Design Patterns - Summary

Semester: Spring 2022

Version: 0.1.0
Date: 2022-04-14 10:49

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Contents

1 Introduction

1.1 Most important quotes

In the introduction the basics of object-oriented programming are explained. Therefore, for many this is easy to read. However, some important statements are made.

Favor object composition over class inheritance

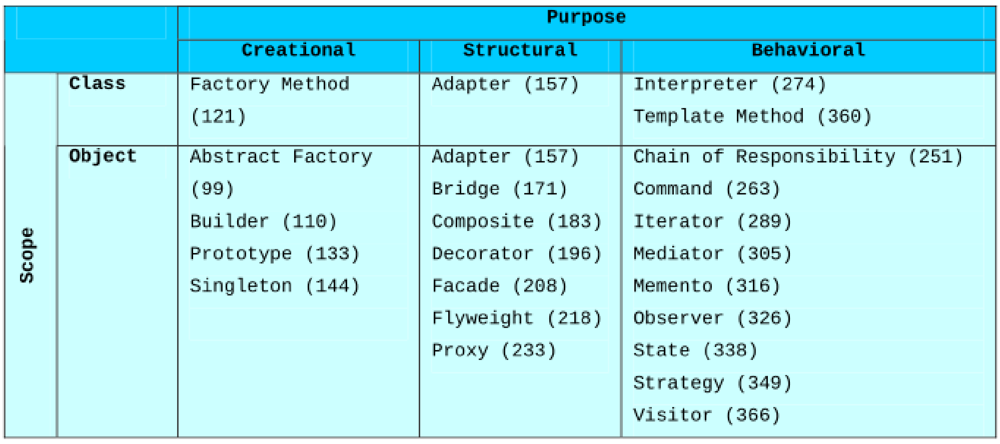
Because it is very difficult to make the correct abstraction often the base class(es) are not complete or have too much in it. With object composition you don't have this problem.

Program to an interface, not an implementation

If you implement against interfaces you can test your class with unit tests. If you implement against a fixed implementation you can not change this behavior at anytime.

1.2 The design patterns

Class Design Patterns deal with the relationships between classes and their subclasses. The Object Design Patterns with the object relationships which can be changed at runtime.



		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method (121)	Adapter (157)	Interpreter (274) Template Method (360)
	Object	Abstract Factory (99) Builder (110) Prototype (133) Singleton (144)	Adapter (157) Bridge (171) Composite (183) Decorator (196) Facade (208) Flyweight (218) Proxy (233)	Chain of Responsibility (251) Command (263) Iterator (289) Mediator (305) Memento (316) Observer (326) State (338) Strategy (349) Visitor (366)

Figure 1: Design pattern space

1.3 The Most common causes of redesign

A list of the most common causes of redesign and how you could solve the problem.

1. Creating an object by specifying a class explicitly.
   Try to avoid creating objects explicitly otherwise you are bound to this decision.
   Design Pattern - Abstract Factory, Design Pattern - Factory Method, Design Pattern - Prototype
2. Dependence on specific operations
   Try to avoid to hard code how you want to satisfy a request. Example: Do not hard code how a specific button should perform.
   Design Pattern - Chain of Responsibility, Design Pattern - Command

3. Dependence on hardware and software platform
   Try to avoid implementing against a specific HW or API.
   Example: Linux & Windows does not have the same Windows System. Provide an abstraction in which the underlying system is not relevant.
   Design Pattern - Abstract Factory, Design Pattern - Bridge
4. Dependence on object representations or implementations
   You should never bother how the inside of class works while using it. Otherwise, you might need to change things when the class is changing its inner live.
   Design Pattern - Builder, Design Pattern - Iterator, Design Pattern - Strategy, Design Pattern - Template Method, Design Pattern - Visitor
5. Algorithmic dependencies
   Do not hard code the algorithm. Try to encapsulate the algorithm in a class. Then you can replace it every time you want.
   Design Pattern - Builder, Design Pattern - Iterator, Design Pattern - Strategy, Design Pattern - Template Method, Design Pattern - Visitor
6. Tight coupling
   Try to avoid tight coupling. Otherwise, it is hard to reuse the class.
   Design Pattern - Abstract Factory, Design Pattern - Bridge, Design Pattern - Chain of Responsibility, Design Pattern - Command, Design Pattern - Facade, Design Pattern - Mediator, Design Pattern - Observer
7. Extending functionality by subclassing
   This is very difficult to made it right. Instead, use object composition to extend functionality.
   Design Pattern - Bridge, Design Pattern - Chain of Responsibility, Design Pattern - Composite, Design Pattern - Decorator, Design Pattern - Observer, Design Pattern - Strategy
8. Inability to alter classes conveniently
   Sometimes you can not modify a class (for example closed source library). The following Design Patterns can help to work around this problem.
   Design Pattern - Adapter, Design Pattern - Decorator, Design Pattern - Visitor