The SOLID Principles

Mel Grubb
Developer

@melgrubb | www.melgrubb.com

The SOLID Principles

S	Single Responsibility Principle
O	Open-Closed Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

For More Information

SOLID Principles for C# Developers

Steve Smith

Overview

Introduction to the SOLID principles

Examine influence on design decisions

Compare that influence with pillars

Single Responsibility Principle

Single Responsibility Principle

“A class should have one, and only one, reason to change.”

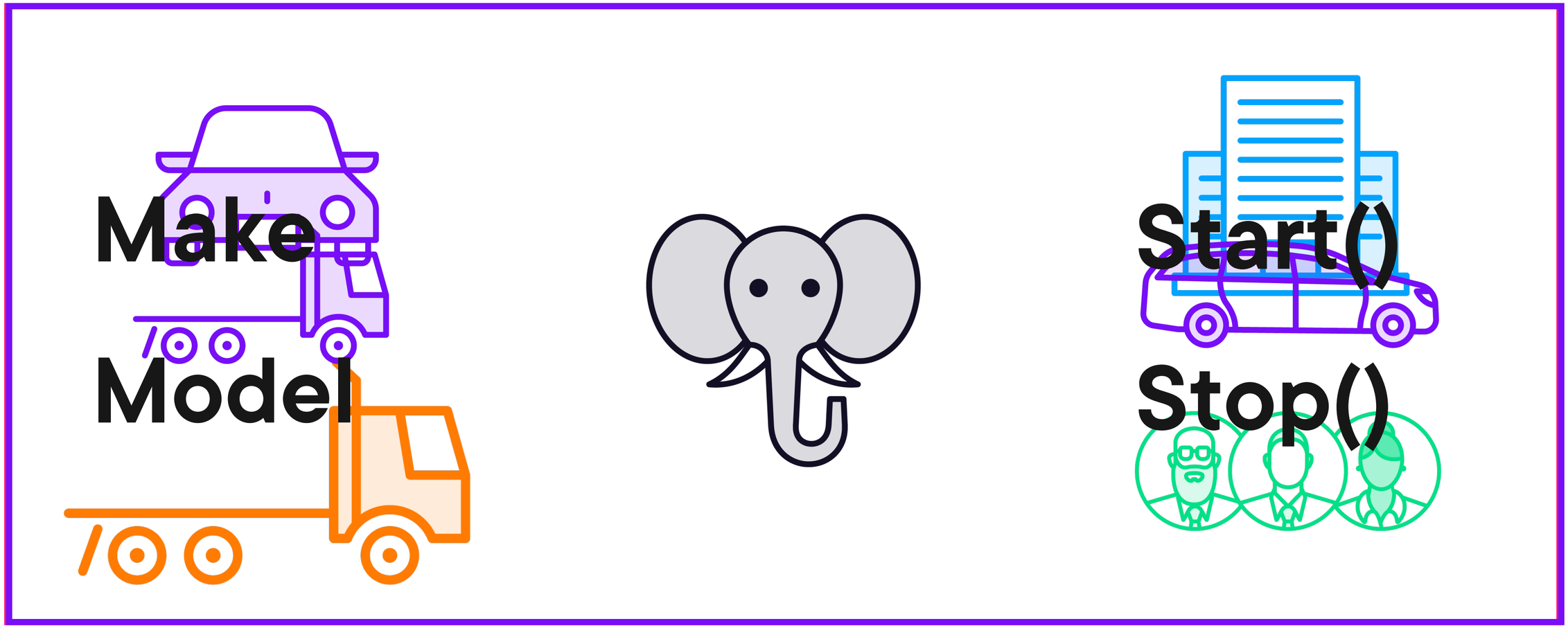
or

“There should never be more than one reason for a class to change.”

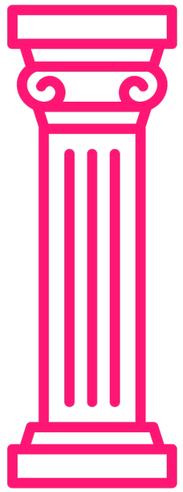
Single Responsibility Principle

“A module should have a clearly-defined job”

Scopes of Responsibility



The Single Responsibility Principle



Abstraction

Encapsulation

Inheritance

Polymorphism

Design Tips

- Identify “responsibilities” and “jobs”
- Watch out for “reasons to change”
- Favor multiple small classes over fewer, more complex ones

Open/Closed Principle

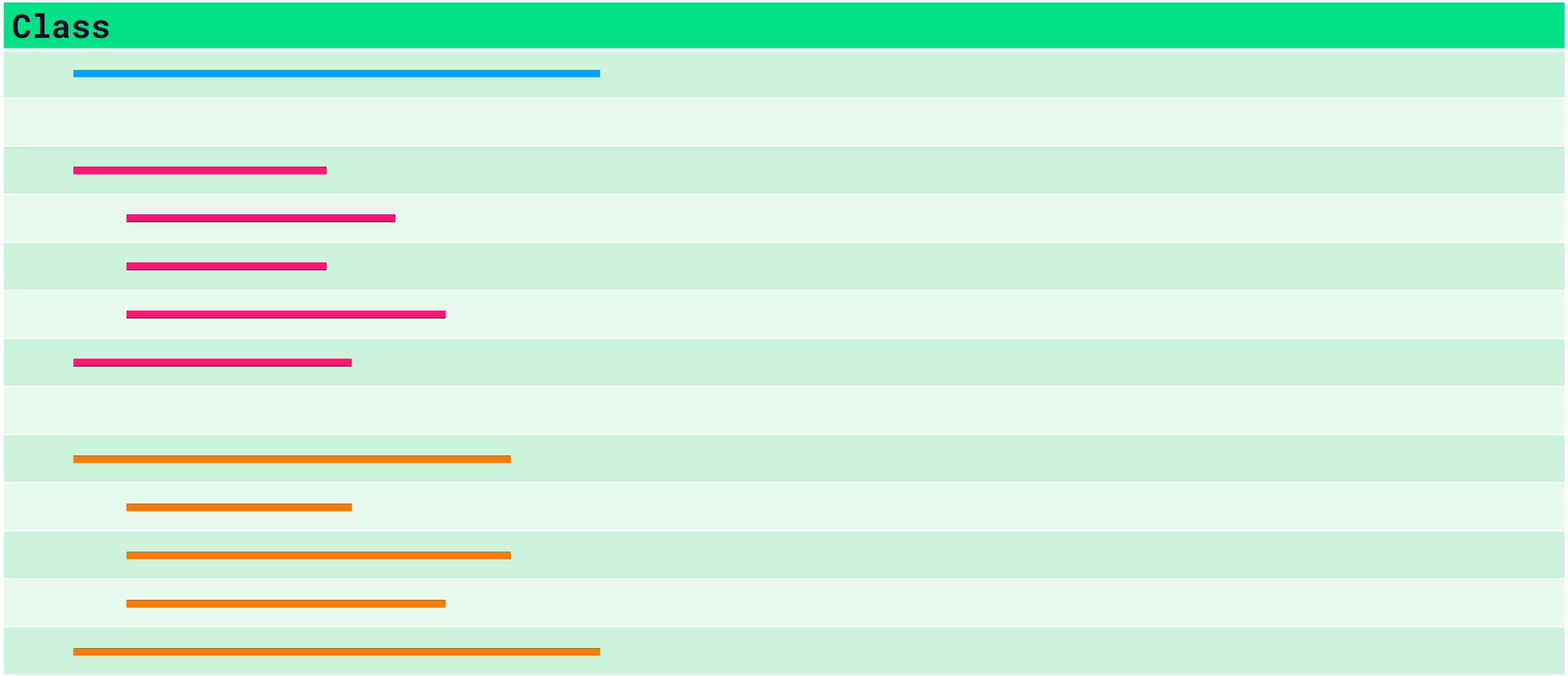
Open/Closed Principle

“Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”

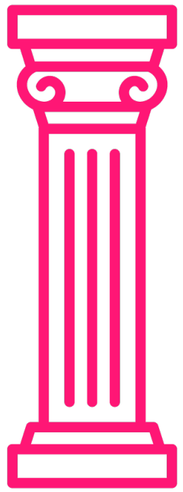
Open/Closed Principle

“You should be able to extend a class’s behavior without modifying it.”

Code Changes

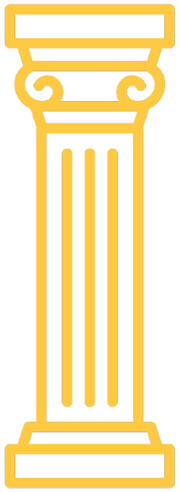


The Open/Closed Principle

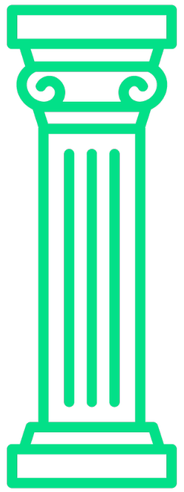


Abstraction

Encapsulation



Inheritance



Polymorphism

Design Tips

- Separate “bedrock” code into base classes
- Implement specializations on top of that bedrock
- Watch for classes with multiple “modes”
- Break these into smaller classes

Liskov Substitution Principle

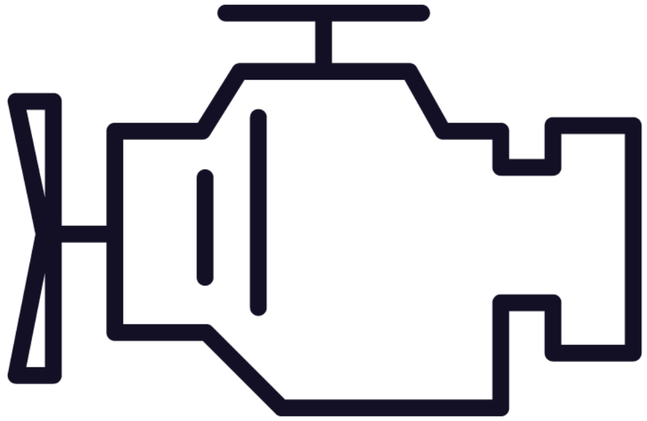
Liskov Substitution Principle

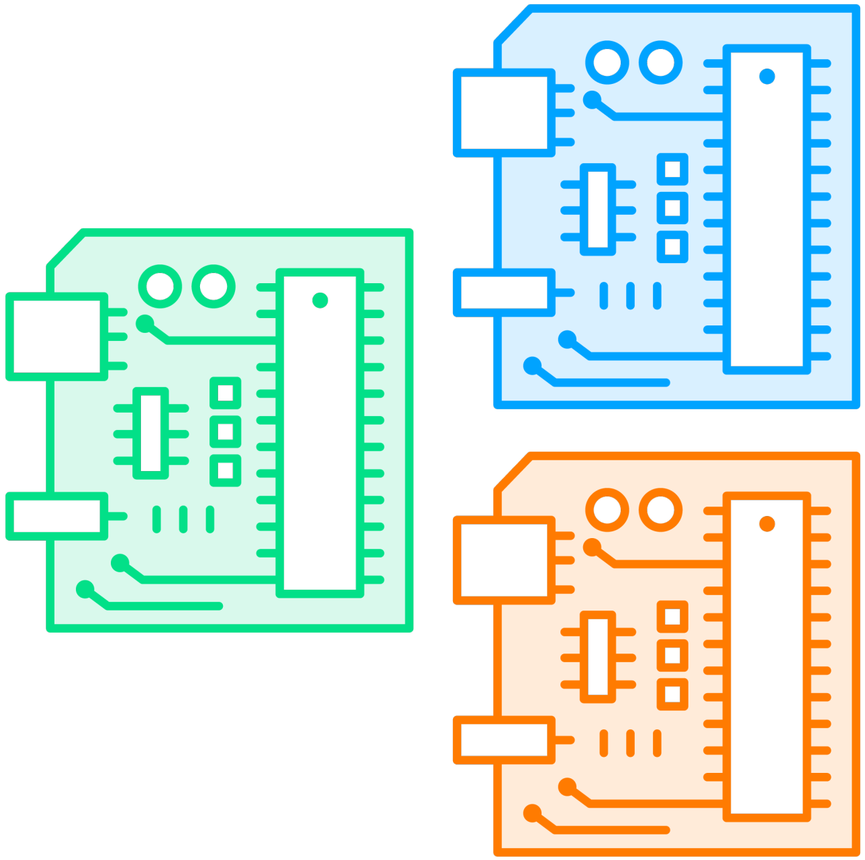
“Let φ(x) be a property provable about objects x of type T. Then φ (y) should also be true for objects y of type S where S is a subtype of T.”

Liskov Substitution Principle

“Derived types must be substitutable for their base types.”

Liskov Substitution Principle



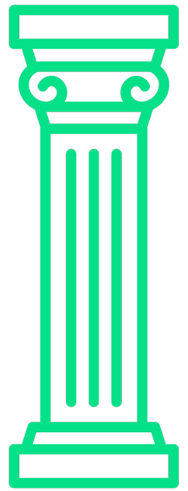


The Liskov Substitution Principle

Abstraction

Encapsulation

Inheritance



Polymorphism

Design Tips

- Test subtypes individually
- Test that they each perform all tasks
- Watch out for NotImplementedExceptions

Interface Segregation Principle

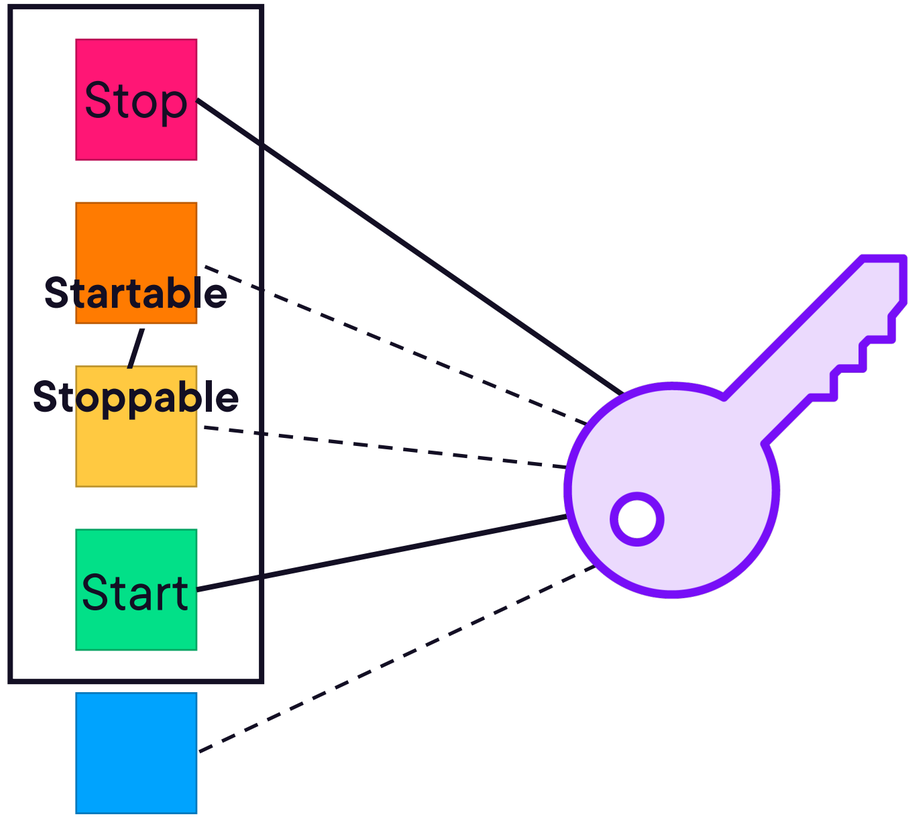
Interface Segregation Principle

“Clients should not be forced to depend upon interfaces that they do not use.”

or

“Make fine-grained interfaces that are client-specific.”

Interface Segregation Principle

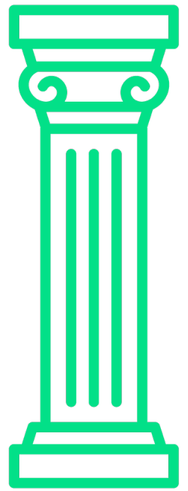


The Interface Segregation Principle

Abstraction

Encapsulation

Inheritance



Polymorphism

Design Tips

- Create interfaces for a class's abilities
- Group related functionality together
- Keep interfaces cohesive

Dependency Inversion Principle

Dependency Inversion Principle

“High-level modules should not depend on low-level modules.

Both should depend on abstractions.”

or

“Depend on abstractions, not concretions.”

Dependency Inversion Principle
!= Dependency Injection
!= Inversion of Control Container

Dependency Injection and IoC containers are just implementation details

Dependency Inversion Principle

Don’t depend on specific classes

Don’t instantiate your own dependencies

Prefer dependencies as interfaces

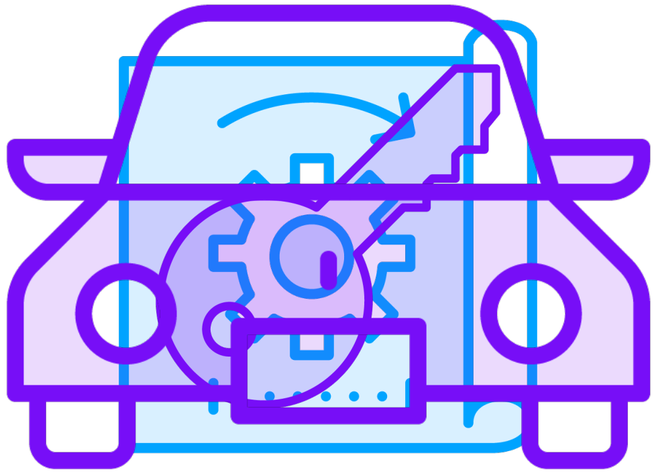
...provided from an outside source

For more information

C# Dependency Injection

Henry Been

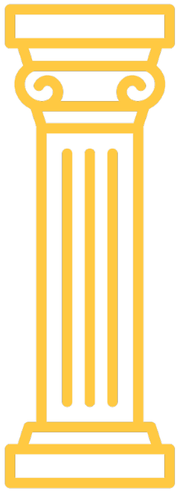
IoC Containers



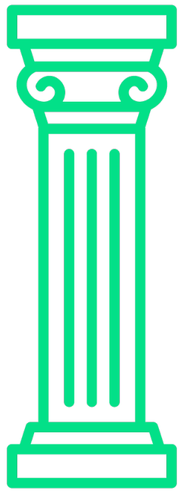
The Dependency Inversion Principle

Abstraction

Encapsulation



Inheritance



Polymorphism

Design Tips

- Don’t instantiate your own dependencies
- Require them as constructor parameters…
- …or inject them as properties
- Consider using an IoC container

Summary

The principles justify the pillars

They span multiple pillars

They can overlap each other

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Introduction to Design Patterns