Fundamentals: The Open / Closed Principle

Steve Smith

http://pluralsight.com/

Outline

- OCP Defined
- The Problem
- An Example
- Refactoring to Apply OCP
- Related Fundamentals

OCP: The Open/Closed Principle

The Open / Closed Principle states that software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

Wikipedia

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

The Open / Closed Principle

Open to Extension

- New behavior can be added in the future

Closed to Modification

- Changes to source or binary code are not required

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

Change behavior without changing code?

Rely on abstractions

No limit to variety of implementations of each abstraction

In .NET, abstractions include:

- Interfaces
- Abstract Base Classes

In procedural code, some level of OCP can be achieved via parameters

Demo

A Price Calculator That Is Not Closed To Change

The Problem

- **Adding new rules require changes to the calculator every time**
- **Each change can introduce bugs and requires re-testing, etc.**
- **We want to avoid introducing changes that *cascade* through many modules in our application**
- **Writing new classes is less likely to introduce problems**
  - Nothing depends on new classes (yet)
  - New classes have no legacy coupling to make them hard to design or test

Three Approaches to Achieve OCP

- **Parameters (Procedural Programming)**
  - Allow client to control behavior specifics via a parameter
  - Combined with delegates/lambda, can be very powerful approach

- **Inheritance / Template Method Pattern**
  - Child types override behavior of a base class (or interface)

- **Composition / Strategy Pattern**
  - Client code depends on abstraction
  - Provides a “plug in” model
  - Implementations utilize Inheritance; Client utilizes Composition

Demo

Refactoring to a Better Design

When do we apply OCP?

- **Experience Tells You**

  If you know from your own experience in the problem domain that a particular class of change is likely to recur, you can apply OCP up front in your design

Otherwise – "Fool me once, shame on you; fool me twice, shame on me"

- **Don't apply OCP at first**
- **If the module changes once, accept it.**
- **If it changes a second time, refactor to achieve OCP**

Remember *TANSTAAFL*

- ***There Ain't No Such Thing As A Free Lunch***
- **OCP adds complexity to design**
- **No design can be closed against all changes**

Summary

- **Conformance to OCP yields flexibility, reusability, and maintainability**
- **Know which changes to guard against, and resist premature abstraction**
- **Related Fundamentals:**
  - Single Responsibility Principle
  - Strategy Pattern
  - Template Method Pattern
- **Recommended Reading:**
  - Agile Principles, Patterns, and Practices by Robert C. Martin and Micah Martin [http://amzn.to/agilepppcsharp]

Credits

- **Images Used Under License**
  - http://www.lostechies.com/blogs/derickbailey/archive/2009/02/11/solid-development-principles-in-motivational-pictures.aspx

For more in-depth online developer training visit

pluralsight
see what you can learn

on-demand content from authors you trust