Software Engineering I (02161)

Week 8

Assoc. Prof. Hubert Baumeister

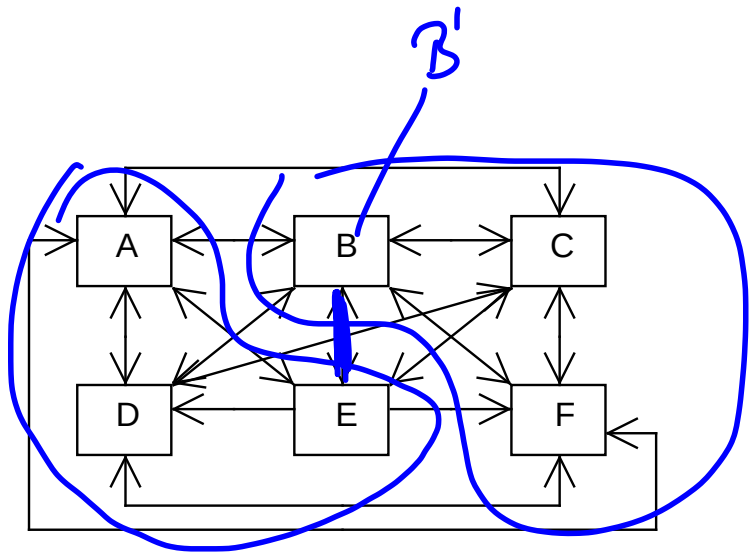
DTU Compute
Technical University of Denmark

Spring 2018

Contents

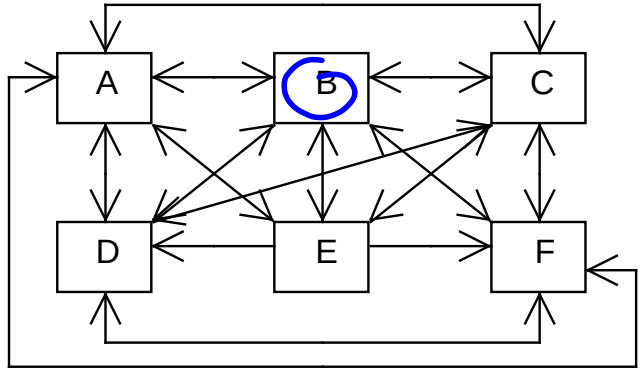
Low Coupling

High coupling

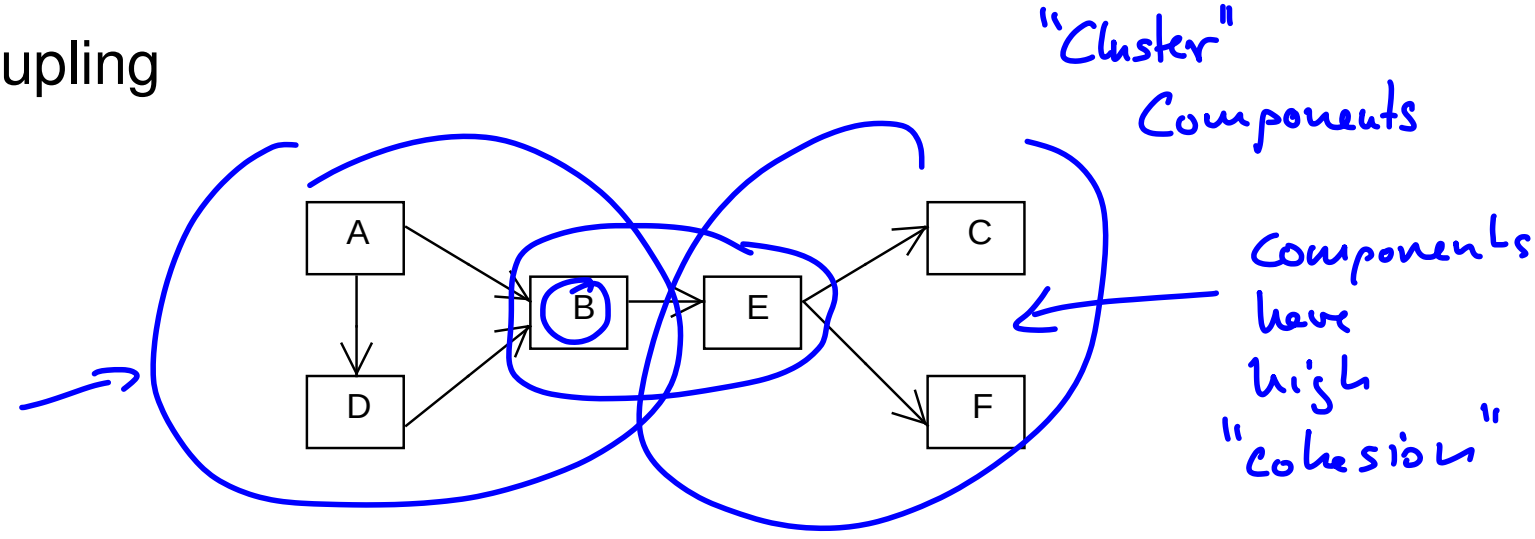


Low Coupling

High coupling

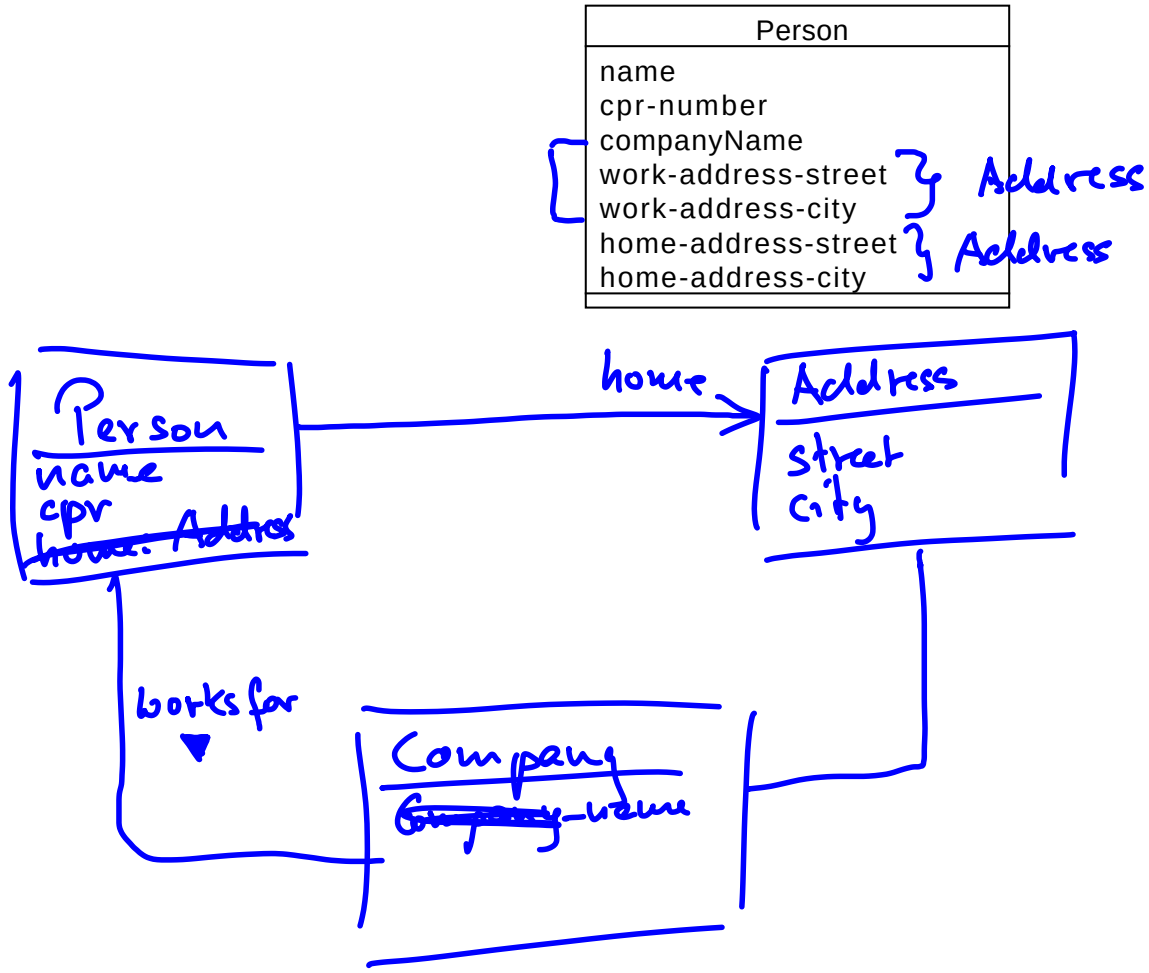


Low coupling



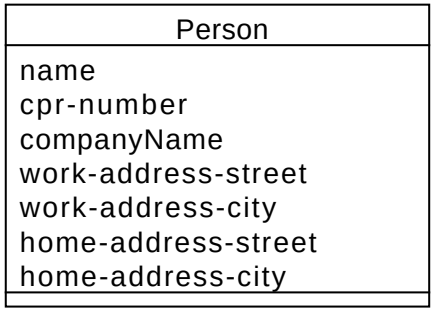
High Cohesion

Low Cohesion

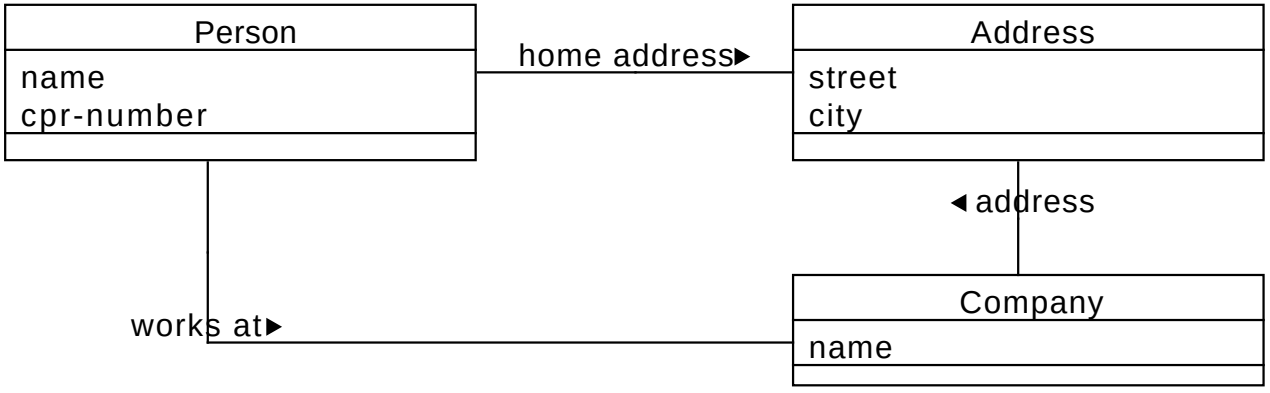


High Cohesion

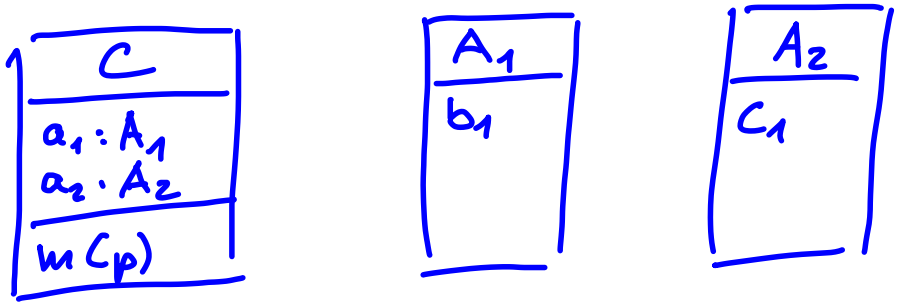
Low Cohesion



High Cohesion



Law of Demeter



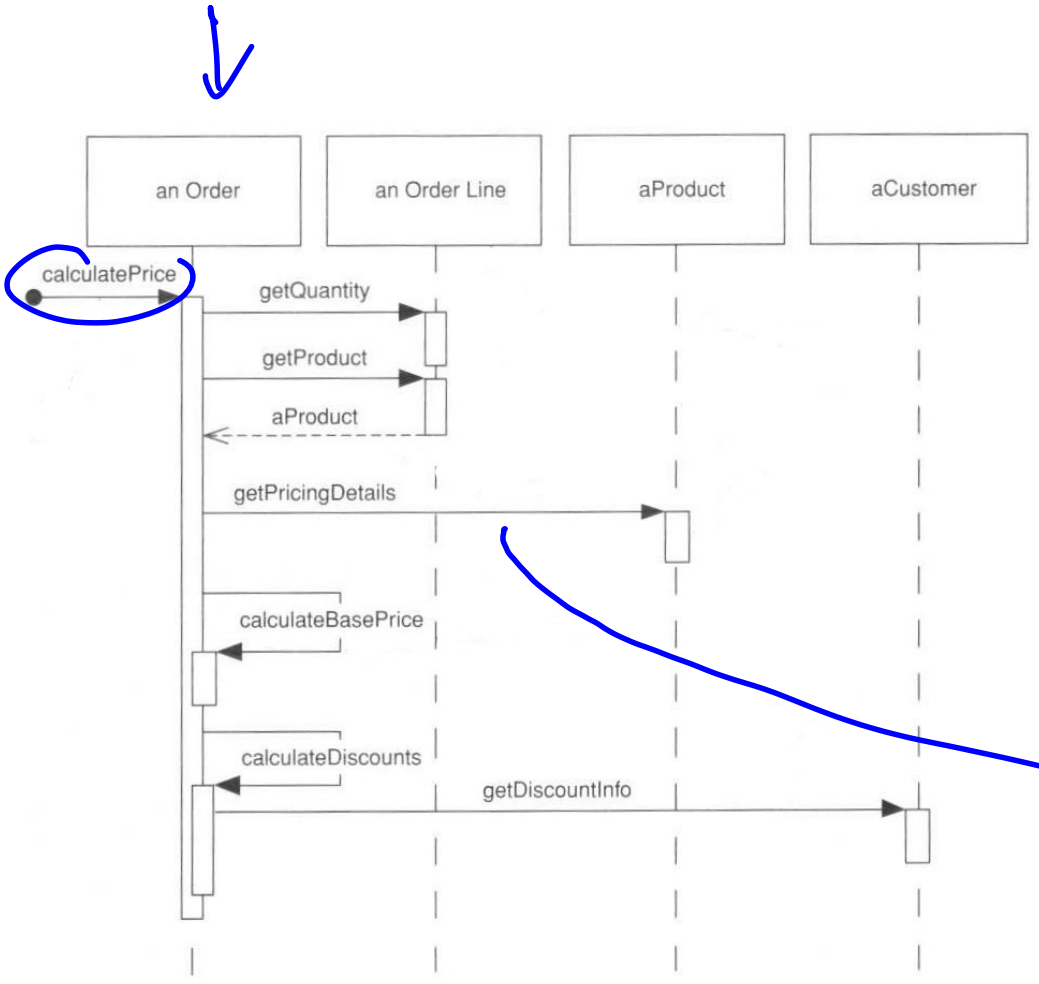
Law of Demeter

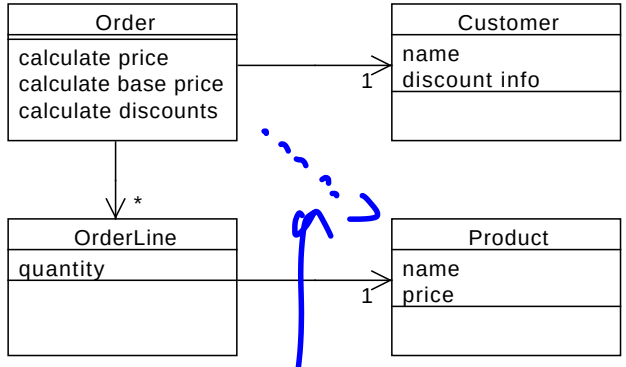
- ”Only talk to your immediate friends”
- Only method calls to the following objects are allowed
  - the object itself
  - its components
  - objects created by that object
  - parameters of methods
- Also known as: **Principle of Least Knowledge**
- Law of Demeter = **low coupling**

→ **delegate functionality**

→ decentralised control

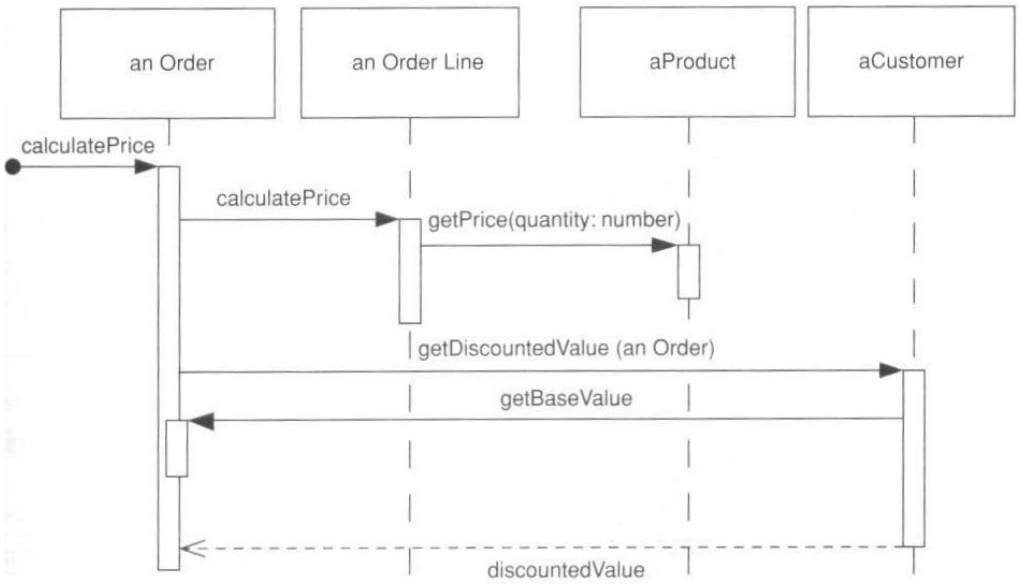
Computing the price of an order

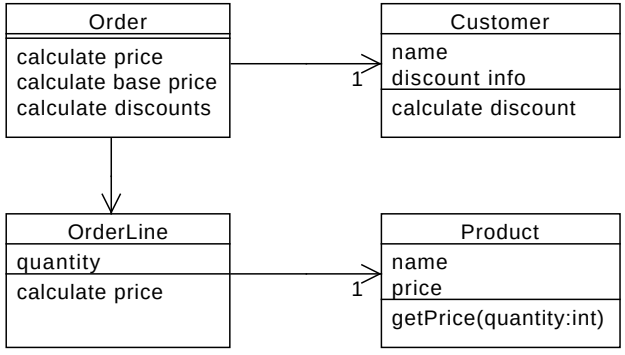




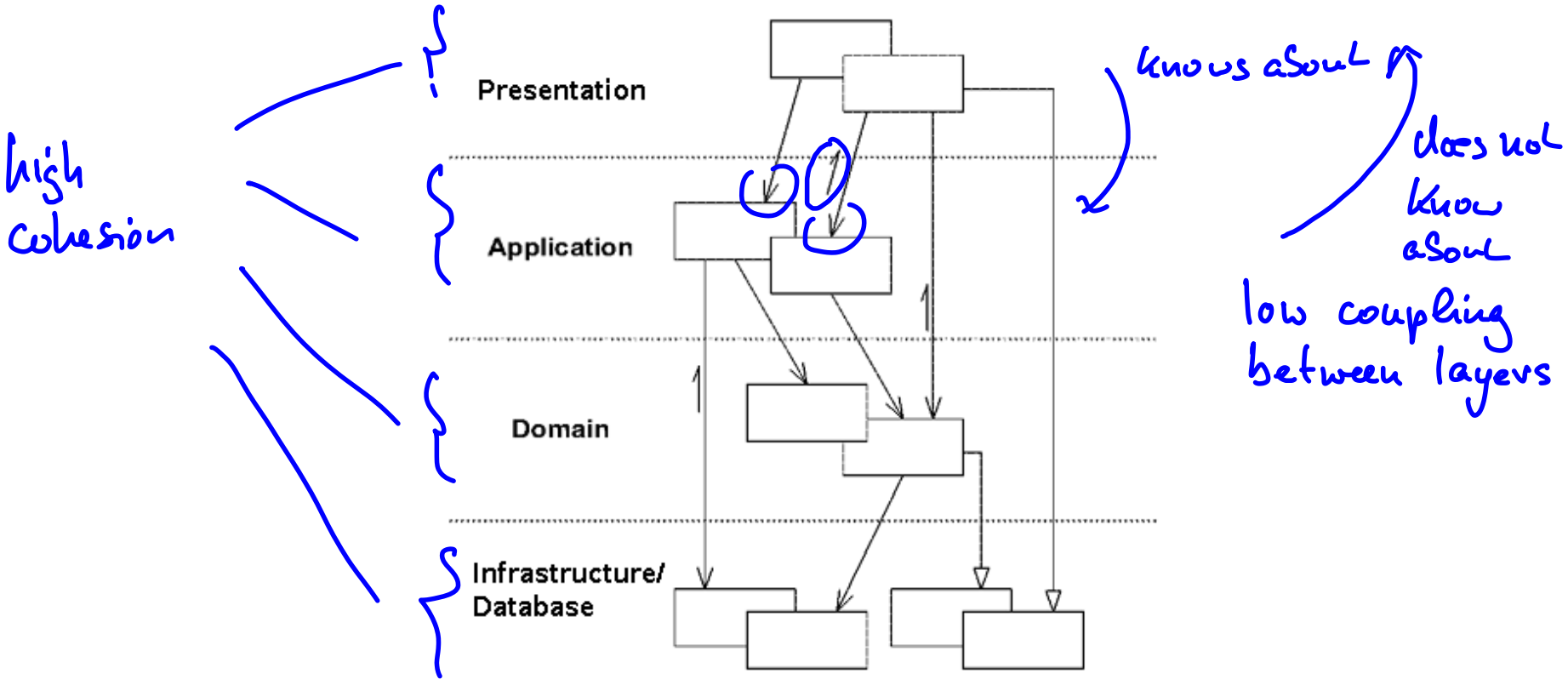
violation of the law of Demeter

Computing the price of an order





Layered Architecture



Eric Evans, Domain Driven Design, Addison-Wesley, 2004

DRY principle

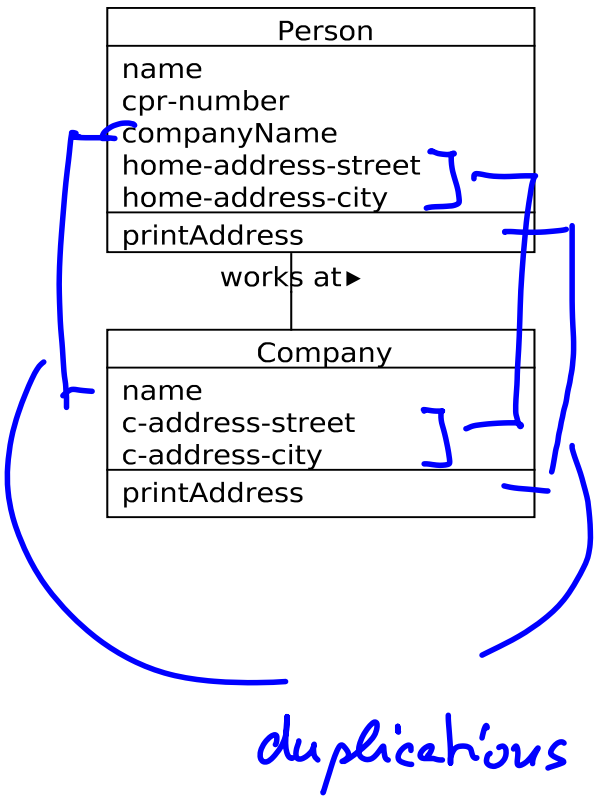
DRY principle

Don’t repeat yourself

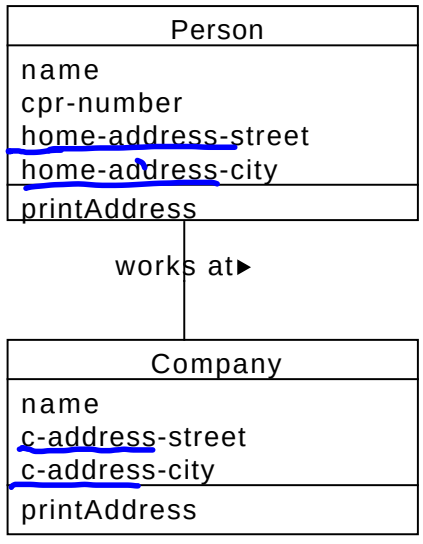
”Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” The Pragmatic Programmer, Andrew Hunt and David Thomas

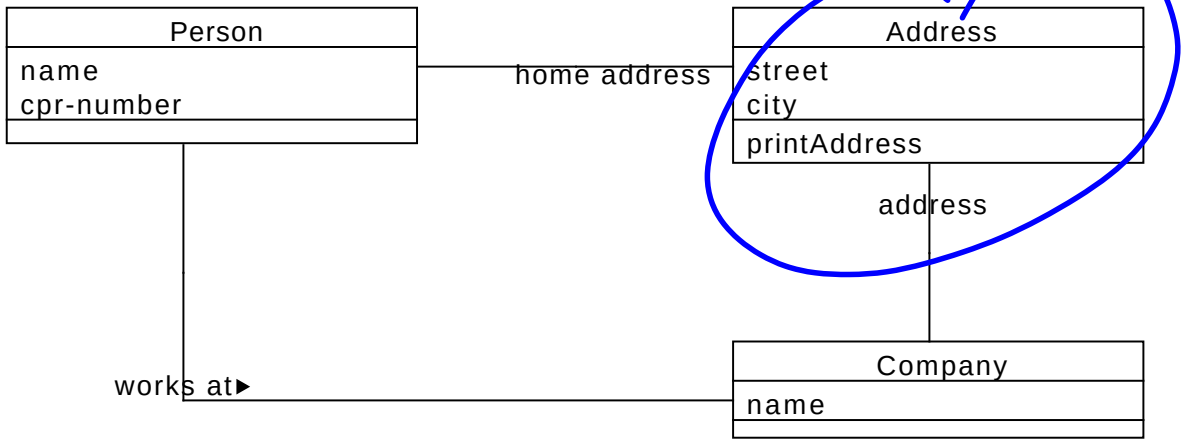
- code
- documentation
- build stystem

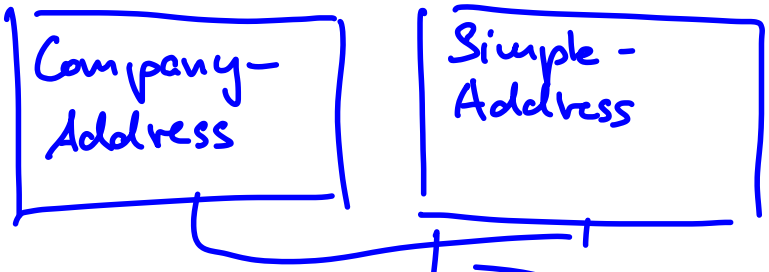
Example: Code Duplication



Example: Code Duplication







DRY

DRY principle

- Techniques to avoid duplication
  - Use appropriate abstractions
  - Inheritance
  - Classes with instance variables
  - Methods with parameters
- Refactor to remove duplication
- Generate artefacts from a common source. Eg. Javadoc

KISS principle

KISS principle

Keep it short and simple (sometimes also: Keep it simple, stupid)

- **simplest solution** *first*
- **Strive** for **simplicity**
  - **Takes time**!!
  - *refactor* for **simplicity**

Antoine de Saint Exupéry

”It seems that perfection is reached not when there is nothing left to add, but when there is nothing left to take away”.

YAGNI principle

YAGNI principle

You ain't gonna needed it

- Focus on the task at hand
  - E.g. using the observer pattern because it **might** be needed

→ **Different kind of flexibility**

- **make your design changable**
  - tests, easy to refactor
- **design for change**
  - Use good OO principles
    - High cohesion, low coupling
    - Decentralized control
    - SOLID principles (next week)

Contents

Patterns in Architecture

182 EATING ATMOSPHERE

. . . we have already pointed out how vitally important all kinds of communal eating are in helping to maintain a bond among a group of people—COMMUNAL EATING (147); and we have given some idea of how the common eating may be placed as part of the kitchen itself—FARMHOUSE KITCHEN (139). This pattern gives some details of the eating atmosphere.

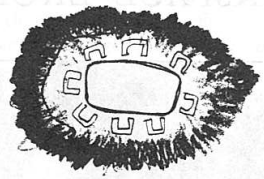
✣ ✣ ✣

When people eat together, they may actually be together in spirit—or they may be far apart. Some rooms invite people to eat leisurely and comfortably and feel together, while others force people to eat as quickly as possible so they can go somewhere else to relax.

Above all, when the table has the same light all over it, and has the same light level on the walls around it, the light does nothing to hold people together; the intensity of feeling is quite likely to dissolve; there is little sense that there is any special kind of gathering. But when there is a soft light, hung low over the table, with dark walls around so that this one point of light lights up people's faces and is a focal point for the whole group, then a meal can become a special thing indeed, a bond, communion.

Therefore:

Put a heavy table in the center of the eating space—large enough for the whole family or the group of people using it. Put a light over the table to create a pool of light over the group, and enclose the space with walls or with contrasting darkness. Make the space large enough so the chairs can be pulled back comfortably, and provide shelves and counters close at hand for things related to the meal.



light in the middle

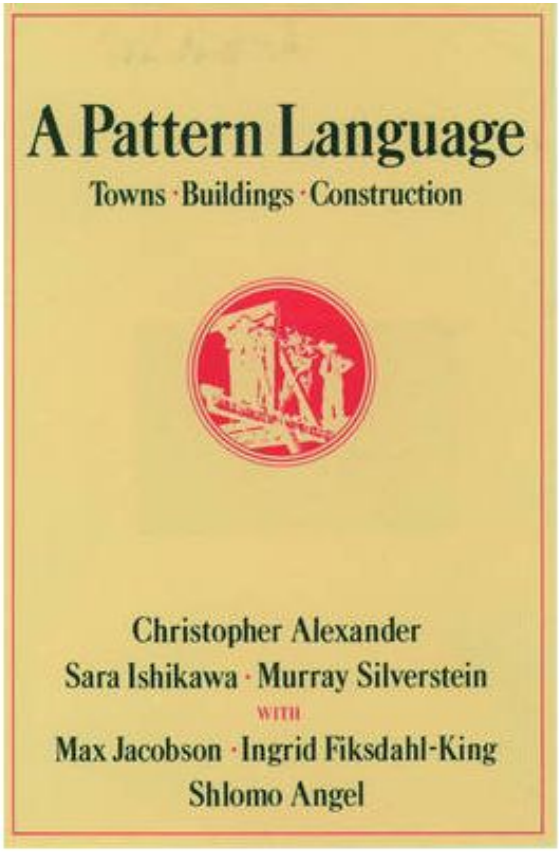
✣ ✣ ✣

Get the details of the light from POOLS OF LIGHT (252); and choose the colors to make the place warm and dark and comfortable at night—WARM COLORS (250); put a few soft chairs nearby—DIFFERENT CHAIRS (251); or put BUILT-IN SEATS (202) with big cushions against one wall; and for the storage space—OPEN SHELVES (200) and WAIST-HIGH SHELF (201). . . .

A Pattern Language, Christopher Alexander, 1977

History of Patterns

- Christopher Alexander: Architecture (1977/1978)
  - Pattern: a *solution* to a *problem* in a context
  - Pattern language: set of related patterns



- Kent Beck and Ward Cunningham: Patterns for Smalltalk applications (1987)

Pattern: ”Objects from the User’s World”

Problem: What are the best objects to start a design with?

Constraints: The way the user sees the world should have a profound impact on the way the system presents information. Sometimes a computer program can be a user’s bridge to a deeper understanding of a domain. However, having a a software engineer second guess the user is a chancy proposition at best.

Kent Beck: ”Birds, Bees, and Browsers—Obvious sources of Objects” 1994 `http://bit.ly/2q4h0GC`

Pattern: ”Objects from the User’s World”

Forces:

- Some people say, ”I can structure the internals of my system any way I want to. What I present to the user is just a function of the user interface.” In my experience, this is simply not so. The structure of the internals of the system will find its way into the thoughts and vocabulary of the user in the most insidious way. Even if it is communicated only in what you tell the user is easy and what is difficult to implement, the user will build a mental model of what is inside the system.
- Unfortunately, the way the user thinks about the world isn’t necessarily the best way to model the world computationally. In spite of the difficulties, though, it is more important to present the best possible interface to the user than to make the system simpler to implement.

Therefore:

Pattern: ”Objects from the User’s World”

Solution: Begin the system with objects from the user’s world. Plan to decouple these objects from the way you format them on the screen, leaving only the computational model.

History of Patterns

- Christopher Alexander: Architecture (1977/1978)
- Kent Beck and Ward Cunningham: Patterns for Smalltalk applications (1987)
- Ward Cunningham: Portland Pattern Repository `http://c2.com/ppr`
  - the Wiki Wiki Web was invented for this purpose
- Gang of four: Design Patterns book (1994) (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)
- Pattern conferences, e.g. PloP (Pattern Languages of Programming) since 1994
- Implementation Patterns, Architectural Patterns, Analysis Patterns, Domain Patterns, Anti Patterns ...

Design Patterns

- Defined in the Design Pattern Book (1994)
- Best practices for object-oriented software
- Creational Patterns
  - Abstract Factory, Builder, Factory Method, Prototype, Singleton
- Structural Patterns
  - Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
- Behavioral Patterns
  - Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

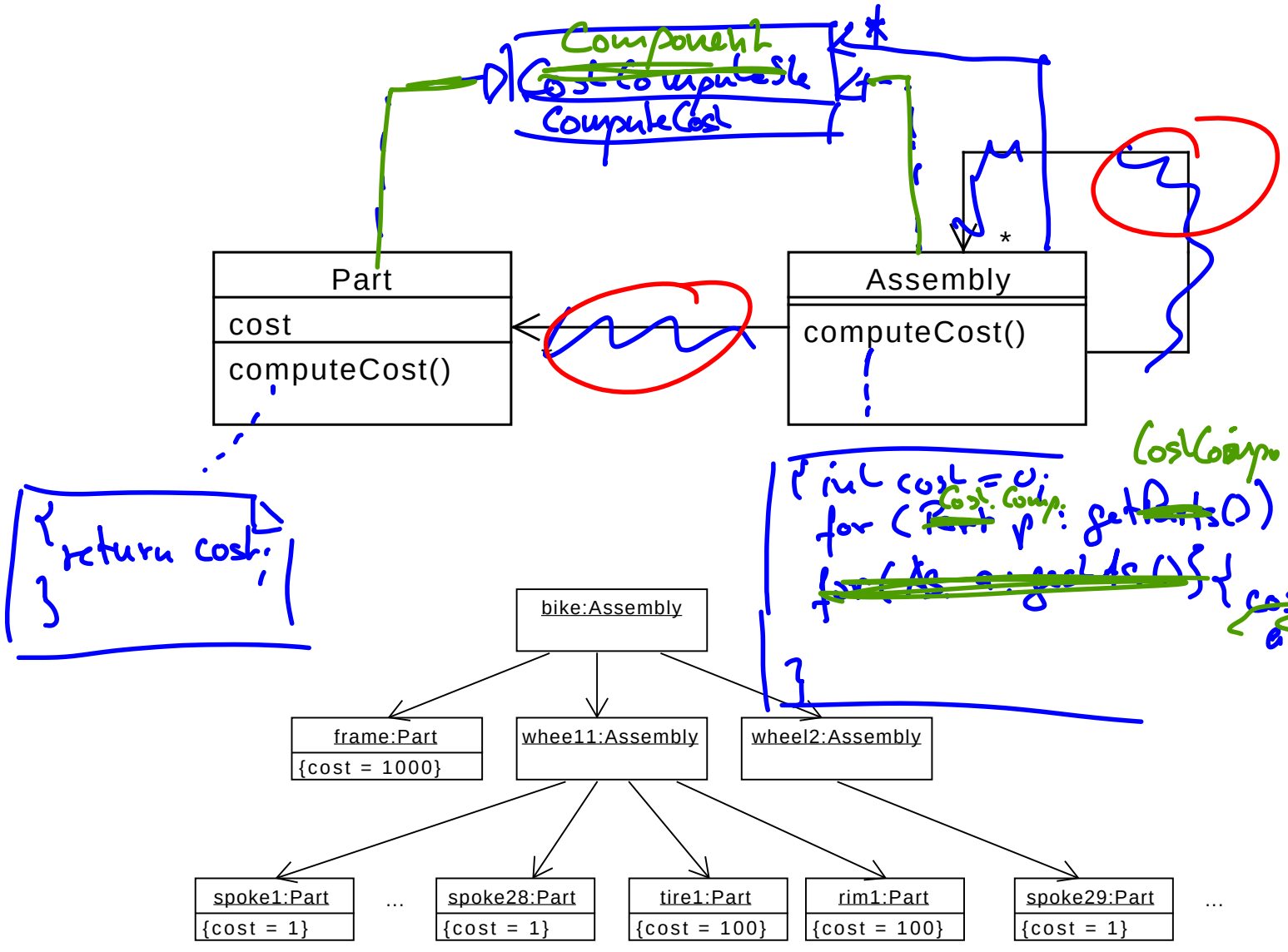
Places to find design patterns:

- Portland Pattern repository `http://c2.com/cgi/wiki?PeopleProjectsAndPatterns` (since 1995)
- Wikipedia `http://en.wikipedia.org/wiki/Design_pattern_(computer_science)`
- Wikipedia `http://en.wikipedia.org/wiki/Category:Software_design_patterns`

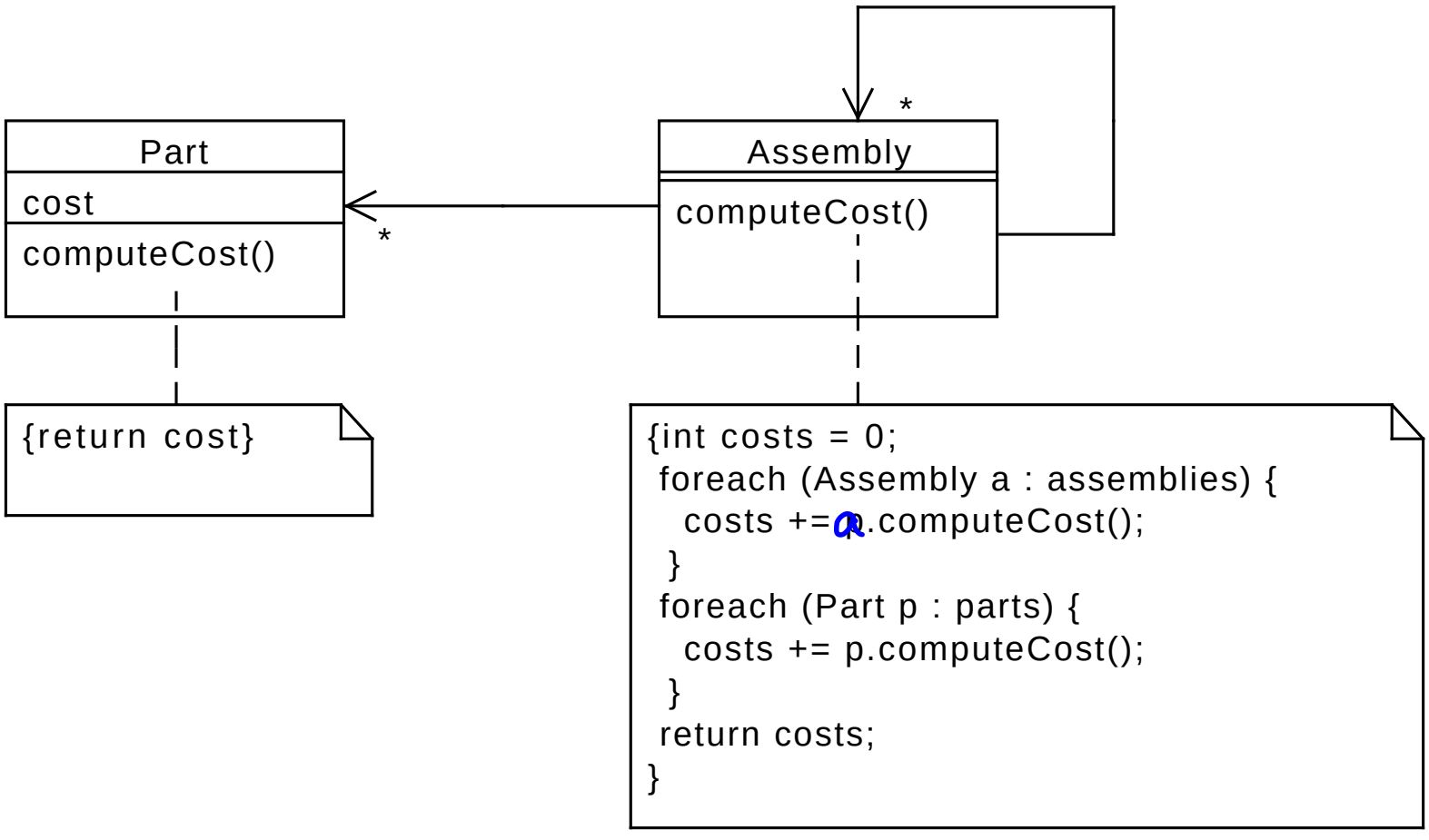
Example: compute costs for components

- Task: compute the overall costs of a bike
- Bike
  - Frame (1000 kr)
  - Wheel: 28 spokes (1 kr), rim (100 kr), tire (100 kr)
  - Wheel: 28 spokes (1 kr), rim (100 kr), tire (100 kr)

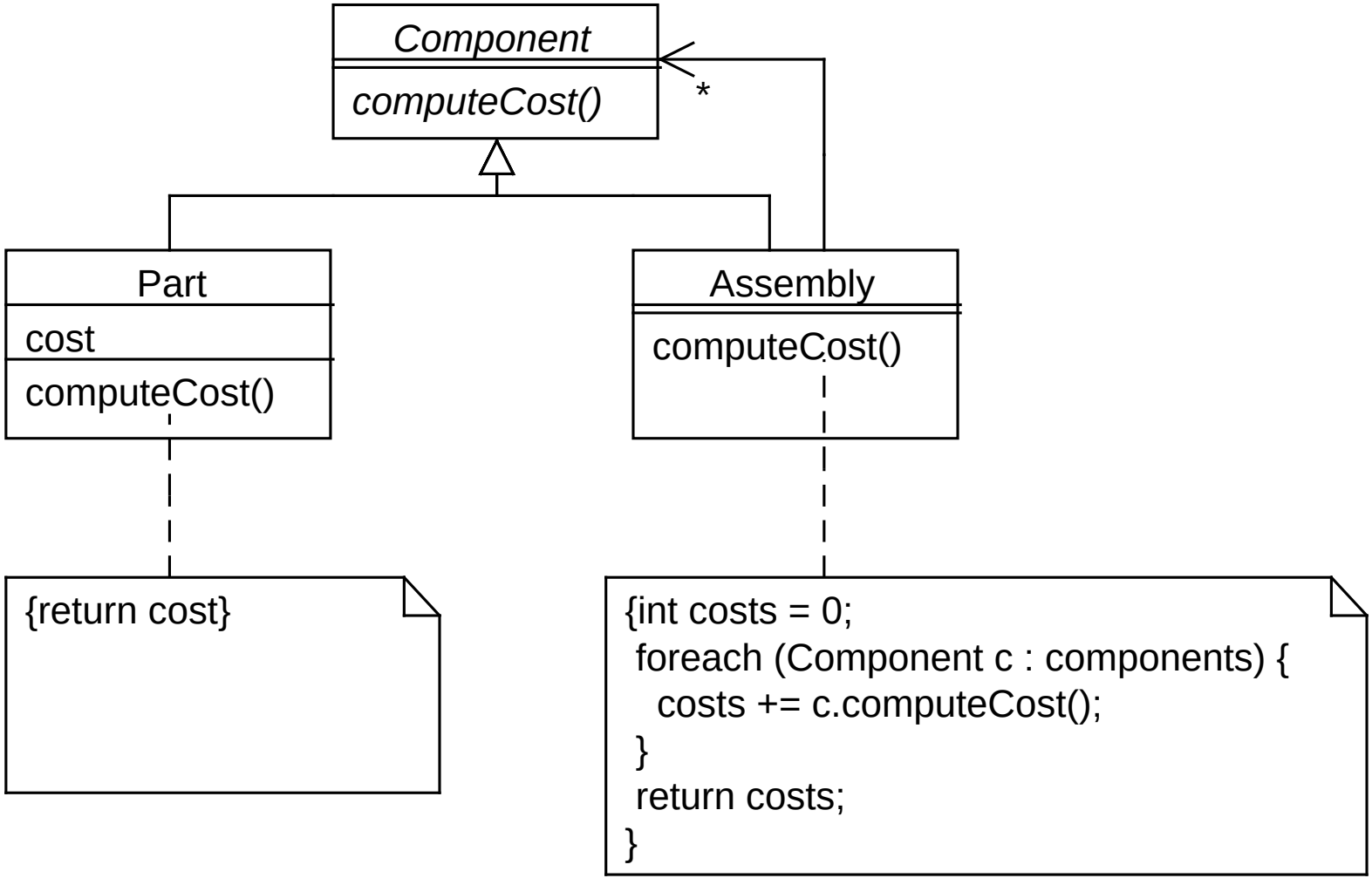
Example: compute costs for components



Example: compute costs for components



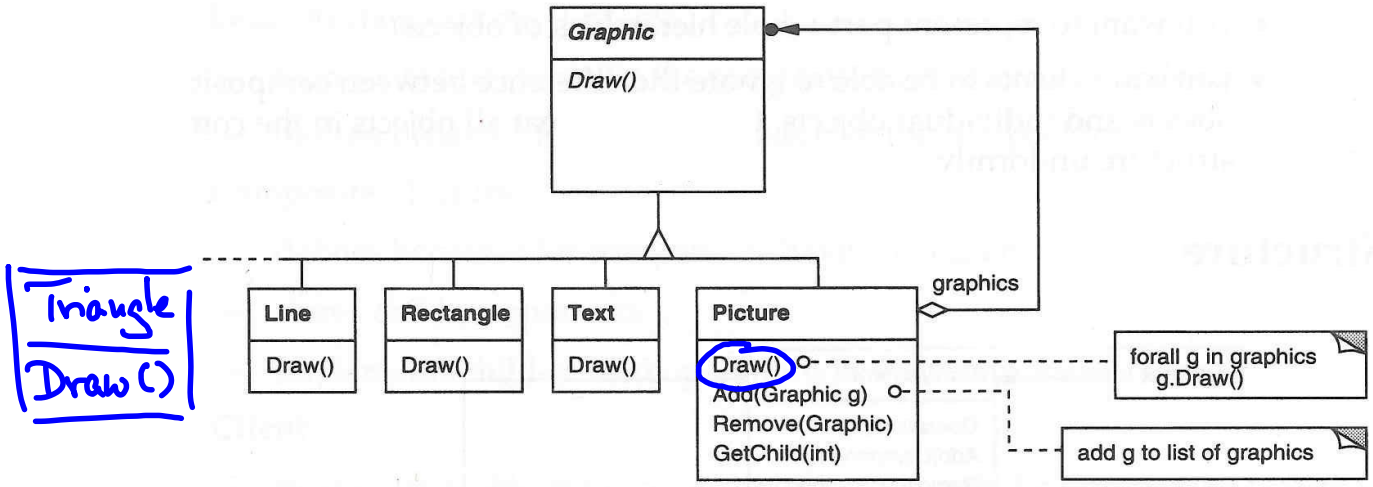
Example: compute costs for components



Composite Pattern

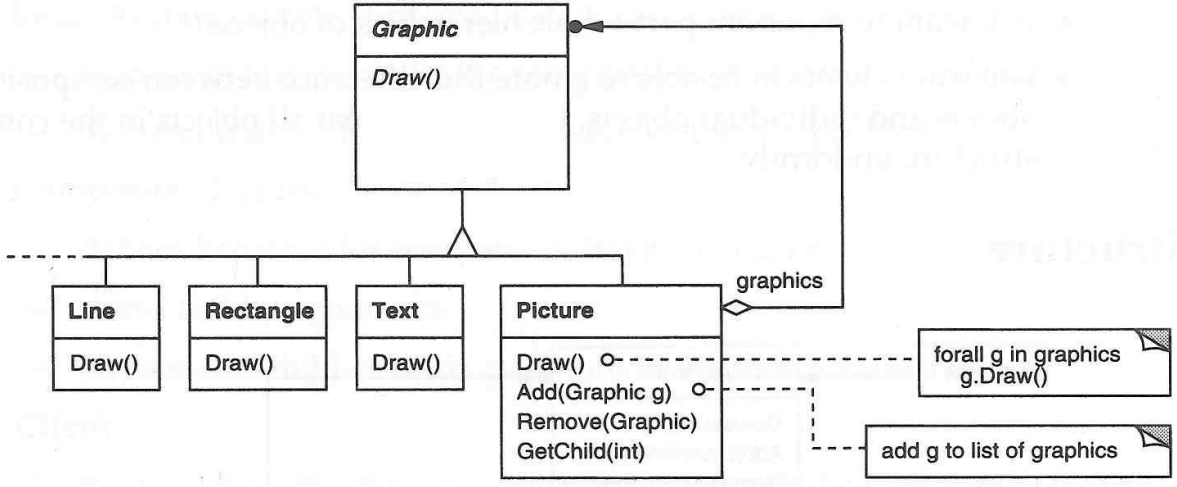
Composite Pattern

”Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

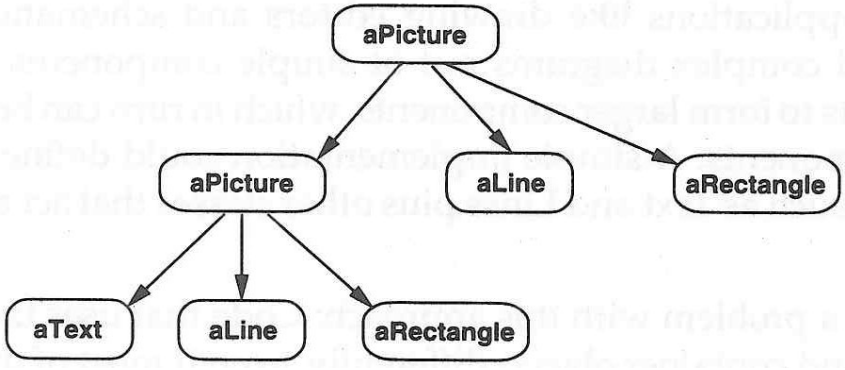


Composite Pattern: Graphics

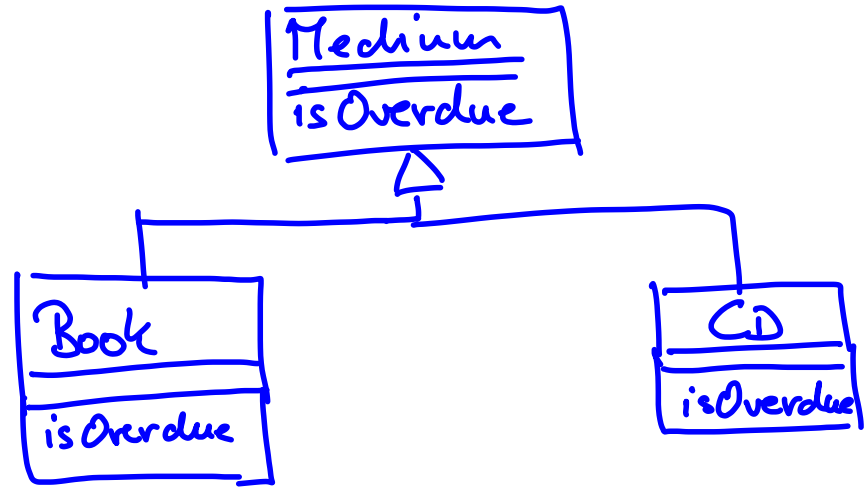
- Class Diagram



- Instance diagram



Template Method Problem



Overdue method for Book:

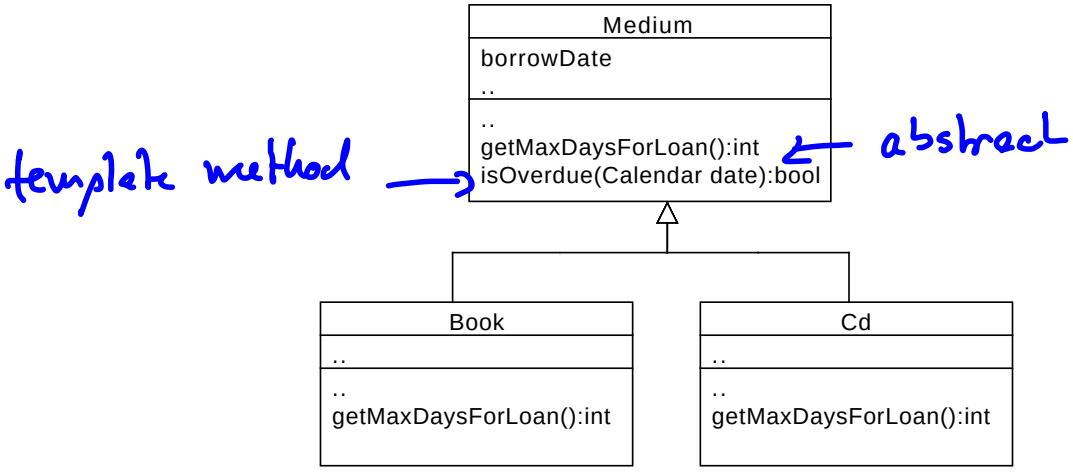
1. compute due date for a book
   a. get the current date
   b. add 4 weeks for the book
2. check if the current date is after the due date

Overdue method for CD:

1. compute due date for a cd
   a. get the current date
   b. add 2 weeks for loan for the cd
2. check if the current date is after the due date

≈

Template Method

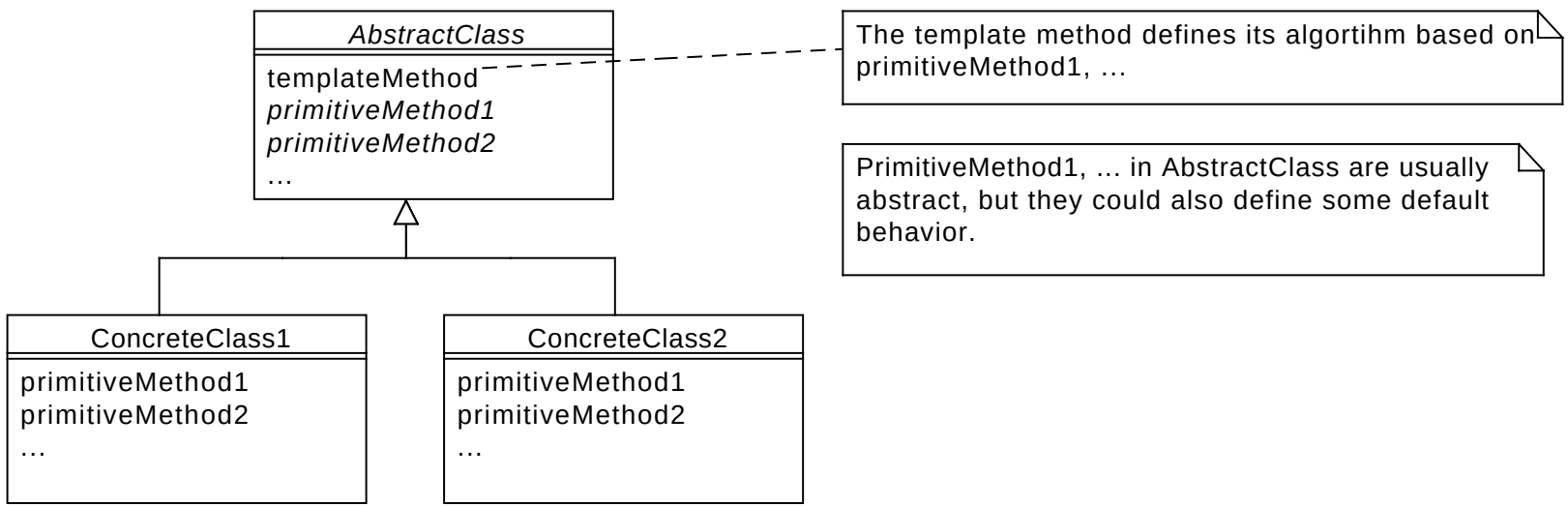



```
public abstract class Medium {

  public boolean isOverdue(Calendar date) {
    if (!isBorrowed()) {
      return false;
    }
    Calendar dueDate = new GregorianCalendar();
    dueDate.setTime(borrowDate.getTime());
    dueDate.add(Calendar.DAY_OF_YEAR, getMaxDaysForLoan());
    return date.after(dueDate);
  }

  public abstract int getMaxDaysForLoan();
}
```

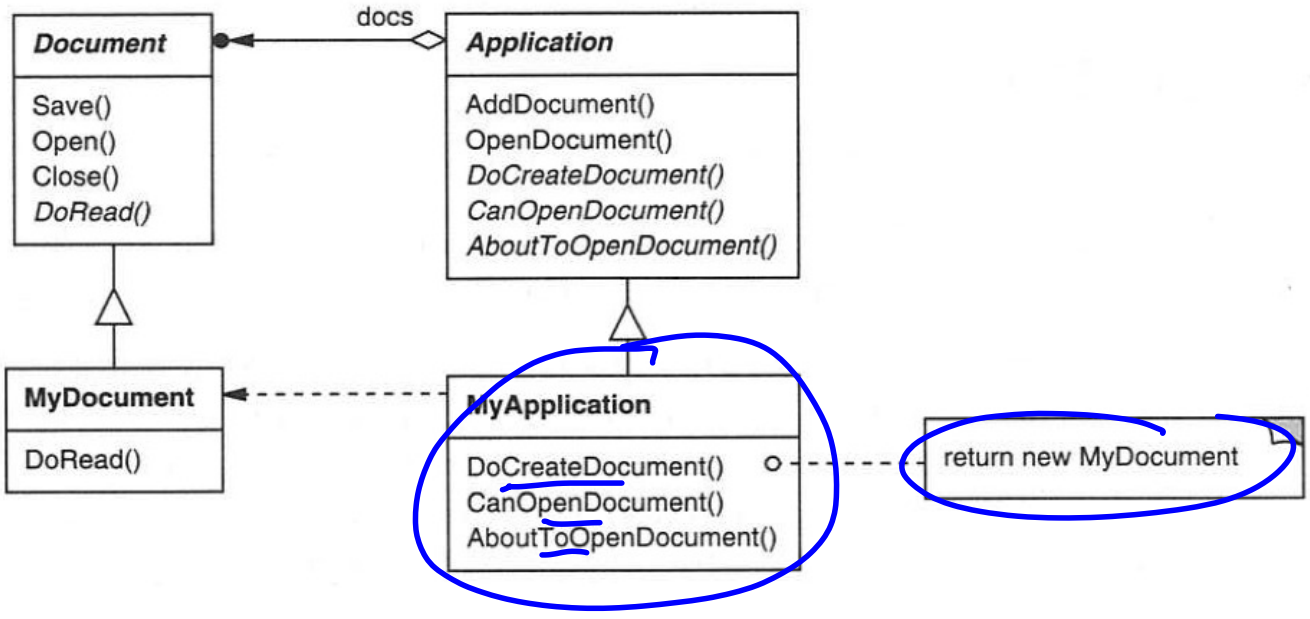
Template Method

Template Method

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



Template Method



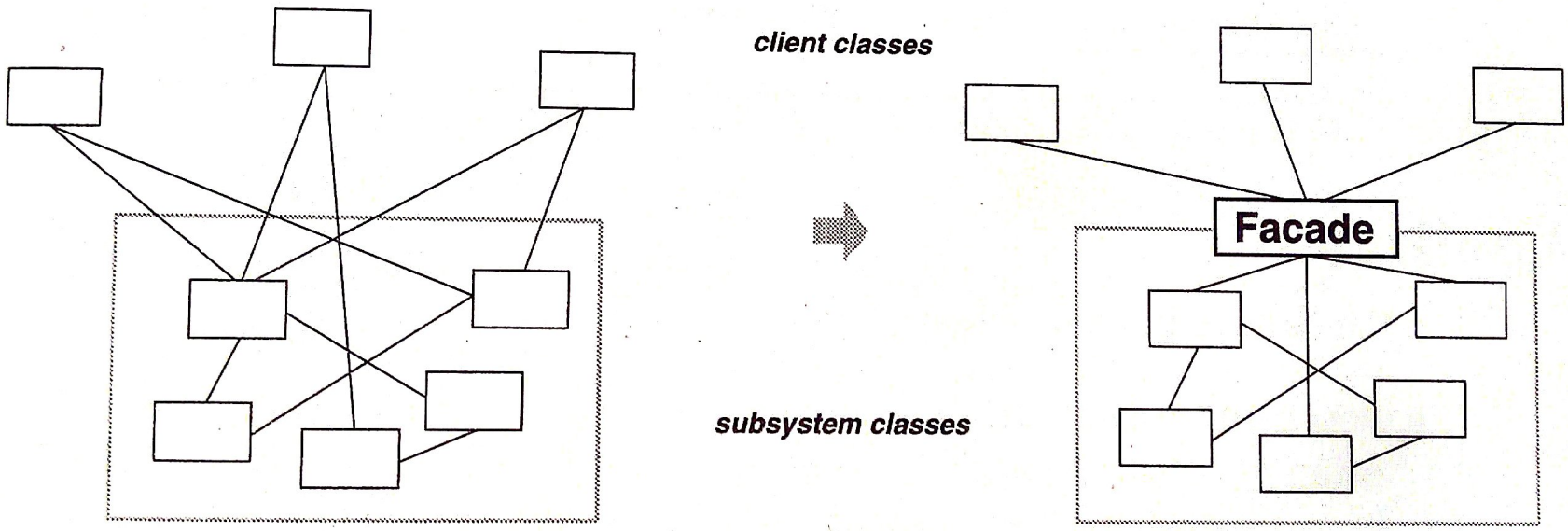

```
public abstract class Application {
  public void openDocument(String name) {
    if (canOpenDocument(name)) {
      Document doc = doCreateDocument(name);
      if (doc != null) {
        docs.add(doc);
        aboutToOpenDocument(doc);
        doc.open();
        doc.read();
      }
    }
  }
}
```

template method

Facade

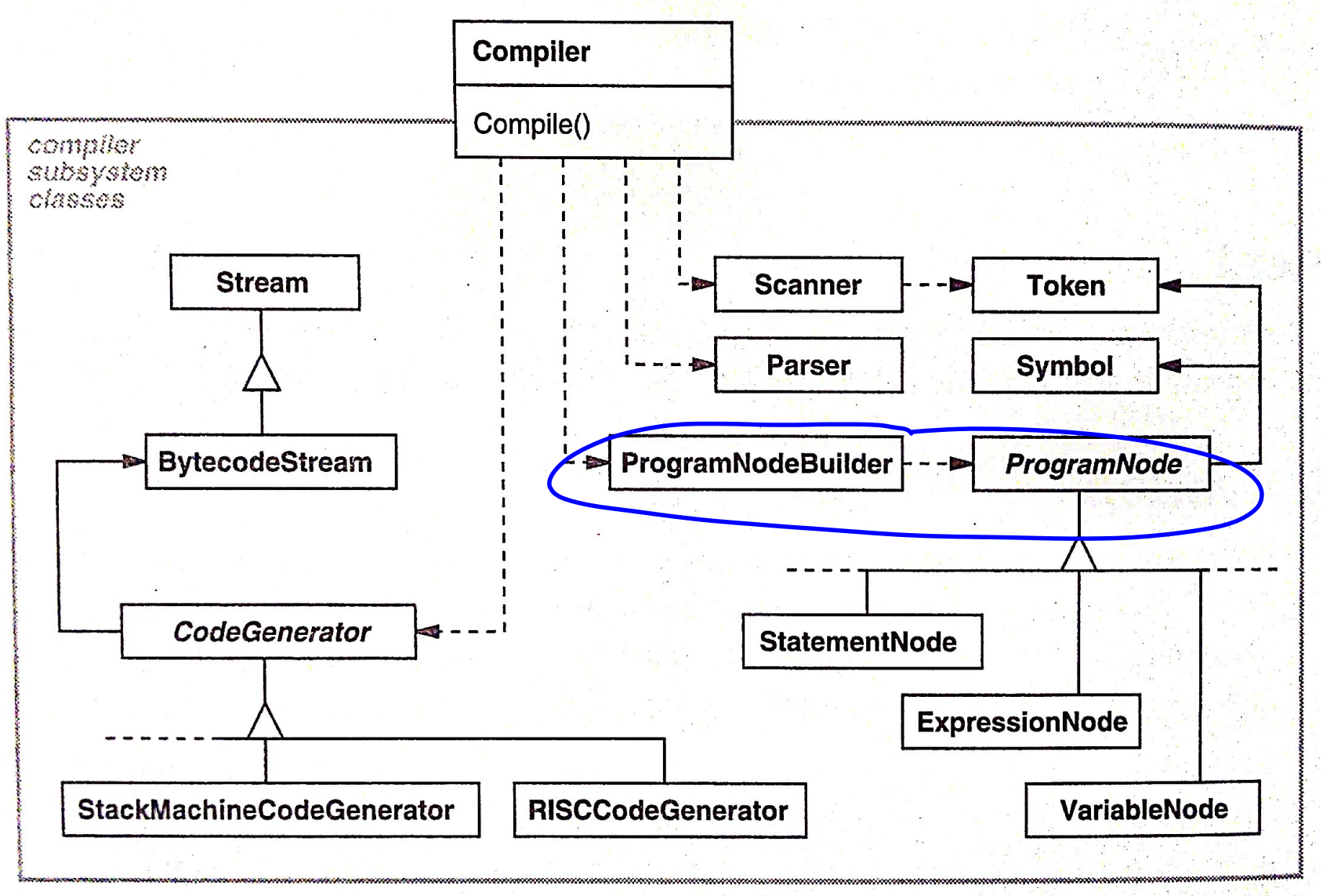
Facade

”Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystems easier to use.”



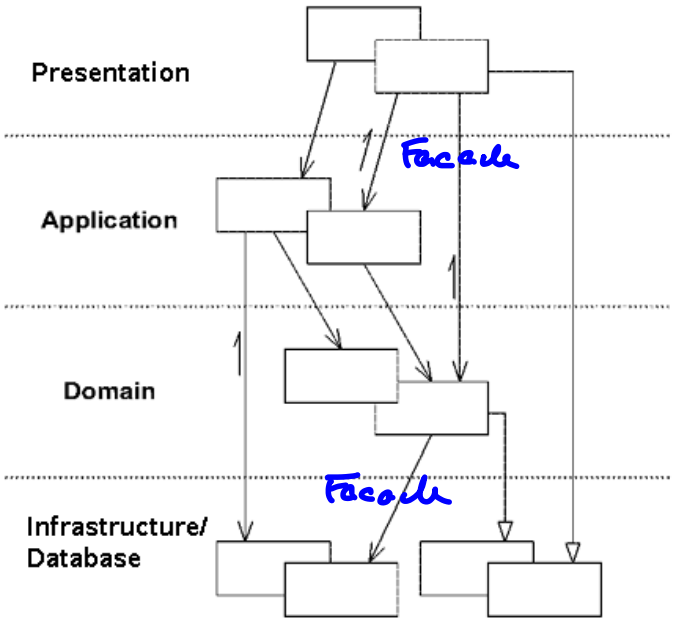
Design Patterns, Addison-Wesley, 1994

Example Compiler



Design Patterns, Addison-Wesley, 1994

Example: Library Application

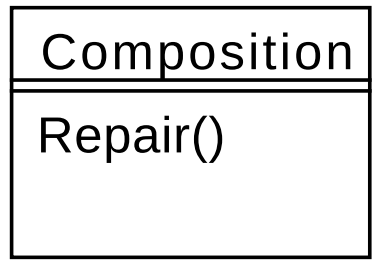


Eric Evans, Domain Driven Design, Addison-Wesley, 2004

- LibApp is the application facade

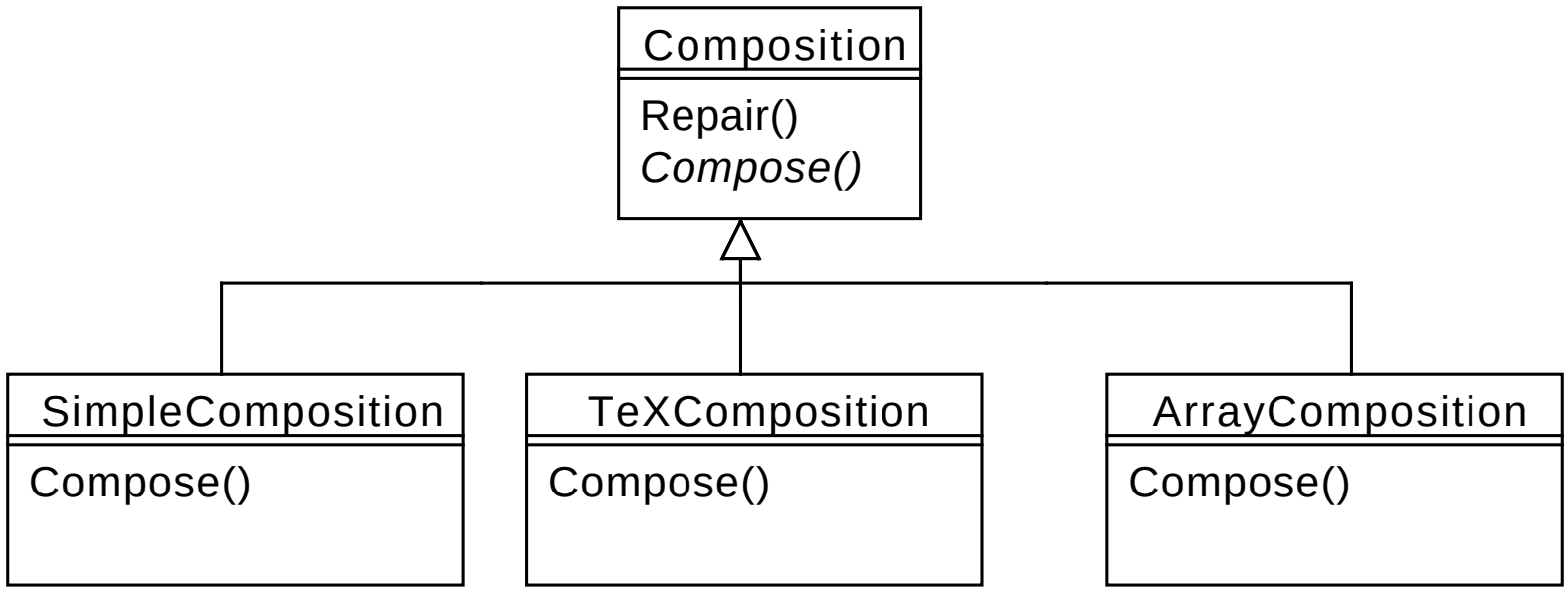
Strategy / Policy: Problem

Different strategies for layouting text: simple, TEX, array, . . . Example: Text formatting



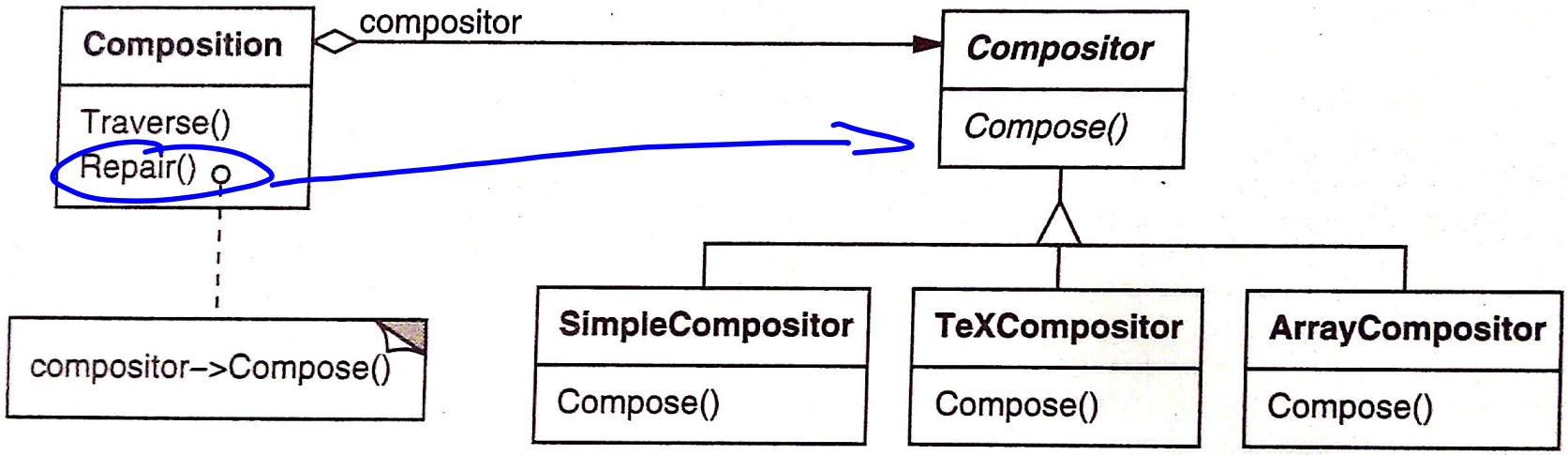

```
public void repair() {
    ...
    if (strategy = "simple")
          // Do simple linebreak
    else if (strategy = "tex")
       // Use TeX's algorithm
    else if (strategy = "array")
       // Do array style linebreak
    ..
}
```

Solution 1: Template Method




```
public void repair() {
   this.compose();
}
```

Strategy Pattern: Solution

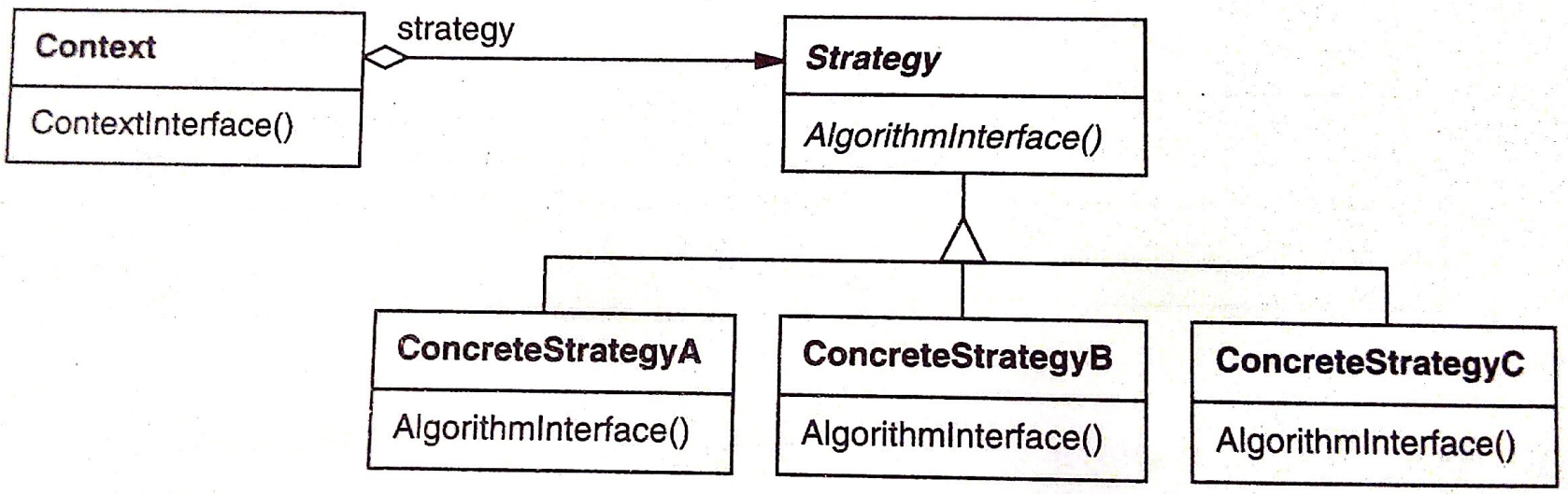


Design Patterns, Addison-Wesley, 1994

Strategy / Policy

Strategy / Policy

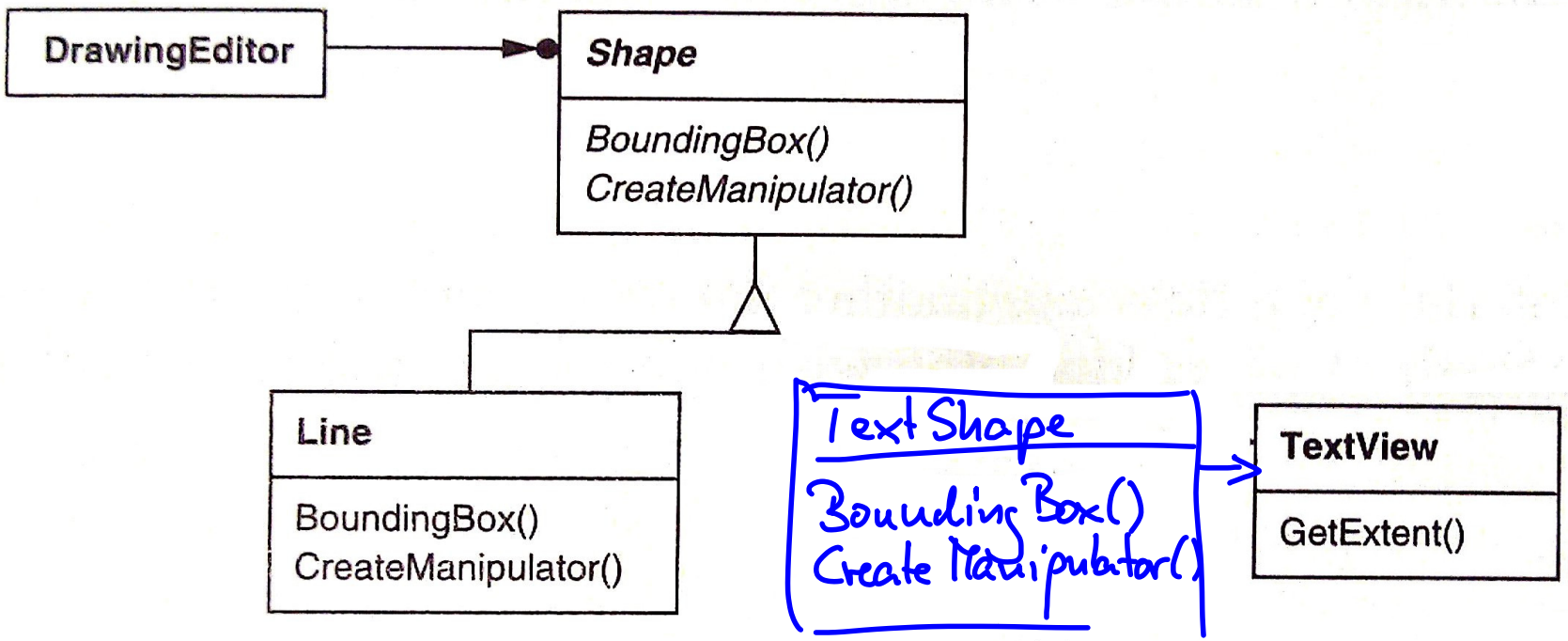
”Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.”



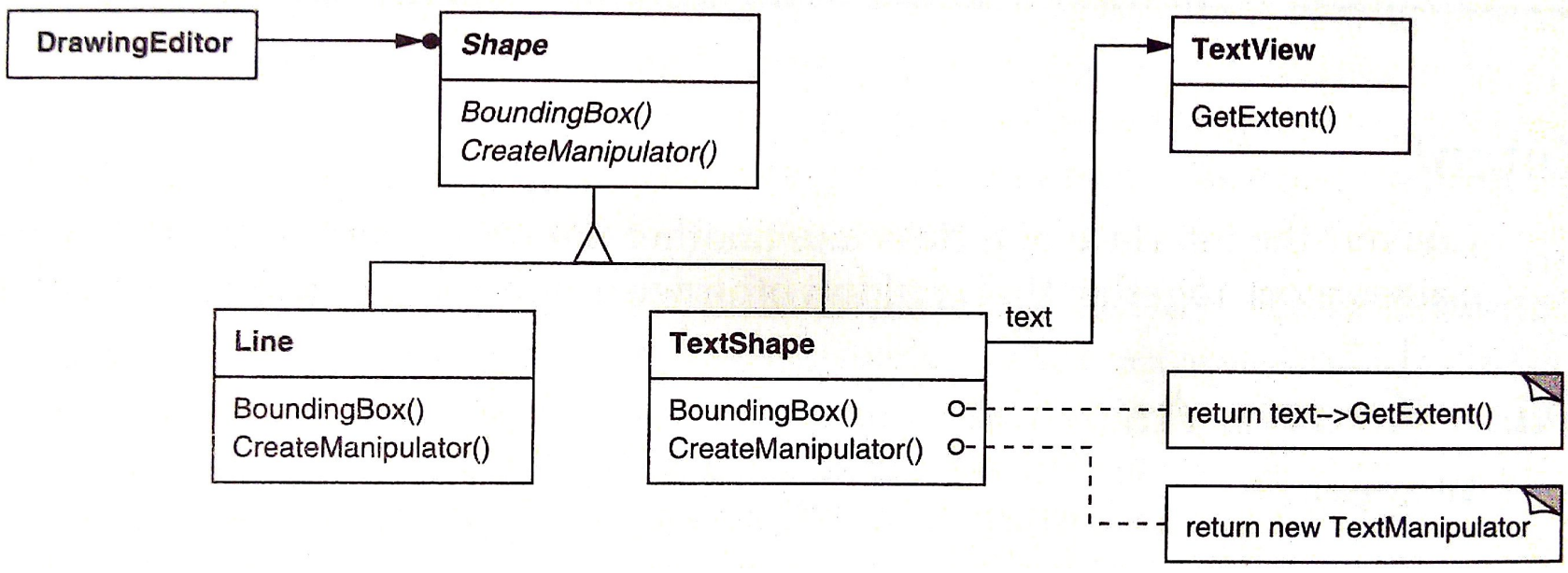
Design Patterns, Addison-Wesley, 1994

Adapter / Wrapper: Problem

- I want to include a text view as part of my graphic shapes
  - Shapes have a bounding box
  - But text views only have an method GetExtent()



Example: Using text views in a graphics editor

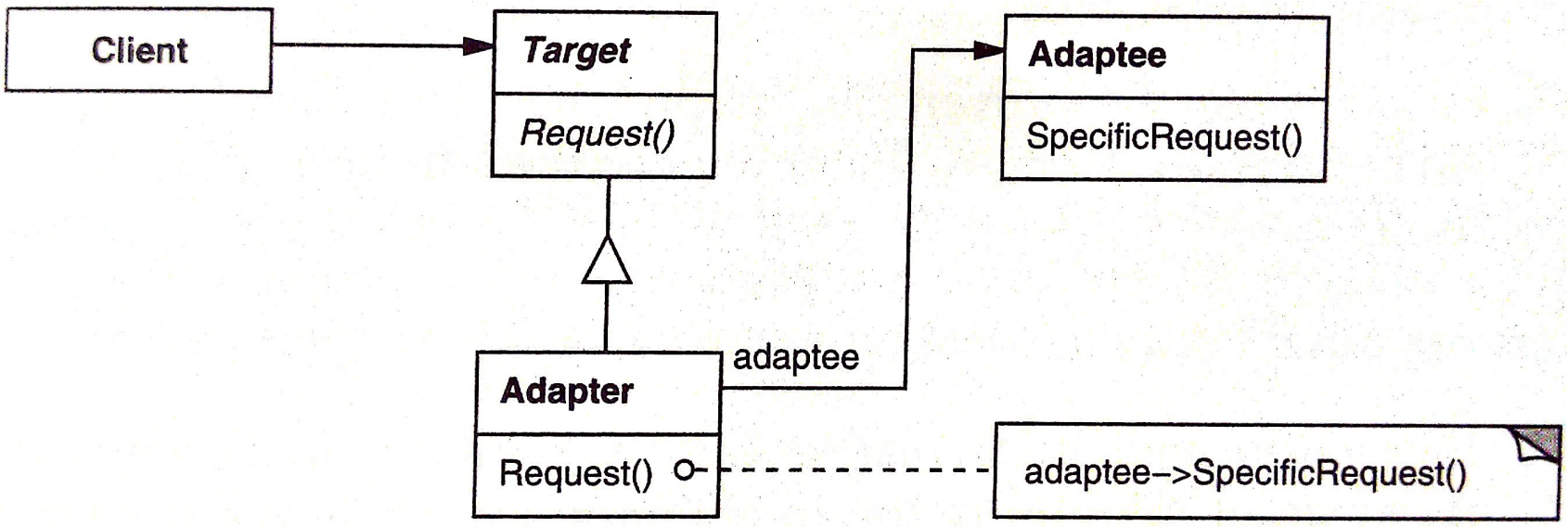


Design Patterns, Addison-Wesley, 1994

Adapter / Wrapper

Adapter / Wrapper

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



Design Patterns, Addison-Wesley, 1994

Anti-Pattern

Anti Pattern

”In computer science, anti-patterns are specific repeated practices that appear initially to be beneficial, but ultimately result in bad consequences that outweigh the hoped-for advantages.” from Wikipedia (`http://en.wikipedia.org/wiki/Anti-pattern`)

- ”Patterns of failure”
- AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis by William J. Brown, Raphael C. Malveau, and Thomas J. Mowbray
- Example: Analysis Paralysis
  - Stuck with developing the analysis model.
  - The model never is good enough.
  - Each time one revisits the same problem, a new variant comes up
  - Solution: Proceed to design and implementation. This gives new insights into the analysis $\rightarrow$ iterative / evolutionary approach
- For a list of anti-patterns see `http://en.wikipedia.org/wiki/Anti-pattern#Recognized.2FKnown_Anti-Patterns`)

Next Week

- User Interface
  - Observer Pattern
  - Model-View-Controller (MVC)
- SOLID Principles