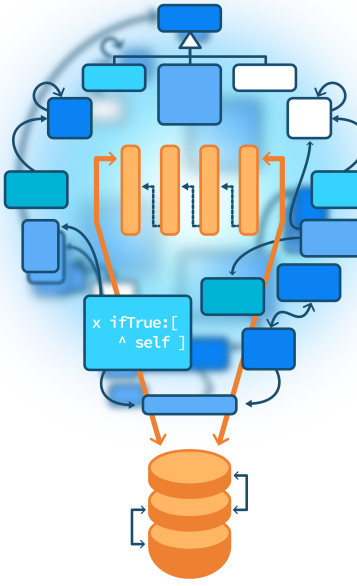
An introduction to design patterns

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone

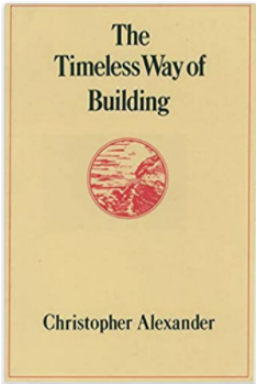


http://www.pharo.org

Goal

- What are design patterns?
- Why they are good?
- But patterns are not panacea
- Examples

Influenced by work on architecture



C. Alexander, *The Timeless Way of Building*, Oxford University Press, 1979

- tried to let people use patterns to build houses and cities
- developed patterns and pattern languages
- failed in architecture but the idea is successful for object-oriented design

Design Patterns

Design patterns are **recurrent solutions to design problems**

- Identification of similar problems at **design** level
- Experts solved **recurring problems** in similar ways
- There are **pros and cons**
- Design Patterns are literature: **Read them!**

What are design patterns?

- **Elegant** solutions that a novice would not think of
- **Generic**
- **Independent** of specific type system or language
- **Well-proven**
  - Successfully tested in several systems
- **Simple**
- Can be **combined** together for more complex solutions

Watchout!

- There are really stupid patterns (e.g., supersuper) in some books
- Stay sceptical

The important points

Design Patterns are **names**

- They create a design **vocabulary**
  - Hook and Template, Factory, Composite, Visitor, Observer, Decorator...
- You can talk at lunch/blackboard about your design in an **abstract but precise** way
- Design patterns are **literature**:
  - the books are good
  - read them, reread them, and read them again!

What design patterns are not

- A design pattern is **not** just one implementation
- A design pattern is **illustrated** using one **possible** implementation
- But alternate implementations exist

Elements of a pattern

- **Pattern name**: Increase design vocabulary
- **Pattern intent**: Describe the goal
- **Problem description**: When to apply it, in what context to use it
- **Solution description** (generic):
  - the elements that make up the design
  - their relationships, responsibilities, and collaborations
- **Consequences**: Results and trade-offs of applying the pattern

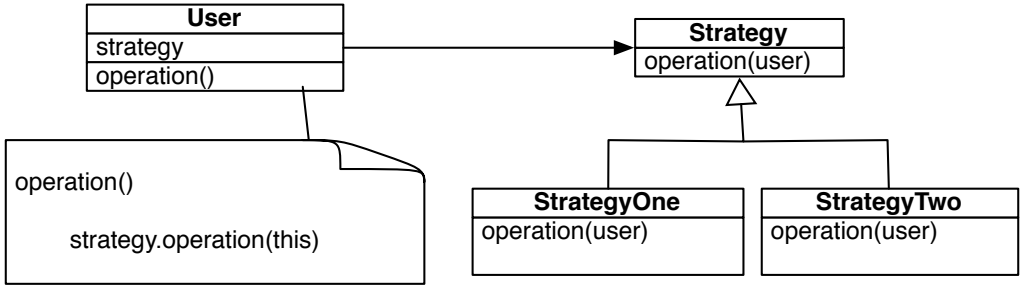
Example: Strategy

Intent:

- Define a family of algorithms
- Encapsulate each in a separate class and
- Define each class with the **same** interface so that they can be **interchangeable**

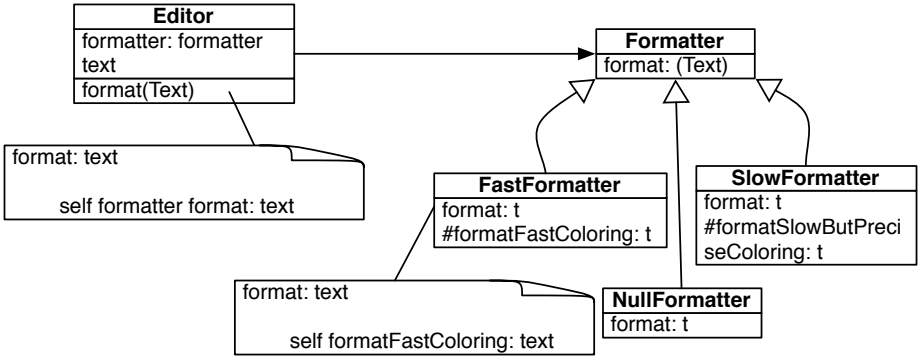
Essence of Strategy

A possible implementation



- Variation: Passing or not the user to the strategy?

Strategy application



Some categories

Creational Patterns

- Instantiation and configuration of classes and objects

Structural Patterns

- Usage of classes and objects in larger structures, separation of interfaces and implementation

Behavioral Patterns

- Algorithms and division of responsibility

Some creational patterns

- Abstract factory (how to create objects)
- **Builder** (provide a programmatic way to create objects)
- Factory Method
- Prototype
- **Singleton** (watch out most people get it wrong)

Some structural patterns

- Adapter
- Bridge
- **Composite** (support nested structure and common api)
- **Decorator**
- Façade (only useful is super limited case)
- **Flyweight** (how to avoid creating too many objects)
- Proxy

Some behavioral patterns

- Chain of responsibility
- **Command** (reify operations)
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- **State**
- **Strategy** (delegate behavior)
- **Template Method** (hook/template)
- **Visitor** (first class operation on composite)

Some others

- **Null Object**
- **Type object**

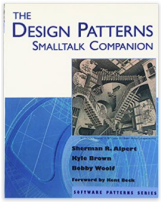
Not a panacea!

- Do not apply Design Patterns when you do not need them
- They make software more complex
  - more classes
  - more indirections, more messages
- Try to understand when NOT applying them!

Alert!

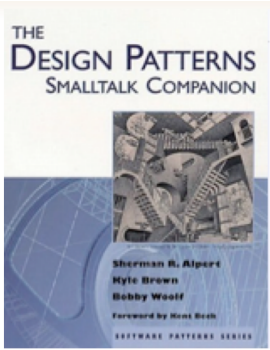
- Patterns are about **intent** and **tradeoffs**
- Do not confuse **intent** and **implementation**

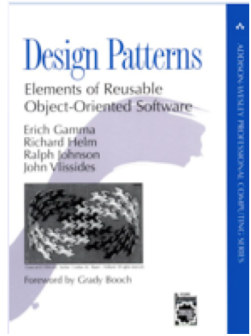
About books



- [Seminal one] **Design Patterns** (Gamma et al), also known as GoF book (Gang of Four)
- [Really excellent] **Smalltalk Design Pattern Companion** (Alpert et al)

What readers said





12 of 12 people found the following review helpful:

★★★★★ **Easier to understand than the original GoF**, February 4, 2000

Reviewer: **Nicolas Weidmann** (Zurich, Switzerland) - See all my reviews

This book gives you a better understanding of the patterns than in its original version (the GoF one). I am not a SmallTalk programmer but a 9 years C++ one. At work I had to use the GoF book and never liked reading it. In contrast to this, the SmallTalk companion is easy to read and you can understand the patterns within the first few lines of their description. Take the Bridge pattern and compare their discussions in the two books. If you really like the Gof one then buy it. But according to me, it would be a big mistake buying the GoF in favour of the SmallTalk companion. Trust a C++ programmer :-)

Conclusion

Reusable solutions to common problems based on experiences from real systems

- Names of abstractions creating a **common** vocabulary for developers
- Often support **modularity** (separation of interface/implementation)
- A basis for frameworks and toolkits basic elements to improve reuse

Produced as part of the course on http://www.fun-mooc.fr

Advanced Object-Oriented Design and Development with Pharo

A course by
S.Ducasse, L. Fabresse, G. Polito, and P. Tesone

Inria LearningLab

2023