# Q&A

## 1. Fondamenti e Tipi di Dato

**1. Cos'è un value type in C#?**

Un *value type* memorizza direttamente il suo valore nello stack. Quando lo assegni a un'altra variabile, viene creata una copia indipendente (es. `int`, `double`, `struct`).

**2. Cos'è un reference type in C#?**

Un *reference type* memorizza un riferimento a un oggetto nello heap. Due variabili che fanno riferimento allo stesso oggetto condividono i dati (es. `class`, `array`, `string`).

**3. Differenza tra shallow copy e deep copy?**

La *shallow copy* copia solo il riferimento (entrambe le variabili puntano allo stesso oggetto), la *deep copy* crea un nuovo oggetto duplicando i dati.

**4. Cosa succede se assegni un array a un altro array?**

Entrambi punteranno allo stesso oggetto in memoria; modificare un elemento in uno modificherà anche l'altro.

---

## 2. Collezioni (Collections)

**5. Quando usare una `List<T>`?**

Quando serve una lista ordinata e dinamica con accesso per indice e inserimenti in coda.

**6. Quando usare un `Dictionary<TKey,TValue>`?**

Quando serve un lookup rapido per chiave univoca.

**7. Quando usare una `Queue<T>`?**

Per gestire sequenze FIFO (First In, First Out), come una coda di richieste.

**8. Quando usare uno `Stack<T>`?**

Per strutture LIFO (Last In, First Out), ad esempio undo/redo o valutazioni di espressioni.

**9. Quando usare un `HashSet<T>`?**

Quando servono elementi unici e controlli rapidi di appartenenza (no duplicati).

---

# 3. LINQ

### 10. Cos'è LINQ?

È un linguaggio integrato che permette di effettuare query su collezioni in modo dichiarativo, leggibile e tipizzato.

### 11. Cosa fa `Where()`?

Filtra gli elementi di una collezione in base a un predicato logico.

### 12. Cosa fa `Select()`?

Proietta ogni elemento in una nuova forma o tipo (trasformazione).

### 13. Cosa fa `FirstOrDefault()`?

Restituisce il primo elemento che soddisfa una condizione o `default` se nessuno la soddisfa.

### 14. Le query LINQ modificano la collezione originale?

No, restituiscono una nuova sequenza (`IEnumerable<T>`).

---

# 4. Delegati ed Eventi

### 15. Cos'è un delegato in C#?

Un tipo che rappresenta un riferimento a uno o più metodi con una certa firma.

### 16. Cosa sono `Action` e `Func`?

Delegati generici pronti all'uso:

- `Action` non ritorna nulla (`void`);
- `Func<T>` ritorna un valore.

### 17. Cos'è un evento in C#?

Un meccanismo per notificare più listener quando avviene un'azione, basato su delegati.

### 18. Come si solleva un evento?

Con `Evento?.Invoke()`, verificando che ci siano sottoscrittori.

### 19. Cosa rappresenta il pattern Observer?

Un modello in cui un *subject* notifica automaticamente più *observer* ai cambiamenti di stato.

---

# 5. Async, Await e Task

### 20. Cosa fa `async` e `await`?

Permette di scrivere codice asincrono non bloccante in modo sequenziale.

### 21. Quando usare async/await?

Per operazioni I/O-bound (rete, file, DB) che altrimenti bloccherebbero il thread principale.

### 22. Differenza tra `Task` e `Thread`?

`Task` è un'astrazione di alto livello che gestisce in automatico i thread del pool. `Thread` è una gestione manuale a basso livello.

### 23. Perché evitare `.Wait()` o `.Result` su un Task?

Possono bloccare il thread o causare deadlock. Usa sempre `await`.

### 24. Cosa fa `ConfigureAwait(false)`?

Evita di catturare il contesto di sincronizzazione, migliorando le performance in app non UI.

---

# 6. TDD (Test-Driven Development)

### 25. Cos'è il TDD?

È una metodologia di sviluppo in cui si scrivono prima i test (che falliscono), poi il codice minimo per farli passare, poi si fa refactoring.

### 26. Quali sono le fasi del ciclo TDD?

Red – Green – Refactor:

1. Scrivi un test che fallisce
2. Implementa codice minimo per farlo passare
3. Pulisci il codice mantenendo i test verdi.

### 27. Cosa si intende per "testare in isolamento"?

Verificare una singola unità di codice senza dipendere da risorse esterne (DB, rete, file).

**28. Cos’è un test doppio (test double)?**

Un oggetto fittizio usato nei test (stub, mock, fake).

**29. Cos’è Moq?**

Una libreria di *mocking* per .NET che permette di creare facilmente oggetti fake di interfacce e verificarne le chiamate.

**30. Cos’è lo schema AAA nei test?**

Arrange – Act – Assert: prepara, agisci, verifica.

---

# 7. Principi SOLID

**31. Cosa afferma il principio SRP?**

Ogni classe deve avere una sola responsabilità e un solo motivo per cambiare.

**32. Cosa afferma OCP?**

Il codice deve essere aperto all’estensione ma chiuso alla modifica.

**33. Cosa afferma LSP?**

Le classi derivate devono poter sostituire le base senza alterare il comportamento atteso.

**34. Cosa afferma ISP?**

Meglio più interfacce specifiche che una sola interfaccia “grassa”.

**35. Cosa afferma DIP?**

Le classi di alto livello devono dipendere da astrazioni, non da implementazioni concrete.

---

# 8. Design Pattern GoF

**36. Cos’è il Singleton Pattern?**

Garantisce che esista una sola istanza di una classe e fornisce un accesso globale a essa.

**37. Cos’è il Factory Method Pattern?**

Definisce un’interfaccia per creare oggetti, lasciando alle sottoclassi la decisione su quale istanziare.

**38. Cos’è l’Abstract Factory Pattern?**

Fornisce un’interfaccia per creare famiglie di oggetti correlati (es. GUI Windows/macOS).

**39. Cos’è lo Strategy Pattern?**

Permette di scegliere dinamicamente un algoritmo tra diversi intercambiabili.

**40. Cos’è l’Observer Pattern?**

Stabilisce una relazione uno-a-molti: quando il soggetto cambia stato, notifica tutti gli osservatori.

**41. Cos’è il Decorator Pattern?**

Aggiunge dinamicamente funzionalità a un oggetto avvolgendolo in un decoratore.

**42. Cos’è l’Adapter Pattern?**

Permette a classi con interfacce incompatibili di collaborare tramite un adattatore.

**43. Cos’è il Facade Pattern?**

Fornisce un’interfaccia unificata e semplice per accedere a un sottosistema complesso.

**44. Cos’è il Command Pattern?**

Incapsula una richiesta come oggetto, permettendo di parametrizzare o annullare operazioni.

---

# 9. Inversione di Controllo e Dependency Injection

**45. Cos’è l’Inversione di Controllo (IoC)?**

È il principio per cui non è l’applicazione a controllare il flusso, ma un framework o container esterno.

**46. Cos’è la Dependency Injection (DI)?**

È una forma di IoC in cui le dipendenze vengono fornite dall’esterno (tramite costruttore, proprietà o container).

**47. Differenza tra IoC e DI?**

IoC è il principio generale, DI è un modo specifico per implementarlo nella gestione delle dipendenze.

**48. Cos’è un contenitore IoC (DI Container)?**

È un sistema che gestisce la creazione e l'iniezione automatica delle dipendenze (es. `ServiceProvider` in .NET Core).

**49. Cos'è il Service Locator?**

Un registro centrale da cui i componenti ottengono servizi; considerato meno chiaro rispetto alla DI esplicita.

---

# 10. Applicazione: Pomodoro Project

**50. Quali principi SOLID applica il progetto Pomodoro?**

- SRP nel Timer e Runner
- OCP nelle strategie di sessione
- DIP nelle notifiche e repository
- Observer nel completamento delle sessioni
- DI nella configurazione finale

---