Esercizio Form con Attributi

I vantaggi principali del fare le validazioni con **attributi** invece che con if sparsi nel codice:

---

1. **Separazione delle responsabilità (SRP)**

Le regole di validazione non sono mischiate con la logica di business:

invece di avere metodi pieni di `if (string.IsNullOrEmpty(nome)) throw ...`,

le regole stanno dichiarate direttamente sopra le proprietà del modello.

```
public class Utente {
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Range(18, 120)]
    public int Età { get; set; }
}
```

➡️ Il codice che usa `Utente` si occupa solo di gestire l'oggetto, non di controllarlo.

---

2. **Riusabilità e consistenza**

Una volta definiti (o importati da `System.ComponentModel.DataAnnotations`), gli stessi attributi possono essere riutilizzati ovunque:

- `[Required]`, `[Range]`, `[StringLength]`, `[EmailAddress]` ecc.
- oppure **custom attribute** come `[CodiceFiscaleValido]`.

➡️ Tutti i modelli condividono le stesse regole, evitando duplicazione.

---

3. **Validazione automatica e riflessione**

Framework come ASP.NET Core, Entity Framework o librerie custom possono:

- leggere gli attributi a runtime con **reflection**,
- applicare automaticamente le regole,

- restituire messaggi di errore coerenti.

➡️ Non serve più scrivere validazioni manuali: basta un `TryValidateObject()` e il sistema legge le regole dichiarative.

---

4. **Dichiaratività → codice più leggibile e self-documenting**

Le regole sono *visibili a colpo d'occhio* nel modello:

chi legge la classe capisce subito cosa è obbligatorio o quali vincoli esistono.

È come una "documentazione eseguibile".

---

5. **Testabilità**

Gli attributi rendono facile testare le regole in isolamento.

Puoi scrivere test che verificano che i tuoi modelli abbiano i giusti attributi,

senza dover instanziare servizi o controller.

Con TDD, puoi anche generare i test in base agli attributi.

---

6. **Estendibilità (Open/Closed Principle)**

Puoi creare **attributi personalizzati** per nuove regole senza toccare il codice esistente:

```
[AttributeUsage(AttributeTargets.Property)]
public class CodiceFiscaleValidoAttribute : ValidationAttribute {
    public override bool IsValid(object value) {
        return Regex.IsMatch(value?.ToString() ?? "", @"^[A-Z0-9]{16}$");
    }
}
```

➡️ Aggiungi una nuova regola, non modifichi il validatore base → OCP rispettato.

In sintesi:

> Gli attributi spostano la validazione dal "come" al "cosa",
>
> rendendo il codice più dichiarativo, riutilizzabile e aderente ai principi SOLID.