Принципы проектирования и дизайна ПО

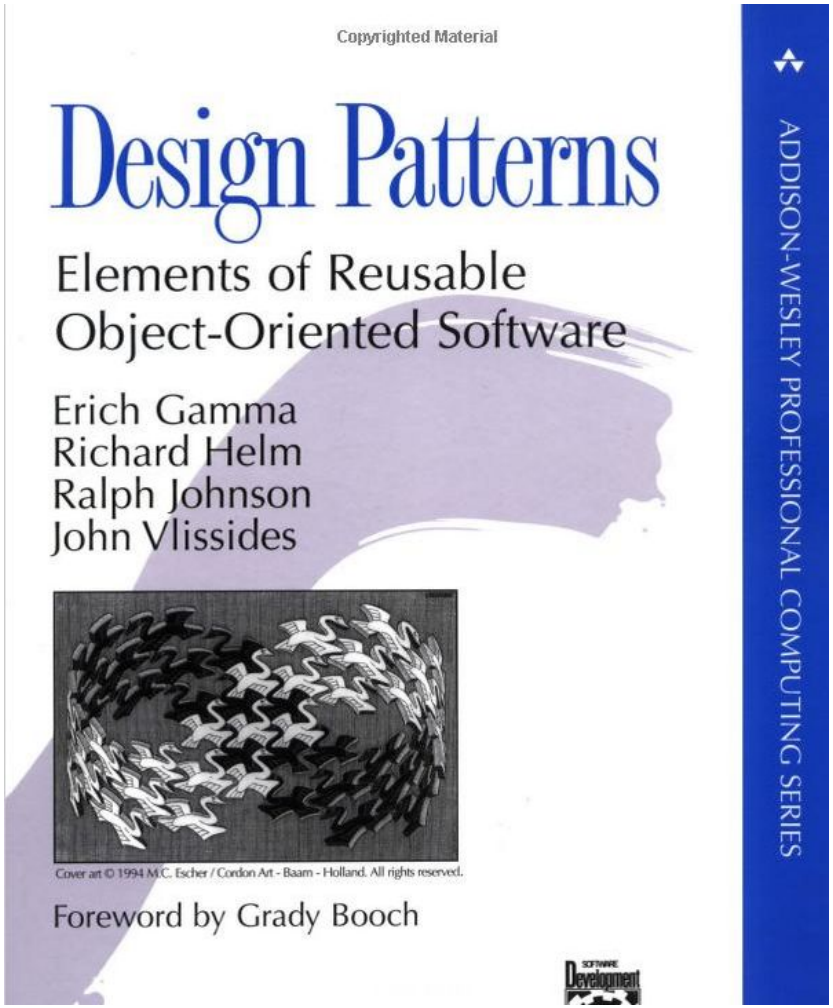
Лекция №9

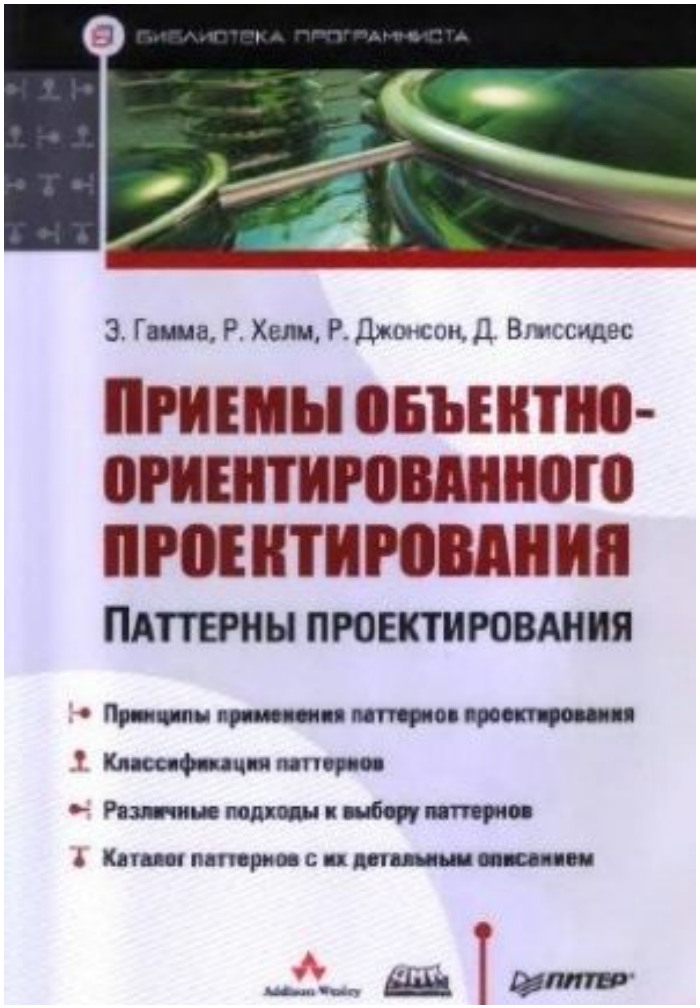
Агошков Илья 2016

History

- 1977 Christopher Alexander authored a book named “A pattern language: Towns, Buildings, Construction”
- 1987 Kent Beck and Ward Cunningham based on Alexander’s ideas created a set patterns used in GUI development in Smalltalk.
- 1988 Erich Gamma started his doctoral dissertation on applying patterns in software engineering.
- 1995 Erich Gamma finished his doctoral dissertation and moved to USA where in cooperation with Richard Helm, Ralph Johnson and John Vlissides publishes the book ***Design Patterns — Elements of Reusable Object-Oriented Software***. This book had 23 patterns described.

GoF design patterns



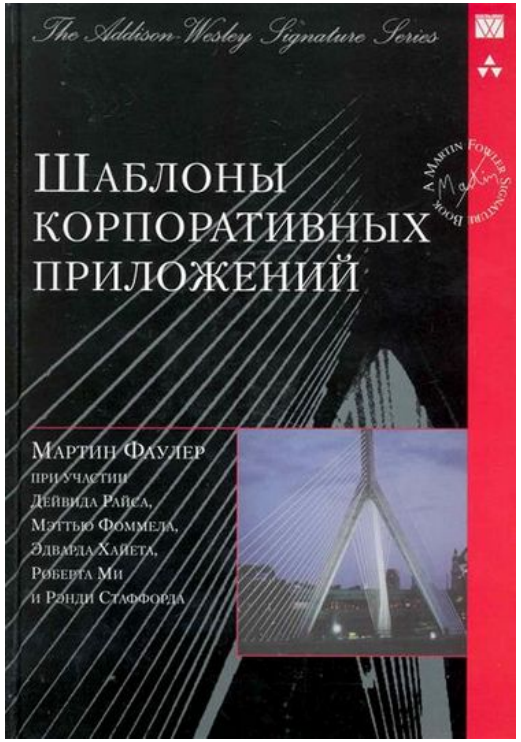


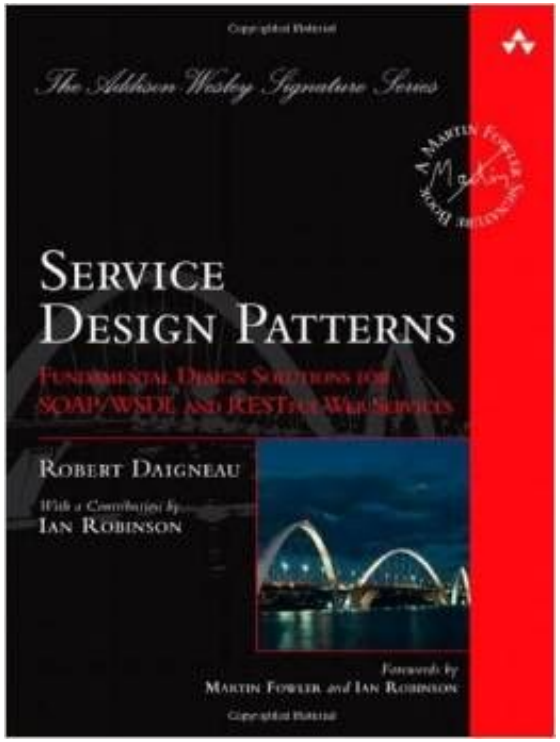
GangOfFour

Martin Fowler

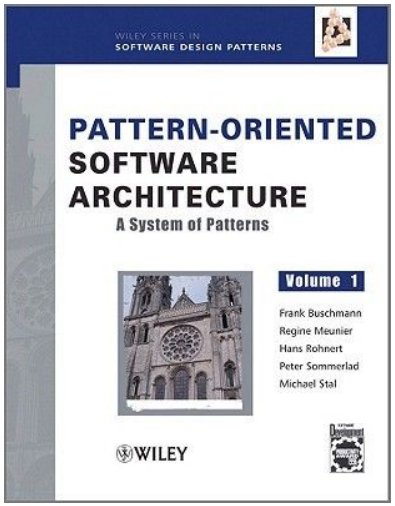
In my view the Gang of Four is the best book ever written on object-oriented design - possibly of any style of design. This book has been enormously influential on the software industry - just look at the Java and .NET libraries which are crawling with GOF patterns.

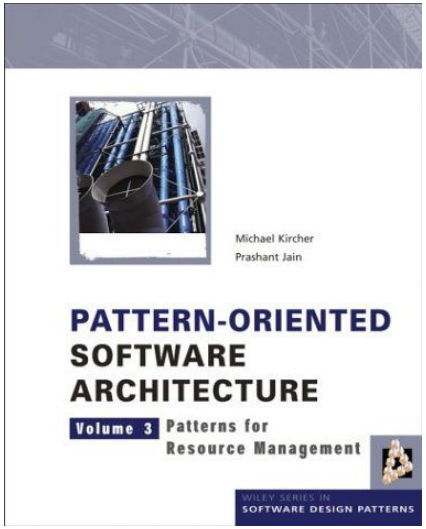
Other patterns

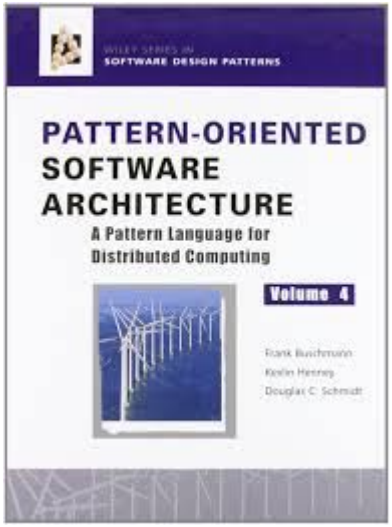




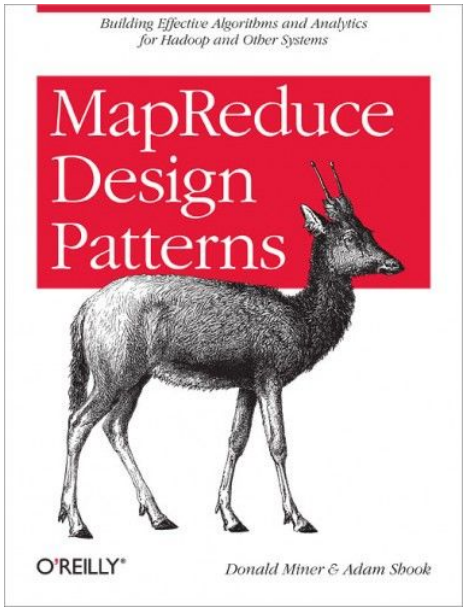
Pattern oriented software architecture

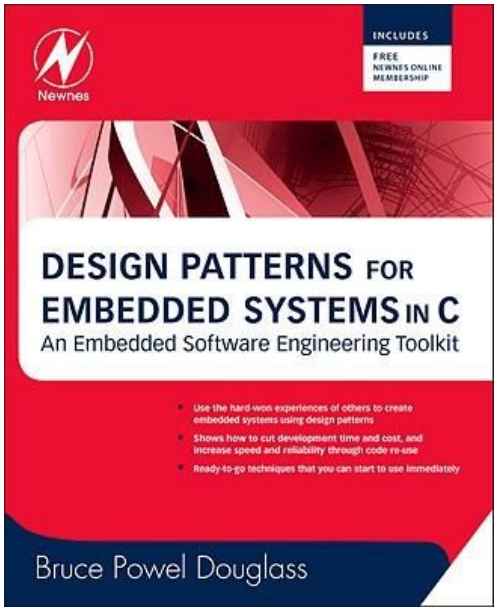






Other patterns





GoF design patterns

Creational	Structural	Behavioral
Factory method Abstract factory Singleton Builder Prototype	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of responsibility Command Interpreter Iterator Mediator Memento Observer State Strategy Template method Visitor

GoF design patterns

Creational	Structural	Behavioral
Factory method	Adapter	Chain of responsibility
Abstract factory	Bridge	Command
Singleton	Composite	Interpreter
Builder	Decorator	Iterator
Prototype	Facade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template method
		Visitor

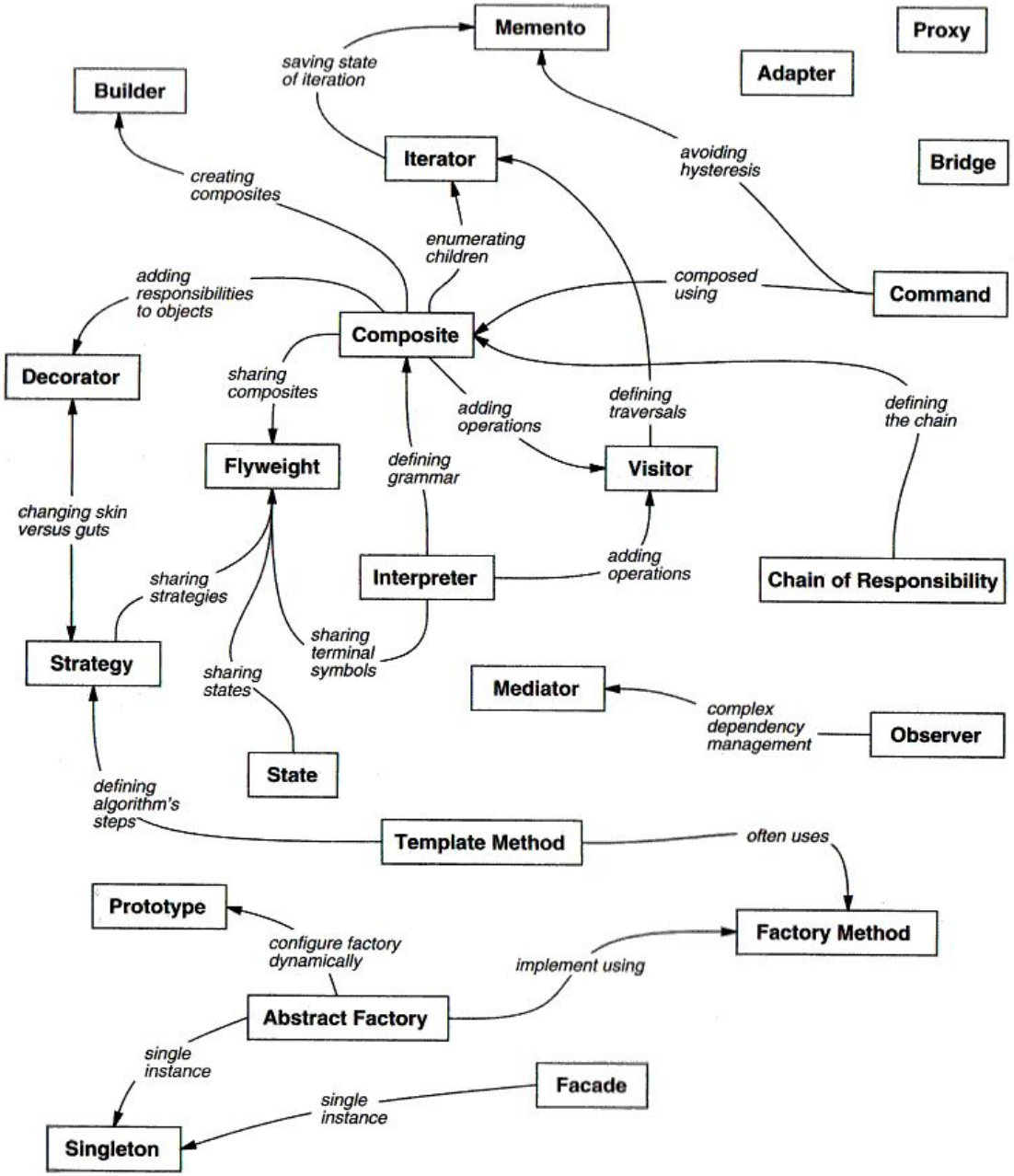


Figure 1.1: Design pattern relationships

Task

op = x + 2 * y

int arr[];

(arr[0] op arr[1]) (arr[2] op arr[3]) arr[5]

arr’[0] arr’[1] arr’[2]

arr’’[0] arr’’[1]

result arr’’’[0]

Task

5 8 2 3 9 2 1

(5 + 2*8) (2 + 2 *3) (9 + 2 *2) 1

21 8 13 1

37 15

67

Task

op = max(x, y)

op = x * y

op = |x| + |y|

https://github.com/sbt-agoshkov-iv/classwork

Strategy

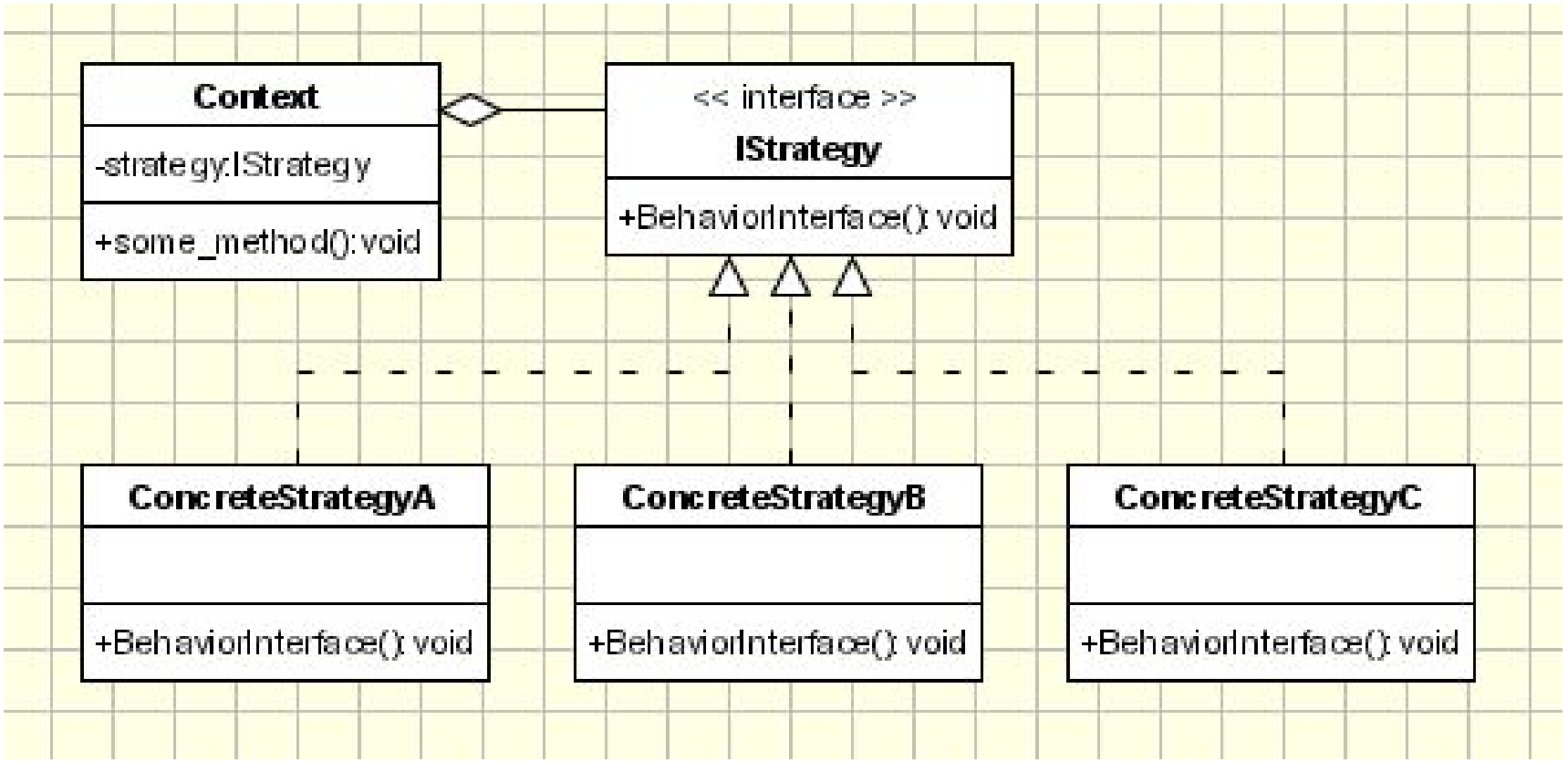
Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

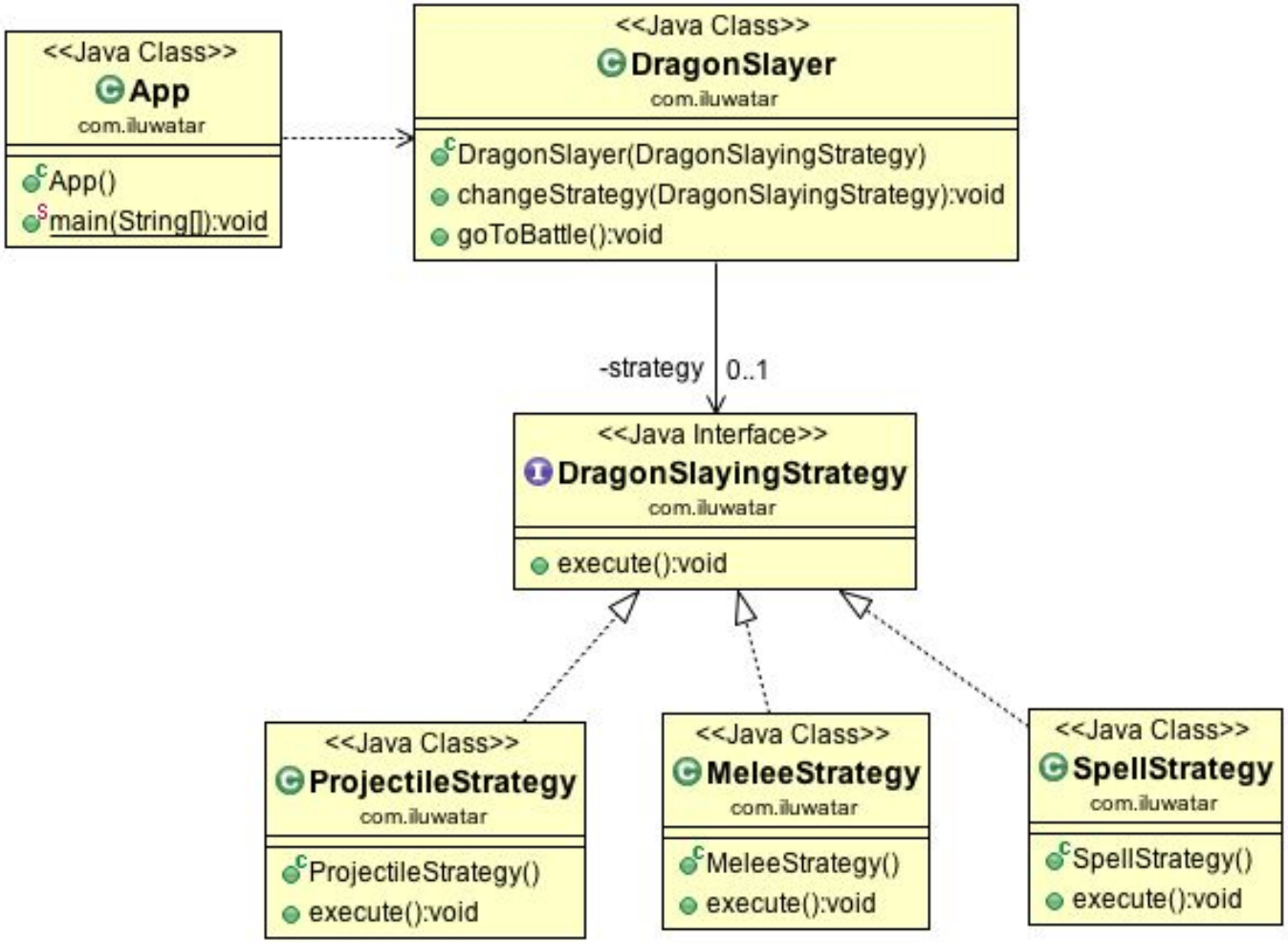
Also known as

Policy

Strategy


Context
-strategy:IStrategy
+some_method():void
<< interface >>
IStrategy
+BehaviorInterface():void
ConcreteStrategyA
+BehaviorInterface():void
ConcreteStrategyB
+BehaviorInterface():void
ConcreteStrategyC
+BehaviorInterface():void

Strategy


<<Java Class>>
App
com.iluwatar
App()
main(String[]):void
<<Java Class>>
DragonSlayer
com.iluwatar
DragonSlayer(DragonSlayingStrategy)
changeStrategy(DragonSlayingStrategy):void
goToBattle():void
-strategy
0..1
<<Java Interface>>
DragonSlayingStrategy
com.iluwatar
execute():void
<<Java Class>>
ProjectileStrategy
com.iluwatar
ProjectileStrategy()
execute():void
<<Java Class>>
MeleeStrategy
com.iluwatar
MeleeStrategy()
execute():void
<<Java Class>>
SpellStrategy
com.iluwatar
SpellStrategy()
execute():void

Strategy

Use the Strategy pattern when

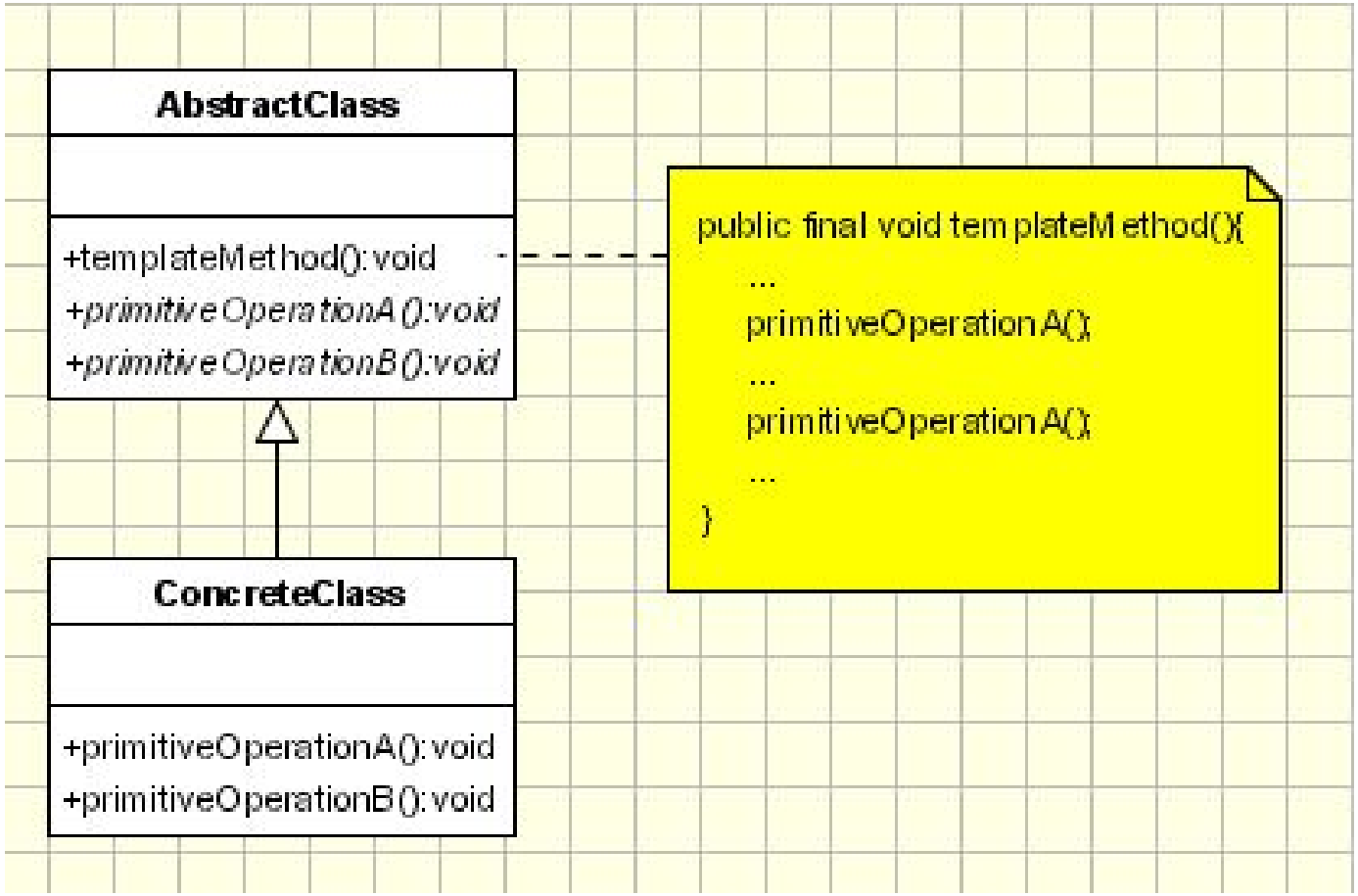
- many related classes differ only in their behavior. Strategies provide a way to configure a class with one of many behaviors.
- you need different variants of an algorithm. For example, you might define algorithms reflecting different space/time trade-offs.
- a class defines many behaviors, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class.

Template method

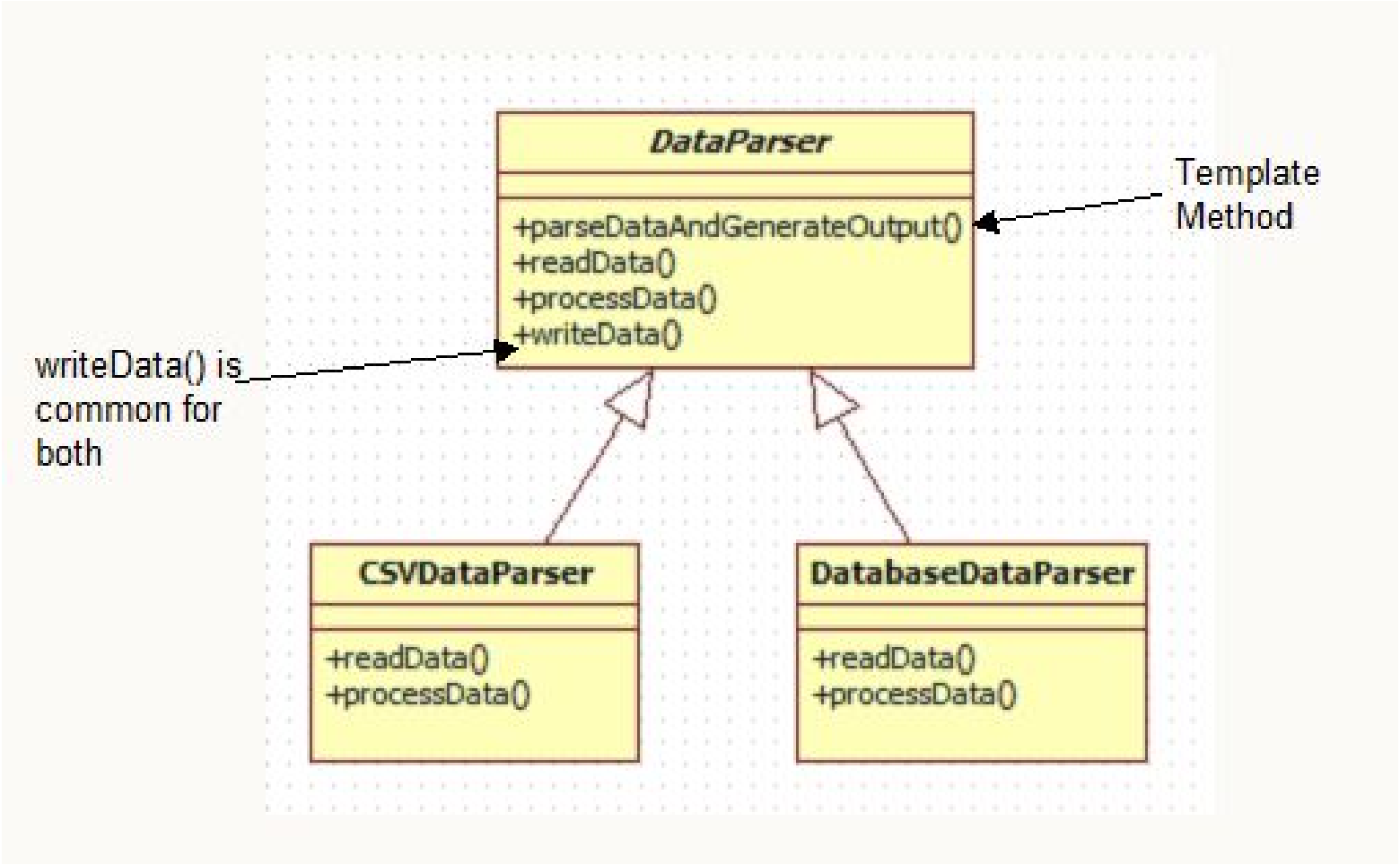
Intent

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm’s structure.

Template method



Template method



Task

```
interface Cache {
  String getValue(String key);
  void putValue(String key, String value);
  void removeValue(String key);
}
```

```
public class MyAlgorithm {
  String doSomeActions(Cache cache) {
    // gets and puts smth from cache
    return “abc”;
  }
}
```

Adapter

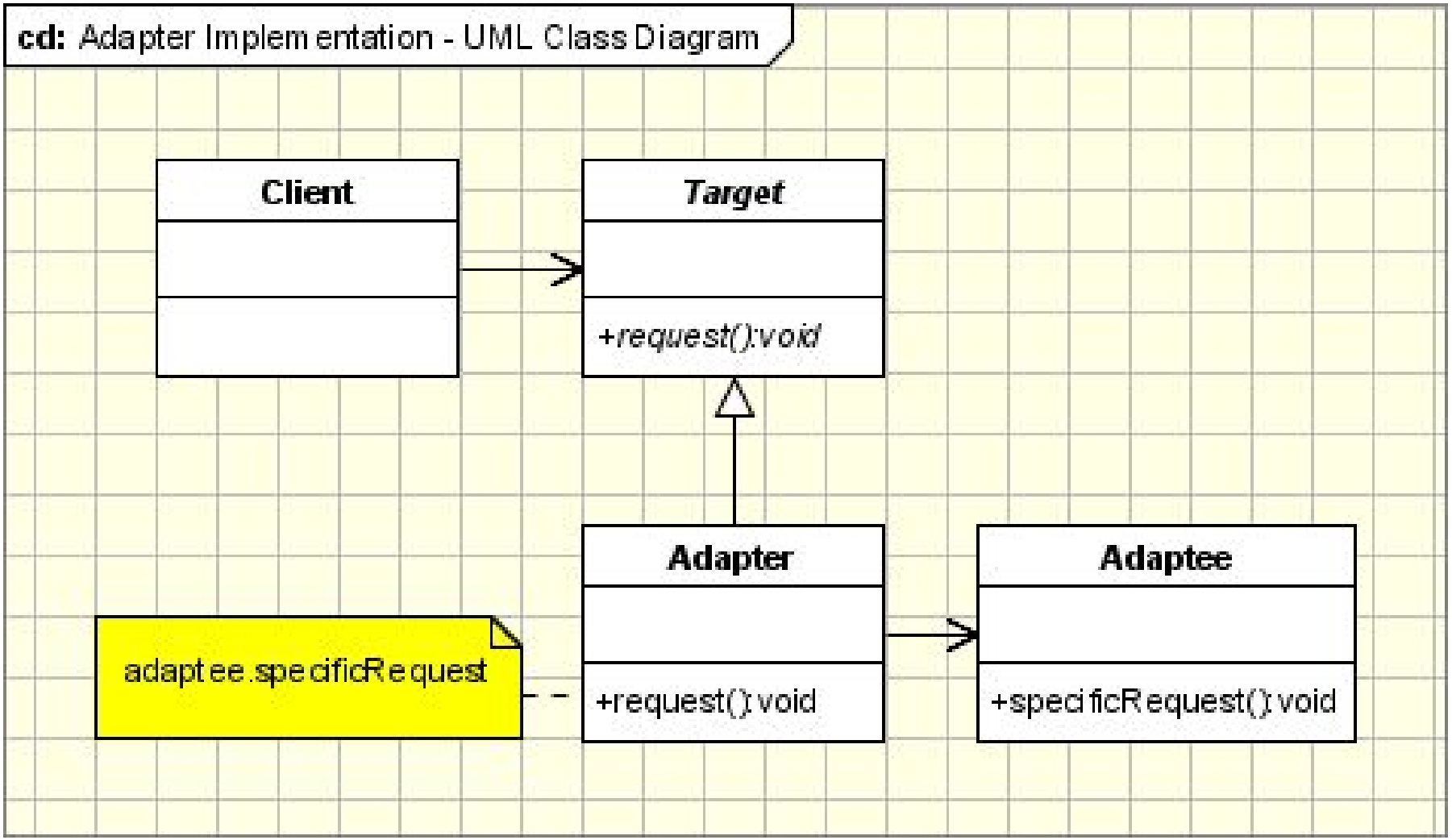
Intent

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

Also known as

Wrapper

Adapter


cd: Adapter Implementation - UML Class Diagram
Client
Target
+request():void
Adapter
+request():void
Adaptee
+specificRequest():void
adaptee.specificRequest