Design Patterns & Software Architecture

A deeper look at patterns

dr. Joost Schalken-Pinkster

University of Applied Science Utrecht

The Netherlands

The contents of these course slides is (in great part) based on:
Chris Loftus, *Course on Design Patterns & Software Architecture for NEU*. Aberystwyth University, 2013.
Jeroen Weber & Christian Köppe, *Course on Patterns and Frameworks*. Hogeschool Utrecht, 2013.
Leo Pruijt, *Course on Software Architecture*. Hogeschool Utrecht, 2010-2013.

Session overview

- ....

A deeper look at patterns

Definition Design Patterns

*A software design pattern is a **solution** to a commonly occurring **problem** in a **context**…*

Elements of patterns

In general, a pattern has four essential elements:

- The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
- The **problem** describes when to apply the pattern. It explains the problem and its context.
- The **solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations. The solution doesn't describe a particular concrete design or implementation.
- The **consequences** are the results and trade-offs of applying the pattern.

Gemma et al.(1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

Documenting patterns
(format by Gamma, Helm, Johnson and Vlissides)

Pattern name	The name of the pattern
Intent	Purpose of the pattern
Also known as	Any other names by which the pattern is known.
Motivation	Illustrates how pattern solves a particular problem.
Applicability	Explains where pattern is and is not applicable.
Structure	This is a description of the relationships.
Participants	Classes and objects and responsibilities in design.
Collaborations	How classes and objects work together.
Consequences	Benefits and drawbacks of pattern.
Implementation	Trade offs, design decisions etc.
Sample Code	Code illustrating how to implement the pattern
Known uses	How the pattern has been used in the past.
Related patterns	Closely related design patterns are listed here.

Gamma et al.(1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

Pattern classification

- Purpose of pattern:
  - **Creational patterns:** Singleton, Factory Method, Abstract Factory, Builder…
  - **Structural patterns:** Adapter, Composite, Decorator, Bridge, Proxy…
  - **Behavioural patterns:** Iterator, Visitor, Observer, State, Strategy, Template Method…

- Scope of pattern:
  - **Object:** deal with object relationships, that can be changed at runtime.
  - **Class:** relationships between classes and their subclasses, that can only be changed at compile time.

What are Creational Design Patterns?

- They abstract the instantiation process…
- They help make the application more independent of how its objects are created…
- Therefore, they help make applications more maintainable…

Three recurring creational themes

- Most creational design patterns encapsulate knowledge about which concrete class is used…
- All hide how and even when objects are created…
- ‘Client’ code is programmed against interfaces or abstract classes that represent those parts of the application which vary over time…

Creational Design Patterns we looked at

- Singleton
- Factory method
- Abstract factory

We did not look at the following:

- Prototype
- Builder

What are structural design patterns?

- Concerned with how classes and objects are composed to form larger structures.

- Two kinds:
  - Structural class patterns use inheritance…
  - Structural object patterns compose objects…

Structural patterns we looked at

- Decorator
- Adapter
- Composite

And only if time permits:

- Façade (when discussing components)

We did not look at the following:

- Flyweight
- Bridge
- Proxy

Behavioural Design Patterns

- Concerned with algorithms and their assignment to classes
- Describe patterns of communication between classes…
- Two main kinds:
  - Behavioural class patterns use inheritance to distribute behaviour between classes…
  - Behavioural object patterns use composition to allow dynamic changing of behaviour…

Behavioural patterns we looked at

We did discuss

- Observer
- Strategy
- Command
- Template method
- Iterator
- Memento
- State
- Chain of responsibility

We did not look at:

- Interpreter
- Mediator
- Visitor

Class vs object based design patterns…

Class-based Patterns

- Factory Method
- Adapter (both object and class)
- Template method
- *Interpreter*

Object-based Patterns

Creational Patterns

- Abstract Factory
- Singleton
- Builder
- *Prototype*

Structural Patterns

- Decorator
- Adapter (both object and class)
- Composite

Object-based Patterns (continued)

Structural Patterns (continued)

- Proxy
- *Façade*
- *Bridge*
- *Flyweight*

Behavioural Patterns

- Strategy
- Observer
- Command
- Memento
- Iterator
- State
- Chain of responsibility
- *Visitor*
- *Mediator*

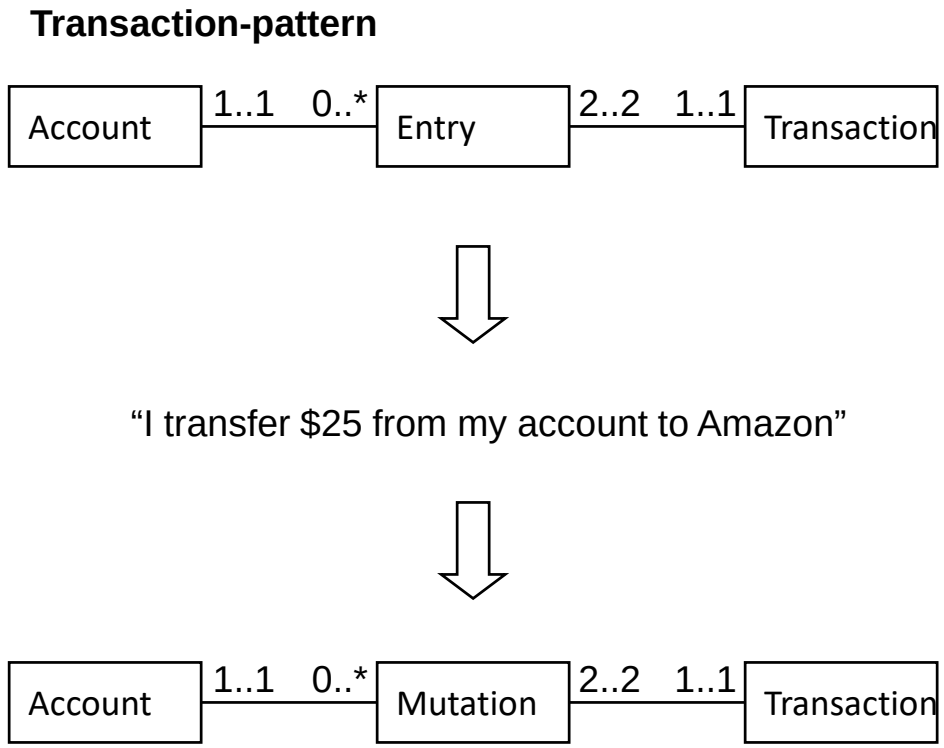
Gamma et al.(1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

Pattern classification

- Level of pattern:
  - **Analysis patterns:** related to requirements and analysis
  - **Architectural patterns/style:** related to software architecture
  - **Design patterns:** related to software design
  - **Implementation patterns/idioms:** related to language specific implementations

Analysis pattern example

- An analysis-pattern is a resource to transform the reality into a suitable analysis-model.
- A analysis-pattern a conceptual solution that was successful in a certain situation and could be helpful in others as well.
- The conceptual solution has to be adapted to the specific situation!

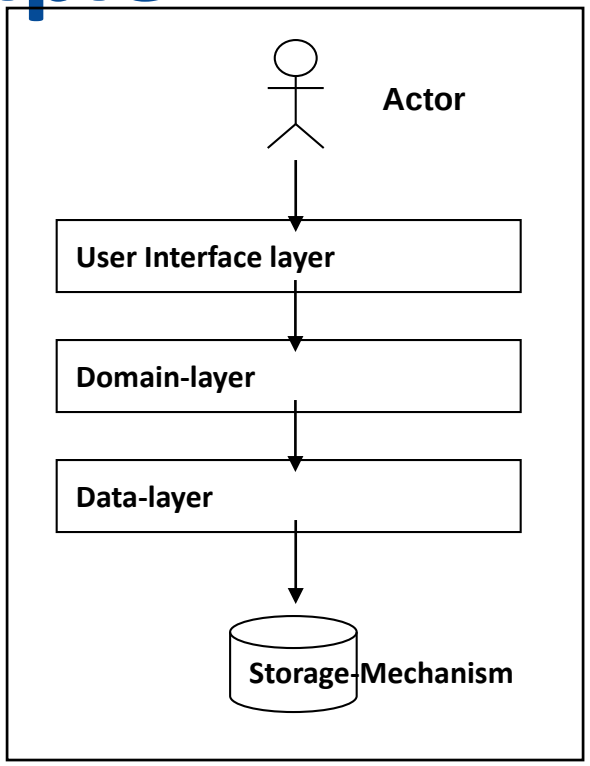


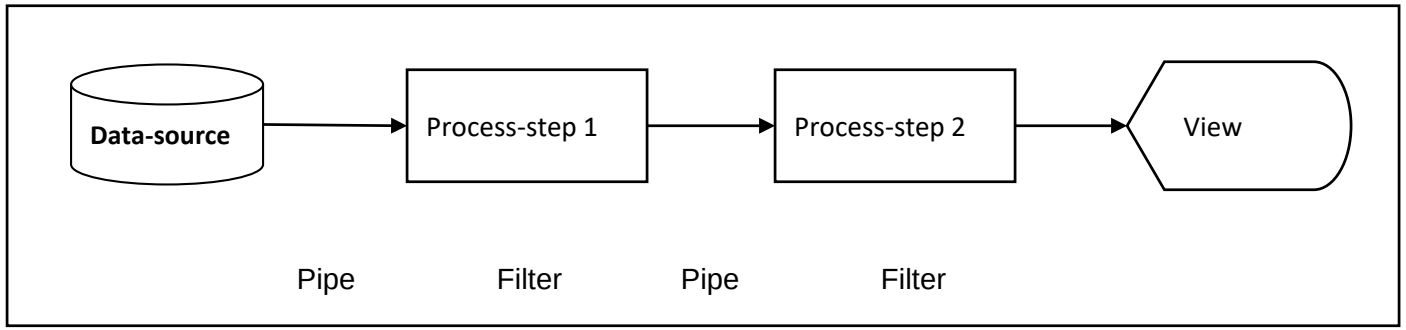
Architecture Patterns example

Architecture-Patterns give solutions to determine the fundamental structure of a software system

Examples:

- Layer-pattern
- Pipes and Filters
- Broker-pattern
- Model-View-Controller (MVC)



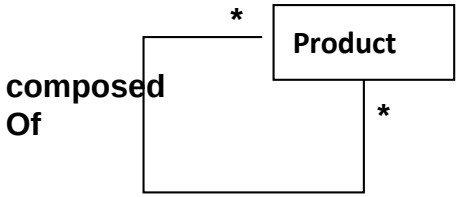


Design Pattern example

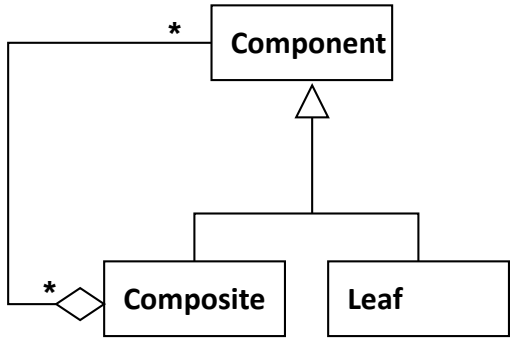
Design-patterns give solutions for small-scale problems.

Examples:

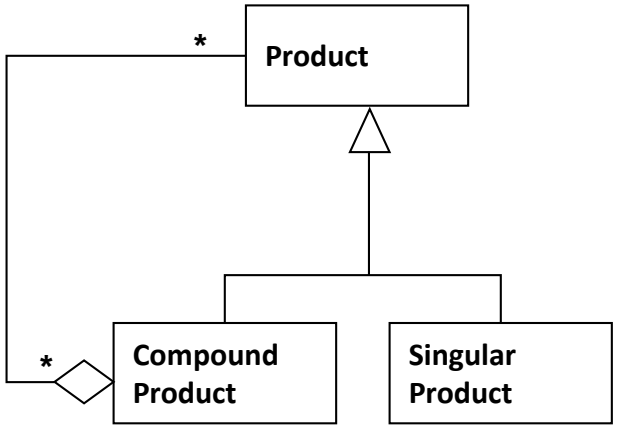
- Composite pattern
- Observer pattern
- Proxy pattern



Analysis model



Composite- Pattern



Design model

Implementation pattern

Implementation-patterns give solutions for language related implementation problems.

Example: PIMPL

```
//header file:
class Handle {
public:
  Handle();                                  // Constructor
  Handle(const Handle&);                     // Copy constructor
  Handle(Handle&&);                          // Move constructor
  Handle& operator=(const Handle&);          // Copy assignment oper
  ~Handle();                                 // Destructor

   // Other operations...
private:
  struct CheshireCat;     // Not defined here
  CheshireCat* smile;     // Handle
};

//CPP file:
#include "handle.h“

struct Handle::CheshireCat {
 int a; int b;
};

Handle::Handle()
 : smile(new CheshireCat()) {
 // do nothing
}

Handle::Handle(const Handle& other)
 : smile(new CheshireCat(*other.smile)) {
 // do nothing
}
```

Design Patterns covered

Creational Patterns

- Abstract Factory
- Factory Method
- Singleton
- ~~*Builder*~~
- ~~*Prototype --*~~

Structural Patterns

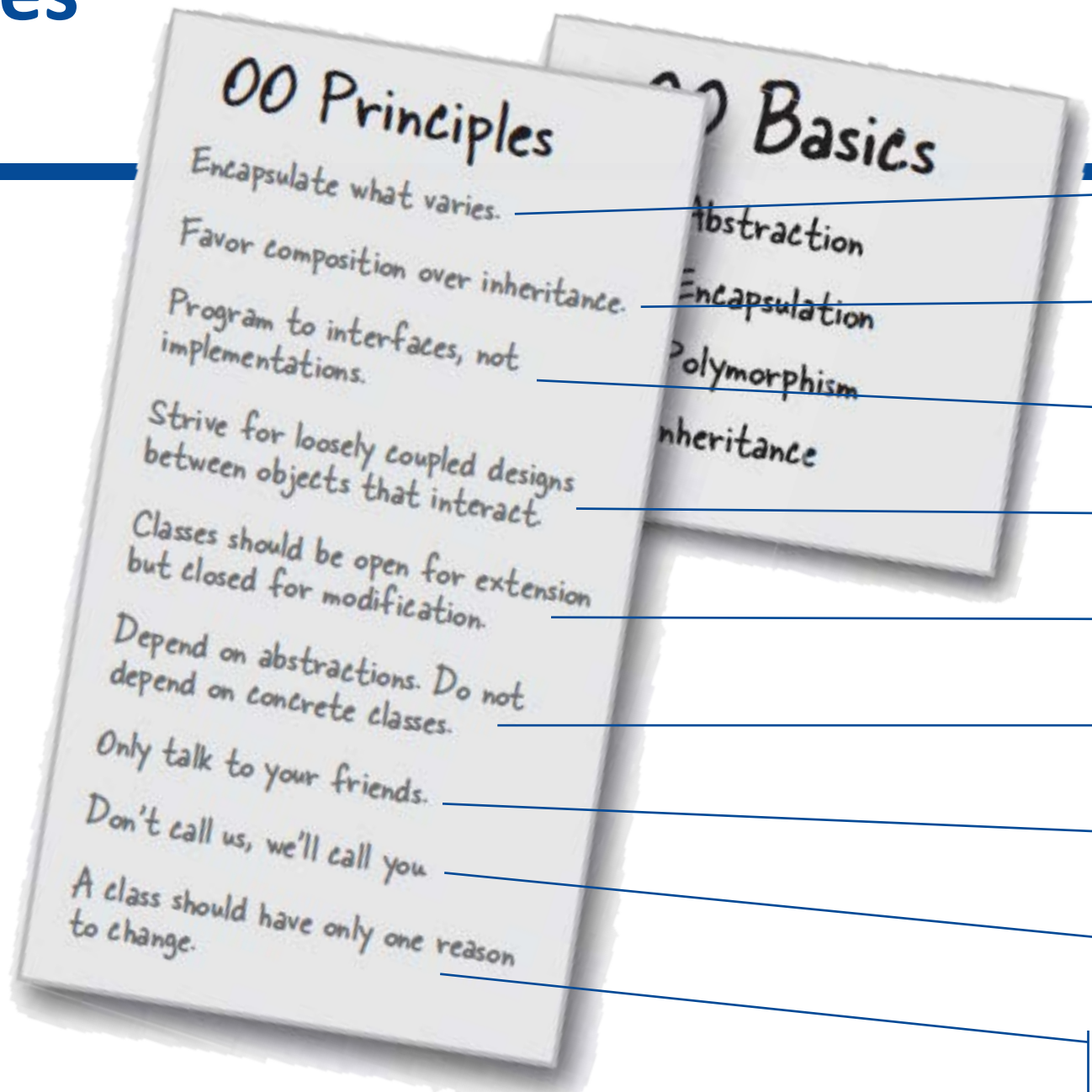
- Decorator
- Adapter
- Composite
- *Façade (time permitting during architecture)*
- ~~*Proxy (time permitting)*~~
- ~~*Bridge*~~
- ~~*Flyweight --*~~

Behavioural Patterns

- Strategy
- Observer
- Command
- Memento
- Template method
- Iterator
- State
- Chain of responsibility
- ~~Visitor --~~
- ~~*Interpreter --*~~
- ~~*Mediator --*~~

Gamma et al.(1995). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

Design Principles covered



Reading

For tomorrow please read:

- Chapter 9 (Well managed collections) of Head First Design Patterns.
- Chapter 13 (Patterns in the Real World) of Head First Design Patterns
- Chapter Chain of Responsibility of Design Patterns: Elements of Reusable Object-Oriented Software, pp 251-262.