Chapter 7
Design Patterns

Jean Privat

IT069IU — Object-Oriented Analysis and Design
2015

Overview

1. Design Patterns
2. GoF Patterns
3. Composite
4. Singleton
5. Visitor

Overview

1. Design Patterns
2. GoF Patterns
3. Composite
4. Singleton
5. Visitor

Pattern

- “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such way that you can use this solution a million times over, without ever doing it the same way twice.”
- “Each pattern is a three part rule, which express a relation between a context, a problem, and a solution.”
- – Christopher Alexander [Alexander, 1977]

Patterns

Advantages of Patterns

- Common vocabulary
- Better reuse
- Capitalize experience
- Higher level of abstraction
- Cookbook of solutions

Patterns

Drawbacks

- Need higher abstraction: recognise problem and to apply a contextualized solutions
- Need to lean them and experiment them
- A lot of patterns
- A lot of patterns based on other patterns

Structure of patterns

Name

- Word or phrase
- Synthesize an idea of the solution

Problem

- The context
- The problem
- The goals

Structure of patterns

Solution

- Elements of solution
- Responsibilities
- Collaborations

Discussion about

- Related effects
- Trade of
- Alternatives

Overview

1 Design Patterns

2 **GoF Patterns**

3 Composite

4 Singleton

5 Visitor

GoF Patterns

Gang of Four

- Gamma, Helm, Johnson and Vlisside
- Design patterns : Elements of Reusable Object-Oriented Software
- Year 1994

Type of GoF Patterns

Creational

- Problems of object creation
- Where do object come from?
- Who is responsible to create (and remember) them?

Structural

- Problems of organization of objects
- How to connect (and access) objects in a flexible and reusable way?

Behavioral

- Problem of organization of operations
- How operations can be more reusable and flexible?

Creational

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Structural

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral

- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Describing Patterns

Identification

- Name
- Classification

Intent

- What is does?
- What is the problem solved?
- Justification

Motivation

- An example
- Illustrate the problem
- Illustrate the pattern

Describing Patterns

Applicability

- When using it?

Structure of the pattern

- Class model

Participants

- What are the elements of the solution?

Collaboration

- How responsibilities are shared?
- Interaction diagram

Describing Patterns

Impact

- Negative effects

Implementation

- Some implementation issues

Example

- Example of code

Known uses

- In real system

Related patterns

- Alternatives, differences and complementarity

How to use use pattern?

Adapt it to your case

- Chose participants and their roles
- Specialize and adapt classes
- Define attributes
- Define methods
- Implement methods

Overview

1. Design Patterns
2. GoF Patterns
3. **Composite**
4. Singleton
5. Visitor

Composite

Context/Problem

- How to treat the wame way (polymorphically):
- a group (or composition) of objects
- a non-composite (atomic object)?

Examples

- Files and directories
- Simple graphical elements and complex graphical elements
- Numbers and operations
- Actions and sequences of actions

Composite

Solution

- Define classes for composite and atomic objects so that they implement the same interface.

Participants

- Component (abstract): The common interface of operations
- Leaf: Atomic component
- Composite: Non-atomic component, aggregates components, forwards operation to them
- Client: Manipulate the objects trough Component

Composite

GRASP

- Based on Polymorphism and provides Protected Variations

Discussion

- Clients do not care if its related objects are atomic or composite.
- Can be used for recursive structures
- Easy to add new leafs or composites

Variation

- Generalize composite management?

Overview

1. Design Patterns
2. GoF Patterns
3. Composite
4. Singleton
5. Visitor

Singleton

Context/Problem

- How to ensure that a class has a single instance?
- That this instance is easily accessible?

Examples

- The root object of the domain layer
- Stateless objects that are pure behavioral

Singleton

Solution

- Define a static method of the class that returns the singleton.
- Hide the constructor

Participant

- Singleton: define a static operation getInstance() that return the uniq instance
- Client: access the singleton object with getInstance() on the globally visible class

Singleton

GRASP

- Based on Information Expert and Protected Variation

Lazy initialization

- Creation work is avoided, if the instance is never actually accessed.
- The lazy initialization sometimes contains complex and conditional creation logic.

Issues

- Concurrency on the object creation
- Hard to get back form static (global)

Overview

1. Design Patterns

2. GoF Patterns

3. Composite

4. Singleton

5. Visitor

Visitor

Context/Problem

- There is a given data-structure (henerogeneous and hierarchical)
- How to add various operations on the data-structure with:
- Keeping a good cohesion and
- Avoiding continuous modification of the classes of the data structure

Example

- Add operations on composite objects

Visitor

Solution

- Encapsulate the operation in a single class
- It knows how to operate each class of the structure

Participants

- Visitor (abstract): declare a specific visit method for each concrete element
- ConcreteVisitor: implement the visit methods
- Element (abstract): define an accept method
- ConcreteElement: implement the accept method by forwarding to the specific visit method
- Client: can apply any visitor to any element

Visitor

GRASP

- There is two applications of Polymorphism
- Pure fabrication helps to keep high cohesion in the data structure
- Indirection makes it possible to add new operations

Discussion

- The visitor objects can contain attributes to track the stat of the compilation

Issues

- The modification of the classes of the data structure will require the update all the visitors

Visitor

Visit

- Automatic
- Manual
- Both

Value passing (arguments and results)

- As objects (or dynamically typed)
- With generics
- Store/retrieve data in the visitor