



Utilizando async y await

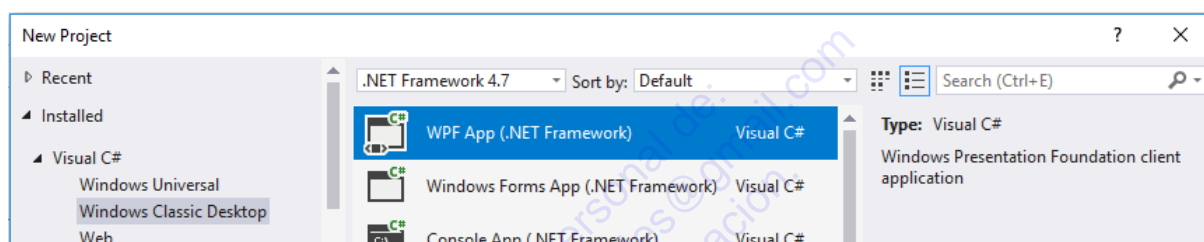
Otra de las técnicas utilizadas para invocar y administrar operaciones asíncronas es mediante el uso de las palabras clave **async** y **await**.

Ejercicio

Utilizando async y await

Realiza los siguientes pasos para conocer el uso del modificador **async** y el operador **await**.

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



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

```
<StackPanel>
    <Label x:Name="lblResult"/>
    <Button x:Name="btnGetResult"
        Content="Obtener resultado" Click="btnGetResult_Click"/>
</StackPanel>
```

El código anterior simula una aplicación sencilla que consiste en un botón y una etiqueta. Cuando el usuario haga clic en el botón, utilizaremos una tarea para obtener un resultado que será mostrado en la etiqueta.

Si un usuario hace clic en el botón, el manejador del evento **Click** del botón será ejecutado en el hilo de la interfaz de usuario. Si el manejador del evento inicia una tarea asíncrona, esa tarea será ejecutada en un hilo de segundo plano.

3. Agrega el siguiente código dentro de la clase **MainWindow** para definir el manejador del evento **Click** del botón **btnGetResult**.

```
private void btnGetResult_Click(object sender, RoutedEventArgs e)
{
    lblResult.Content = "Calculando un número aleatorio...";
    Task<int> T = Task.Run<int>() =>
    {
        System.Threading.Thread.Sleep(10000);
        return new Random().Next(5000);
    };
}
```



```
lblResult.Content += $"Número obtenido: {T.Result}";  
}
```

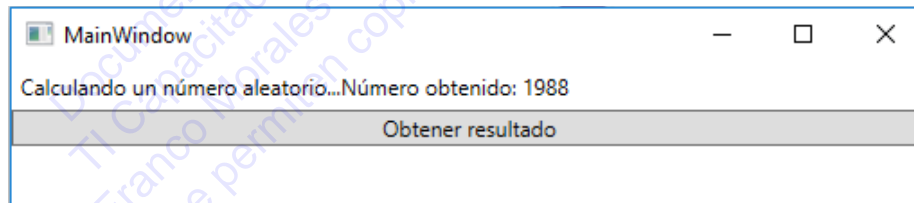
En el código anterior, cuando el usuario hace clic en el botón, el manejador de eventos invoca una operación de larga duración a través de una tarea *T*. El hilo de la tarea *T* simplemente duerme durante 10 segundos y después devuelve un numero entero aleatorio menor que 5000.

La instrucción ***lblResult.Content = T.Result*** bloquea el hilo de la interfaz de usuario hasta que el resultado de la tarea esté disponible.

En la práctica podríamos querer realizar algún otro proceso tal como hacer una llamada a un servicio web o recuperar información desde una base de datos. Cuando la tarea *T* haya finalizado, el manejador de evento, actualizará la etiqueta *lblResult* con el resultado de la operación.

4. Ejecuta la aplicación y haz clic en el botón **Obtener resultado**. Puedes notar que no se muestra el primer mensaje “Calculando un número aleatorio...” debido a que el hilo de la interfaz de usuario se ha congelado. En este ejemplo, la instrucción final en el manejador de evento bloquea el hilo de la interfaz de usuario hasta que el resultado de la tarea esté disponible. Esto significa que la interfaz de usuario quedará congelada completamente y el usuario será incapaz de cambiar el tamaño de la ventana, minimizar la ventana, etc.

Al finalizar la tarea, podrás ver el valor devuelto por la tarea.



Para permitir que la interfaz de usuario siga respondiendo, podemos convertir el manejador de evento en un método asíncrono.

Las palabras clave ***async*** y ***await*** se introdujeron en el .NET Framework 4.5 para facilitar la realización de operaciones asíncronas. Utilizamos el modificador ***async*** para indicar que un método puede ejecutarse de forma asíncrona.

5. Modifica el código que define al manejador del evento **Click** del botón para permitir que pueda ejecutarse de forma asíncrona.

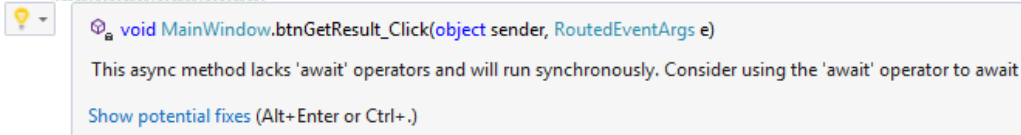
```
private async void btnGetResult_Click(object sender, RoutedEventArgs e)
```



Un método asíncrono se ejecuta síncronamente hasta que encuentra el primer operador **await**. En ese punto, el método es suspendido hasta que la tarea esperada sea completada y mientras tanto, el control regresa al código que invocó al método.

Puedes notar el mensaje que indica que debido a que el método no tiene una expresión con un operador **await**, será ejecutado de forma síncrona.

```
private async void btnGetResult_Click(object sender, RoutedEventArgs e)
{
```



Dentro del método **async** utilizamos el operador **await** para indicar los puntos en los cuales la ejecución del método puede ser suspendida mientras esperamos que una operación de larga duración sea completada.

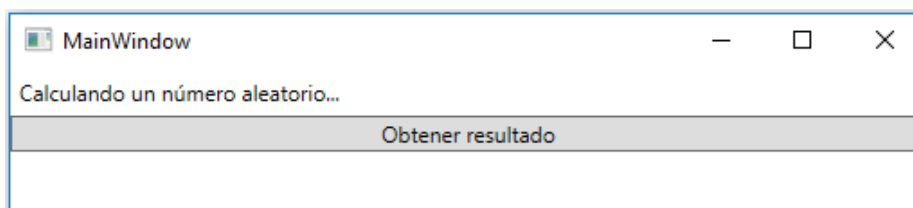
6. Modifica la siguiente línea del método.

```
lblResult.Content += $"Número obtenido: {await T}";
```

Mientras el método es suspendido en un punto **await**, el hilo que invoca al método puede hacer otro trabajo.

A diferencia de otras técnicas de programación asíncrona, las palabras clave **async** y **await** permiten ejecutar la lógica asíncrona en un solo hilo. Esto es particularmente útil cuando queremos ejecutar lógica en el hilo de la interfaz de usuario debido a que permite ejecutar lógica de manera asíncrona en el mismo hilo sin bloquear la interfaz de usuario.

7. Ejecuta la aplicación y haz clic en el botón **Obtener resultado**. Puedes notar que el primer mensaje es mostrado y que puedes mover o cambiar el tamaño de la ventana ya que ahora no está congelada.



El código actual incluye 2 cambios clave:

- La declaración del método ahora incluye la palabra clave **async**.
- Se le ha agregado el operador **await** a la instrucción que bloquea la ejecución.



Nota que cuando utilizamos el operador **await**, no esperamos el resultado de la tarea, esperamos a la tarea misma, esto es, ya no utilizamos la propiedad **Result** del hilo. Cuando el motor de ejecución de .NET ejecuta un método **async**, realmente no espera el resultado de la instrucción **await**. El método retorna a su invocador y el hilo es libre para realizar otro trabajo. Cuando el resultado de la tarea se encuentra disponible, el motor de ejecución regresa al método y continua la ejecución a partir de la instrucción **await**.

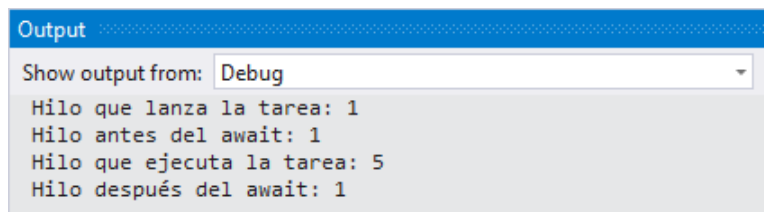
8. Agrega las siguientes instrucciones al inicio del archivo **MainWindow.xaml.cs** para importar los espacios de nombres de las clases **Debug** y **Thread** respectivamente.

```
using System.Diagnostics;  
using System.Threading;
```

9. Modifica el código del método para mostrar los identificadores de las tareas.

```
private async void btnGetResult_Click(object sender, RoutedEventArgs e)  
{  
    lblResult.Content = "Calculando un número aleatorio...";  
    Debug.WriteLine(  
        $"Hilo que lanza la tarea: {Thread.CurrentThread.ManagedThreadId}");  
    Task<int> T = Task.Run<int>(  
        () =>  
        {  
            Debug.WriteLine(  
                $"Hilo que ejecuta la tarea: {Thread.CurrentThread.ManagedThreadId}");  
            Thread.Sleep(10000);  
            return new Random().Next(5000);  
        }  
    );  
    Debug.WriteLine(  
        $"Hilo antes del await: {Thread.CurrentThread.ManagedThreadId}");  
    lblResult.Content += $"Número obtenido: {await T}";  
    Debug.WriteLine(  
        $"Hilo después del await: {Thread.CurrentThread.ManagedThreadId}");  
}
```

10. Ejecuta la aplicación y haz clic en el botón **Obtener resultado**. Puedes notar que la tarea de larga duración es ejecutada en otro hilo y después el hilo principal regresa cuando el resultado está disponible para seguir ejecutando las instrucciones del método asíncrono a partir de la instrucción **await**.





En este ejercicio, mostramos el uso de las palabras clave ***async*** y ***await*** para facilitar la realización de operaciones asíncronas.



Para obtener más información sobre el uso de ***async*** y ***await*** se recomienda consultar el siguiente enlace:

Asynchronous programming with async and await (C#)

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/index>

Documento creado por:
TI Capacitación para uso personal de:
Franco Morales franco.e.morales@gmail.com
No se permiten copias SIN autorización.