Compendio Pomodoro C#

Molestone 1 — Timer minimo (SRP)

Teoria

Questa milestone introduce il principio di responsabilità singola (SRP).

L'obiettivo è creare una classe che si occupi solo di una cosa: contare il tempo.

Non deve sapere cosa succede dopo (notifica, salvataggio, ecc.) — solo gestire il countdown.

Implementazione

- Crea una classe TimerService che riceve un ITickProvider nel costruttore.
- ITickProvider incapsula il concetto di "attesa di 1 secondo".
- Il metodo Countdown(int seconds, Action<int> onTick, Action onCompleted) deve:
  1. Chiamare onTick() per ogni secondo rimanente (da seconds a 0).
  2. Dopo il conteggio, eseguire onCompleted().
  3. Usare _tick.Tick() per scandire il tempo (senza Thread.Sleep diretto).
-

Il risultato deve essere un timer riutilizzabile e facile da testare, separato dalla logica dell'app.

Milestone 2 — Notifiche (DIP + ISP)

Teoria

Questa fase riguarda i principi DIP (Dependency Inversion) e ISP (Interface Segregation).

L'obiettivo è introdurre un livello di astrazione per le notifiche: l'app deve "sapere che deve notificare", non "come notificare".

Implementazione

- Crea un'interfaccia INotifier con un metodo Notify(string message).
- Implementa almeno una classe concreta (ConsoleNotifier o BeepNotifier).

- Ogni notifica sarà iniettata nelle classi che la useranno (non istanziata direttamente).

Il vantaggio è che in futuro potrai sostituire ConsoleNotifier con, ad esempio, una EmailNotifier, senza cambiare la logica del timer.

Milestone 3 — Strategie di sessione (OCP)

Teoria

Questa milestone introduce il principio OCP (Open/Closed Principle): il codice dev'essere aperto all'estensione ma chiuso alla modifica.

Nel contesto del Pomodoro, significa poter aggiungere nuove strategie di sessione (durata focus/break) senza modificare il codice esistente.

Implementazione

- Crea l'interfaccia ISessionStrategy con un metodo GetDurations() che restituisce (focusSec, breakSec).
- Implementa alcune strategie: Classic25_5, Deep50_10 e Custom.
- Ogni strategia incapsula la propria logica di durata.

Così l'app potrà funzionare con strategie intercambiabili, rispettando OCP e favorendo la scalabilità.

Milestone 4 — Evento di completamento (Observer Pattern)

Teoria

Qui si introduce il concetto di eventi e il pattern Observer:

quando una sessione termina, il sistema deve notificare altri componenti interessati (es. repository, log, UI).

Implementazione

- All'interno di Pomodoro, dichiara un evento PomodoroCompleted.
- Quando la sessione di focus termina, invoca questo evento.

- Chi è interessato (ad esempio CsvSessionRepository) può “iscriversi” all’evento per reagire di conseguenza.

Questo approccio riduce l’accoppiamento: Pomodoro non sa chi ascolta né cosa farà.

Milestone 5 — Persistenza su file (SRP + DIP)

Teoria

Applichiamo ancora SRP e DIP: la logica di salvataggio deve essere separata dal resto dell’app e iniettata come dipendenza.

Pomodoro o il suo runner non devono sapere come i dati vengono salvati, ma solo che verranno salvati.

Implementazione

- Definisci un’interfaccia ISessionRepository con un metodo Save(SessionLog log).
- Implementa una classe CsvSessionRepository che scrive i log su file.
- Il formato sarà semplice, es. 2025-01-01T12:00:00.0000000Z,focus.
- Sottoscrivi l’evento PomodoroCompleted per salvare i log quando la sessione finisce.

Questo garantisce modularità e testabilità: puoi sostituire il repository con una versione in memoria o su database senza modificare il codice principale.

Milestone 6 — Runner (coordinamento)

Teoria

Il Runner è la parte che orchestra il flusso del Pomodoro: focus + break.

Segue il principio SRP, avendo la sola responsabilità di coordinare le fasi in base alla strategia scelta.

Implementazione

- Crea una classe PomodoroRunner che riceve Pomodoro, TimerService e INotifier.

- Nel metodo Run(ISessionStrategy strategy), esegui:
    1. Una sessione di focus.
    2. Una pausa (break) con countdown separato.
-
- Usa il notifier per i messaggi di inizio/fine sessione.

Questo è il “collante” del progetto: unisce logica, notifiche e strategie in un flusso coerente.

Milestone 7 — Progress bar (optional “wow”)

Teoria

Milestone opzionale ma d’effetto.

Serve per dare un feedback visivo all’utente, mostrando l’avanzamento del tempo direttamente in console.

Implementazione

- Durante il countdown, disegna una barra di progresso (es. ########----).
- Aggiornala in base al tempo rimanente.
- Usa la console in modalità “riscrittura” (\r) per animare la barra.

Questo tocco finale rende il progetto più interattivo e mostra l’uso creativo delle API console.

Milestone Extra — Dependency Injection “reale”

Teoria

In questa estensione, si introduce la DI container nativa di .NET.

Serve a far capire come i principi SOLID si applicano anche a progetti professionali.

Implementazione

- Usa Microsoft.Extensions.DependencyInjection.

- Registra i servizi (INotifier, ISessionRepository, TimerService, Pomodoro).
- Crea il ServiceProvider e risolvi il PomodoroRunner.

Questo collega l'esercizio pratico a concetti aziendali come IoC Container, scopes e lifetime.

Conclusione

Completando le milestone, gli studenti avranno:

- Applicato tutti i principi SOLID in un contesto reale.
- Costruito un progetto completo, testabile e scalabile.
- Capito come si collegano pattern classici (Observer, Strategy) con l'architettura moderna di .NET.