

◆ أولاً: إمّى نحتاج نستخدم Asynchronous Programming؟

يعنى الكود يقدر يكمل شغله من غير ما يستنى حاجة تخلص، وده بيساعد البرمجة الـ Asynchronous البرنامج يفضل سريع وميروحش في حالة “مش بيرد”.

أكتر 3 حالات بنحتاج فيها الـ `async`:

1. الاتصال بقاعدة البيانات:
لما البرنامج يبعث استعلام كبير أو يضيف بيانات، ممكن العملية تأخذ وقت، زي `async` بدل ما التطبيق يفضل مستنى، بنخليلها:
2. `await db.SaveChangesAsync();`
3. قراءة أو كتابة الملفات:
لو الملف كبير، القراءة ممكن تبطأ التطبيق.
فبنستخدم:
4. `await File.ReadAllTextAsync("data.txt");`
5. أو الإنترنت API الاتصال بـ:
الطلبات دي بتأخذ وقت بسبب الشبكة، فبنستخدم:
6. `await httpClient.GetAsync(url);`

كده البرنامج يكمل شغله عادي لحد ما الرد يجي

◆ ثانياً: الفرق بين Thread و Task

عندنا طريقتين لتشغيل كود في الخلفية C# في Thread و Task.
الاثنين بيشعقاو كود في نفس الوقت، بس الفرق بينهم في الطريقة والإدارة.

- **Thread:**
دي الطريقة القديمة، وإنانت اللي بتتحكم فيها بنفسك
مثال:
`Thread t = new Thread(MyMethod);
t.Start();`

كثير Threads بس التعامل معها ممكن يكون معقد خصوصاً لو عندك

- **Task:**
دي الطريقة الحديثة والأسهل
، وإنانت بس بتقول Thread Pool هو اللي بيديرها عن طريق .NET:
`Task.Run(() => MyMethod());`

هتشتغل لوحدها من غير ما تشيل هم الإداره والـ Task
كمان بتشتغل حلو جداً مع `await`.

الخلاصة:

من جوا Task بتسخدم كل Thread

You said:

عايزنا مختلفه بقا خالص

ChatGPT said:

تمام يا محمد

فيها نفس المعلومات بس مكتوبة بطريقة روائية — دي نسخة ريبورت جديدة ومختلفة تماماً في الأسلوب

وحكانية كأنك بتحكي لحد فاهم شوية كود، عايز تفهمه المفاهيم بطريقة خفيفة ومتربطة.

اللغة عامية بس مرتبة ومناسبة للتسليم برضو

والـ Task والـ Thread عن الـ Delegates في C#

أول جزء: فكرة الـ Asynchronous Programming

تخيل إنك فاتح برنامج، ودوست على زرار بيعمل عملية تقيلة زي جلب بيانات من السيرفر أو قراءة ملف كبير ، هيف يشتني العملية تخلص وبعدين يكمل — يعني (Synchronous) لو البرنامج شغال بطريقة عادي المستخدم هيحس إن التطبيق هنجر.

الي بتخليل التطبيق يشتغل "كانه بيعمل كذا حاجة في Asynchronous Programming وهذا بيجي دور الـ ".نفس الوقت

يعني العملية الطويلة تشتغل في الخلفية، والبرنامج يكمل باقي الشغل عادي

أمثلة بسيطة على الحالات دي ⤵

- لما التطبيق يتصل بـ Database Query ويسنتي نتيجة
- لما يقرأ أو يكتب ملف كبير من المارد
- لما يبعث أو يستقبل بيانات من الإنترن特 أو API.

بالتالي، المستخدم يفضل شايف التطبيق شغال وسرريع، حتى لو في كود بيأخذ وقت في الخلفية وده بيخلّي تجربة الاستخدام أحسن بكثير

ثاني جزء: الفرق بين الـ Thread وـ Task

بعض كده، الاثنين شبه بعض من حيث إنهم بيشغلا كود في الخلفية، بس طريقة الشغل مختلفة شوية.

◆ الـ Thread

هو وحدة تنفيذ بيشغلها النظام نفسه.

إنت اللي بتعملها وتتحكم فيها بنفسك، زي مثلاً:

```
Thread t = new Thread(MyMethod);  
t.Start();
```

بس دي طريقة متعبة شوية لأنك مسؤول عن كل حاجة فيها: إدارة الموارد، التوقف، المزامنة... إلخ.

◆ الـ Task

دي بقى طريقة أذكي وأحدث.

هو اللي بيتكفل بإدارة الثريادات عن طريق حاجة اسمها Thread Pool.

بتخلِي النظام هو اللي يختار ويضبط كل حاجة Task كتير بإيدك، الـ Threads يعني بدل ما تعمل

مثلاً:

```
Task.Run(() => MyMethod());
```

كده الميثود هتشغل في الخلفية على طول، وانت مش محتاج تتحكم في أي تفاصيل.

● باختصار:

زي إنك تسوق عربية مانيوال، الـ Thread

زي إنك تسوق أوتوماتيك — نفس الفكرة، بس أسهل وأنضف في الإدارة والـ Task.

□ الجاهزة في Delegates ثالث جزء: أنواع الـ C#

هو زي متغير يقدر يشاور على ميثود، # C# في الـ Delegate

يعني بدل ما بتعد اسم الميثود، تتبع المتغير اللي شايلها

وده مفيد لما تحب تمرر ميثود كـ parameter.

علشان نسهل على نفسنا بقى جهز لنا 3 أنواع جاهزة من الـ C# delegates:

1. Action

لما الميثود ما بترجعش حاجة (void).

مثال:

```
2. Action<string> sayHi = name => Console.WriteLine($"Hi {name}");  
3. sayHi("Mohamed");
```

4. Func

لما الميثود بترجع قيمة

آخر نوع في البراميتر هو نوع الـ return.

مثال:

```
5. Func<int, int, int> add = (a, b) => a + b;  
6. Console.WriteLine(add(3, 5)); // 8
```

7. Predicate

يعني اختبار أو شرط (true أو false) لما الميثود بترجمة.

مثال:

```
8. Predicate<int> isEven = x => x % 2 == 0;
```

```
9. Console.WriteLine(isEven(4)); // True
```

الخلاصة ← END

يتخلّي التطبيق أسرع ومايهمش وقت العمليات البطيئة Asynchronous Programming.
إن الأولى محتاجة إدارة يدوية، والثانية ذكى وأسهل Thread و Task الفرق بين Delegates.
فهي طريقة مرنة نقدر نستخدم فيها الميثودز كقيم، أما الـ Action، Func، و Predicate، فهى الأنواع الجاهزة زي.