Multithreading

- الـ Multithreadingهو تقنية في البرمجة تسمح بتشغيل أجزاء متعددة من الكود (تسمى " ("Threadsفي نفس الوقت.

الفكرة هي تقسيم البرنامج إلى أجزاء صغيرة يمكن تنفيذها بشكل متوازي، مما يحسن الأداء خاصة في الأجهزة متعددة المعالجات.

المفاهيم الأساسية:

- **Thread:** هو وحدة من العمل يمكن أن تعمل بشكل متوازٍ مع خيوط أخرى.
- **Main Thread:** هو الخيط الرئيسي الذي يبدأ البرنامج به .
- **Concurrency vs Parallelism:**
- **التزامن (Concurrency):** عندما يتم التعامل مع أكثر من مهمة واحدة في نفس الوقت، لكن قد لا تنفذ كلها بالتوازي.
- **التوازي (Parallelism):** عندما يتم تنفيذ أكثر من مهمة في نفس الوقت باستخدام معالجات متعددة.
- **Thread Pool:** مجموعة من الخيوط الجاهزة للاستخدام بدلاً من إنشاء خيوط جديدة في كل مرة .

الفوائد:

- تحسين أداء التطبيقات التي تحتاج إلى التعامل مع مهام متعددة في وقت واحد مثل التطبيقات التي تعمل على الشبكة أو تتعامل مع الملفات.

- تسريع المعالجة عبر المعالجات المتعددة .

Example :

```
using System;
using System.Threading;

class Program
{
    static void Main(string[] args)
    {
        // إنشاء Threads
        Thread thread1 = new Thread(PrintNumbers);
        Thread thread2 = new Thread(PrintLetters);

        // بدء تشغيل Threads
        thread1.Start();
        thread2.Start();

        // انتظار انتهاء Threads
        thread1.Join();
        thread2.Join();

        Console.WriteLine("تم الانتهاء من جميع المهام!");
    }

    static void PrintNumbers()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine($"الرقم: {i}");
            Thread.Sleep(500); // تأخير لمحاكاة عمل معقد
        }
    }

    static void PrintLetters()
    {
        for (char c = 'A'; c <= 'E'; c++)
        {
            Console.WriteLine($"الحرف: {c}");
            Thread.Sleep(500); // تأخير لمحاكاة عمل معقد
        }
    }
}
```

1. 📌 ما هي البرمجة غير المتزامنة؟

- البرمجة غير المتزامنة تتيح للبرنامج تنفيذ أكثر من عملية في نفس الوقت (متوازيًا) دون أن يتوقف النظام بالكامل.

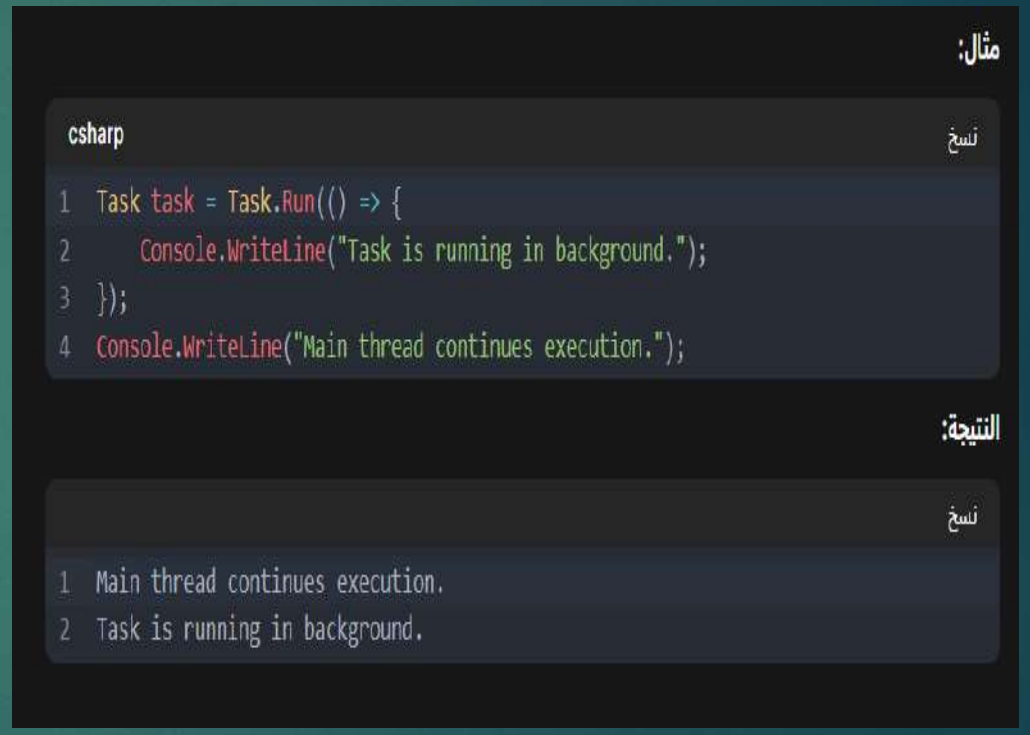
- تستخدم عندما تحتاج لتنفيذ عمليات تستغرق وقتًا (مثل الوصول إلى قاعدة البيانات، أو الاتصال بشبكة الإنترنت).

2. 📌 الأدوات الأساسية:

- **(1) Task.Run:**

يستخدم لإنشاء مهمة (Task) تعمل ف الخلفيه

المهمة مش بتوقف الـ Main Thread.



- (2) Task.WaitAll:
- يستخدم لانتظار انتهاء عدة مهام قبل المتابعة.

مثال:

csharp نسخ

```csharp
Task task1 = Task.Run(() => Console.WriteLine("Task 1 completed."));
Task task2 = Task.Run(() => Console.WriteLine("Task 2 completed."));

Task.WaitAll(task1, task2); // انتظر حتى تنتهي كل المهام

Console.WriteLine("All tasks are done.");
```

النتيجة:

نسخ

```
Task 1 completed.
Task 2 completed.
All tasks are done.
```

- (3) Async/Await:

  - async: يجعل الدالة قادرة على التعامل مع العمليات غير المتزامنة.

  - await: يجبر الكود على الانتظار لحد ما العملية غير المتزامنة تخلص.

مثال:

csharp نسخ

```csharp
static async Task ExampleAsync()
{
    Console.WriteLine("Starting async operation...");
    await Task.Delay(2000); // انتظر 2 ثانية
    Console.WriteLine("Async operation completed.");
}

static async Task Main(string[] args)
{
    await ExampleAsync(); // انتظر لحد ما ExampleAsync تخلص
    Console.WriteLine("Main method finished.");
}
```

النتيجة:

نسخ

```
Starting async operation...
[انتظار لمدة 2 ثانية]
Async operation completed.
Main method finished.
```

3. 📌 لماذا نستخدمها؟

- تحسين الأداء: السماح للنظام بتنفيذ عمليات متوازية.
- تجربة مستخدم أفضل: عدم تجميد واجهة المستخدم أثناء تنفيذ العمليات الثقيلة.