

Unidad 2 - Capítulo 4 - Laboratorio 1: Programación Asíncrona

1. Introducción a la Asincronía

Objetivos

Comprender el uso de los métodos asíncronos en .NET, mediante el uso de Task, async, await, y la comparación frente a métodos sincrónicos.

Duración Aproximada

30 minutos

Pasos

1. Crear una nueva aplicación de consola en Visual Studio llamada `LabAsincronia1`.
2. Implementar dos métodos:
 - `SimularOperacionPesada()`: que simule una operación que tarde 3 segundos utilizando `Thread.Sleep`.
 - `SimularOperacionPesadaAsync()`: que realice la misma operación, pero utilizando `await Task.Delay`.
3. Cree un nuevo método que se llame `CompararSincronoVsAsincrono()` e invóquelo desde el método `Main`, que realice lo siguiente:
 - Llame a la versión sincrónica, midiendo el tiempo total de ejecución con `Stopwatch`.
 - Muestre por consola un mensaje antes y después de la ejecución.
 - Llame luego a la versión asíncrona, pero sin esperarla inmediatamente. Mientras la tarea está en ejecución, permitir que el usuario escriba por consola. Luego, esperar a que finalice la tarea asíncrona.
 - Medir y mostrar el tiempo total de ejecución.
4. Compilar y ejecutar. Observar la diferencia entre el comportamiento bloqueante y no bloqueante.
5. Responder:
 - A. ¿Qué ventajas observás en el uso del código asíncrono?
 - B. ¿Qué inconvenientes podría tener si el código asíncrono no se maneja adecuadamente?

2. Ejecución Paralela de Tareas

Objetivos

Aplicar `Task` y `Task.WhenAll` para ejecutar múltiples tareas en paralelo de forma controlada.

Duración Aproximada

30 minutos

Pasos

1. Dentro del mismo proyecto, agregar tres métodos que simulen tareas pesadas:

- `OperacionMediaAsync()`: simula una operación de 2 segundos.
 - `OperacionLargaAsync()`: simula una operación de 3 segundos.
 - `OperacionCortaAsync()`: simula una operación de 1 segundo.
2. Crear un método `EjecutarTareasParalelasAsync()` que utilice `Task.WhenAll` para ejecutar las tres operaciones de forma paralela.
 3. Medir el tiempo de ejecución y mostrar por pantalla cuánto tarda el programa en finalizar todas las tareas.
 4. Comparar ejecutando las tres tareas en forma secuencial y luego en paralelo. Observar y analizar la diferencia de tiempos.

3. Manejo de Excepciones en Tareas Asincrónicas

Objetivos

Comprender cómo manejar las excepciones que se producen en métodos asincrónicos y cómo capturarlas correctamente para evitar bloqueos inesperados o pérdidas de control.

Duración Aproximada

30 minutos

Pasos

1. Agregar un método llamado `OperacionConErrorAsync()` que simule una operación asincrónica que lanza una excepción después de 2 segundos, por ejemplo, `throw new InvalidOperationException("Error simulado en operación asincrónica");`.
2. Crear un método llamado `ProbarManejoExcepcionesAsync()` que:
 - Invoca `OperacionConErrorAsync()`.
 - Capture la excepción usando un bloque `try-catch`.
 - Muestre un mensaje indicando que la excepción fue capturada y su mensaje.
3. Llamar a `ProbarManejoExcepcionesAsync()` desde el `Main` y observar que la aplicación no se detiene inesperadamente.

4. Reporte de Progreso en Tareas Asincrónicas

Objetivos

Aprender a utilizar `IProgress<T>` para reportar el progreso de una operación asincrónica y notificar al usuario mientras la tarea está corriendo.

Duración Aproximada

30 minutos

Pasos

1. Crear un método llamado `OperacionLargaConProgresoAsync(IProgress<int> progreso)` que:
 - Simule una operación de 10 pasos, con una demora de 500 ms en cada paso.

- Después de cada paso, reporte el porcentaje de avance (por ejemplo, 10%, 20%, ..., 100%) usando el objeto `progreso`.
2. Desde el `Main`, crear un `Progress<int>` que capture los reportes y los imprima por consola.
 3. Ejecutar `OperacionLargaConProgresoAsync()` pasando el objeto `Progress<int>` y observar cómo se actualiza la consola mientras la tarea está en curso.

Preguntas para reflexionar:

- A. ¿En qué situaciones prácticas usarías el progreso?
- B. ¿Cuál es la ventaja de `IProgress<T>` respecto a pasar simplemente una acción (un `Action<int>`)?

5. Cancelación de Tareas Asincrónicas

Objetivos

Utilizar `CancellationToken` para permitir que una tarea asincrónica pueda ser cancelada en tiempo de ejecución.

Duración Aproximada

30 minutos

Pasos

1. Crear un método llamado `OperacionCancelableAsync(CancellationToken token)` que simule 10 pasos de trabajo, uno por segundo. En cada iteración debe verificar si el token ha solicitado cancelación (`token.ThrowIfCancellationRequested()`).
2. En `Main`, utilizar `CancellationTokenSource` para cancelar la operación luego de que el usuario presione una tecla.
3. Capturar la excepción `OperationCanceledException` y mostrar un mensaje al usuario.

Preguntas para reflexionar:

- A. ¿Qué sucede si no se utiliza un `try-catch` alrededor de la llamada asincrónica?
- B. ¿Por qué es importante capturar las excepciones dentro del método asincrónico en lugar de hacerlo solo en `Main`?

6. (Opcional) Escritura y Lectura de Archivos Asincrónicos

Objetivos

Familiarizarse con las operaciones de entrada/salida asincrónicas mediante `File.WriteAllTextAsync` y `File.ReadAllTextAsync`.

Duración Aproximada

30 minutos

Pasos

1. Agregar al proyecto dos métodos:
 - o `EscribirArchivoAsync(string ruta, string contenido)`: que escriba contenido en un archivo de texto.
 - o `LeerArchivoAsync(string ruta)`: que lea el contenido del archivo.
2. Desde `Main`, invocar estos métodos para escribir y luego leer un archivo llamado `datos.txt`.
3. Mostrar por consola el contenido leído.
4. Simular que el archivo no existe y capturar el error correspondiente.

Nota: Para el **manejo de archivos** puede consultar el siguiente link oficial de Microsoft:

<https://learn.microsoft.com/es-es/troubleshoot/developer/visualstudio/csharp/language-compilers/read-write-text-file>