Trabajando con Operaciones APM

Muchas de las clases del .NET Framework que soportan operaciones asíncronas realizan estas operaciones mediante la implementación de un patrón de diseño conocido como ***Modelo de Programación Asíncrona*** o simplemente ***APM*** por sus siglas en inglés (***Asynchronous Programming Model***).

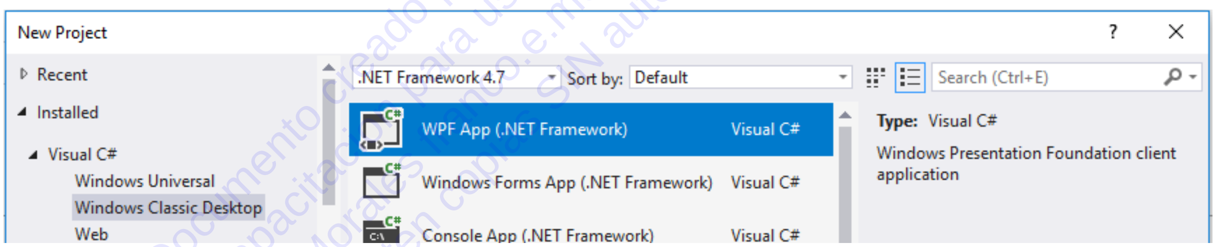
El patrón ***APM*** es implementado típicamente como 2 métodos: un método ***Begin****NombreDeLaOperación* que inicia la operación asíncrona y un método ***End****NombreDeLaOperación* que proporciona el resultado de la operación asíncrona.

Normalmente, invocamos al método ***End****NombreDeLaOperación* dentro del método *Callback*.

Ejercicio
Trabajando con Operaciones APM

Realiza los siguientes pasos para conocer la forma de trabajar con operaciones ***APM***.

1. Crea una aplicación WPF utilizando la plantilla ***WPF App (.NET Framework)***.



Consideremos una aplicación que verifica los URLs que el usuario proporciona. La aplicación consistirá en un cuadro de texto, un botón de comandos y una etiqueta.

2. Dentro del archivo ***MainWindow.xaml***, remplaza el elemento ***<Grid>*** por el siguiente código.

```
<StackPanel >
    <TextBox x:Name="URLToValidate" Height="38"/>
    <Button x:Name="ValidateURL" Content="Validar URL"
            Width="100" Height="38" Click="ValidateURL_Click"/>
    <Label x:Name="Result" Height="38" />
</StackPanel>
```

El usuario podrá proporcionar un URL en el cuadro de texto y después podrá hacer clic en el botón. El manejador del evento clic del botón, enviará una petición web asíncrona al URL y posteriormente mostrará en la etiqueta el código de estatus de la respuesta.

Para nuestro ejemplo, podemos utilizar la clase ***HttpWebRequest*** para validar un URL.

3. Agrega la siguiente instrucción al inicio del archivo ***MainWindow.xaml.cs*** para importar el espacio de nombre de la clase ***HttpWebRequest***.

```
using System.Net;
```

4. Agrega el siguiente código en la clase ***MainWindow*** para definir el manejador del evento ***Click*** del botón ***ValidateURL***. Este código crea una instancia ***WebRequest*** para el esquema URI especificado.

```
private void ValidateURL_Click(object sender, RoutedEventArgs e)
{
    try
    {
        HttpWebRequest Request =
            (HttpWebRequest)WebRequest.Create(URLToValidate.Text);
    }
    catch(Exception ex)
    {
        Result.Content = ex.Message;
    }
}
```

El patrón *APM* es implementado típicamente como 2 métodos: un método ***Begin****NombreDeLaOperación* que inicia la operación asíncrona y un método ***End****NombreDeLaOperación* que proporciona el resultado de la operación asíncrona.

La clase ***HttpWebRequest*** incluye métodos llamados ***BeginGetResponse*** y ***EndGetResponse***. El método ***BeginGetResponse*** envía una solicitud asíncrona a un recurso de la intranet o de internet y el método ***EndGetResponse*** devuelve la respuesta que el recurso proporciona.

El método ***Begin****NombreDeLaOperación* recibe como parámetro un delegado ***AsyncCallback*** que hace referencia a un método que será invocado cuando la operación asíncrona correspondiente se haya completado.

El método ***Begin****NombreDeLaOperación* recibe también un parámetro ***object*** de estado, útil para procesar la respuesta de la operación asíncrona.

Las clases que implementan el patrón ***APM*** utilizan una instancia ***IAsyncResult*** para representar el estatus de la operación asíncrona. El método ***Begin****NombreDeLaOperación* devuelve un objeto ***IAsyncResult***.

5. Agrega el siguiente código dentro del método ***ValidateURL_Click*** para iniciar una petición asíncrona a un recurso de la intranet o de internet.

```
HttpWebRequest Request =
    (HttpWebRequest)WebRequest.Create(URLToValidate.Text);
IAsyncResult Result =
    Request.BeginGetResponse(GetResponse, Request);
```

El código anterior indica que el método ***GetResponse*** (no implementado aún) será invocado cuando la operación asíncrona se haya completado. Puedes notar también que el objeto ***Request*** es proporcionado para que se pueda procesar la respuesta de la operación asíncrona.

6. Agrega el siguiente código para definir el método *Callback* ***GetResponse***. El método *Callback* recibe un parámetro ***IAsyncResult*** que nos permite procesar el resultado de la operación asíncrona.

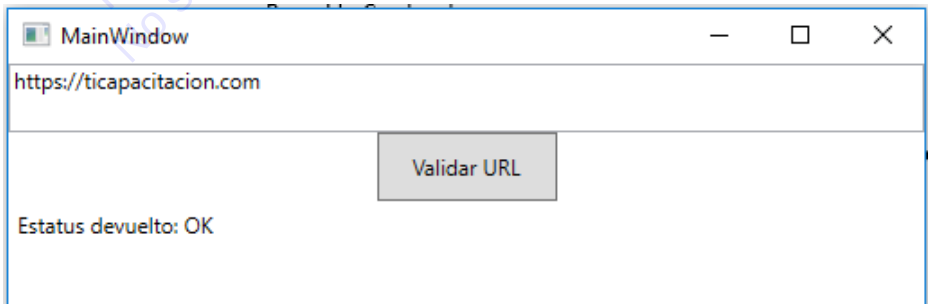
```
private void GetResponse(IAsyncResult result)
{
    // Recuperamos el objeto Request proporcionado como argumento en la
    // llamada al método Request.BeginGetResponse.
    HttpWebRequest Request = (HttpWebRequest)result.AsyncState;
    // Con el objeto Request podemos obtener el resultado de la petición.
    HttpWebResponse Response =
        (HttpWebResponse)Request.EndGetResponse(result);
    Result.Dispatcher.Invoke(() =>
    {
        Result.Content =
            $"Estatus devuelto: {Response.StatusCode}";
    });
}
```

Podemos probar ahora el manejo de operaciones de clases que implementan el patrón de diseño ***APM***.

7. Ejecuta la aplicación.

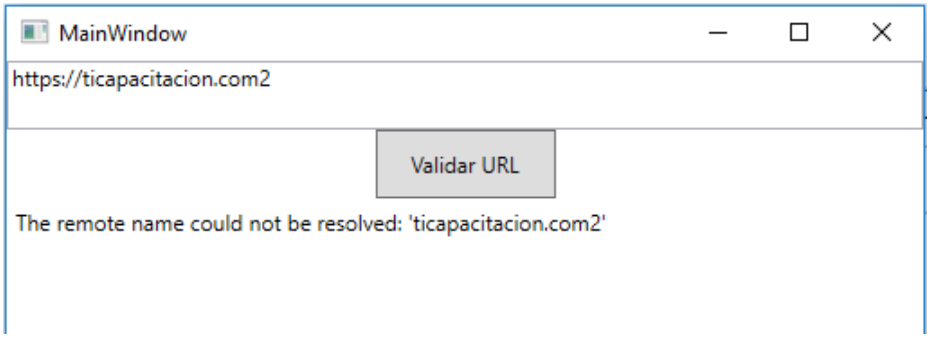
8. Escribe un URL existente dentro del cuadro de texto, por ejemplo, ***https://ticapacitacion.com***.

9. Haz clic en el botón ***Validar URL***. Podrás ver el resultado ***OK***.



10. Escribe un URL ***no*** existente dentro del cuadro de texto, por ejemplo, ***https://ticapacitacion.com2***

11. Haz clic en el botón ***Validar URL***. Podrás ver el resultado indicando que el nombre no puede ser resuelto.



La biblioteca ***Task Parallel*** facilita trabajar con las clases que implementan el patrón ***APM***. En lugar de implementar un método *Callback* para invocar al método ***End****NombreDeLaOperación* como en el caso del método *GetResponse* que implementamos anteriormente, podemos utilizar el método ***TaskFactory.FromAsync*** para invocar la operación asíncronamente y devolver el resultado en una sola instrucción.

12. Modifica la declaración del manejador de evento para que pueda ejecutarse de forma asíncrona.

```
private async void ValidateURL_Click(object sender, RoutedEventArgs e)
```

13. Modifica el bloque ***try*** para utilizar el método ***FromAsync*** para invocar la operación asíncronamente y devolver la respuesta de la petición web.

```
try
{
    HttpWebRequest Request =
        (HttpWebRequest)WebRequest.Create(URLToValidate.Text);
    HttpWebResponse Response =
        await Task<WebResponse>.Factory.FromAsync(
            Request.BeginGetResponse,
            Request.EndGetResponse, Request) as HttpWebResponse;
}
```

El método ***FromAsync*** incluye varias sobrecargas para soportar los métodos *APM* que requieren un número variable de argumentos.

En lugar de que implementemos un método *Callback* para procesar la respuesta, podemos utilizar el método ***FromAsync*** para realizar la operación completa.

14. Elimina el método ***GetResponse***.

15. Agrega el siguiente código dentro del bloque ***try*** para mostrar el resultado de la petición en la etiqueta ***Result***.

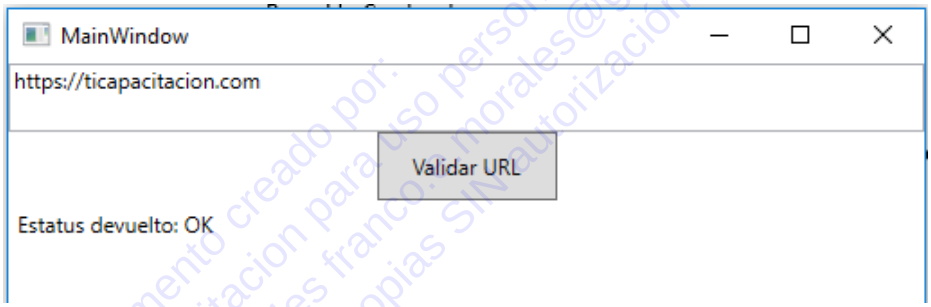
```
try
{
    HttpWebRequest Request =
        (HttpWebRequest)WebRequest.Create(URLToValidate.Text);
    HttpWebResponse Response =
        await Task<WebResponse>.Factory.FromAsync(
            Request.BeginGetResponse,
            Request.EndGetResponse, Request) as HttpWebResponse;

    Result.Content = $"Estatus devuelto: {Response.StatusCode}";
```

Puedes notar que ahora el código es más simple y conciso.

16. Ejecuta la aplicación.

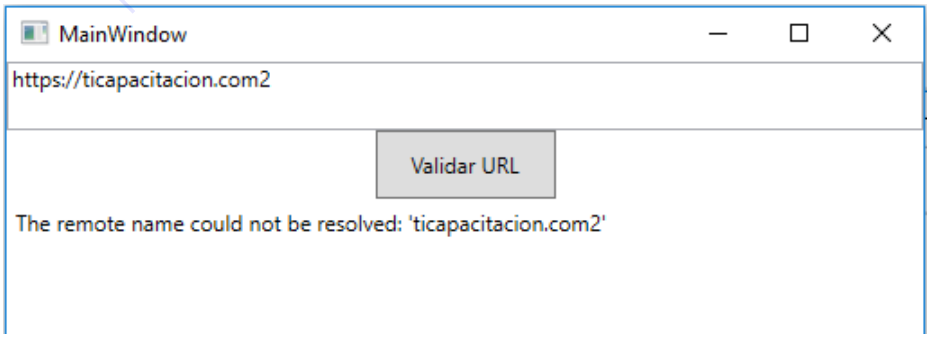
17. Escribe un URL existente dentro del cuadro de texto, por ejemplo, ***https://ticapacitacion.com***.

18. Haz clic en el botón ***Validar URL***. Podrás ver el resultado ***OK***.



19. Escribe un URL ***no*** existente dentro del cuadro de texto, por ejemplo, ***https://ticapacitacion.com2***

20. Haz clic en el botón ***Validar URL***. Podrás ver el resultado indicando que el nombre no puede ser resuelto.



La aplicación funciona como era esperado.

En este ejercicio, mostramos la forma de trabajar con operaciones ***APM*** utilizando ***Task Parallel Library*** que facilita trabajar con las clases que implementan el patrón de diseño ***APM***.

Para obtener más información sobre el uso de la biblioteca *Task Parallel* con patrones *APM*, se recomienda consultar el siguiente enlace:

TPL and Traditional .NET Framework Asynchronous Programming
http://go.microsoft.com/fwlink/?LinkID=267847