



Programovanie v jazyku C#

Návrhové vzory

prednáška 10
Ing. Ján Magyar, PhD.
ak. rok. 2022/2023 ZS

Návrhový vzor

- všeobecné znovupoužiteľné riešenie pre opakujúce sa problémy v softvérovom inžinierstve
- popis riešenia, nie samotné riešenie
- formalizovaný best practice
- nie sú nevyhnutné, ale zjednodušujú implementáciu

Štruktúra návrhových vzorov

- definuje komponenty a ich rolu
- definuje vzťah medzi komponentmi
- nešpecifikuje funkcionálnosť (závisí od prípadu použitia)
- nešpecifikuje implementáciu (je to na programátorovi)

Použitie návrhových vzorov

- pre časté problémy
- programovací jazyk môže ponúkať implicitné riešenie problému
- zvyčajne pre objektovo-orientované jazyky a programy
- osvedčený spôsob písania kvalitného softvéru

Výhody použitia návrhových vzorov

- rýchlejší vývoj
- bezpečnejšie a overené riešenia
- rieši aj skryté problémy
- lepšia čitateľnosť a štruktúra riešenia
- rozdelenie do komponentov pre znovupoužitie kódu
- zdokumentované riešenie

Dokumentácia návrhových vzorov

- meno a klasifikácia
- cieľ - prečo by sa mal použiť?
- alternatívne mená
- motivácia - ukázkový problém
- použiteľnosť - popis problémov pre ktoré sa vzor dá použiť
- štruktúra - diagram triedy
- členovia - zoznam komponentov

Dokumentácia návrhových vzorov (pokračovanie)

- spolupráca - ako interagujú triedy a objekty
- dôsledky - výsledky, následky, kompromis
- implementácia - ukázkový prípad použitia
- ukázkový kód - ako sa implementuje v niektorom jazyku
- známe prípady použitia - reálne využitia
- podobné vzory - porovnanie s ďalšími vzormi

Kritika návrhových vzorov

- môžu naznačiť chýbajúcu podporu v programovacom jazyku
- môžu byť implementované inými prístupmi
- zvyšuje zložitosť riešenia pri nevhodnom použití

Typy návrhových vzorov

- kreačné - ako vytvoriť objekt?
- štrukturálne - ako realizovať vzťah medzi objektmi?
- behaviorálne - ako môžu komponenty komunikovať?
- konkurenčnosť - pre viacvláknové programy

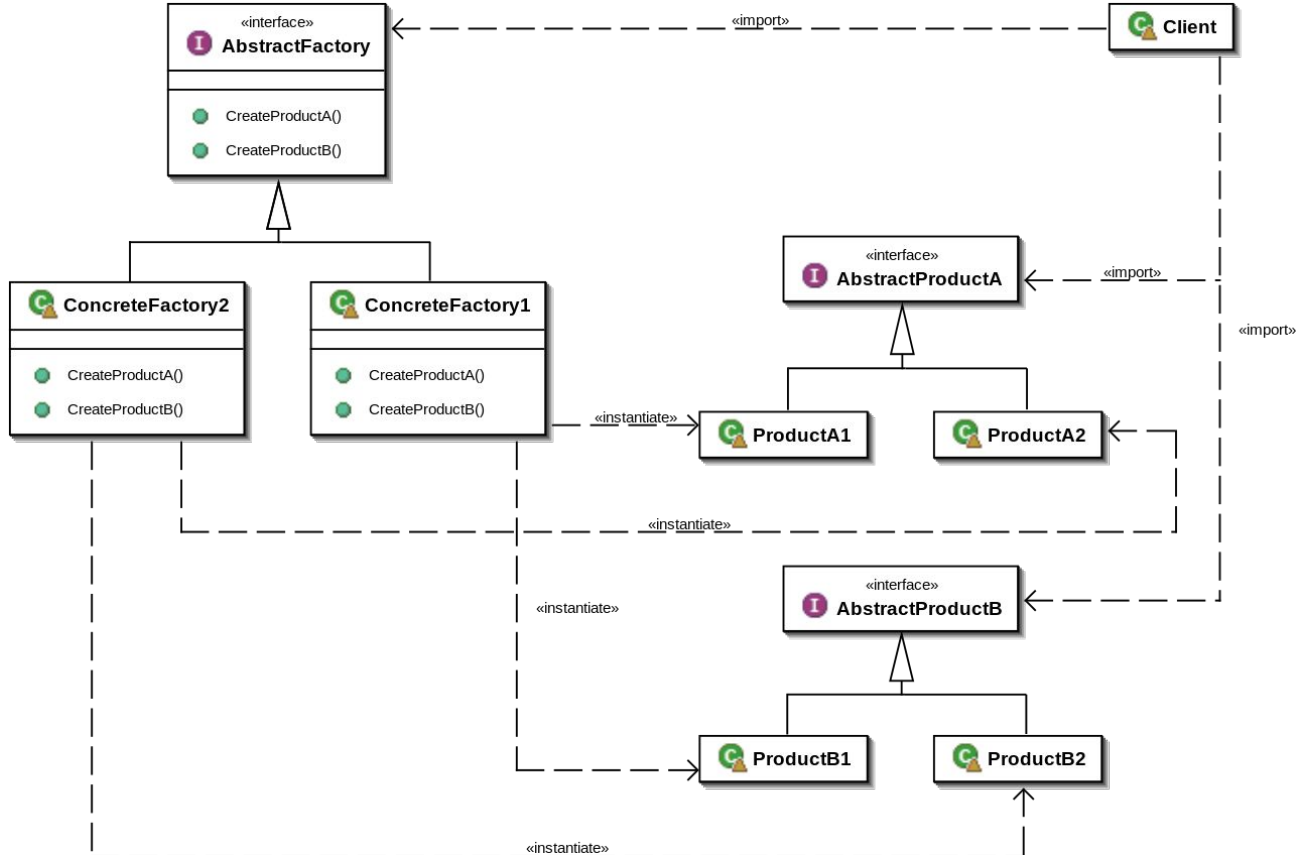
Kreačné návrhové vzory

- abstract factory
- builder
- factory method
- prototype
- singleton

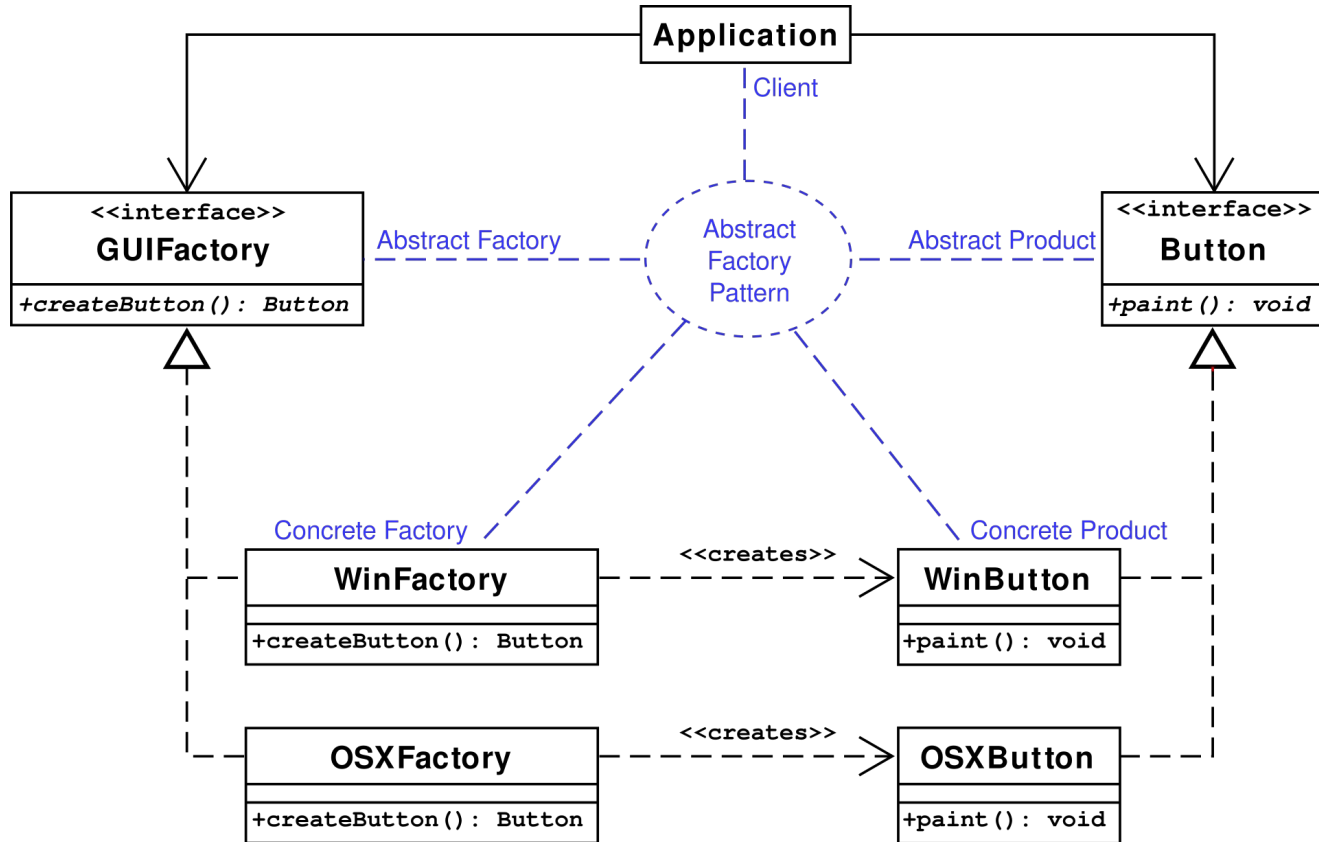
Abstract factory

- poskytnúť rozhranie pre vytvorenie skupiny závislých objektov bez špecifikácie konkrétnej triedy
- Ako môže byť aplikácia nezávislá od toho, ako sa vytvoria jej objekty a objekty ktoré potrebuje?
- vytvorenie objektov je skryté v osobitnom objekte
- úloha vytvorenia objektov je delegovaná factory objektu

Abstract factory



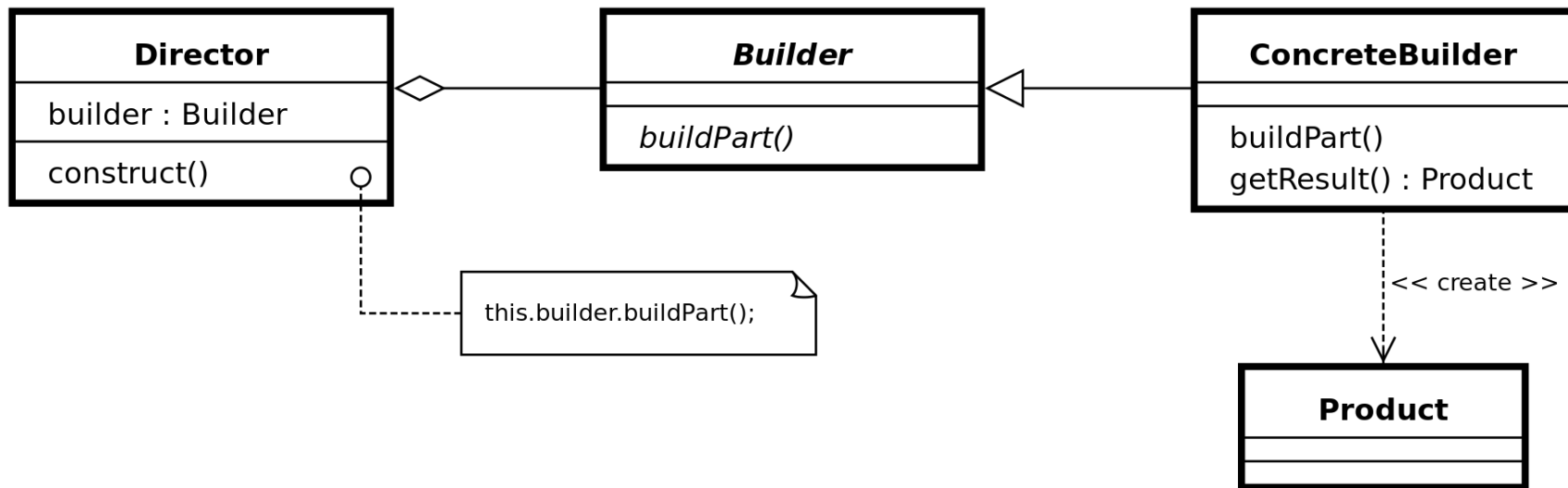
Abstract factory



Builder

- pre zložité objekty, oddelíme ich reprezentáciu od ich vytvorenia
- rovnaký proces môže vytvoriť rôzne reprezentácie
- Ako môžeme zjednodušiť triedu, ktorá obsahuje vytvorenie zložitého objektu?

Builder



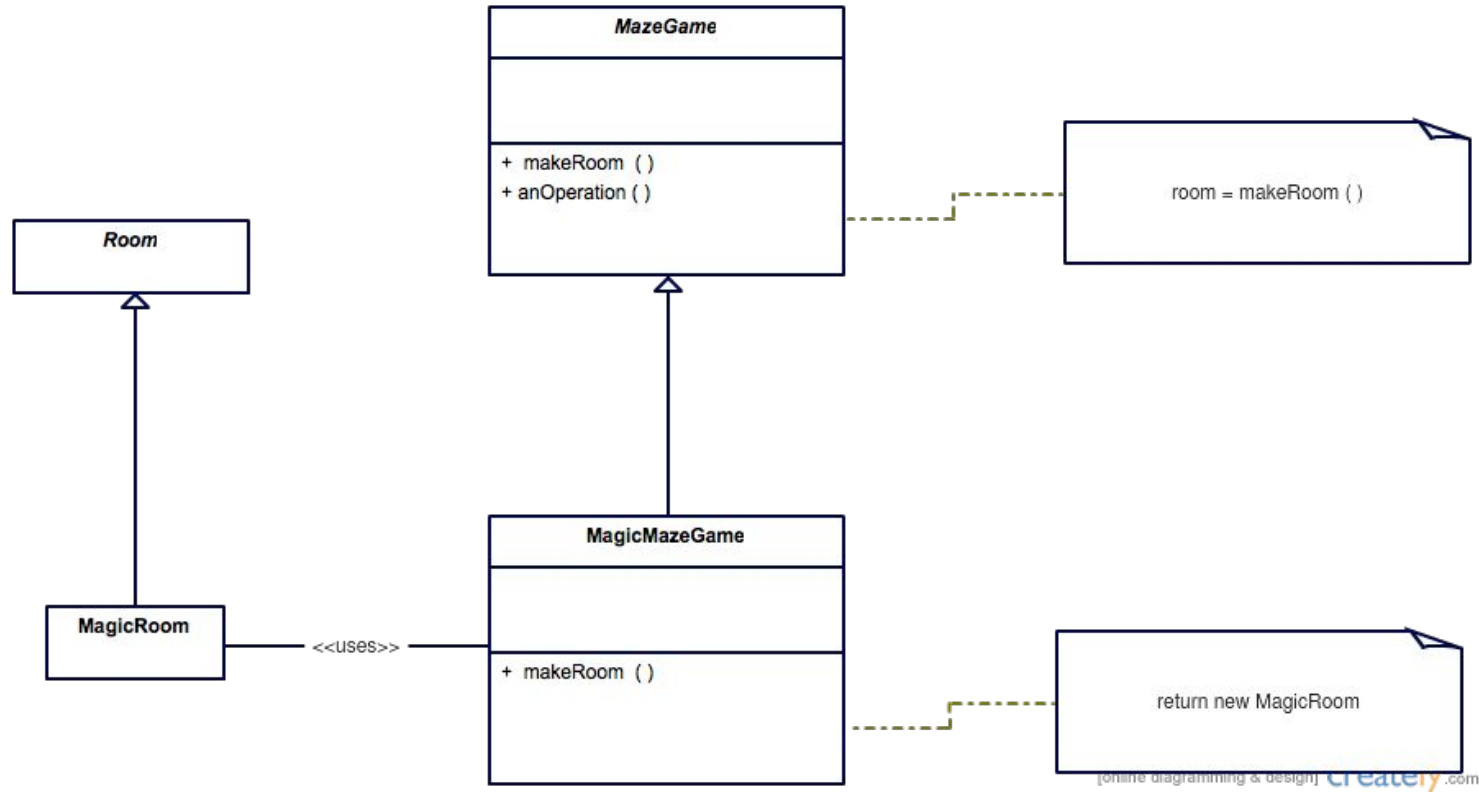
Použitie vzoru Builder

- môžeme meniť vnútornú reprezentáciu objektu
 - enkapsulovaný kód pre vytvorenie a reprezentáciu
 - môžeme kontrolovať proces vytvorenia
-
- musíme zdefinovať builder pre každý typ produktu
 - triedy buildera môžeme meniť
 - ťažší dependency injection

Factory method

- rozhranie pre vytvorenie jediného objektu, ale podtriedy rozhodujú, inštanciu ktorej triedy majú vytvoriť
- vytvorenie inštancií je úlohou podtried
- osobitná operácia (factory method) je zodpovedná za vytvorenie objektu, objekt vytvoríme zavolaním tejto metódy

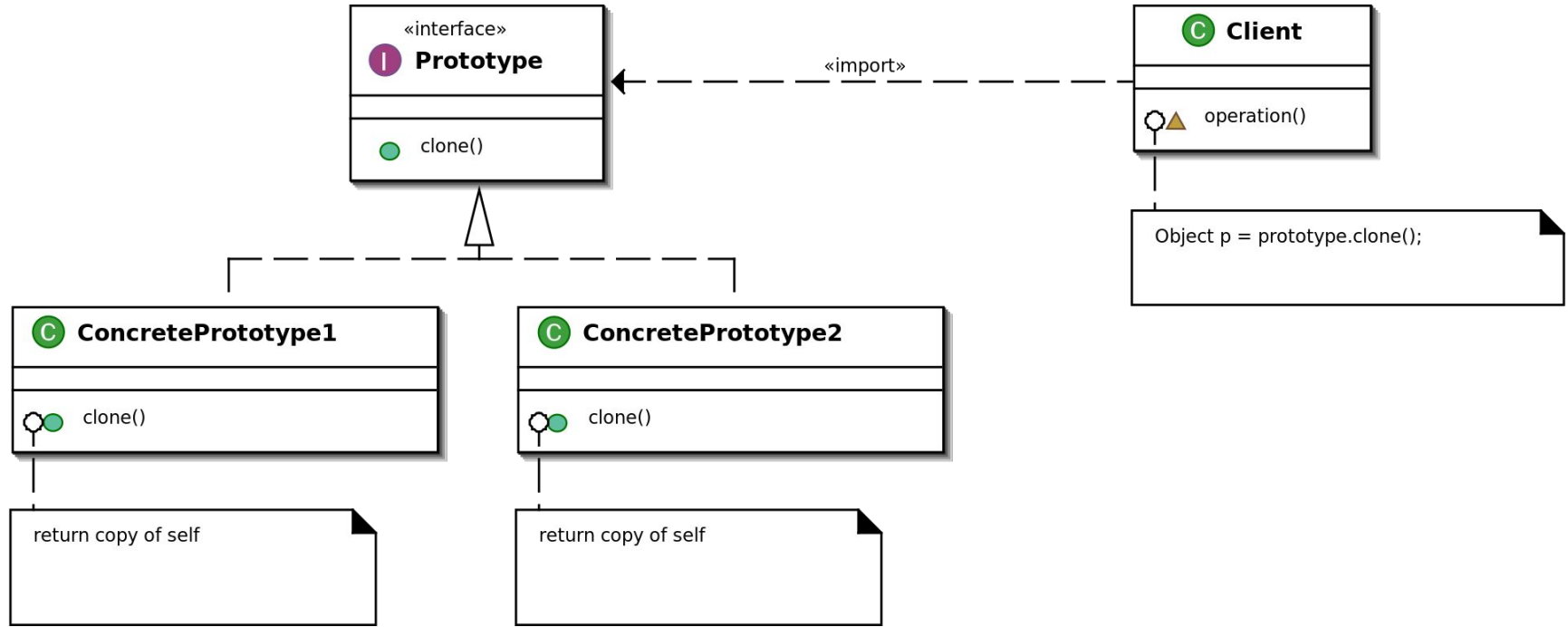
Factory method



Prototype

- objekty vytvárame na základe prototypovej inštancie
- vytváranie inštancií použitím už existujúceho objektu
- lepší výkon, menšia záťaž na pamäť
- môžeme špecifikovať počas behu, ktorý objekt sa má vytvoriť - dynamicky načítané triedy
- definujeme abstraktnú triedu s metódou `clone()`, ktorú implementujú konkrétne podtriedy

Prototype



Singleton

- zabezpečuje, že trieda má iba jednu inštanciu (alebo žiadnu)
- poskytuje prístup k jedinej inštancii
- môžeme kontrolovať vytvorenie inštalácie
- možné lazy initialization
- globálny stav

Singleton

Singleton

- singleton : Singleton
- Singleton()
- + getInstance() : Singleton

Ďalšie kreačné návrhové vzory

- dependency injection
- lazy initialization
- multiton
- object pool
- resource acquisition is initialization

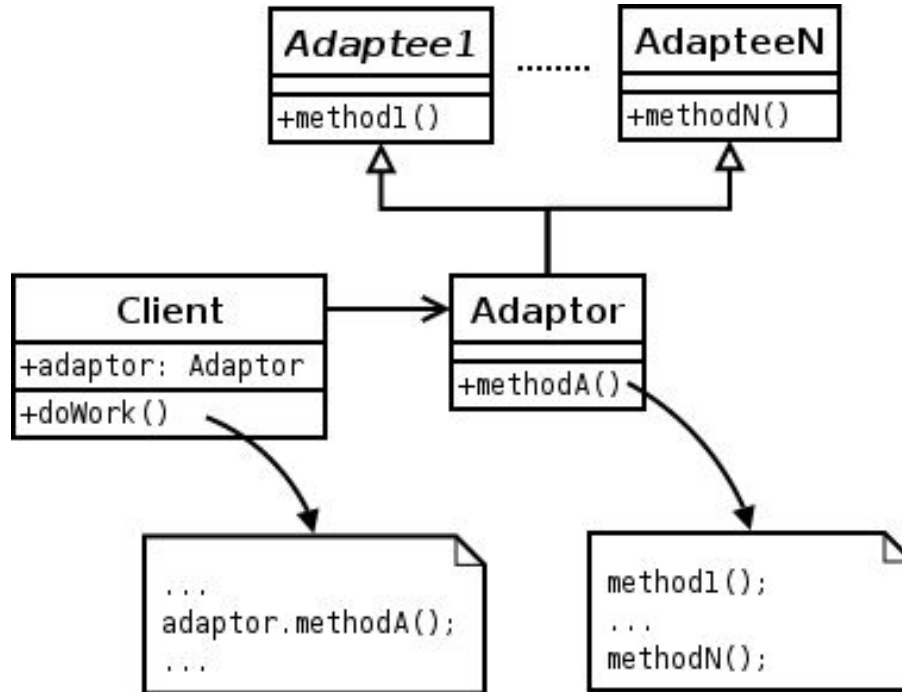
Štrukturálne vzory

- adapter
- bridge
- composite
- decorator
- facade
- proxy

Adapter

- konvertovanie rozhrania triedy na iné rozhranie, ktoré očakáva ďalšia trieda
- umožňuje spoluprácu dvoch tried s nekompatibilnými rozhraniami
- adapter konvertuje rozhranie adaptee na target
- možný na úrovni triedy alebo inštancie

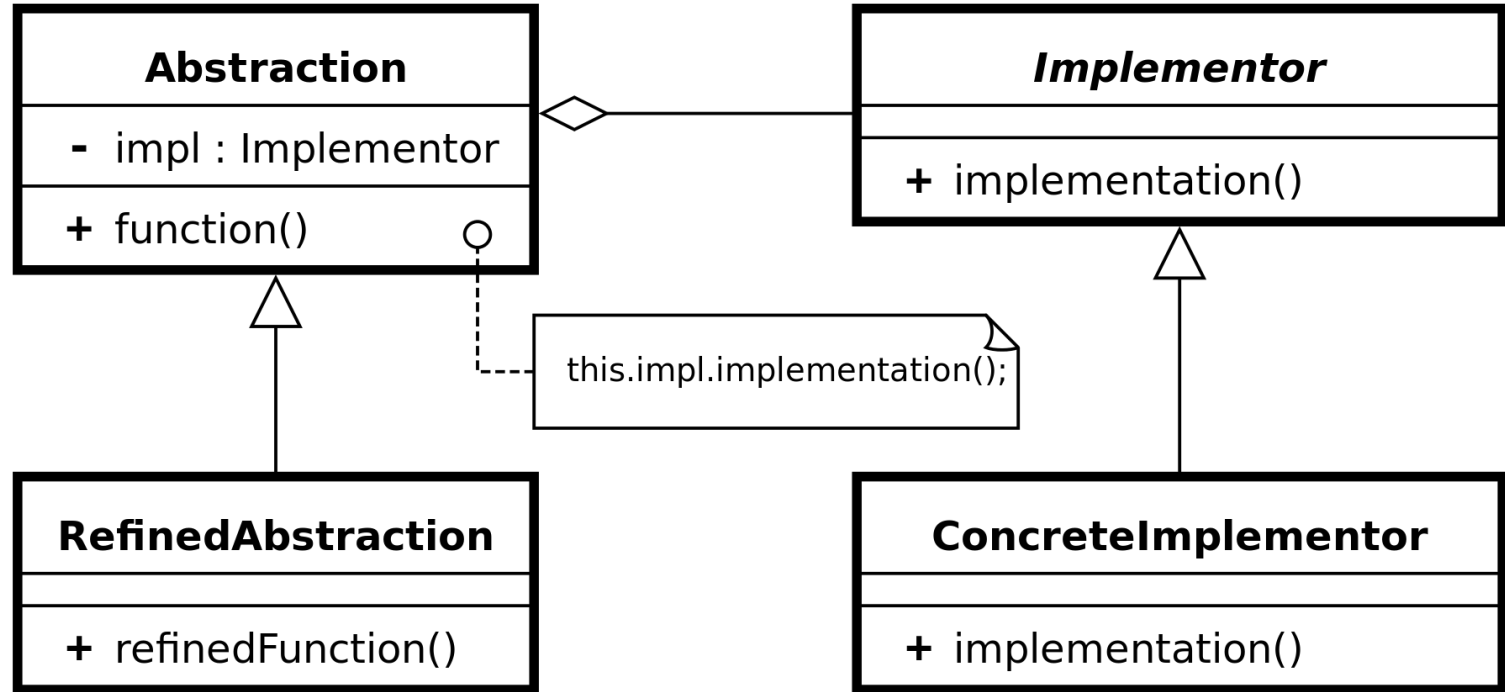
Adapter



Bridge

- oddelí abstrakciu od jej implementácie
- môžeme zmeniť abstrakciu alebo implementáciu bez potreby zmeniť druhú
- užitočné ak trieda a jej úloha sa často mení
- pridá druhú vrstvu abstrakcie
- často implementované ako adaptor na úrovni objektu

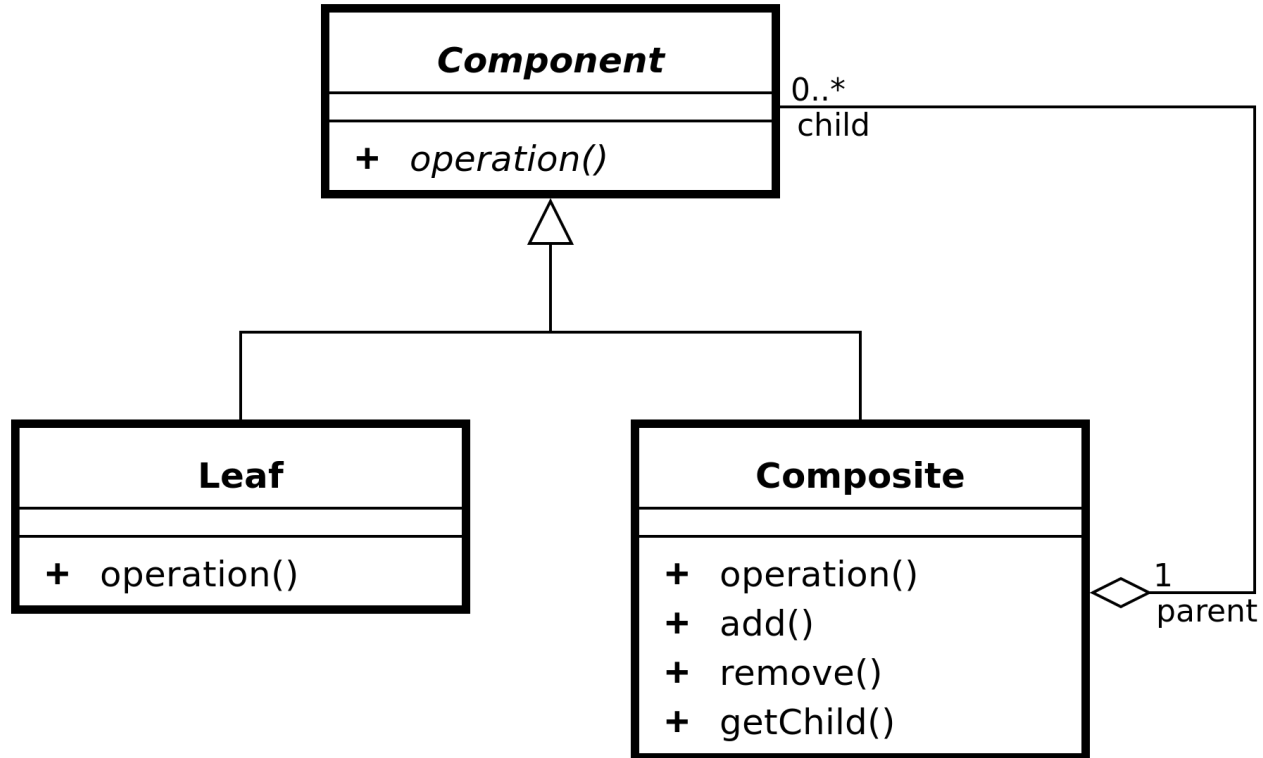
Bridge



Composite

- rozdelí objekty do stromovej štruktúry
- individuálne objekty a ich kompozity sa používajú jednotne
- definujeme zjednotené rozhranie `Component`, ktoré je implementované časťami objektu aj celkovým objektom
- kompozit prepošle požiadavky svojim potomkom

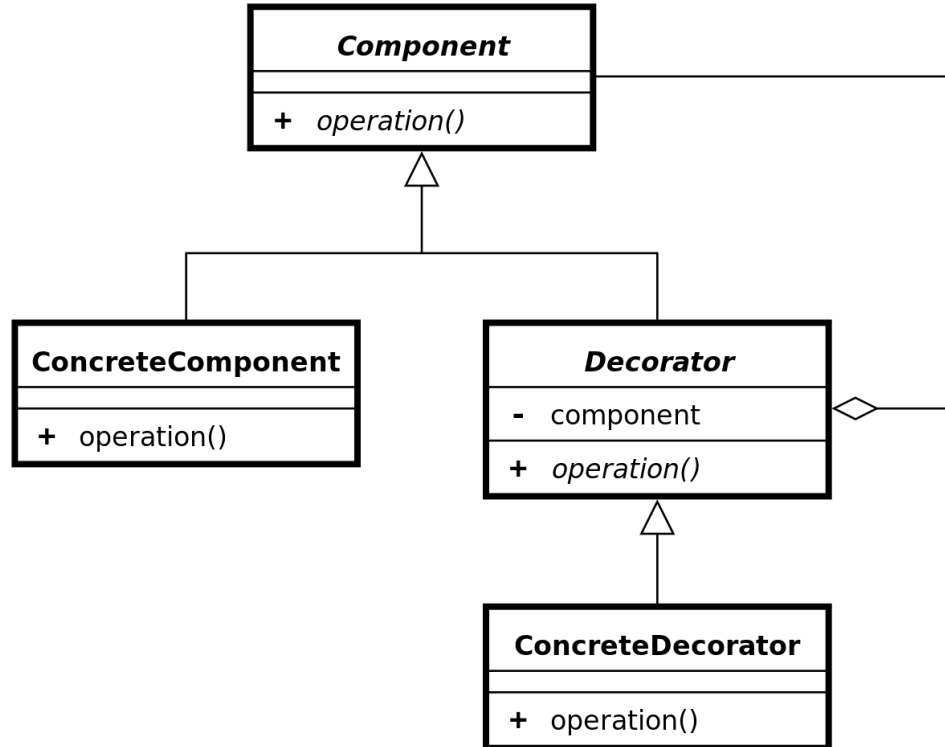
Composite



Decorator

- pridáme nové zodpovednosti objektom dynamicky bez zmeny ich rozhrania
- alternatíva k dedičnosti s cieľom rozšíriť funkcionality
- rozhranie rozšíreného objektu (`Component`) implementujeme preposlaním všetkých požiadaviek a následným vykonaním ľubovoľnej dodatočnej funkcionality

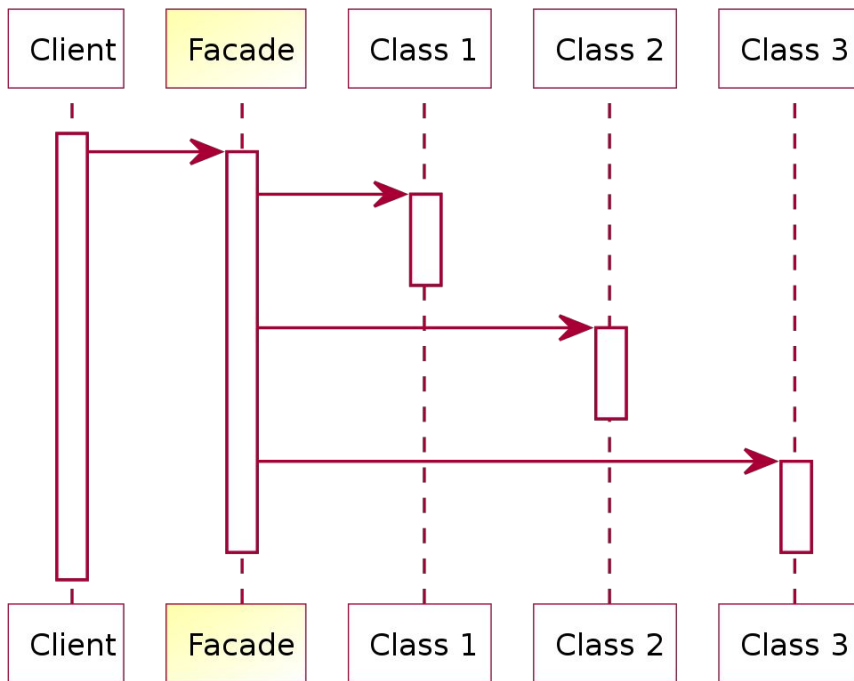
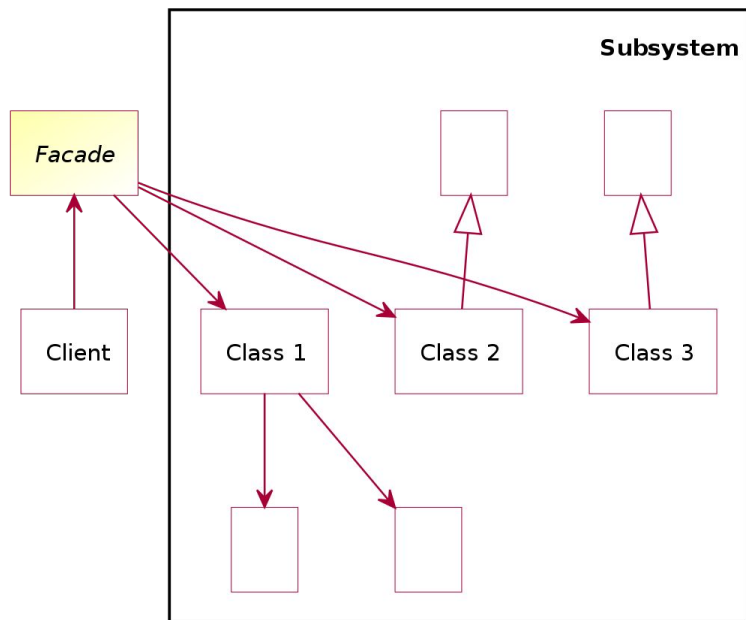
Decorator



Facade

- poskytuje zjednotené rozhranie pre sadu rozhraní definíciou rozhrania vyššej úrovne
- subsystemy sa použijú ľahšie
- lepšia čitateľnosť
- loose coupling

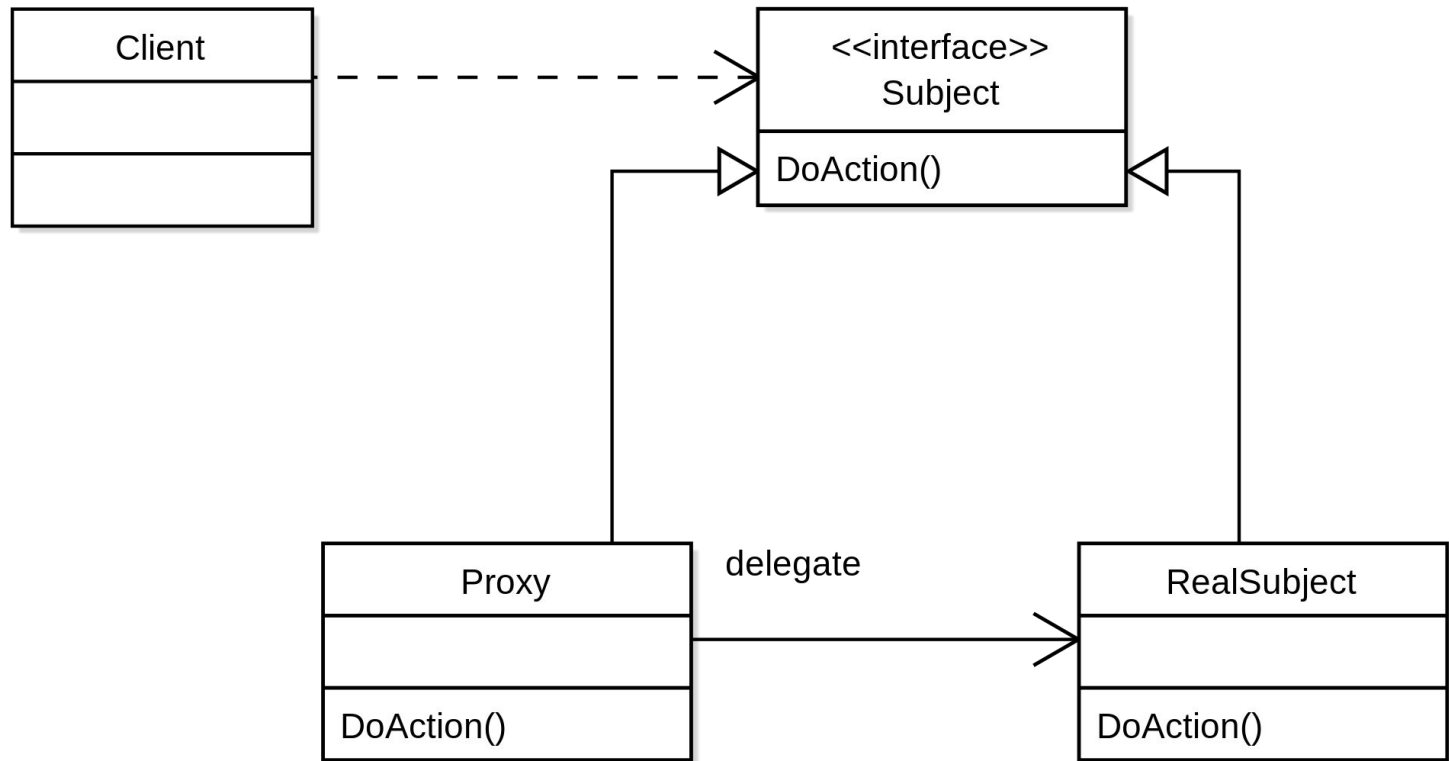
Facade



Proxy

- kontroluje prístup k objektu poskytnutím placeholdera, cez ktorý prechádza komunikácia
- proxy objekt môžeme použiť ako náhradu za iný objekt
- proxy môže definovať ďalšiu funkcionálnosť pre kontrolu prístupu k objektu
- proxy môže byť
 - remote
 - virtual
 - protection

Proxy



Ďalšie štrukturálne návrhové vzory

- extension object
- flyweight
- front controller
- marker
- module
- twin

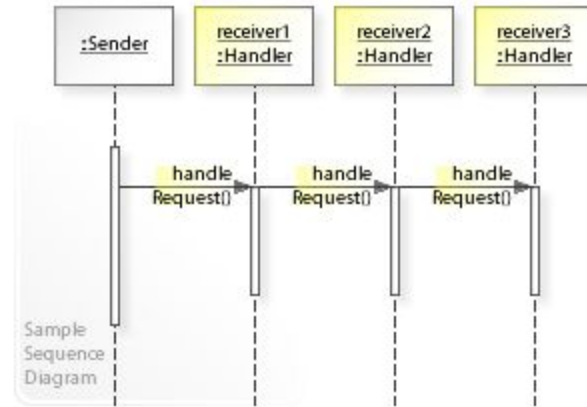
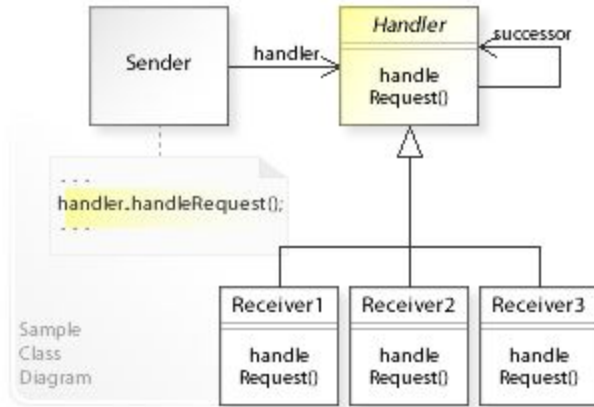
Behaviorálne vzory

- chain of responsibility
- command
- iterator
- mediator
- observer and publish/subscribe
- strategy
- visitor

Chain of responsibility

- viac objektov môže spracovať požiadavku
- vytvoríme zreteženie procesorov a správa sa posielá ďalej kým nie je spracovaná
- máme zdroj command objektov a sériu procesorov
- každý procesor požiadavku buď spracuje alebo ju prepošle na základe podmienok počas behu

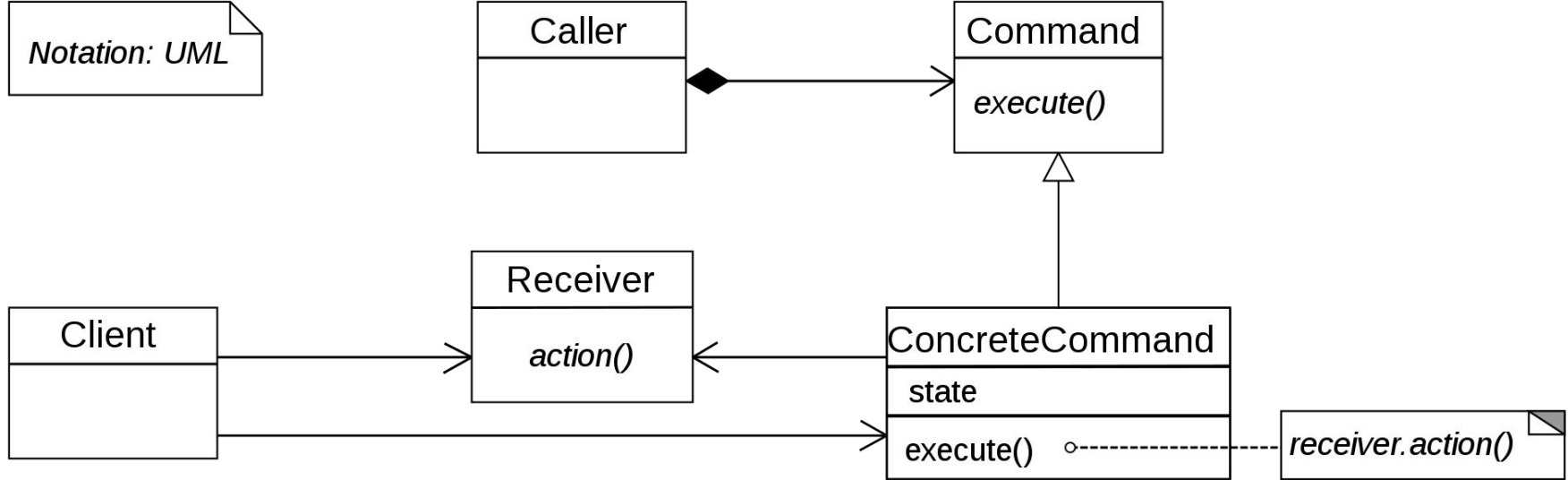
Chain of responsibility



Command

- enkapsulácia požiadavky do objektu
- umožňuje parametrizáciu klientov s rôznymi požiadavkami
- pre queueing, logging, a nezvratiteľné operácie
- požiadavka je delegovaná do command objektu namiesto priameho spracovania
- napr. GUI buttons, progress bars, transactions, wizards

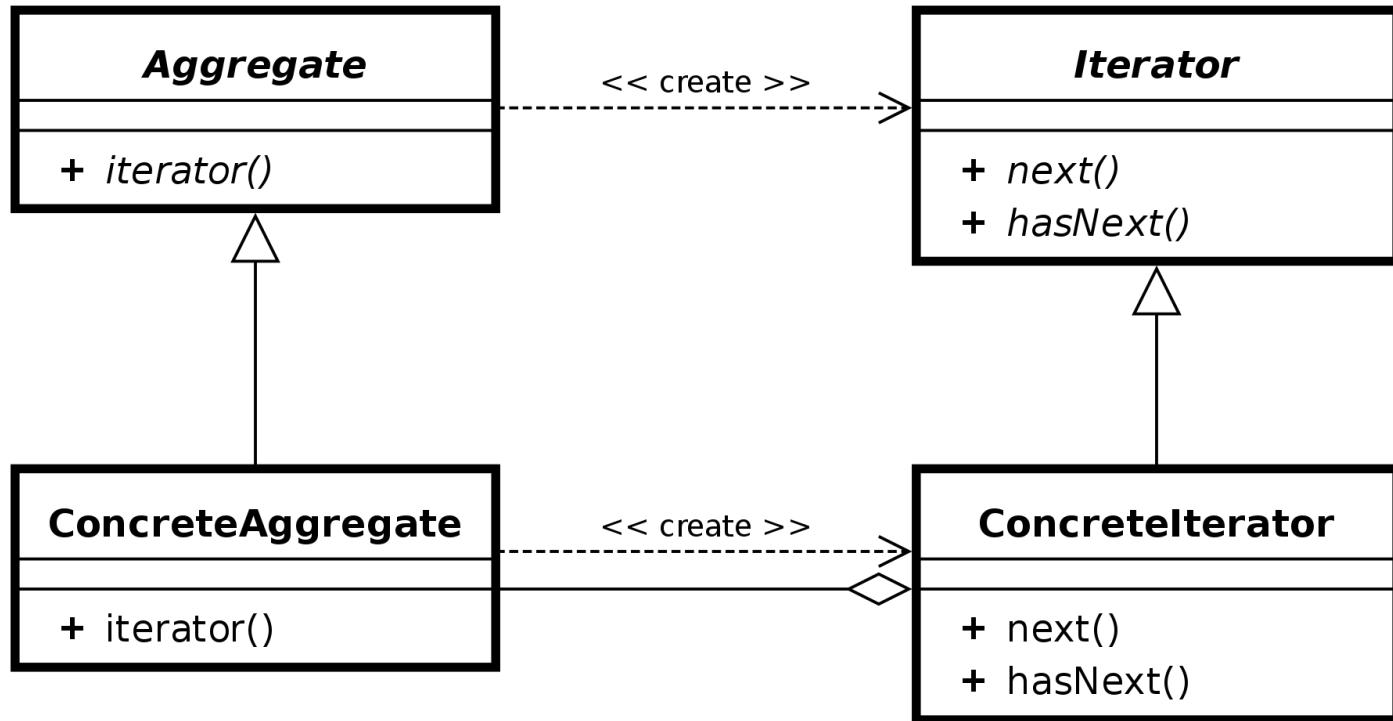
Command



Iterator

- poskytuje spôsob prístupu k prvkom agregátu sekvenčne bez odhalenia vnútornej reprezentácie
- oddelí algoritmus od kontajnera
- aj keď poskytuje jednotný prístup, nie vždy je optimálny
- trieda iterátor zapuzdrí prístup k a prechádzanie agregátom, klient používa iterátor

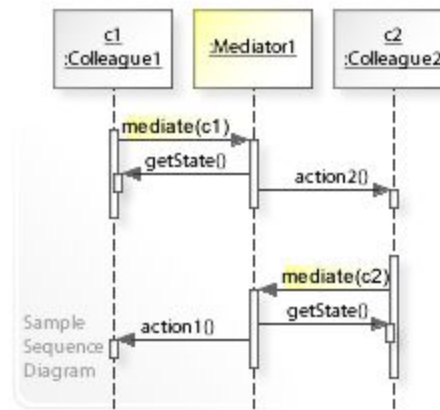
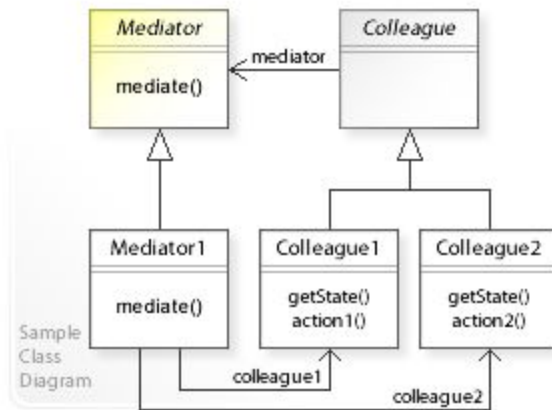
Iterator



Mediator

- enkapsulácia interakcie medzi sadou objektov
- umožňuje loose coupling medzi objektmi, keďže neodkazujú jeden na druhý explicitne
- môžeme meniť interakciu objektov
- zdefinujeme triedu mediátor, ktorá sa potom používa pre komunikáciu medzi objektmi

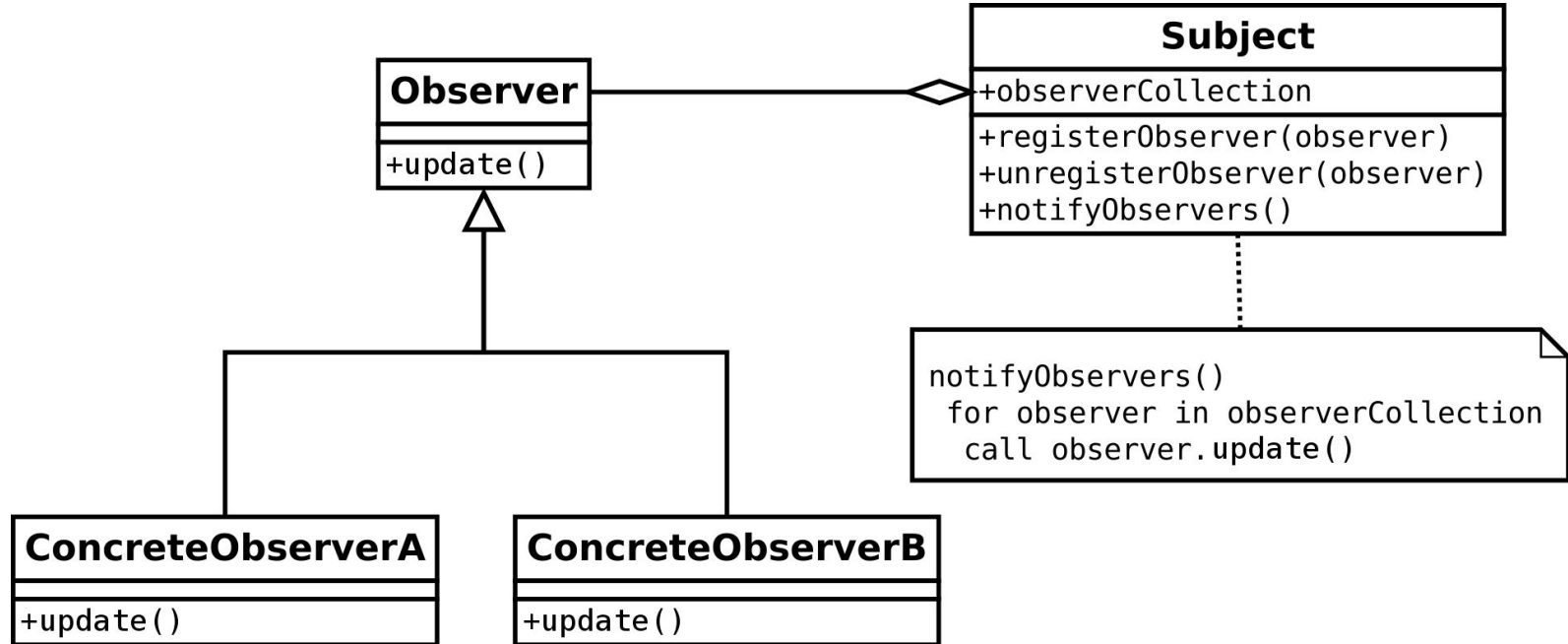
Mediator



Observer

- definícia závislosti one-to-many
- zmena jedného objektu vyžaduje notifikáciu ďalších závislých objektov
- implementuje vzor publisher/subscriber
- **subject** udržiava zoznam svojich **observerov** a notifikuje ich automaticky, tie následne zmenu spracujú vhodným spôsobom

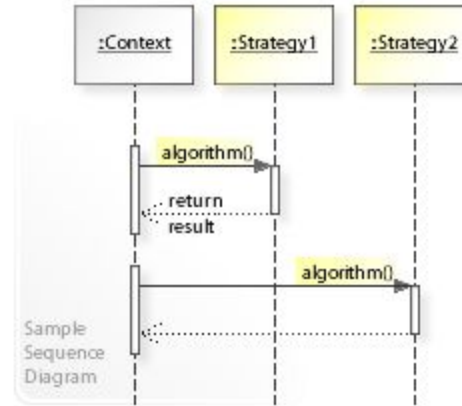
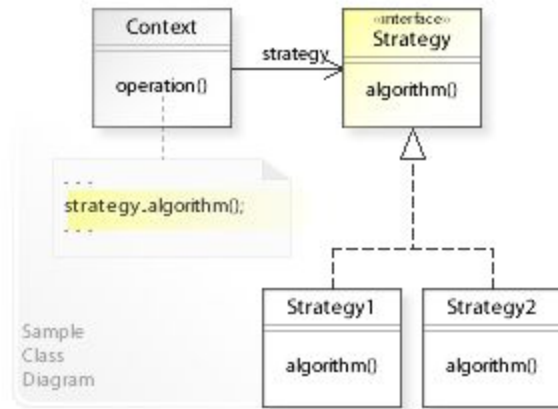
Observer



Strategy

- zdefinujeme skupinu podobných algoritmov s rovnakým rozhraním
- algoritmy potom vieme používať zameniteľne
- vyberieme algoritmus počas behu na základe podmienky
- pred zavolaním algoritmu objekt dostane informáciu o tom, ktorý má použiť a následne zavolá zodpovedajúcu metódu

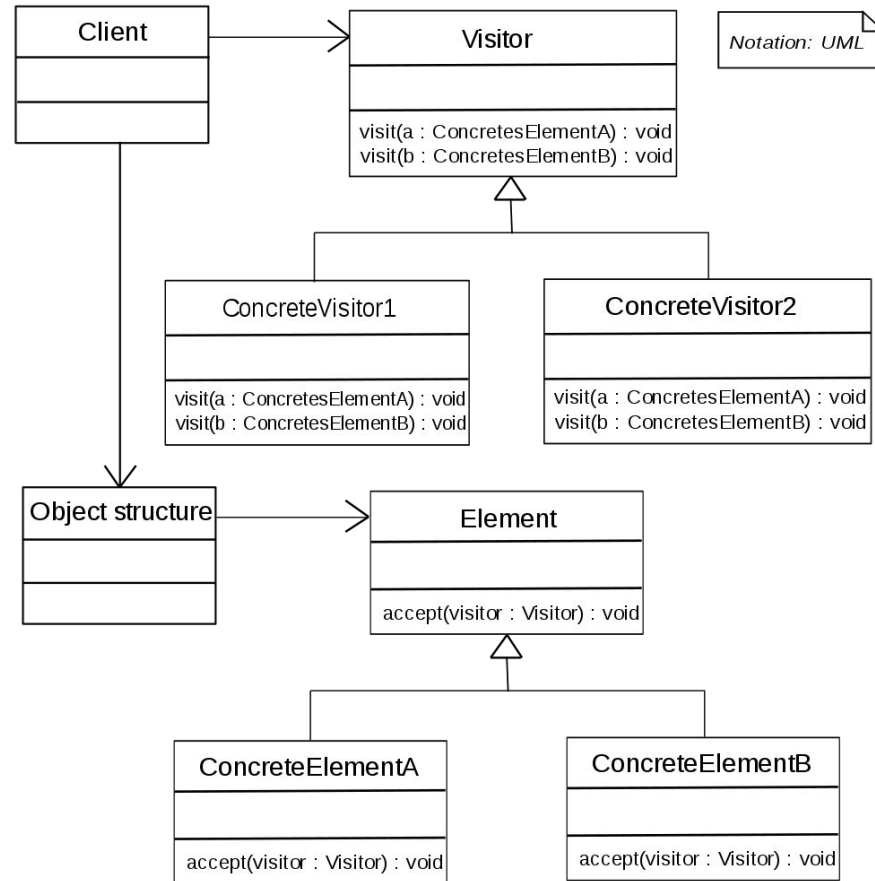
Strategy



Visitor

- pre operácie, ktoré sa vykonávajú nad prvkami štruktúry
- môžeme zadať novú operáciu bez toho aby sme museli zmeniť triedu prvku
- oddelíme algoritmus od objektovej štruktúry ktorou pracuje
- klient neinteraguje priamo s prvkom, namiesto toho pošle visitor objekt, ktorý následne vykoná operáciu nad prvkom

Visitor



Ďalšie behaviorálne návrhové vzory

- blackboard
- memento
- null object
- servant
- specification
- state
- template method

otázky?