Design patterns 2

IT Academy 2014

Rozdelenie návrhových vzorov

Creational patterns - riešia problémy súvisiace s vytváraním objektov v systéme. Podstatou je vybrať správnu triedu pre vytvorenie objektu a zaistenie správneho počtu týchto objektov. Väčšinou sa jedná o dynamické rozhodnutia počas behu programu. Patria sem napríklad:

- Factory Method Pattern
- Vylepšený - abstract factory method pattern
- Builder pattern
- Prototype pattern
- Singleton Pattern

Rozdelenie návrhových vzorov

Structural Patterns - predstavujú skupinu návrhových vzorov zameriavajúcich sa na možnosti usporiadania jednotlivých tried alebo komponent v systéme. Snahou je sprehľadniť systém a využiť možnosť štrukturalizácie kódu. Medzi tieto návrhové vzory patrí napr.:

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

Rozdelenie návrhových vzorov

Behavioral patterns - sa zaujímajú o chovanie systému. Môžu byť založené na triedach, alebo objektoch. Pri triedach využívajú pri návrhu riešení najmä princíp dedičnosti. V druhom postupe je riešená spolupráca medzi objektom a skupinou objektov, ktorá zaisťuje dosiahnutie požadovaného výsledku. Medzi tento typ vzorov môžeme zaradiť:

- **Chain of resposibility pattern**
- **Command Pattern**
- **Interpreter pattern** - http://objekty.vse.cz/Objekty/Vzory-Interpreter
- **Iterator pattern**
- **Mediator pattern**
- **Memento pattern**
- **Observer pattern**
- **State pattern**
- **Strategy pattern**
- **Template Pattern**
- **Visitor Pattern**

http://objekty.vse.cz/Objekty/Vzory-prehled

Factory pattern

factory.java

Factory pattern

Factory pattern - v tomto návrhovom vzore vytvárame objekty v inom objekte. Dôvody:

- Chceme oddeliť kód, ktorý vytvára objekty, od kódu objektu.
- Nechceme, alebo nemôžeme použiť konštruktor.
- Chceme môcť dynamicky vytvárať nové inštancie, bez potreby vedieť konkrétne meno triedy.

Abstract factory pattern

Ako bude fungovať abstract factory?

Prototype pattern

prototype.pseudo

Prototype pattern

Prototype pattern - design pattern ktorý nám umožňuje vytvárať nové objekty klonovaním. Umožňuje nám tak vyhnúť sa vytváraniu inštancie pomocou new, ak je takéto vytvorenie inštancie drahé vzhľadom na svoje použitie, alebo ak nemôžeme vytvárať odvodené triedy.

Builder pattern

builder.pseudo

Builder pattern

Builder pattern - design pattern ktorý nám umožňuje vytvárať nové objekty tak, že v mieste, kde potrebujem nový objekt, si ho môžem manuálne inicializovať v mieste, kde ho vytváram. Navyše, ku každému kroku viem naviazať funkcionalitu.

Rozdiel oproti factory je ten, že inicializácia objektu sa vo factory môže meniť tak, že dedíme factory, ale v builderovi si to môžeme my sami "nakonfigurovat'" v mieste použitia objektu (vytvorenia jeho inštancie).

Decorator pattern

decorator.pseudo

Decorator pattern

Decorator pattern - pattern, pomocou ktorého môžeme k objektu pridávať novú funkčnosť. Spoločne s Adapter pattern sa spoločne označujú aj ako Wrapper pattern. Novú funkčnosť môžeme pridávať staticky alebo dynamicky.

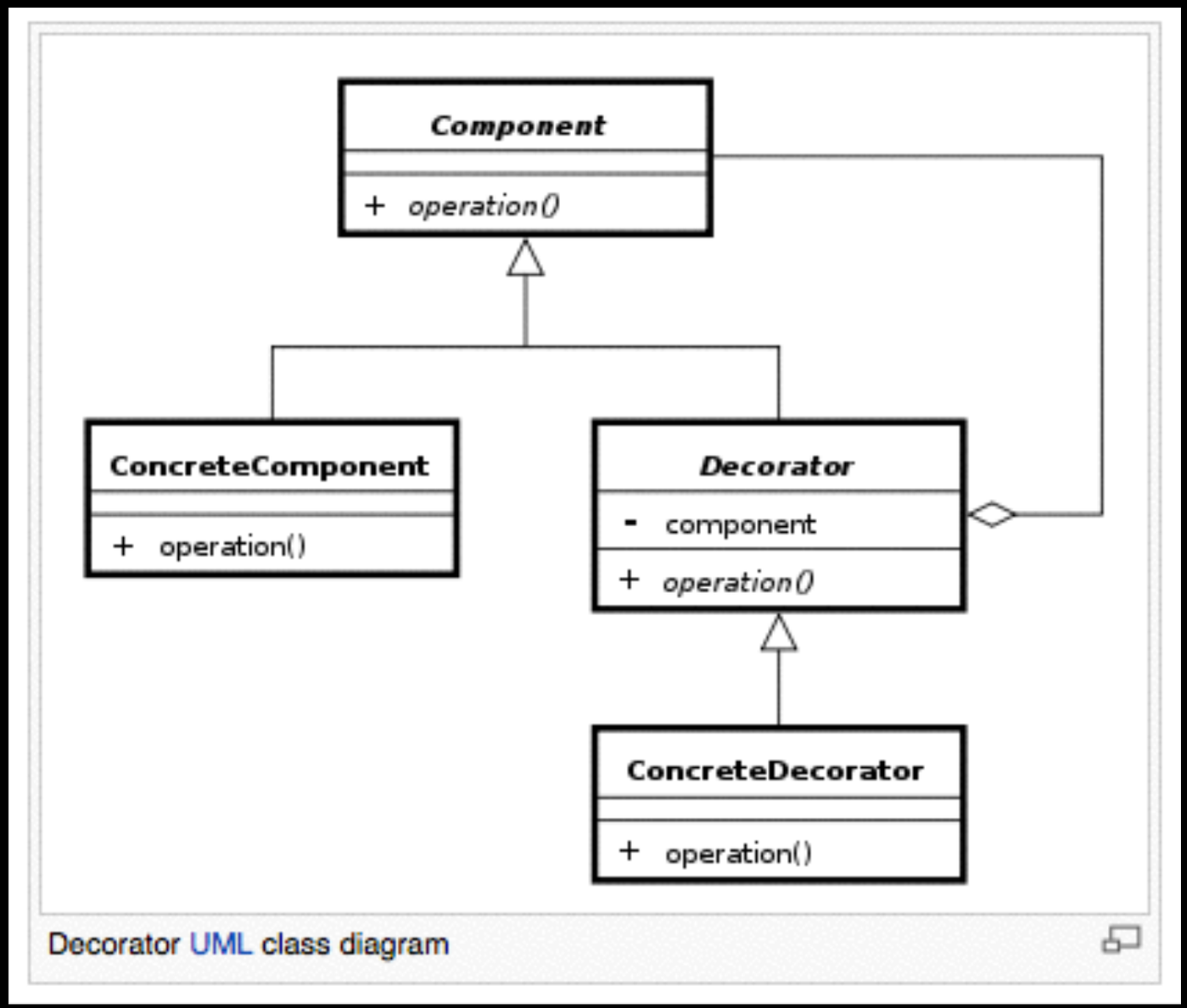
Decorator pattern

Ako vytvorit’ decorator?

1. Subclass the original "Component" class into a "Decorator" class (see UML diagram);
2. In the Decorator class, add a Component pointer as a field;
3. Pass a Component to the Decorator constructor to initialize the Component pointer;
4. In the Decorator class, redirect all "Component" methods to the "Component" pointer; and
5. In the ConcreteDecorator class, override any Component method(s) whose behavior needs to be modified.

Decorator pattern

Ako to vyzerá na obrázku? :D



Decorator UML class diagram

Chain of responsibility design pattern

chain-of-responsibility.java

Chain of responsibility design pattern

Pri písaní aplikácii sa často stane, že potrebujeme, aby objekt predal správu nejakému inému objektu, ale nevie, ktorý konkrétny objekt dokáže správu spracovať.

Riešením je chain of responsibility pattern. Tento pattern nám umožňuje vytvoriť reťaz objektov, ktoré volám cez spoločný vstupný bod. Vstupu predám správu, a on potom odovzdá správu tomu objektu, ktorý ju dokáže prijať. To mi dovolí každého z prijímateľov správy vymeniť, ak to budem potrebovať.

Observer pattern

observer.cs

Observer pattern

Podstata implementácie návrhového vzoru **Observer** je v tom, že vytváram observerov (pozorovateľov) a observable objekty (pozorovateľné objekty), ktoré si samé udržujú referenciu na všetkých observerov, a tým potom odosielajú správy. Ak chcem, môžem pridávať ku inštanciám nových observerov. Aj dynamicky.

Mediator pattern

mediator.cpp

Mediator pattern

v skratke:

Vytvorím objekt, ktorý predávam objektom, a ony pomocou neho spolu komunikujú. V princípe vytvorím centrálny komunikačný uzol medzi objektami. Takto môžem sprehľadniť komunikáciu medzi objektami.

diagram: http://zamjad.wordpress.com/2012/04/25/mediator-design-pattern-using-c/

Bridge pattern

bridge.java

Bridge pattern

Niekedy potrebujeme, aby nami definovaná abstrakcia využívala viaceré implementácie.

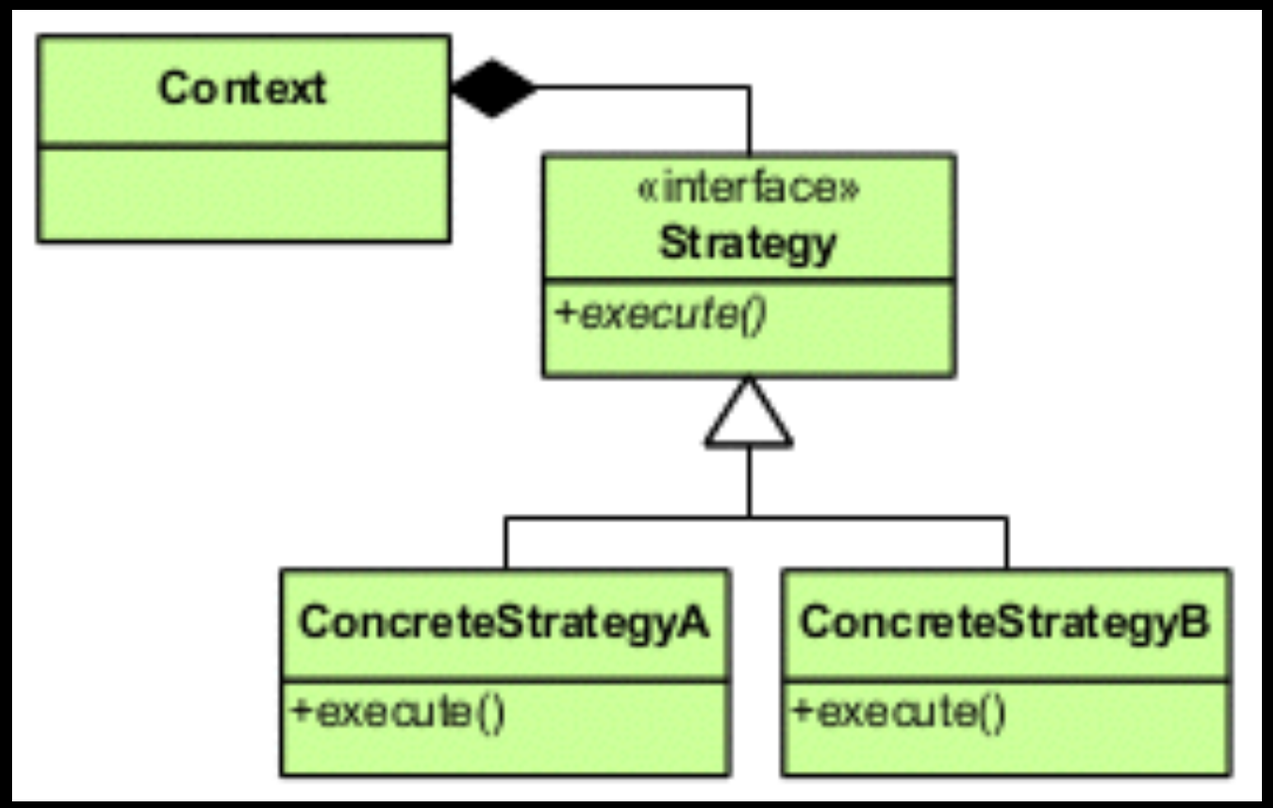
V java príklade vidíme, ako potrebujeme používať objekt Shape s viacerými API. Výhoda tejto implementácie oproti podedeniu Shape a rozšíreniu o 2 metódy, ktoré by ho vykreslovali cez obe API, je tá, že v tomto prípade môžem túto funkcionalitu zapúzdriť mimo objektu Shape. Pri použití Shape si v konečnom dôsledku môžem stále vybrať, aké API použijem, ale nemusím mať všetky API implementované v triede Shape.

Strategy pattern

strategy.java

Strategy pattern

Umožňuje nám prepínať algoritmy, ktoré použijeme v danom kontexte, v danom momente, s danými parametrami.



Visitor pattern

visitor.java

Visitor pattern

http://stackoverflow.com/a/255300

Môžeme pridávať novú funkcionalitu triedam bez toho, aby sme ich museli modifikovať.

http://www.artima.com/cppsource/top_cpp_aha_moments.html

Memento pattern

memento.java

Memento pattern

Umožňuje nám ukladať vnútorný stav objektu a neskôr sa k nemu vrátiť. Takýmto spôsobom môžeme implementovať napríklad UNDO.

(pri dodržaní korektného OOP návrhu)

Interpreter pattern

C++:

http://sourcemaking.com/design_patterns/interpreter/cpp/1

http://en.wikipedia.org/wiki/Interpreter_pattern