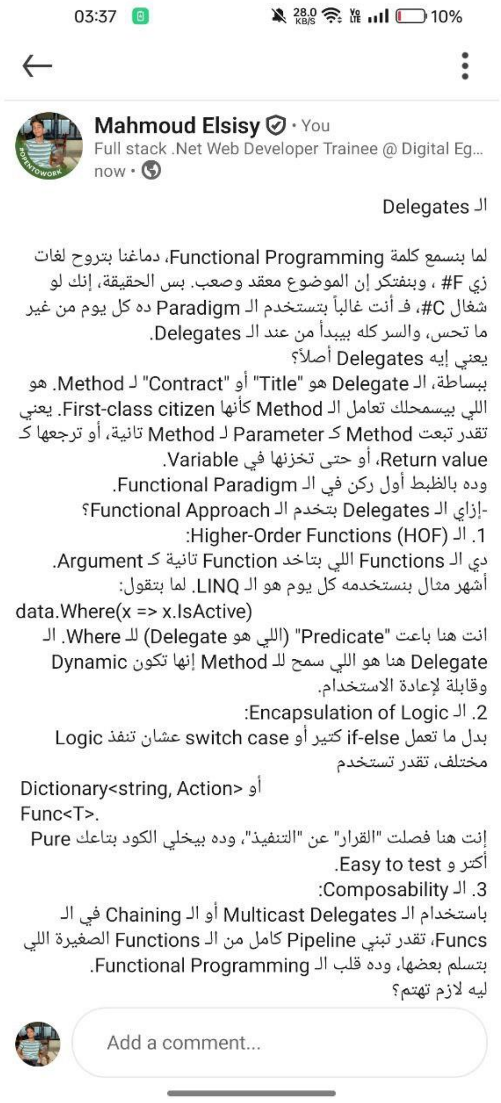
C#AssignmentDay10P02

1. LinkedIn article about Delegates in implementing functional paradigm.

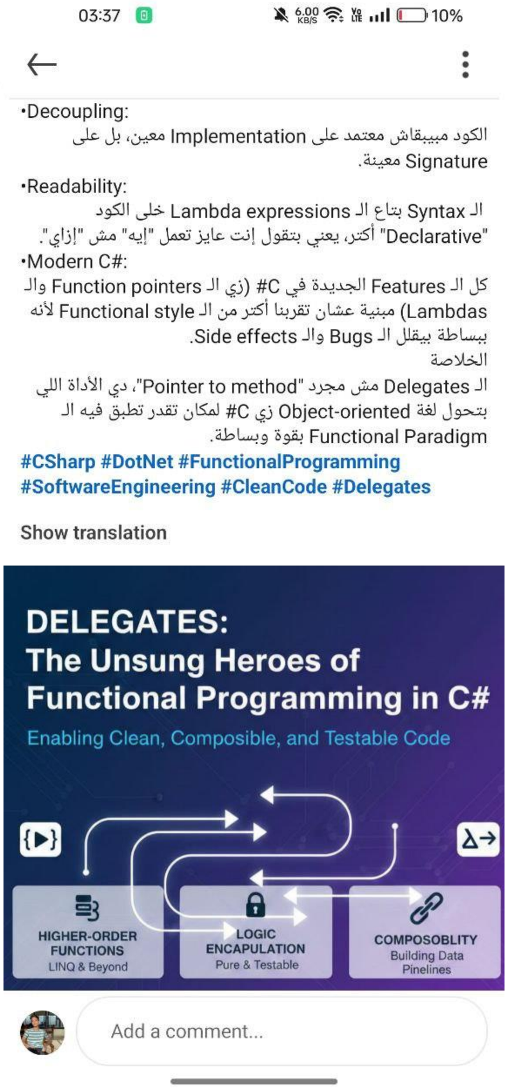


Mahmoud Elsisy · You
Full stack .Net Web Developer Trainee @ Digital Eg...
now ·

الـ Delegates

لما بنسمع كلمة Functional Programming، دماغنا بتروح لغات
زي F# ، وبنفتكر إن الموضوع معقد وصعب. بس الحقيقة، إنك لو
شغال C#، فـ أنت غالباً بتستخدم الـ Paradigm ده كل يوم من غير
ما تحس، والسر كله بيبدأ من عند الـ Delegates.
يعني إيه Delegates أصلاً؟
ببساطة، الـ Delegate هو "Title" أو "Contract" لـ Method. هو
اللي بيسمحلك تعامل الـ Method كأنها First-class citizen. يعني
تقدر تبعت Method كـ Parameter لـ Method تانية، أو ترجعها كـ
Return value، أو حتى تخزنها في Variable.
وده بالظبط أول ركن في الـ Functional Paradigm.
-إزاي الـ Delegates بتخدم الـ Functional Approach؟
1. الـ Higher-Order Functions (HOF):
دي الـ Functions اللي بتاخد Function تانية كـ Argument.
أشهر مثال بنستخدمه كل يوم هو الـ LINQ. لما بتقول:
data.Where(x => x.IsActive)
انت هنا باعت "Predicate" (اللي هو Delegate) للـ Where. الـ
Delegate هنا هو اللي سمح للـ Method إنها تكون Dynamic
وقابلة لإعادة الاستخدام.
2. الـ Encapsulation of Logic:
بدل ما تعمل if-else كتير أو switch case عشان تنفذ Logic
مختلف، تقدر تستخدم

Dictionary<string, Action> أو
Func<T>.
إنت هنا فصلت "القرار" عن "التنفيذ"، وده بيخلي الكود بتاعك Pure
أكتر و Easy to test.
3. الـ Composability:
باستخدام الـ Multicast Delegates أو الـ Chaining في الـ
Funcs، تقدر تبني Pipeline كامل من الـ Functions الصغيرة اللي
بتسلم بعضها، وده قلب الـ Functional Programming.
ليه لازم تهتم؟



•Decoupling:
الكود مبيبقاش معتمد على Implementation معين، بل على
Signature معينة.
•Readability:
الـ Syntax بتاع الـ Lambda expressions خلى الكود
"Declarative" أكتر، يعني بتقول إنت عايز تعمل "إيه" مش "إزاي".
•Modern C#:
كل الـ Features الجديدة في C# (زي الـ Function pointers والـ
Lambdas) مبنية عشان تقربنا أكتر من الـ Functional style لأنه
ببساطة بيقلل الـ Bugs والـ Side effects.
الخلاصة
الـ Delegates مش مجرد "Pointer to method"، دي الأداة اللي
بتحول لغة Object-oriented زي C# لمكان تقدر تطبق فيه الـ
Functional Paradigm بقوة وبساطة.
#CSharp #DotNet #FunctionalProgramming
#SoftwareEngineering #CleanCode #Delegates

Show translation

2. Parallel Programming and Concurrency

Concurrency and parallel programming are related concepts in software and systems design that help programs handle multiple things at once — but they're not the same thing:

- **Concurrency** means a system can manage multiple tasks *in overlapping time periods* — tasks *start, run, and complete* in an interleaved fashion. It's about structuring software so it can deal with lots of work without waiting for one thing to finish before starting the next. Tasks may or may not actually run at the exact same instant.
- **Parallel programming** is a specific form of concurrency where tasks truly run *simultaneously* on multiple processors/cores. This can significantly speed up compute-heavy work.
- **Modern software often mixes both**: you write a concurrent program that can scale and remain responsive, and if hardware allows, tasks may execute in parallel. The key challenges include coordination, avoiding race conditions, and ensuring correct shared state.

In short:

- Concurrency is about *dealing with* many tasks.
- Parallelism is about *doing* many tasks at the same time.

3. Unit Testing and Test-Driven Development (TDD)

-Unit Testing

- A **unit test** checks one small unit of code (often a single function or class) in isolation to verify it behaves as expected.
- It’s automated, repeatable, and serves as documentation and a guard against regressions (bugs introduced by later changes).

-Test-Driven Development (TDD)

- **TDD** is a development practice where **tests are written before the code**. The typical cycle is:
    1. Write a small test that *fails*.
    2. Write the minimal code to make it *pass*.
    3. Refactor the code and tests for clarity and design.
    4. Repeat with the next test.
- TDD encourages thinking about requirements first, leads to better test coverage, and can improve design because you focus on how code *should behave* before how it *works*. It’s also part of many agile workflows.
- Limitations include writing many tests (which takes time) and risk that tests reflect incorrect assumptions if not designed well.

4. Asynchronous Programming with async and await

-Asynchronous programming is a way for programs to handle long-running operations (like I/O, network calls, file access, timers, etc.) *without blocking* the rest of the program.

-async and await Keywords

- Many languages (such as JavaScript, Python, C#, Dart, Rust) provide **async/await** as language features that simplify writing asynchronous code.
- An **async** function is one that can be paused and resumed.
- The **await** keyword tells the program to *wait for a particular async operation to complete* while letting the system do other work in the meantime.

-What This Does

- Instead of freezing the whole program while something slow happens (blocking), async/await lets the program **continue handling other tasks** and come back to finish work when ready.
- Under the hood, the language or runtime usually turns these into state machines and “promises/futures” so that one piece of work can pause and later resume without consuming a thread.

-Why It Matters

- Especially useful for server applications, UI apps, or services that handle many requests simultaneously where waiting on one task shouldn’t freeze everything else.