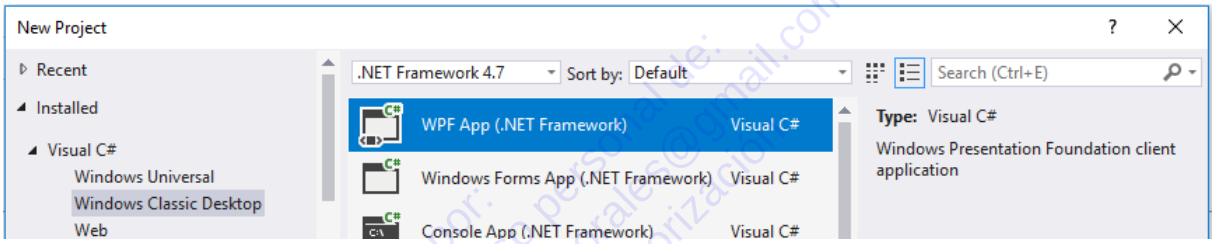
Manejando Excepciones de Métodos Esperables

Cuando realizamos operaciones asíncronas con las palabras clave ***async*** y ***await***, podemos manejar las excepciones de la misma forma en que manejamos las excepciones de código síncrono mediante el uso de bloques ***try/catch***.

Ejercicio
Manejando Excepciones de Métodos Esperables

Realiza los siguientes pasos para conocer la forma de manejar excepciones de *métodos esperables*.

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



   Consideremos una aplicación que nos permita mostrar el contenido de un sitio Web.

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

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Button x:Name="GetContent" Content="Obtener contenido Web"
            Click="GetContent_Click"/>
    <Label x:Name="WebContent" Grid.Row="1"/>
</Grid>
```

   Para realizar la petición web, en el manejador del evento *Click* del botón podríamos invocar al método asíncrono ***WebClient.DownloadStringTaskAsync*** utilizando el operador ***await***.

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

```
using System.Net;
```

4. Agrega el siguiente código en la clase ***MainWindow*** para definir el manejador del evento ***Click*** del botón ***GetContent***. Este código crea una instancia ***WebClient*** para realizar una petición web y mostrar el resultado obtenido en el control ***WebContent***. Si el URL proporcionado en el código no es válido, el método lanzará una excepción ***WebException***

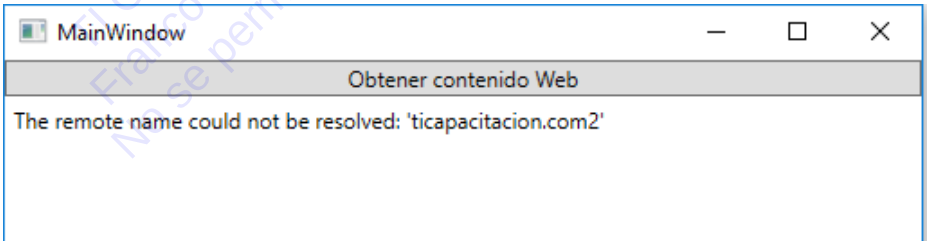
que será atrapada en el bloque ***catch*** y el mensaje de la excepción será mostrado en el control *WebContent*.

```
private async void GetContent_Click(object sender, RoutedEventArgs e)
{
    using (WebClient Client = new WebClient())
    {
        try
        {
            WebContent.Content =
                await Client.DownloadStringTaskAsync(
                    "https://ticapacitacion.com2");
        }
        catch(WebException ex)
        {
            WebContent.Content = ex.Message;
        }
    }
}
```

En este ejemplo de código se muestra la forma de atrapar una excepción generada dentro de un método *esperable (awaitable)*. Aunque la operación es asíncrona, se devuelve el control al método *GetContent_Click* cuando la operación asíncrona es completada y la excepción es manejada correctamente. Esto funciona porque detrás de escenas, la biblioteca *Task Parallel* captura las excepciones asíncronas y vuelve a lanzarlas en el hilo de la interfaz de usuario.

5. Ejecuta la aplicación.

6. Haz clic en el botón ***Obtener contenido Web***. Puedes notar que debido a que el URL no es válido, la excepción es atrapada y el mensaje es mostrado en la pantalla.



Excepciones no Observadas

Cuando una tarea genera una excepción, solo podemos manejar la excepción cuando el hilo que inicia la tarea (*joining thread*) accede a la tarea, por ejemplo, mediante el uso del operador ***await*** o mediante el llamado al método ***Task.Wait***. Si el *joining thread* nunca accede a la tarea, la excepción permanecerá como no observada. Cuando el recolector de basura (*GC*) del .NET Framework detecta que una tarea ya no es requerida, el planificador de tareas lanzará una excepción si alguna excepción de la tarea permanece como no observada. De manera predeterminada, esta excepción es ignorada,

sin embargo, podemos implementar un controlador de excepciones de último recurso mediante la suscripción al evento ***TaskScheduler.UnobservedTaskException***. En el manejador de excepciones, podemos establecer el estatus de la excepción a ***Observed*** para evitar que sea propagada.

7. Agrega el siguiente código dentro del constructor de la clase ***MainWindow*** para realizar una suscripción al evento ***UnobservedTaskException*** del objeto ***TaskScheduler***.

```
public MainWindow()
{
    InitializeComponent();
    TaskScheduler.UnobservedTaskException +=
        TaskScheduler_UnobservedTaskException;
}
```

8. Agrega el siguiente código dentro de la clase ***MainWindow*** para definir el manejador del evento ***UnobservedTaskException***. Observa que en el manejador de excepciones establecemos el estatus de la excepción a ***Observed*** para evitar que se propague.

```
private void TaskScheduler_UnobservedTaskException(
    object sender, UnobservedTaskExceptionEventArgs e)
{
    // Obtenemos las excepciones de la tarea
    var UnobservedExceptions = e.Exception.InnerExceptions;
    // Mostramos los mensajes de las excepciones
    foreach (var ex in UnobservedExceptions)
    {
        WebContent.Dispatcher.Invoke(()=>
        WebContent.Content +=
            $"{ex.Message}{Environment.NewLine}");
    }
    // Establecemos la excepción a Observed para que no se propague.
    e.SetObserved();
}
```

Para ejemplificar, podemos ahora lanzar una tarea que dispare una excepción no observada.

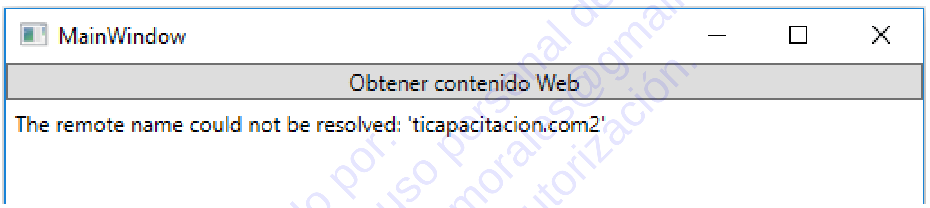
9. Modifica el método ***GetContent_Click*** para que se ejecute de forma síncrona y lance una tarea que dispare una excepción no observada (el código no maneja la excepción).

```
private void GetContent_Click(object sender, RoutedEventArgs e)
{
    Task.Run(async () =>
    {
        using (WebClient Client = new WebClient())
        {
            string Content =
            await Client.DownloadStringTaskAsync(
                "https://ticapacitacion.com2");
            WebContent.Dispatcher.Invoke(() =>
            {
                WebContent.Content = Content;
            });
```

```
            }
        });
        // Con fines de prueba:
        // Damos tiempo a que se ejecute la tarea.
        System.Threading.Thread.Sleep(3000);
        // Forzamos la recolección de basura.
        GC.WaitForPendingFinalizers();
        GC.Collect();
    }
```

10. Ejecuta la aplicación.

11. Haz clic en el botón ***Obtener contenido Web***. Puedes notar que la excepción no observada (no manejada) es procesada en el manejador del evento ***TaskScheduler.UnobservedTaskException*** y el mensaje es mostrado en el control ***WebContent***.



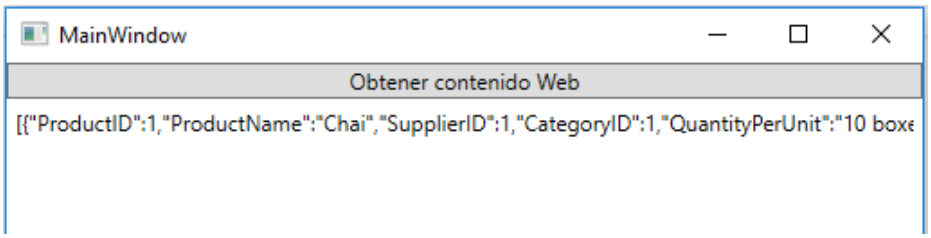
12. Regresa a Visual Studio y detén la ejecución.

13. Proporciona un URL válido por ejemplo https://ticapacitacion.com/webapi/northwind/products.

```
using (WebClient Client = new WebClient())
{
    string Content =
    await Client.DownloadStringTaskAsync(
        "https://ticapacitacion.com/webapi/northwind/products");
    WebContent.Dispatcher.Invoke(() =>
    {
        WebContent.Content = Content;
    });
}
```

14. Ejecuta la aplicación.

15. Haz clic en el botón ***Obtener contenido Web***. Puedes notar que se muestra la información correcta del URL válido.



16. Regresa a Visual Studio y detén la ejecución.

17. Proporciona nuevamente un URL no válido por ejemplo https://ticapacitacion.com2.

```
using (WebClient Client = new WebClient())
{
    string Content =
    await Client.DownloadStringTaskAsync(
        "https://ticapacitacion.com2");
    WebContent.Dispatcher.Invoke(() =>
    {
        WebContent.Content = Content;
    });
}
```

18. Comenta el código que realiza la suscripción al evento ***TaskScheduler.UnobservedTaskException***.

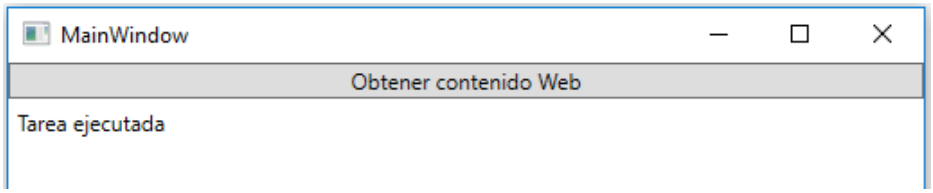
```
public MainWindow()
{
    InitializeComponent();
    //TaskScheduler.UnobservedTaskException +=
    //    TaskScheduler_UnobservedTaskException;
}
```

19. Agrega el siguiente código al final del método para indicar cuando la tarea haya finalizado.

```
    // Con fines de prueba:
    // Damos tiempo a que se ejecute la tarea.
    System.Threading.Thread.Sleep(3000);
    // Forzamos la recolección de basura.
    GC.WaitForPendingFinalizers();
    GC.Collect();
    WebContent.Content += "Tarea ejecutada";
}
```

20. Ejecuta la aplicación.

21. Haz clic en el botón ***Obtener contenido Web***. Puedes notar que la excepción generada por ser un URL no válido no es mostrada.



De manera predeterminada, a partir del .NET Framework 4.5, el motor de tiempo de ejecución .NET ignora las excepciones de tareas no observadas y permite que la aplicación se siga ejecutando. Este comportamiento contrasta con el comportamiento predeterminado en el .NET Framework 4.0, donde el motor de tiempo de ejecución .NET podía terminar cualquier proceso que lanzara excepciones de tareas no observadas. Podemos revertir al enfoque de terminación del proceso mediante la adición de un elemento ***ThrowUnobservedTaskExceptions*** en el archivo de configuración de la aplicación.

22. Regresa a Visual Studio y detén la ejecución.

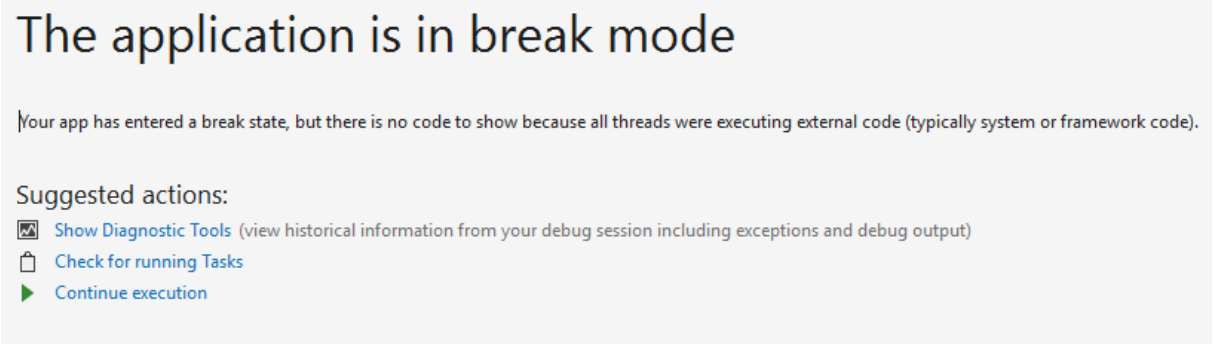
23. Agrega el siguiente código dentro del archivo ***App.config***.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7" />
    </startup>

    <runtime>
      <ThrowUnobservedTaskExceptions enabled="true"/>
    </runtime>

</configuration>
```

Si establecemos ***ThrowUnobservedTaskExceptions*** en ***true***, el motor de tiempo de ejecución .NET terminará cualquier proceso que contenga excepciones de tareas no observadas.

24. Ejecuta la aplicación.

25. Haz clic en el botón ***Obtener contenido Web***. Puedes notar que Visual Studio detiene la aplicación debido a la excepción no observada (no manejada).



Una práctica recomendada es establecer el indicador ***ThrowUnobservedTaskExceptions*** en ***true*** durante el desarrollo de la aplicación y retirar el indicador antes de liberar el código.

En este ejercicio, mostramos la forma de manejar excepciones en métodos *esperables* y la forma de manejar excepciones *no observadas*.