o interrumpir un proceso.

Ejemplo:

```
Private Sub InitializeMyButton()
    ' Create and initialize a Button.
    Dim button1 As New Button
    ' Set the button to return a value of OK when clicked.
    button1.DialogResult = DialogResult.OK
    ' Add the button to the form.
    Controls.Add(button1)
End Sub 'InitializeMyButton
```

LinklLabel - Muestra texto con un vínculo estilo Web e inicia un evento cuando el usuario hace clic. Por lo general el texto es un vínculo a otra ventana o sitio Web.

NotifyIcon - Muestra un icono en el área de notificación del estado de la barra de tareas que representa una aplicación que se ejecuta en el fondo.

ToolBar - Contiene una colección de controles de botón.

Establecimiento de valores

CheckBox - Muestra un cuadro de verificación y una etiqueta para texto. Por lo general se usa para configurar opciones.

CheckedListBox - Muestra una lista desplazable de elementos, cada uno acompañado de un cuadro de verificación.

RadioButton - Muestra un botón que se puede activar o desactivar.

Trackbar - Permite que los usuarios establezcan valores en una escala al mover un "indicador" en la escala. *(por ejemplo el control de volumen)*

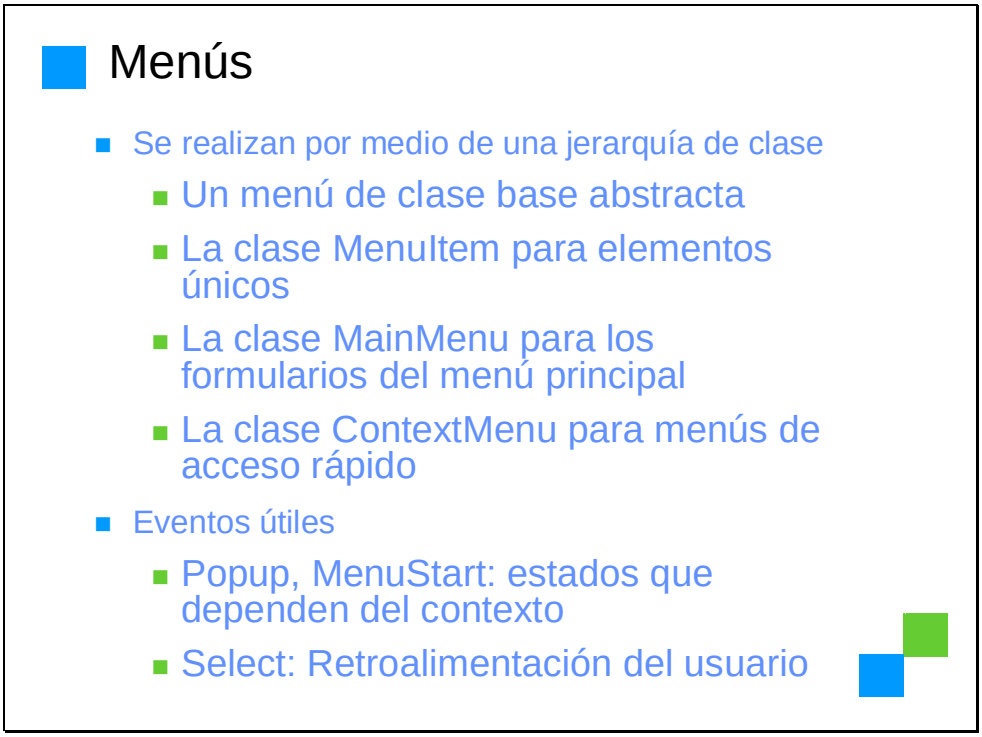
DataGrid - El control DataGrid de Windows Forms proporciona una interfaz para conjuntos de datos de ADO.NET, muestra datos tabulares y permite actualizaciones a la fuente de datos. Cuando el control DataGrid se configura con una fuente de datos valida, se llena automáticamente creando columnas y filas con base en la forma de los datos. El control DataGrid se puede utilizar para mostrar ya sea una tabla única o relaciones jerárquicas entre un conjunto de tablas.

Ejemplo:

```
Public Class DataGridSample
   Inherits Form
   Private ds As DataSet
   Private myGrid As DataGrid
   Shared Sub Main()
      Application.Run(New DataGridSample)
   End Sub
   Public Sub New()
      InitializeComponent()
   End Sub
   Public Sub InitializeComponent()
      Me.ClientSize = New Size(550, 450)
      myGrid = New DataGrid
      myGrid.Location = New Point(10, 10)
      myGrid.Size = New Size(500, 400)
      myGrid.CaptionText = "Microsoft .NET DataGrid"
      Me.Controls.Add(myGrid)
      Me.Text = "Visual Basic Grid Example"
      ConnectToData()
      myGrid.SetDataBinding(ds, "Suppliers")
   End Sub
   Private Sub ConnectToData()
      ' Create the ConnectionString and create a SqlConnection.
      ' Change the data source value to the name of your computer.
      Dim cString As String = "Data Source=localhost;Integrated Security=SSPI;Initial
Catalog=northwind"
      Dim cnNorthwind As SqlConnection = New SqlConnection(cString)
      ' Create a SqlDataAdapter for the Suppliers table.
      Dim adpSuppliers As SqlDataAdapter = New SqlDataAdapter
      ' A table mapping tells the adapter what to call the table.
      adpSuppliers.TableMappings.Add("Table", "Suppliers")
      cnNorthwind.Open()
      Dim cmdSuppliers As SqlCommand = _
      New SqlCommand("SELECT * FROM Suppliers", cnNorthwind)
      cmdSuppliers.CommandType = CommandType.Text
      adpSuppliers.SelectCommand = cmdSuppliers
      Console.WriteLine("The connection is open.")
      ds = New DataSet("Customers")
      adpSuppliers.Fill(ds)
      ' Create a second SqlDataAdapter and SqlCommand to get
      ' the Products table, a child table of Suppliers.
      Dim adpProducts As SqlDataAdapter = New SqlDataAdapter
      adpProducts.TableMappings.Add("Table", "Products")
      Dim cmdProducts As SqlCommand = _
      New SqlCommand("SELECT * FROM Products", cnNorthwind)
```

```
        adpProducts.SelectCommand = cmdProducts
        adpProducts.Fill(ds)
        cnNorthwind.Close()
        Console.WriteLine("The connection is closed.")
        ' You must create a DataRelation to link the two tables.
        Dim dr As DataRelation
        Dim dc1 As DataColumn
        Dim dc2 As DataColumn
        ' Get the parent and child columns of the two tables.
        dc1 = ds.Tables("Suppliers").Columns("SupplierID")
        dc2 = ds.Tables("Products").Columns("SupplierID")
        dr = New System.Data.DataRelation("suppliers2products", dc1, dc2)
        ds.Relations.Add(dr)
    End Sub
End Class
```

Diapositiva 22



Menús

Los menús se realizan por una jerarquía de clases. El menú es la clase de base abstracta para las clases MainMenu, MenuItem y ContextMenu.

MenuItem

La clase más importante para los programadores es MenuItem. MenuItem actúa como un solo elemento de menú, pero puede contener sub-elementos. Un sub-elemento se representa también mediante MenuItem.

MenuItem tiene un conjunto de constructores sobrecargado. La variante más sencilla requiere una cadena de captura. Pero en la mayoría de los casos utilizará un constructor con un manejador de eventos como segundo argumento.

```
Dim mi As New MenuItem("&File")
mi.MenuItems.Add(New MenuItem("&New", New EventHandler(this.DoNew)))
mi.MenuItems.Add(New MenuItem("&Save", New EventHandler(this.DoSave)))
```

Puede asignar un manejador de evento después de la creación de un evento OnClick. Otros constructores ofrecen otros argumentos, como un acceso rápido o un arreglo de sub-elementos.

El evento Popup le permite llevar a cabo tareas antes de presentar el menú, como deshabilitar la opción Copy si no se tiene algo seleccionado. El evento Select le permite realizar tareas tales como proporcionar una ayuda detallada para sus

elementos de menú de la aplicación cuando el usuario coloca el cursor del *mouse* sobre un elemento del menú.
Más detalles acerca de MenuItem están disponibles a través de sus propiedades.

Checked - Obtiene o establece un valor que indica si aparece una marca junto al texto en un elemento de menú.
Enabled - Obtiene o establece un valor indicando si el elemento del menú está habilitado.
MenuID - Establece un valor indicando el identificador de Windows para este elemento del menú.
Mnemonic - Obtiene un valor que indica el carácter mnemónico asociado con este elemento del menú.
ShortCut - Obtiene o establece un valor que indica la tecla de acceso rápido asociado con ese elemento del menú.
ShowShortcut - Obtiene o establece un valor que indica si la tecla de acceso rápido que está asociada con el elemento del menú se debe mostrar junto al nombre del elemento del menú.
RadioCheck - Obtiene o establece un valor que indica si el MenuItem, de estar marcado, muestra un botón de radio (•) en lugar de una marca (✓).
MdiList - Obtiene o establece un valor que indica si el elemento del menú se llenará con una lista del hijo de la Interfaz de documentos múltiples (MDI) que aparece dentro del formulario asociada.

MainMenu

Para asignar un menú a un formulario, deberá crear un objeto MainMenu. Puede asignar este objeto a la propiedad Menu de un formulario:

```
Me.Menu = New MainMenu() 'this refers to the form
```

ContextMenu

La clase ContextMenu representa menús de acceso rápido que se pueden presentar cuando el usuario hace clic en el botón alterno del *mouse* sobre un control o área del formulario. Los controles (y por lo tanto los formularios) tiene una propiedad ContextMenu que puede vincular a ese menú. Si utiliza esta propiedad, el menú de Contexto se abre automáticamente si el usuario hace clic con el botón alterno del mouse en el área cliente de un control. También puede mostrar manualmente un menú de contexto con el método Show.

Más eventos

Si la propiedad ContextMenu se modifica, entonces ocurre un evento ContextMenuChanged.

El evento MenuStart aparece cuando se hace clic en algún elemento del menú. Puede utilizar este evento de manera similar al evento Popup.

El evento MenuComplete señala que todos los menús están cerrados. Por ejemplo, este es el momento adecuado de reiniciar el texto de descripción en la barra de estado.

Diapositiva 23

Barra de estado

- Es una ventana horizontal en la parte inferior de la ventana padre.
- Controles StatusBar, StatusBarPanel

```
Private Dim sb  As StatusBar sb
Private Dim sbp0 As StatusBarPanel
Private Dim sbp1 As StatusBarPanel
sb = New StatusBar()
sbp0 = sb.Panels.Add("")
sbp0.AutoSize = StatusBarPanelAutoSize.Contents
sbp1 = sb.Panels.Add("")
sbp1.Text = “Panel Text”
sbp1.AutoSize = StatusBarPanelAutoSize.Contents
sbp1.Alignment = HorizontalAlignment.Center
sb.ShowPanels = true
```

StatusBar and StatusBarPanel

Una barra de estado es una ventana horizontal que se abre en la parte inferior de la ventana padre en la cual la aplicación puede mostrar diferentes tipos de información de estado. La barra de estado se puede dividir en diferentes partes para mostrar más de un tipo de información.

.NET Framework ofrece el control StatusBar de la barra de estado y el control StatusBarPanel para paneles individuales.

Puede crear un panel mediante el método Add del grupo de Paneles.

Para mostrar los paneles, debe establecer la propiedad ShowPanels como verdadera.

Puede indicar el tamaño y alineación de cada panel a través de algunas propiedades.

Ejemplo:

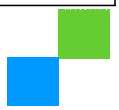
```
Private Dim sb  As StatusBar sb
    Private sbp0 As StatusBarPanel
    Private sbp1 As StatusBarPanel
    sb = New StatusBar
    sbp0 = sb.Panels.Add("")
    sbp0.AutoSize = StatusBarPanelAutoSize.Contents
    sbp1 = sb.Panels.Add("")
    sbp1.Text = "Panel Text"
    sbp1.AutoSize = StatusBarPanelAutoSize.Contents
    sbp1.Alignment = HorizontalAlignment.Center
    sb.ShowPanels = True
```

Diapositiva 24

Cuadros de diálogo comunes

- OpenFileDialog, SaveFileDialog
  - Derivados de FileDialog
  - Ofrecen el método OpenFile
- FontDialog
  - Presenta un evento Apply para la funcionalidad de vista previa
- ColorDialog
- PageSetupDialog
- PrintDialog

Clase	Descripción
ColorDialog	Representa un cuadro de diálogo común que muestra los colores disponibles junto con los controles que permiten al usuario definir los colores personalizados.
FontDialog	Representa un cuadro de diálogo común que muestra una lista de fuentes instaladas actualmente en el sistema.
PageSetupDialog	Representa un cuadro de diálogo que permite a los usuarios manipular las configuraciones de la página, incluyendo márgenes y la orientación del papel.
PrintDialog	Permite que los usuarios seleccionen una impresora y elijan qué porciones del documento imprimir.
FileDialog	Muestra una ventana de diálogo a partir de la cual el usuario puede seleccionar un archivo. De esta clase se derivan OpenFileDialog, que representa un cuadro de diálogo común el cual permite al usuario abrir un archivo, y SaveFileDialog para guardar el archivo.



Cuadros de diálogo comunes

Windows ofrece algunos cuadros de diálogo comunes. .NET Framework encapsula esta funcionalidad en algunas clases.

En su centro, existe una clase base abstracta llamada CommonDialog. Puede mostrar un cuadro de diálogo mediante ShowDialog lo cual devuelve un valor DialogResult. FileOpenDialog ofrece el método OpenFile, el cual abre un archivo con permiso de lectura/escritura seleccionado por el usuario. El valor de retorno es un objeto Stream. Esta es una simplificación, de otra manera obtiene el nombre del archivo por medio de la propiedad FileName y abre el archivo usted mismo. La clase OpenFileDialog ofrece también el método OpenFile, pero abre el archivo seleccionado por el usuario con permiso de sólo lectura. FontDialog genera un evento Apply cuando el usuario hace clic en el botón Apply. Manejar este evento para hacer una vista previa de la fuente elegida.

ColorDialog

Representa un cuadro de diálogo común que muestra los colores disponibles junto con los controles que permiten al usuario definir los colores personalizados.

FontDialog

Representa un cuadro de diálogo común que muestra una lista de fuentes instaladas actualmente en el sistema.

PageSetupDialog

Representa un cuadro de diálogo que permite a los usuarios manipular las configuraciones de la página, incluyendo márgenes y la orientación del papel.

PrintDialog

Permite que los usuarios seleccionen una impresora y elijan qué porciones del documento imprimir.

Ejemplo de OpenFileDialog:

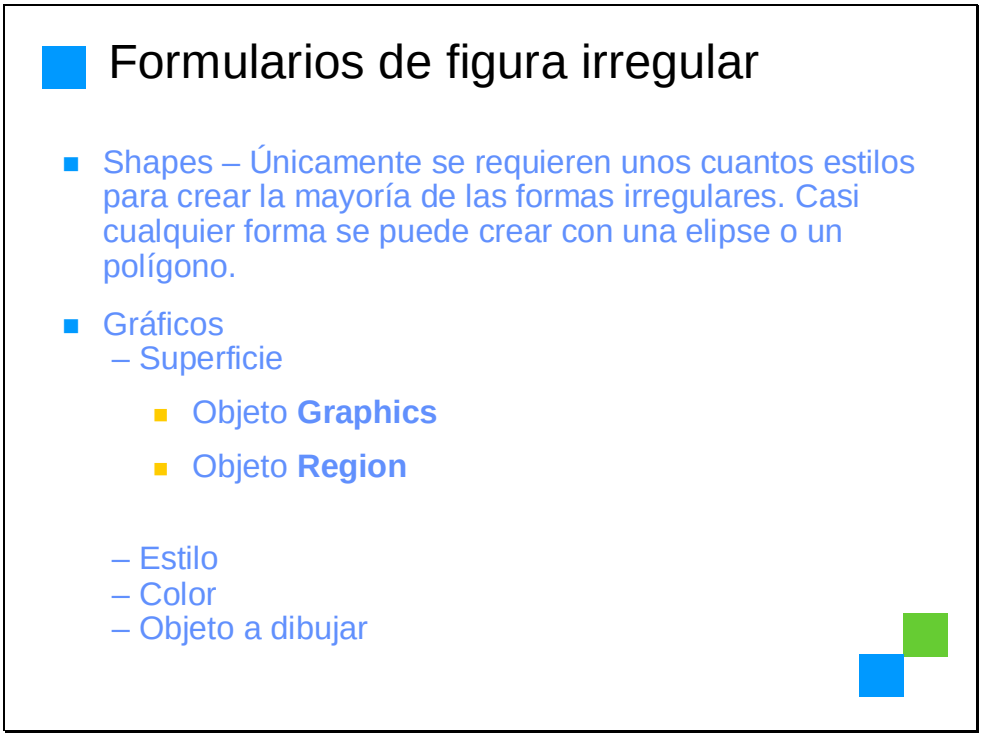
```
Public Sub OpenFile()
        Dim myStream As System.IO.Stream
        Dim openFileDialog1 As New OpenFileDialog

        'Set the initial directory
        openFileDialog1.InitialDirectory = "c:\\"
        'Only display text files
        openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*"
        openFileDialog1.FilterIndex = 2
        openFileDialog1.RestoreDirectory = True
        If openFileDialog1.ShowDialog() = DialogResult.OK Then
           'Verify if the user select any file.
           myStream = openFileDialog1.OpenFile()
           If Not (myStream = null) Then
                  MessageBox.Show( myStream.ToString() );  'Display a file name
selected.
           End If
        End If
    End Sub
```

FileDialog

Muestra una ventana de diálogo a partir de la cual el usuario puede seleccionar un archivo. De esta clase se derivan OpenFileDialog, que representa un cuadro de diálogo común el cual permite al usuario abrir un archivo, y SaveFileDialog para guardar el archivo.

Diapositiva 25



Formularios de figura irregular

Este es un tema muy interesante. Los formularios irregulares no responden a cualquier problema difícil dentro del desarrollo de una GUI, pero son emocionantes y divertidos Necesita cuatro elementos para crear un formulario irregular: una superficie con una figura, un estilógrafo (pen), el color y el objeto a dibujar.

Shapes – Únicamente se requieren unos cuantos estilos para crear la mayoría de las formas irregulares. Casi cualquier forma se puede crear con una elipse o un polígono. Las formas como círculos y triángulos son en realidad casos especiales o combinaciones de estas dos figuras. Otros elementos que se pueden dibujar incluyen arcos, Beziers, curvas y líneas.

Gráficos

Superficie – La superficie sobre la cual dibuja es equivalente a dos objetos diferentes dentro de .NET Framework: los objetos Graphics y Region. El objeto Graphics proporciona los métodos para dibujar en el contexto.

Stylus - El estilógrafo es la herramienta con la cual dibuja. .NET Framework le ofrece las herramientas con las cuales puede dibujar: una pluma o pincel.

Color – Los colores se utilizan con frecuencia para dar diferentes efectos. Los colores disponibles en .NET se agrupan bajo dos encabezados: KnownColor y SystemColors.

Objeto a dibujar – Después de que ha dicho y hecho todo, hay otras cosas que no pueden ser insignificantes si no tiene algo que dibujar. Si encuentra algo interesante en dibujar programáticamente, tome un punto vista cubista, y vea lo que sucede. Este ejemplo dibuja una señal de alto para la figura de nuestro formulario

Ejemplo:

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim grphPath As GraphicsPath = New GraphicsPath
    Dim grphSurface As Graphics
    Dim rgnSurface As Region
    Dim rectText As RectangleF
    Dim octArray(8) As PointF
    Dim fntText As Font
    Me.BackColor = Color.Red

    'Octagon Point Array
    octArray(0) = New PointF(70, 0)
    octArray(1) = New PointF(170.0)
    octArray(2) = New PointF(240.7)
    octArray(3) = New PointF(240, 170)
    octArray(4) = New PointF(170, 240)
    octArray(5) = New PointF(70, 240)
    octArray(6) = New PointF(0.17)
    octArray(7) = New PointF(0.7)
    octArray(8) = New PointF(70, 0)
    'this turns the form itself into the octagon
    grphPath.AddPolygon(octArray)
    rgnSurface = New Region(grphPath)
    Me.Region = rgnSurface

    'This handles the borders
    grphSurface = Me.CreateGraphics()
    grphSurface.DrawPolygon(New Pen(Color.White, 5), octArray)
    grphSurface.DrawPolygon(New Pen(Color.Black, 3), octArray)

    'This handles the text.
    rectText = New RectangleF(8, 80, 240, 100)
    fntText = New Font(Me.Font.FontFamily, 55, FontStyle.Bold)
    grphSurface.DrawString("STOP", fntText, New SolidBrush(Color.Black), rectText)
    'Clean Up
    fntText.Dispose()
    grphSurface.Dispose()
    grphPath.Dispose()
    rgnSurface.Dispose()
End Sub
```

Diapositiva 26

Lecturas recomendadas

- Troelsen. Visual Basic .NET and the .NET Platform: An Advanced Guide. APRESS, 2002.
- Beres, Evjen. *Visual Basic .NET Bible*. WiWiley, 2002gTagliaferri.sVisual Basic.NET Codemaster's Library.ybSybex, 2002.
- WWW.ASP.NET
- WWW.GOTDOTNET.COM
- WWW.DOTNETJUNKIES.COM
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchasyncprocvb.asp
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconthreadinginvisualbasic.asp
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontowfcforms.asp
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netstart/html/cpframeworkref_start.asp