# C# OOP, SOLID, and Design Fundamentals Cheat Sheet

## OOP Basics

- Encapsulation: Bundle data + methods, hide internal state via access modifiers.
- Inheritance: Share behavior between base and derived classes.
- Polymorphism: Override methods or use interfaces for different behaviors.
- Abstraction: Hide complexity, expose essential features only.

## OOP Advanced

- Favor composition over inheritance for flexibility.
- Use interfaces and dependency injection to decouple implementations.
- Use sealed classes to prevent further inheritance where needed.

## SOLID Principles

- S: Single Responsibility - one reason to change.
- O: Open/Closed - open for extension, closed for modification.
- L: Liskov Substitution - derived types must be substitutable.
- I: Interface Segregation - use smaller, specific interfaces.
- D: Dependency Inversion - depend on abstractions, not concretions.

## Static vs Readonly vs Const

- const: Compile-time constant. Implicitly static. Must be assigned at declaration.
- readonly: Assigned at declaration or in constructor. Runtime constant.
- static: Shared across all instances. Belongs to the type, not the instance.

## Dependency Injection (DI)

- Design pattern to inject dependencies instead of hardcoding them.
- Use constructor injection for required dependencies.
- Use Microsoft.Extensions.DependencyInjection for built-in DI in ASP.NET Core.
- Promotes testability, modularity, and decoupling.

## Composition

- 'Has-a' relationship (e.g., Car has Engine).
- Preferable over inheritance for flexibility and loose coupling.
- Enables runtime changes in behavior using strategy or decorator pattern.

## Design Best Practices

- Minimize class responsibilities.
- Prefer interfaces for dependencies.
- Avoid deep inheritance trees.
- Use SOLID + DRY + YAGNI principles.
- Unit test business logic independently of infrastructure.