C# Assignment Day03

Part02

1-LinkedIn article about string immutability:

لما بنشتغل في C#، هنتعامل كتير مع النوع string، وواحدة من أهم الخصائص اللي لازم نفهمها هي immutability أو عدم قابلية التغيير. بس يعني إيه الكلام ده بالظبط؟

إيه هي الـ Immutability؟

لما نقول إن الـ string immutable، ده معناه إن بعد ما بننشئ string معين، القيمة بتاعته مش هتتغير. أي تعديل على الـ string مش هيغير القيمة الأصلية، لكن هيطلع string جديد بالقيمة الجديدة.

مثال بسيط:

```
string name = "Mahmoud";
name = name + " Elsisi";
Console.WriteLine(name);
```

هنا، لما بنعمل name + " Elsisi"، مش بيتغير الـ string الأصلي "Mahmoud"، لكن بيتخلق object جديد في الذاكرة بقيمة "Mahmoud Elsisi"، والـ variable name دلوقتي بيشاور على الـ string الجديد.

ليه C# عملت الـ strings immutable؟
1) الأمان (Safety):
لأن strings بتتشارك أحيانًا بين كذا object أو thread، لو كانت mutable، أي تغيير ممكن يبوظ بيانات تانية.

2) الأداء مع الـ Hashing:
الـ strings بتستخدم كتير في الـ Dictionaries أو الـ HashTables، ووجودها immutable بيخلي الـ hash ثابت طول عمر الـ string.

3) التعامل مع الـ literals:
اللي strings يعني، string interning بتعمل C# ليها نفس القيمة بتتخزن مرة واحدة في الذاكرة. لو كانت mutable، ده كان هيخلي الموضوع معقد جدًا.

لكن فين المشكلة؟

الموضوع كله حلو لحد ما نيجي نعدل على strings كتير في loops أو عمليات متكررة. كل مرة بنعدل فيها string، بيتخلق object جديد، وده ممكن يأثر على الأداء.

الحل؟

للتعامل مع تغييرات كتير على نصوص، الأفضل استخدام:

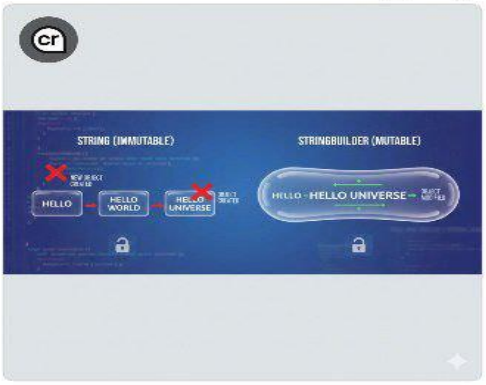
StringBuilder:
مخصوص للتعامل مع النصوص اللي Designed محتاجة تعديل كتير، لأنه بيعدل النصوص على نفس الذاكرة من غير ما يعمل copies كتير.

مثال:

```
StringBuilder sb = new
StringBuilder("Mahmoud");
sb.Append(" Elsisi");
Console.WriteLine(sb.ToString());
```

هنا sb اتغير من غير ما يتخلق object جديد في كل مرة، وده أكثر كفاءة.

الخلاصة:
الـ string immutability في C# ميزة مهمة بتحمي البيانات وبتسهل التعامل مع النصوص في حاجات زي الـ hashing والـ interning، لكن لو هتعمل تعديلات كتير على نصوص، استخدم StringBuilder أو أي طريقة مناسبة لتجنب استهلاك الذاكرة..



2- What's Enum data type, when is it used? And name three common built-in Enums used frequently?

Enum (short for *enumeration*) is a special value type in C# that defines a set of named constants.

It allows you to give meaningful names to numeric values, improving readability and maintainability.

Enums are used when you have a **fixed set of related values**.

Examples:

- Days of the week
- Months of the year
- Status codes (Pending, Approved, Rejected)
- Options for user roles (Admin, User, Guest)

Why use Enums?

- Improves **code readability** (DaysOfWeek.Monday instead of 1)
- Prevents **invalid values**
- Makes **switch statements and logic** easier

Three common built-in Enums in C#

1. **ConsoleColor** – Represents colors for console text and background.

   Ex: Console.ForegroundColor = ConsoleColor.Green;

2. **DayOfWeek** – Represents days of the week.

   Ex: DayOfWeek today = DateTime.Now.DayOfWeek;

3. **DateTimeKind** – Represents time kinds (Local, UTC, Unspecified).

   Ex: DateTime dt = DateTime.UtcNow;

DateTimeKind kind = dt.Kind;

3- what are scenarios to use string Vs StringBuilder?

When to use string:

- Strings are immutable, so every modification creates a new object.
- Best for scenarios where text does not change frequently.

Use cases:

1. Fixed messages or labels:

Ex: string greeting = "Hello World";

2. User input stored once (like username, email).
3. Simple concatenations (only a few + operations).
4. Comparison operations or dictionary keys.

When to use StringBuilder

- StringBuilder is **mutable**, so modifications **happen in-place**.
- Ideal when **many string modifications** are needed (loops, repeated concatenations, inserts, deletes).

Use cases:

1. Building long text dynamically:

Ex:

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 1000; i++)
{
```

```
    sb.Append(i + ", ");
}
```

2. Modifying text frequently: Append, Insert, Remove, Replace.
3. Large-scale reports, logs, or file content construction.