Les design patterns en C++14

But : proposer une implémentation en C++ “moderne” (ou expliquer pourquoi un design pattern n'est plus pertinent en C++ “moderne”). ie mettre a jour l'article de Davide Come : http://come-david.developpez.com/tutoriels/dps/

Dans un premier temps, principalement les design patterns du GoF.

Sources :

- http://en.wikipedia.org/wiki/Software_design_pattern
- https://sourcemaking.com/design_patterns
- http://www.dofactory.com/net/design-patterns
- http://www.oodesign.com/
- http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns
- http://www.bogotobogo.com/DesignPatterns/introduction.php
- http://openclassrooms.com/courses/programmez-en-oriente-objet-en-php/les-design-patterns
- http://openclassrooms.com/courses/programmation-objet-avancee-la-conception-avant-tout-design-patterns-a-l-emploi
- https://www.ics.com/designpatterns/book/index.html
- ... il y a tellement de sources qui parlent de cela...

Creational patterns

Abstract factory

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Builder

Separate the construction of a complex object from its representation, allowing the same construction process to create various representations.

Factory method

Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses (dependency injection[15]).

Lazy initialization

Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as “virtual proxy”, an implementation strategy for the Proxy pattern.

Multiton

Ensure a class has only named instances, and provide global point of access to them.

Object pool

Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns.

Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Resource acquisition is initialization

Ensure that resources are properly released by tying them to the lifespan of suitable objects.

Singleton

Ensure a class has only one instance, and provide a global point of access to it.

Structural patterns

Adapter

Or Wrapper or Translator.

Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the translator.

Bridge

Decouple an abstraction from its implementation allowing the two to vary independently.

Composite

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Decorator

Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.

Facade

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Flyweight

Use sharing to support large numbers of similar objects efficiently.

Front Controller

The pattern relates to the design of Web applications. It provides a

centralized entry point for handling requests.

Module

Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.

Proxy

Provide a surrogate or placeholder for another object to control access to it.

Twin

Twin allows modeling of multiple inheritance in programming languages that do not support this feature.

Behavioral patterns

Blackboard

Generalized observer, which allows multiple readers and writers. Communicates information system-wide.

Chain of responsibility

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects

and pass the request along the chain until an object handles it.

Command

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Interpreter

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Iterator

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Mediator

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Memento

Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.

Null object

Avoid null references by providing a default object.

Observer

Or Publish/subscribe

Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.

Servant

Define common functionality for a group of classes

Specification

Recombinable business logic in a Boolean fashion

State

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template method

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Visitor

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Concurrency patterns

(plus tard)