Software Engineering I (02161)

Week 9

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2013

Contents

Sequence Diagrams

Object-orientation: Centralized vs Decentralized Control/Computation

Basic Principles of Good Design

Design Patterns

Sequence Diagram

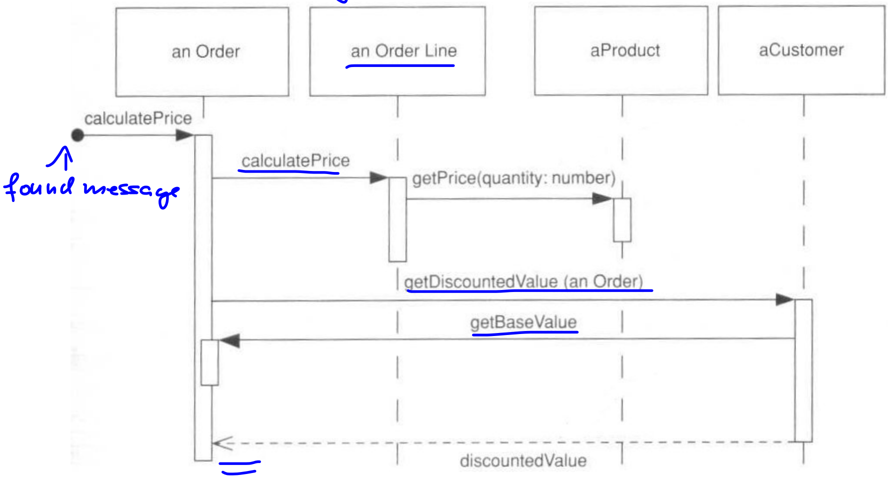
```
public class Order {

  List<OrderLine> orderLines = new ArrayList<OrderLine>();
  private Customer customer;
  double baseValue = 0;

  public double calculatePrice() {
    for (OrderLine ol : orderLines) {
      baseValue += ol.calculatePrice();
    }
    return customer.getDiscountedValue(this);
  }

  public double getBaseValue() {
    return baseValue;
  }
}
```

Sequence Diagram

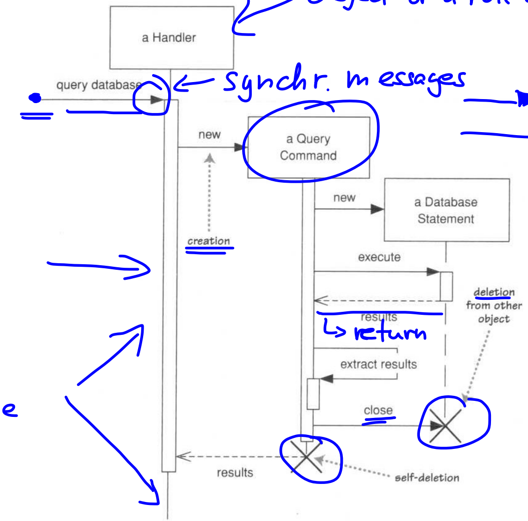
```
public class OrderLine {

  private int quantity;
  private Product product;

  public double calculatePrice() {
    return product.getPrice(quantity);
  }

}


public class Product {

  private double pricingDetails;

  public double getPrice(int quantity) {
    return pricingDetails * quantity;
  }
}


public class Customer {

  public double getDiscountedValue(Order order) {
    return (1 - 5/100)*order.getBaseValue(); // 5% discount
  }
}
```

Sequence diagram

One scenario only



Creation and deletion of participants



Object or a role an object plays

synchr. messages

lost message

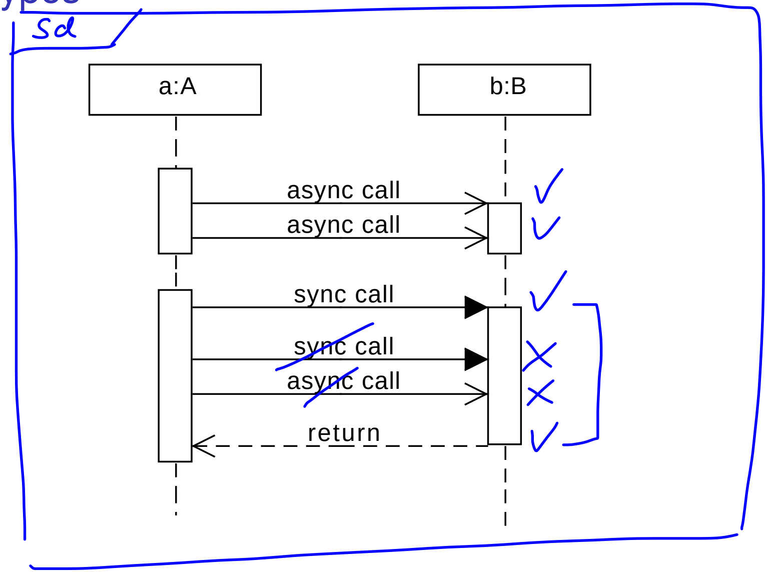
asynch. messages

Activation

↳ return

Lifeline

Arrow types

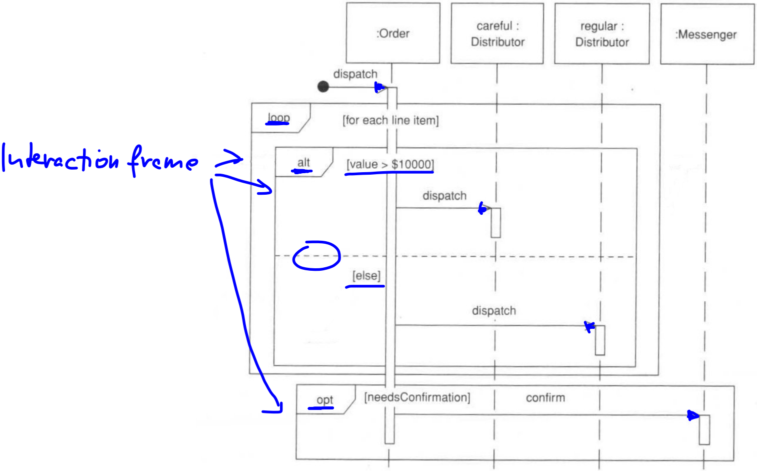

sd
a:A
b:B
async call
async call
sync call
~~sync call~~
~~async call~~
return
✓
✓
✓
✗
✗
✓

Interaction Frames Example

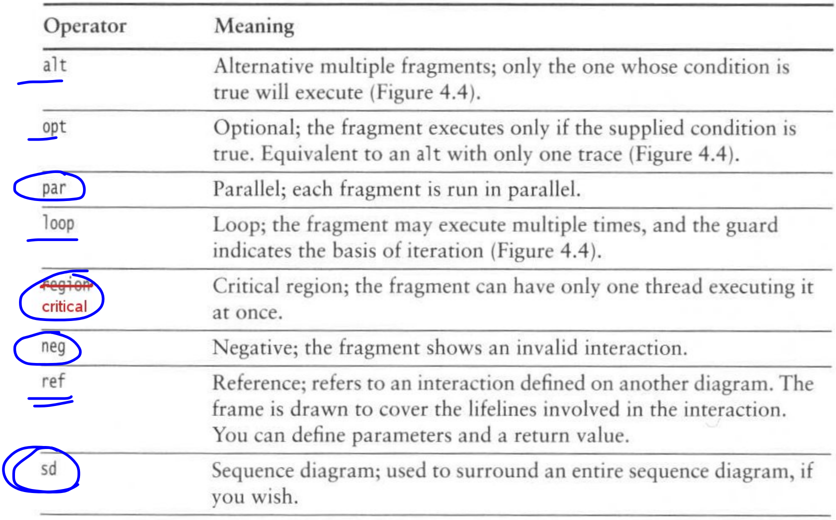
Realising an algorithm using a sequence diagram

```
public void dispatch() {
  for (LineItem lineItem : lineItems) {
    if (lineItem.getValue() > 10000) {
      careful.dispatch();
    } else {
      regular.dispatch();
    }
  }
  if (needsConfirmation()) {
    messenger.confirm();
  }
}
```

Realisation with Interaction Frames



Interaction Frame Operators I

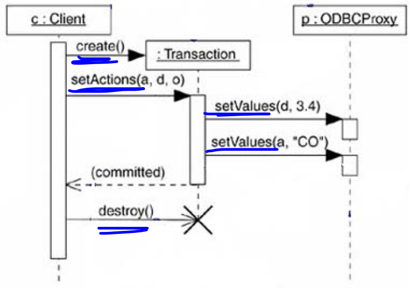


Operator	Meaning
alt	Alternative multiple fragments; only the one whose condition is true will execute (Figure 4.4).
opt	Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (Figure 4.4).
par	Parallel; each fragment is run in parallel.
loop	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4).
~~region~~ critical	Critical region; the fragment can have only one thread executing it at once.
neg	Negative; the fragment shows an invalid interaction.
ref	Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram; used to surround an entire sequence diagram, if you wish.

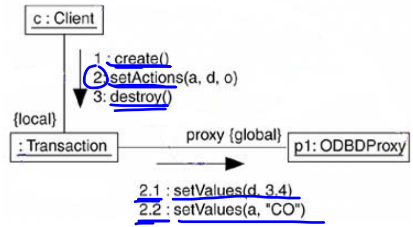
Interaction Diagrams

Interaction Diagrams = Sequence + Communication Diagrams

Sequence Diagram



Communication Diagram



Focus:

timeing of msgs.

if object struct. is important

Usages of sequence diagrams

- Abstract: show the execution (i.e. exchange of messages) of a system
- Concrete
  - Design (c.f. CRC cards)
  - Visualize program behaviour

Use Case: borrow book

Use of seq. diagr. examination report

name: borrow book
description: the user borrows a book
actor: user

main scenario:

1. the user borrows a book

alternative scenario

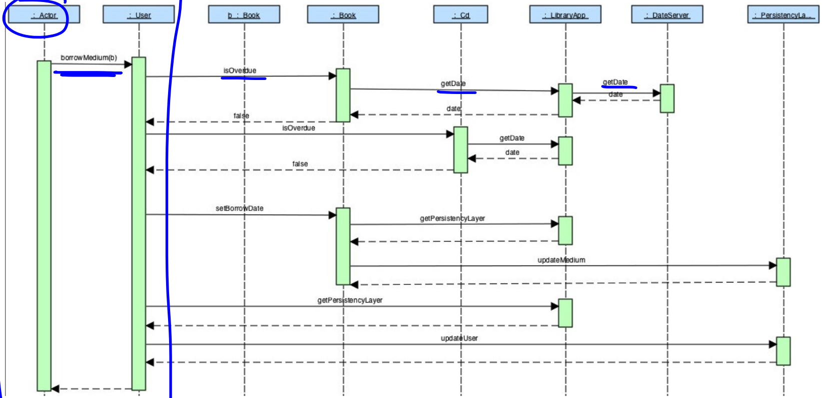
1. the user wants to borrow a book, but has already 10 books borrowed
2. the system presents an error message

Program: borrow book

```
public class User extends PersistentObject {
  public void borrowMedium(Medium medium) throws BorrowException {
    if (medium == null)
      return;
    if (borrowedMedia.size() >= 10) {
      throw new TooManyBooksException();
    }
    for (Medium mdm : borrowedMedia) {
      if (mdm.isOverdue()) {
        throw new HasOverdueMedia();
      }
    }
    medium.setBorrowDate(libApp.getDate());
    borrowedMedia.add(medium);
    try {
      libApp.getPersistencyLayer().updateUser(this);
    } catch (IOException e) {
      throw new Error(e);
    }
  }
}
```

Sequence diagram: borrow book success scenario

Actor



Use case scenario

Contents

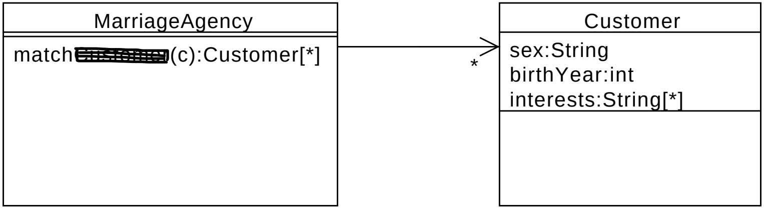
Sequence Diagrams

Object-orientation: Centralized vs Decentralized Control/Computation

Basic Principles of Good Design

Design Patterns

Marriage Agency: centralized control



MarriageAgency

match~~Customer~~(c):Customer[*]

*

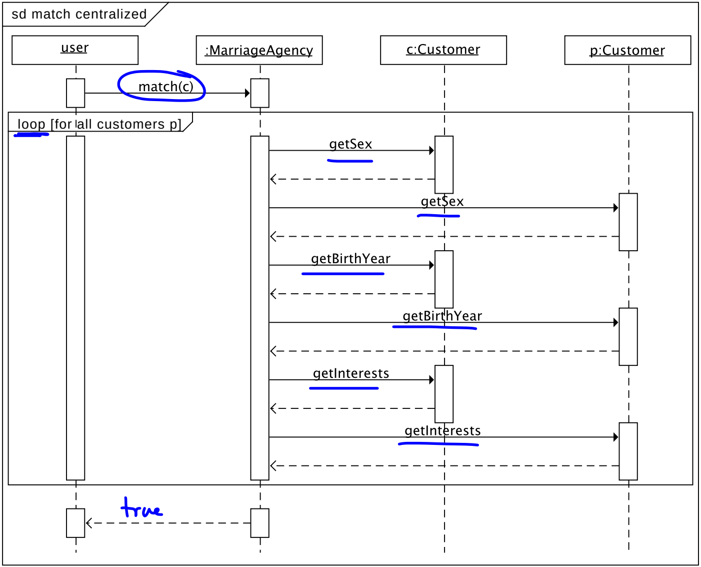
Customer

sex:String

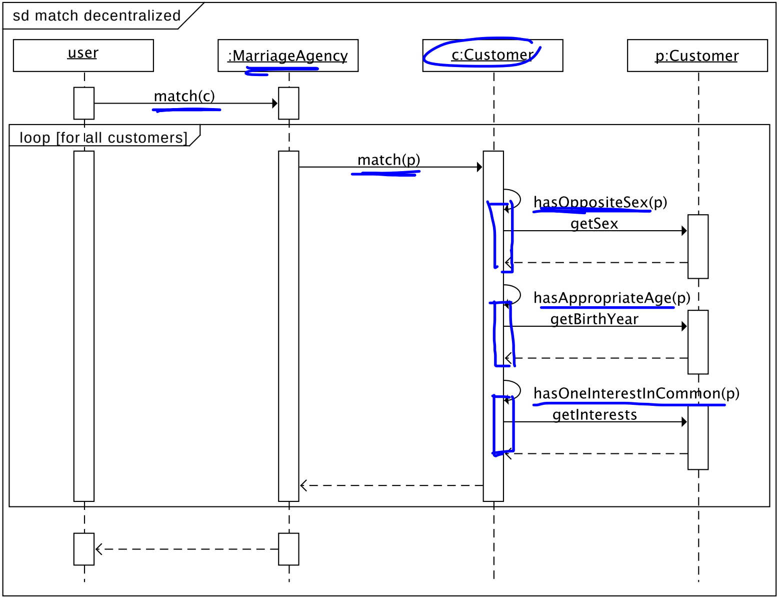
birthYear:int

interests:String[*]

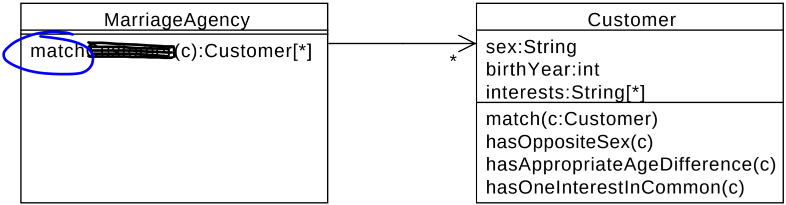
Marriage Agency: centralized control



Marriage Agency: decentralized/distributed control



Marriage Agency: decentralized/distributed control



MarriageAgency

match~~Customers~~(c):Customer[*]

*

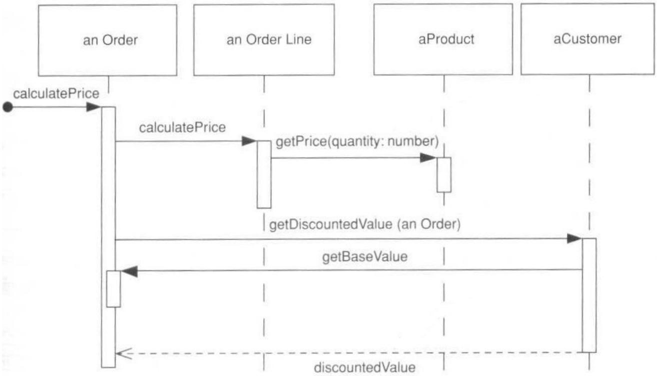
Customer

sex:String
birthYear:int
interests:String[*]

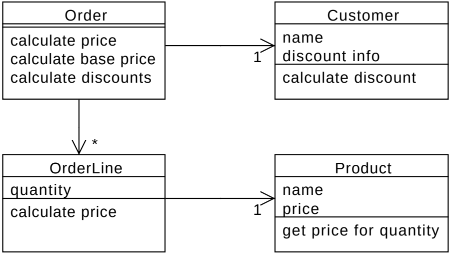
match(c:Customer)
hasOppositeSex(c)
hasAppropriateAgeDifference(c)
hasOneInterestInCommon(c)

Real objects : state + operations

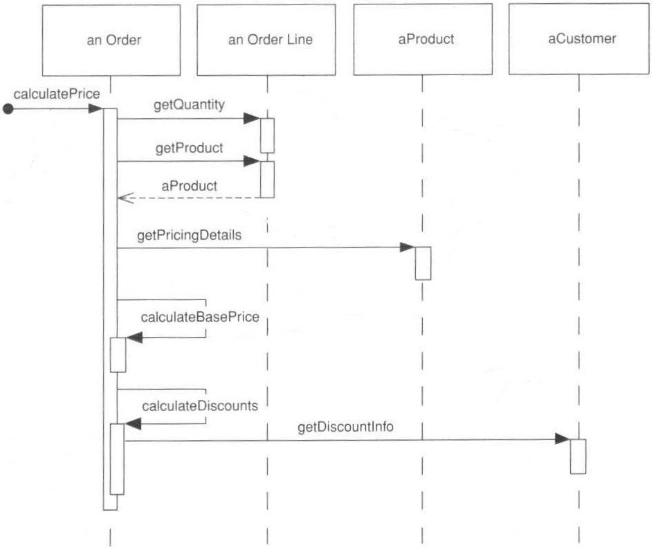
Distributed control



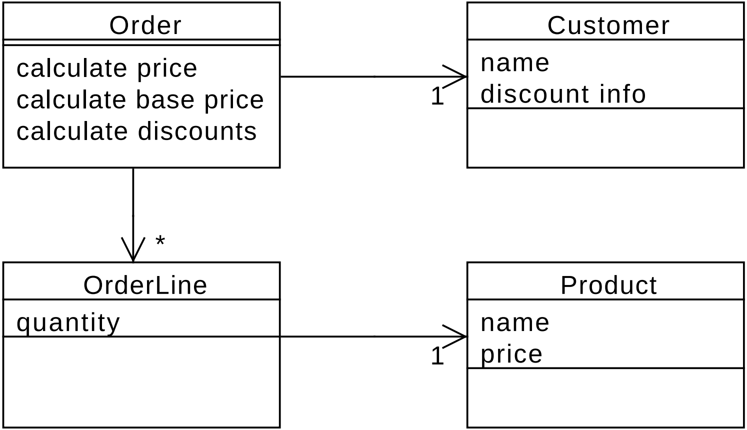
Distributed Control: Class diagram



Centralised control



Centralized control



Centralized vs Distributed control

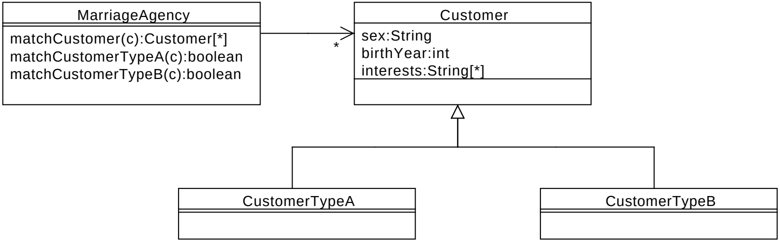
- Centralized control
  - **One method**
  - Data objects

  → procedural programming language
- Distributed control
  - Objects **collaborate**
  - Objects = data *and* behaviour

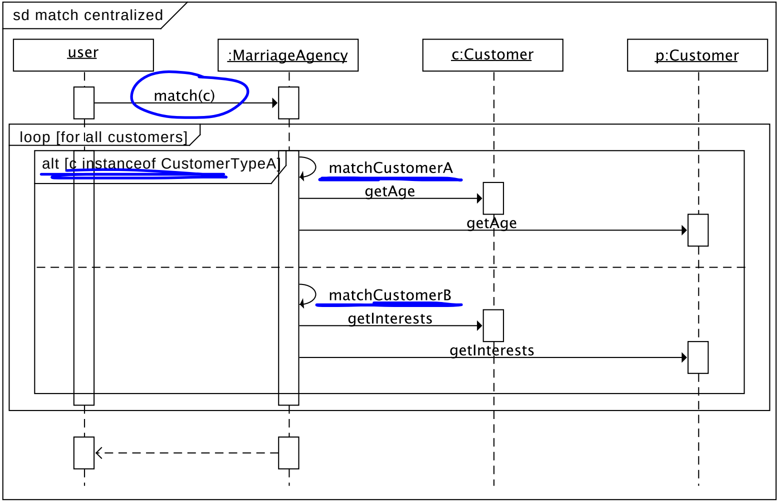
  → Object-orientation
- Advantage
  - Easy to adapt

  → Design for *change*

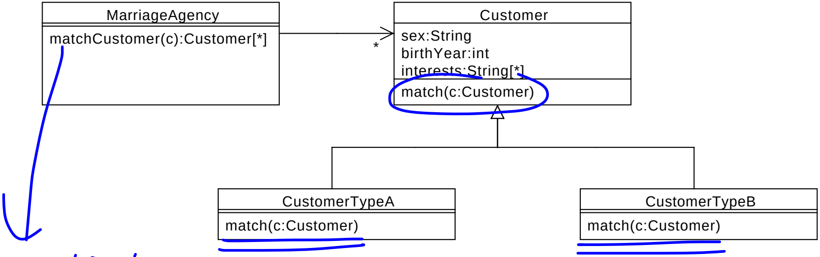
Design for change: centralized control



Design for change: centralized control

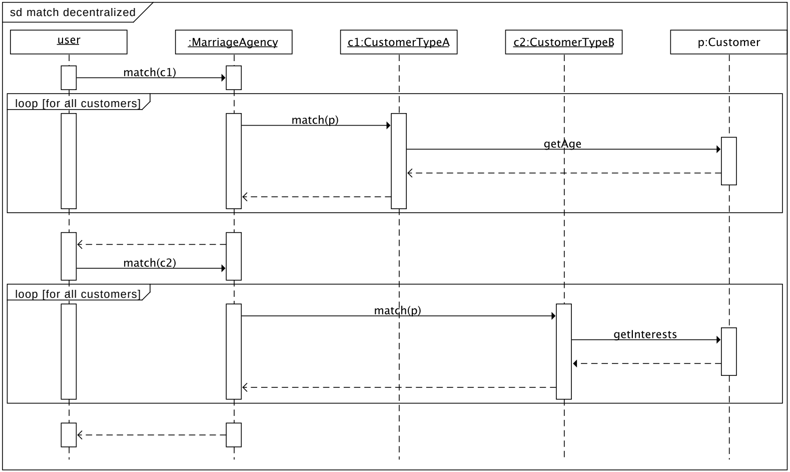


Design for change: decentralized control



generic match alg.

Design for change: decentralized control



Contents

Sequence Diagrams

Object-orientation: Centralized vs Decentralized Control/Computation

Easy to understand

Basic Principles of Good Design

Intention revealing names

Adaptable

Performance ?

low coupling / high cohesion

"Reuse"

Design Patterns

DRY principle

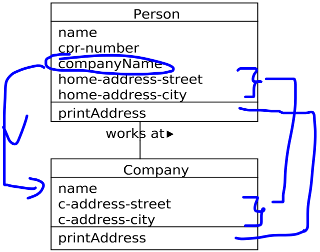
DRY principle

Don’t repeat yourself

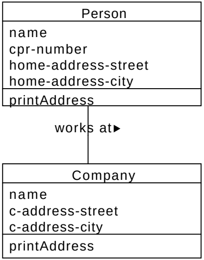
”Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” The Pragmatic Programmer, Andrew Hunt and David Thomas

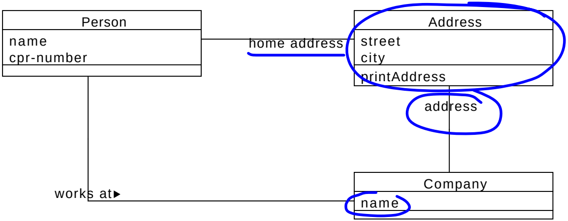
- code
- documentation
- build stystem

Example: Code Duplication



Example: Code Duplication





DRY 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

balance

Contents

Sequence Diagrams

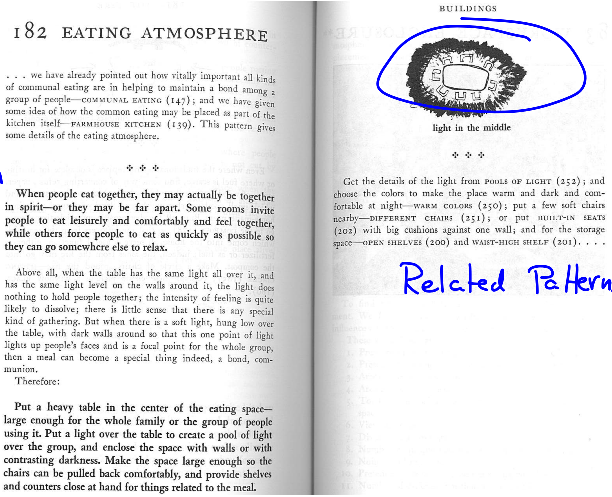
Object-orientation: Centralized vs Decentralized Control/Computation

Basic Principles of Good Design

Design Patterns

- Observer Pattern

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.

❖ ❖ ❖

Problem

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.

Forces

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, a communion.

Therefore:

Solution

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.

BUILDINGS

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). . . .

Related Patterns

A Pattern Language, Christopher Alexander, 1977

Pattern and pattern language

- Pattern: a *solution* to a *problem* in a context
- Pattern language: set of related patterns

Design Patterns

History of Patterns

- Chstiopher Alexander: Architecture (1977/1978)
- Kent Beck and Ward Cunningham: Patterns for Smalltalk applications (1987)
- Design Patterns book (1994) GoF
- Portland Pattern Repository `http://c2.com/ppr`
  → origin of *wikis*

Design Patterns

- Defined in the Design Pattern Book
- Best practices for object-oriented software
  → use of *distributed control*
- Types: Creational Patterns, Structural Patterns, Behavioral Patterns
- Places to find patterns:
  - Wikipedia `http://en.wikipedia.org/wiki/Design_pattern_(computer_science)`
  - Portland Pattern repository `http://c2.com/cgi/wiki?PeopleProjectsAndPatterns` (since 1995)
  - Wikipedia `http://en.wikipedia.org/wiki/Category:Software_design_patterns`

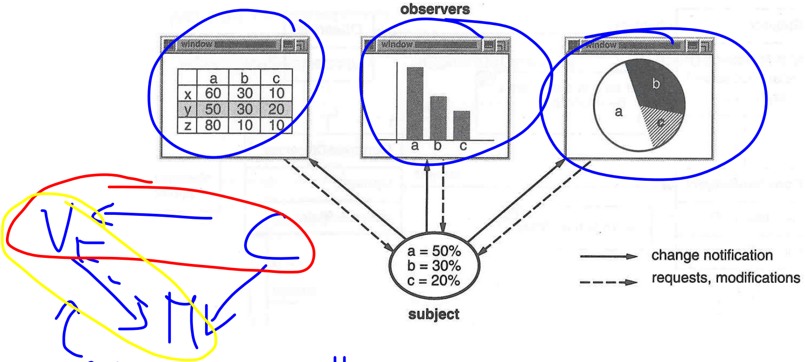
Design Pattern structure

- Alexander: Context, Problem, Forces, Solution, Related Pattern
- Design Patterns: Intent, Motiviation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses, Related Patterns

Observer Pattern

Observer Pattern

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

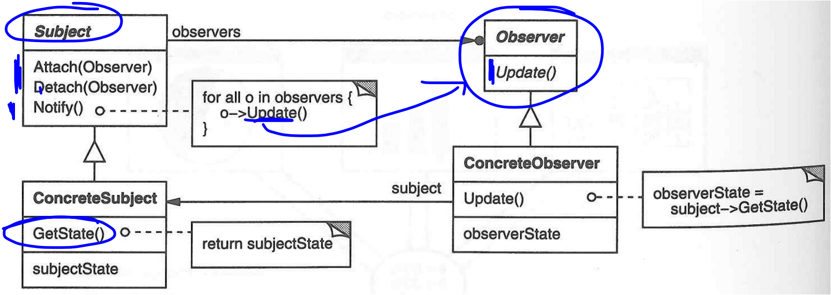


	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10

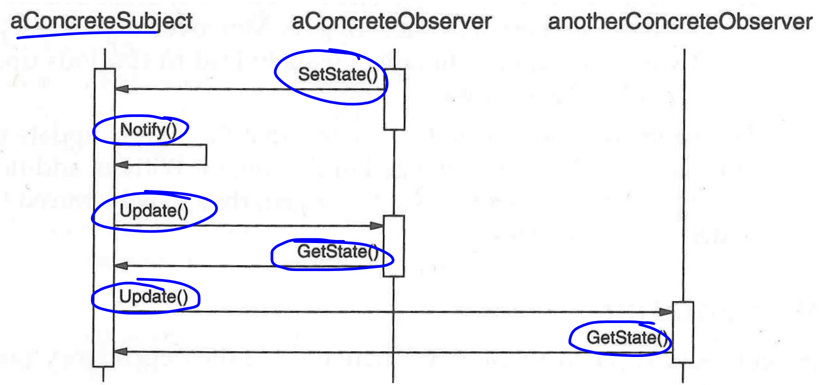
Observer pattern

Observer Pattern

Observasle = Susject



Observer Pattern



Implementation in Java

- java.util.Observer: Interface
  - update(Observable o, Object aspect)
- java.util.Observable: Abstract class
  - addObserver, deleteObserver
  - setChanged
  - notifyObservers(Object aspect)

have an impl.