Design Patterns - Summary

Semester: Spring 2022

Version: 0.1.0
Date: 2022-05-10 16:43

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

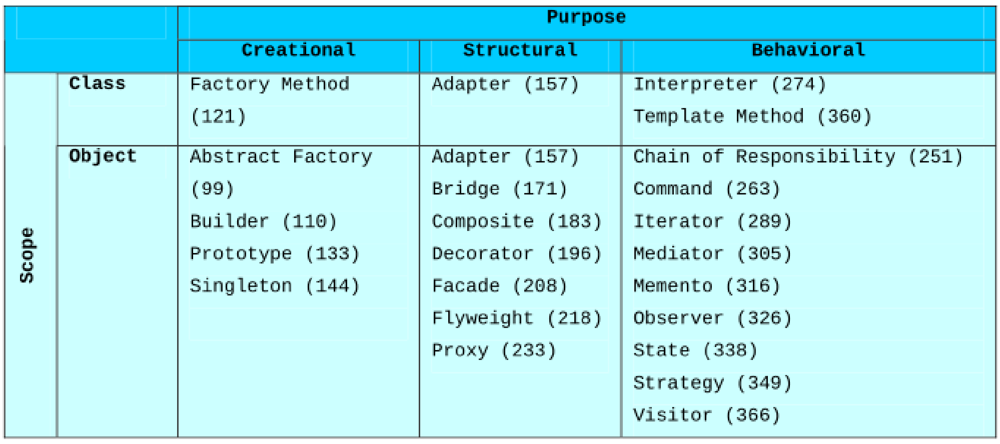
Because it is very difficult to make the correct abstraction often the base class(es) are not complete or have too much in it. With object composition you don't have this problem.

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

2 Creational

2.1 Abstract Factory

The Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete class. It is a good choice, when you want to create big class hierarchy from a factory. In general the Simple Factory (Design Pattern - Simple Factory) is more common today.

Motivation:

- You want to create concrete objects without knowing which concrete type you get.
- You are only interested in the interface.

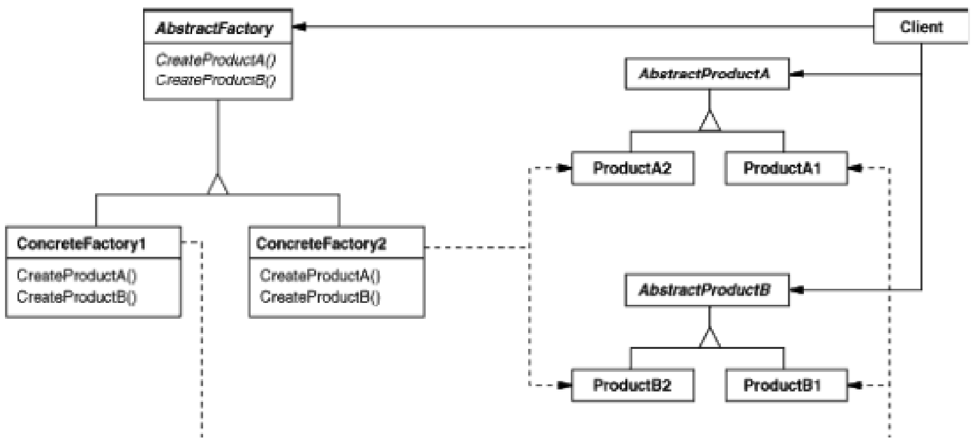


Figure 2: Abstract Factory Class Diagram

2.1.1 Important Notes

The Factory must be created at the very beginning of the application. After creation only this factory must be used to create objects. Otherwise, the benefit of the factory are gone.

2.2 Builder

The Builder Pattern seperates the construction of a complex class from its representation. Therby you can hidde the creation of complex objects behind functions (see listening 1).

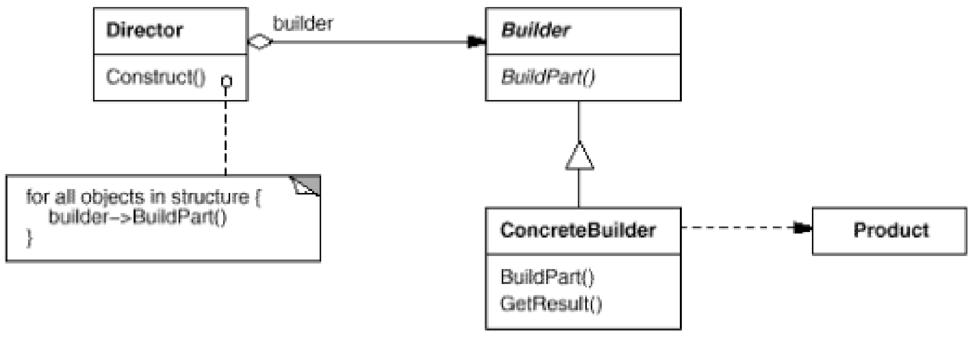


Figure 3: Builder Class Diagram

```
Maze* MazeGame::CreateMaze () {
  Maze* aMaze = new Maze;
  Room* r1 = new Room(1);
  Room* r2 = new Room(2);
  Door* theDoor = new Door(r1, r2);
  aMaze->AddRoom(r1);
  aMaze->AddRoom(r2);
  r1->SetSide(North, new Wall);
  r1->SetSide(East, theDoor);
  r1->SetSide(South, new Wall);
  r1->SetSide(West, new Wall);
  r2->SetSide(North, new Wall);
  r2->SetSide(East, new Wall);
  r2->SetSide(South, new Wall);
  r2->SetSide(West, theDoor);
  return aMaze;
}

// With Builder Pattern
Maze* MazeGame::CreateMaze (MazeBuilder& builder) {
  builder.BuildMaze();
  builder.BuildRoom(1);
  builder.BuildRoom(2);
  builder.BuildDoor(1, 2);
  return builder.GetMaze();
}
```

Listing 1: The Builder Pattern in Action

2.3 Factory Method

The Factory Method is used to create an object, but the subclass decides which class is instantiated. With this pattern you can create concrete creators which can decide which concrete product should be created. The user of the creator only has to use the interface.

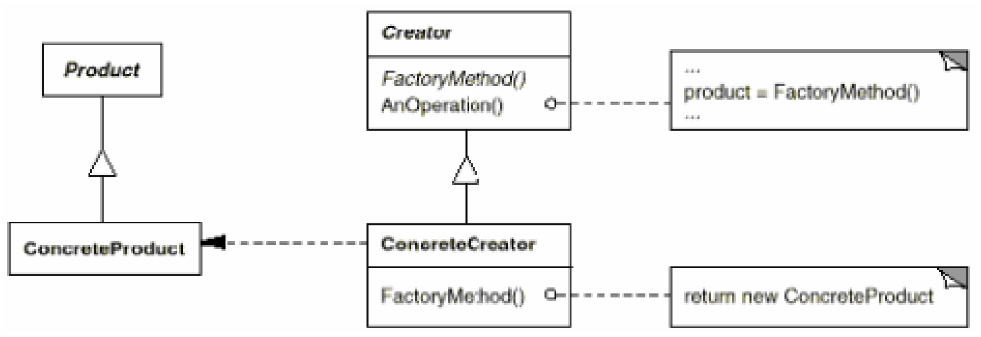


Figure 4: Factory Method Class Diagram

2.4 Prototype