Université de Paris

Génie Logiciel
Software design

Sylvain Lobry

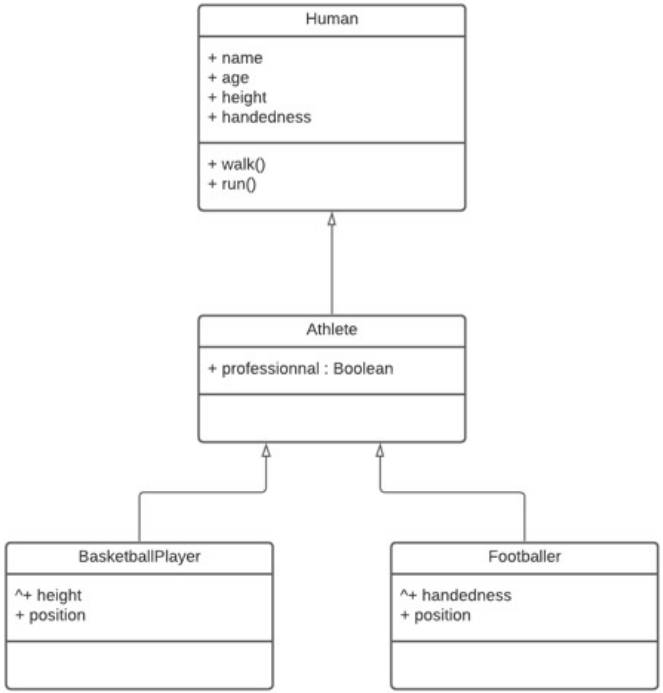
26/11/2021

Resources: www.sylvainlobry.com/GenieLogiciel

UML to model the structure

Specialization - Reminders

- Pay attention to the arrow!
- You can explicitely state inherited elements with “^”. In most cases you *do not have to*.
- Useful for complex hierarchies

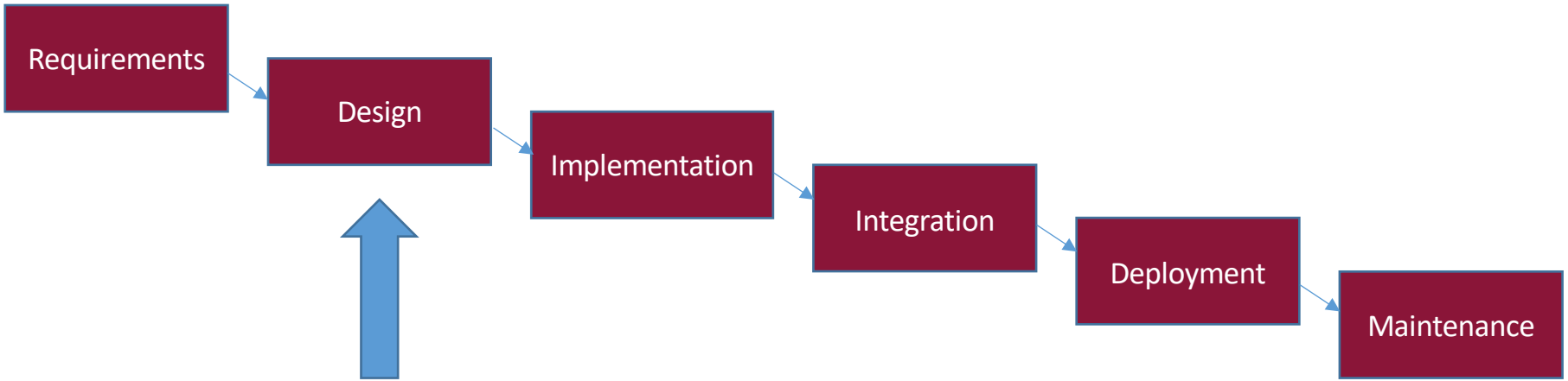


Software design

1. Introduction
2. Important concepts of software design
3. Reusable solutions: design patterns

Back to SDLC

Waterfall model



Software design

Definitions

- Definition: **Software design** is the process of defining the overall structure and interaction of your code so that the resulting product will satisfy the requirements

- Modeling is different from design!
- Modeling: a way to express the design

- It is an important step to agree:
  - on the usage of the software product
  - on the structure of the software product

Software design

1. Introduction
2. Important concepts of software design
3. Reusable solutions: design patterns

Software design

Software design concepts

- Abstraction
- Coupling and cohesion
- Decomposition and modularization
- Encapsulation and information hiding
- Separation of interface and implementation
- Sufficiency, completeness and primitiveness
- Separation of concerns

Software design

Abstraction

- **Abstraction**: a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information
- Without abstraction, your program becomes too complex, not usable
- You should always ask yourself the question “what complexity is right for my target?”

Software design

Encapsulation

- **Encapsulation**: hiding the details of an abstraction to an external entity
- Essential component of an abstraction, but different from abstraction

- Good practice: if some component is subject to change, encapsulate it!

Software design

Modularization

- Modularization: to divide the software in small components, each having a well defined interface
- Divide and conquer principle
- Achieved through encapsulation: what is not relevant to the external user is hidden.

- Advantages:
  - Easier to maintain
  - Easier to work in parallel
  - Handle well complexity

Software design

Separation of concerns

- Separation of concerns is a design principle that states that a program should be divided in sections, each adressing a different concern.
- Term coined by Dijkstra in 1974
- Each section is a module

- Allows for re-usability of the modules

Software design

Separation of interface and implementation

- Separation of interface and implementation: the interface is public, not the implementation details
- The interface describes what services a client of the class can use, and how to request them.
- The implementation describes how these services are provided

Software design

Coupling and cohesion

- **Coupling** is the measure of the degree of interdependence between modules of the software.
- In general, aim for low coupling.

- **Cohesion** is the measure of the degree to which the elements of the module of the software are functionally related.
- In general, aim for high cohesion.

Software design

Sufficiency, completeness and primitiveness

- A software component should be:
  - sufficient and complete: it captures **all the important characteristics** of an abstraction and **nothing more**.
  - Primitiveness: the design should be based on patterns that are easy to implement.

Software design

1. Introduction
2. Important concepts of software design
3. Reusable solutions: design patterns

Software design

Design patterns: why?

- Most software design challenges are **common** to many projects
- No need to reinvent the wheel
- Allow for **common** solutions

Software design

Design patterns: what?

- Definition: « general, reusable solution to a commonly occurring problem within a given context in software design »
- Concept proposed by Christopher Alexander in 1977
- Formalized (and popularized) by the « Gang of Four » in:

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). Design Patterns: Elements of Reusable Object-Oriented Software.

Software design

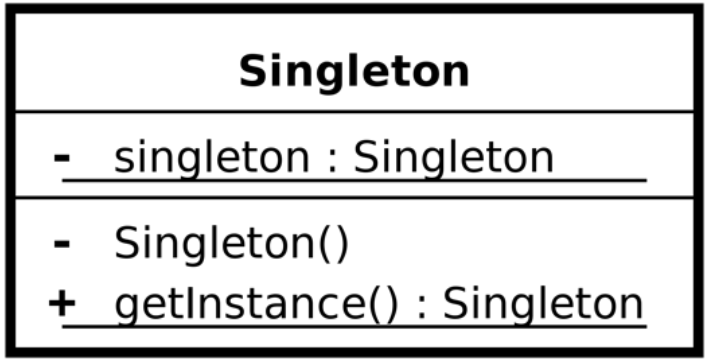
Design patterns: what?

- 23 design patterns in 3 categories:
  - Creational : patterns to create object
  - Structural: patterns that compose classes to obtain new functionalities
  - Behavorial: patterns that deal with the communication between objects.

Software design

Creational design patterns - Singleton

- Ensures that a class has only one instance
- Better alternative to global variables
- Exemple of use case: logger

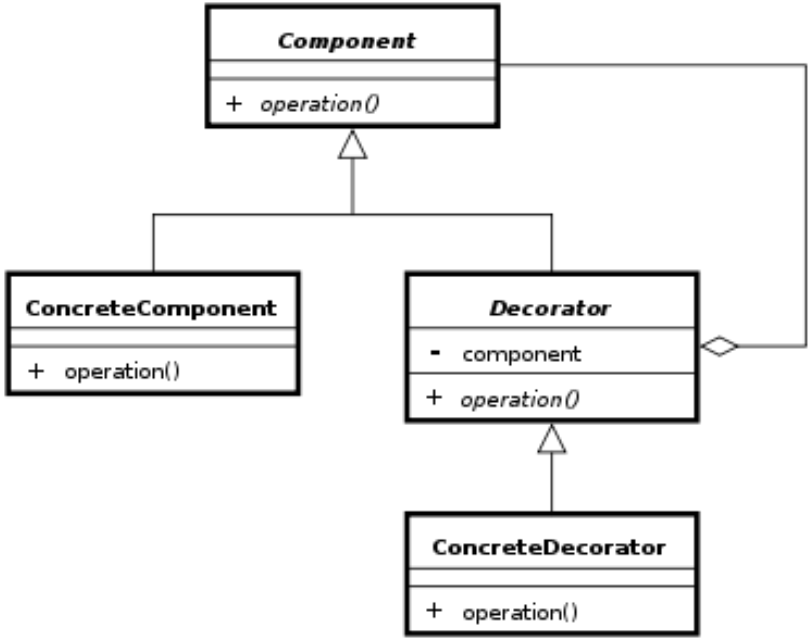


Software design

Structural design patterns - Decorator

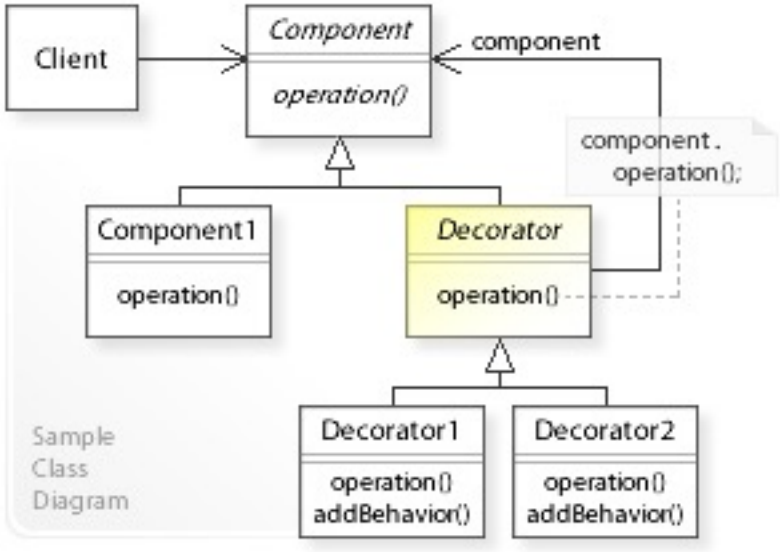
- Adds behavior to an individual object dynamically
- Can be seen as dynamic specialization

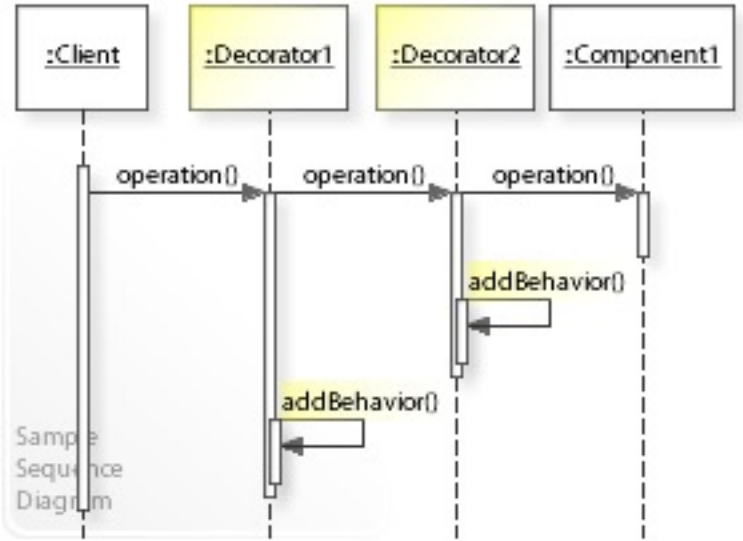
- Often used for GUI



Software design

Structural design patterns - Decorator



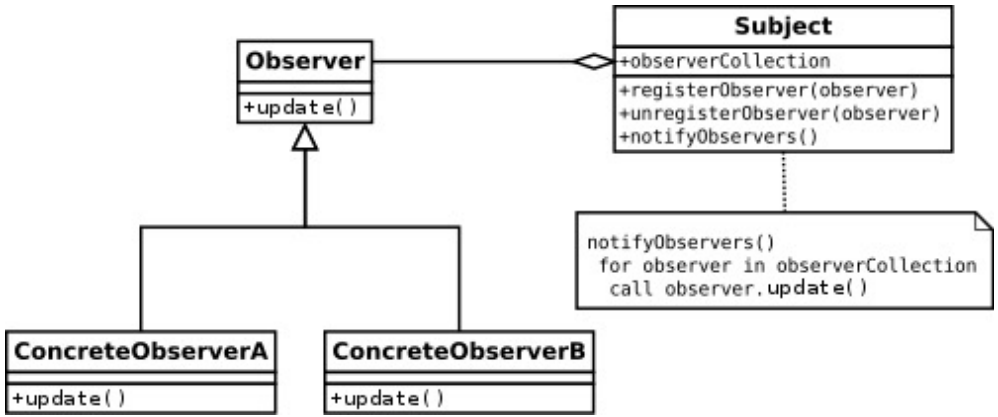


Source: http://w3sdesign.com/

Software design

Behavorial design patterns - Observer

- Allows dependencies without coupling
- Exemple of use case: subscribe for availability of a product



Software design

Criticisms on design patterns

- High-level programming language often gives solutions to these problems.
- Often, people try to apply them *too much*. Design patterns are not the solutions to all problems.
- Specific solution might better fit your project.
- Use them sparsly

Software design

Conclusion

- Software design is a critical step towards building quality software.
- Allows to make sure that requirements will be fulfilled.
- Allows for an easier implementation.

- You should:
  - Aim at fulfilling classical design concepts
  - Aim at a model that everyone agrees on
  - Go with the simplest solutions as possible