Open / Closed Principle

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS
@ardalis | ardalis.com | weeklydevtips.com

SOLID Principles

SRP

Single Responsibility Principle

OCP

Open / Closed Principle

LSP

Liskov Substitution Principle

Interface Segregation Principle

ISP

Dependency Inversion Principle

DIP

Open / Closed Principle

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

Dr. Bertrand Meyer originated the term in his 1988 book, *Object-Oriented Software Construction*

It should be possible to change the behavior of a method without editing its source code.

Open / Closed

Open to extension

New behavior can be added in the future

Code that is closed to extension has fixed behavior

Closed to modification

Changes to source or binary code are not required

The only way to change the behavior of code that is closed to extension is to change the code itself

Why Should Code Be Closed to Modification?

Less likely to introduce bugs in code we don't touch or redeploy

Less likely to break dependent code when we don't have to deploy updates

Fewer conditionals in code that is open to extension results in simpler code

Bug fixes are ok

How Do We Add Another Policy Type?

```
switch (policy.Type)
{
    case PolicyType.Auto:

    case PolicyType.Land:

    case PolicyType.Life:
}
```

Balance abstraction and concreteness

Abstraction adds complexity

Predict where variation is needed and apply abstraction as needed

Extremely Concrete

```
public class DoOneThing
{
    public void Execute()
    {
        Console.WriteLine(“Hello world.”);
    }
}
```

Extremely Concrete

```
public class DoSomethingElse
{
    public void SomethingElse()
    {
        var doThing = new DoOneThing();
        doThing.Execute();
        // other stuff
    }
}
```

Extreme Extensibility

```
public class DoAnything<TArg, TResult>
{
    private Func<TArg, TResult> _function;
    public DoAnything(Func<TArg, TResult> function)
    {
        _function = function;
    }
    public TResult Execute(TArg a)
    {
        return _function(a);
    }
}
```

Extreme Extensibility

```
public abstract class DoAnything<TResult, TArg>
{
    public abstract TResult Execute(TArg a);
}
```

How Can You Predict Future Changes?

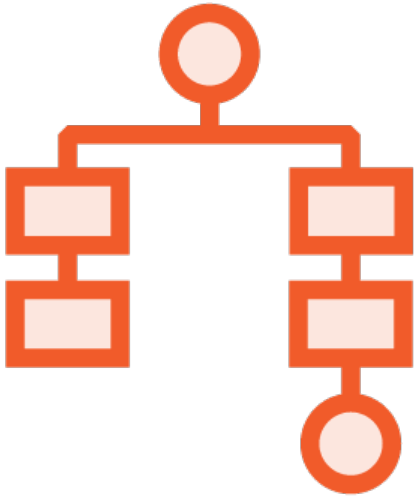
Start concrete

Modify the code the first time or two

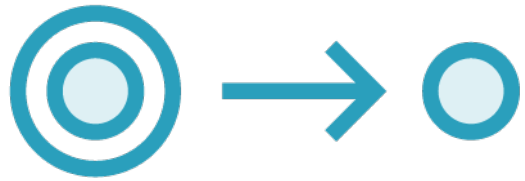
By the third modification, consider making the code open to extension for that axis of change

Typical Approaches to OCP

Parameters



Inheritance



Composition / Injection

Extremely Concrete

```
public class DoOneThing
{
    public void Execute()
    {
        Console.WriteLine(“Hello world.”);
    }
}
```

Parameter-Based Extension

```
public class DoOneThing
{
    public void Execute(string message)
    {
        Console.WriteLine(message);
    }
}
```

Inheritance-based Extension

```
public class DoOneThing
{
    public virtual void Execute()
    {
        Console.WriteLine(“Hello world.”);
    }
}
public class DoAnotherThing
{
    public override void Execute()
    {
        Console.WriteLine(“Goodbye world!”);
    }
}
```

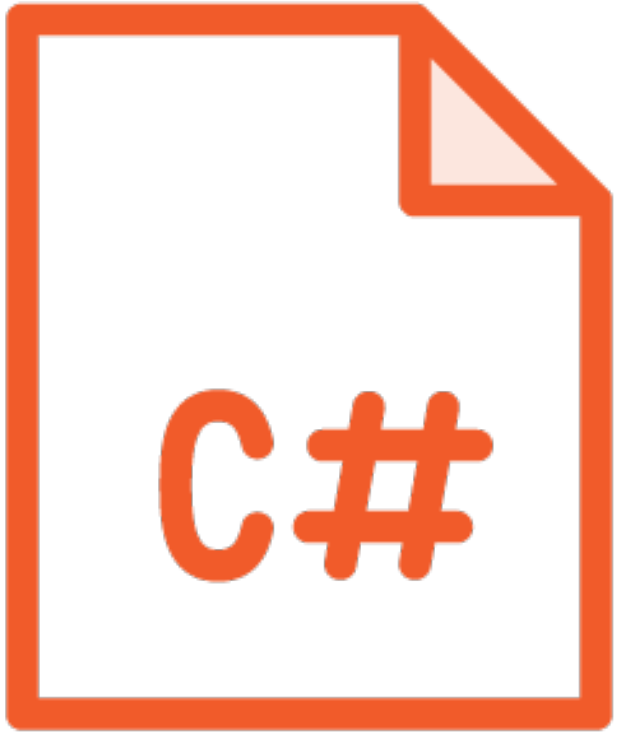
Composition/Injection Extension

```
public class DoOneThing
{
    private readonly MessageService _messageService;
    public DoOneThing(MessageService messageService)
        => _messageService = messageService;

    public void Execute()
    {
        Console.WriteLine(_messageService.GetMessage());
    }
}
```

Prefer implementing
new features in new classes.

Why Use a New Class?



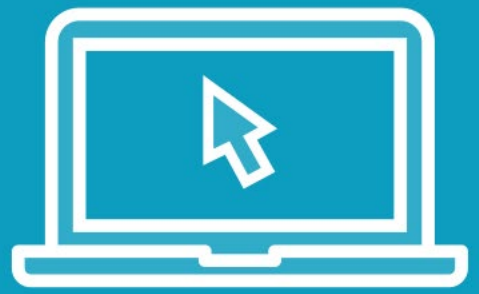
- Design class to suit problem at hand
- Nothing in current system depends on it
- Can add behavior without touching existing code
- Can follow Single Responsibility Principle
- Can be unit-tested

Learn More

Maintain Legacy Code with New Code

- weeklydevtips.com/015

Demo



Applying OCP to RatingService

Available at
https://github.com/ardalis/solidsample

Packages and Libraries

Closed for modification

- Consumers cannot change package contents

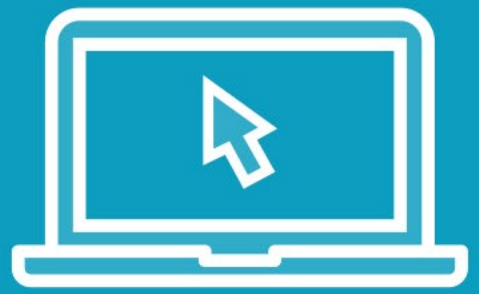
Closed for modification

- Should not break consumers when new behavior is added

Open to extension

- Consumers should be able to extend the package to suit their own needs

Demo



NuGet package example: Guard Clauses

Available at
https://github.com/ardalis/guardclauses

More Resources on OCP

Why you need to know OCP but don't

- https://bit.ly/2LSXOuo

Open Closed Principle by Robert Martin

- https://bit.ly/2Gmxg1Z

Open Closed Principle by Jon Skeet

- https://bit.ly/2AMmprC

SOLID Principles

Single Responsibility Principle

Open / Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Key Takeaways

Solve the problem first using simple, concrete code

Identify the kinds of changes the application is likely to continue needing

Modify code to be extensible along the axis of change you've identified

- Without the need to modify its source each time