Návrhové vzory GoF

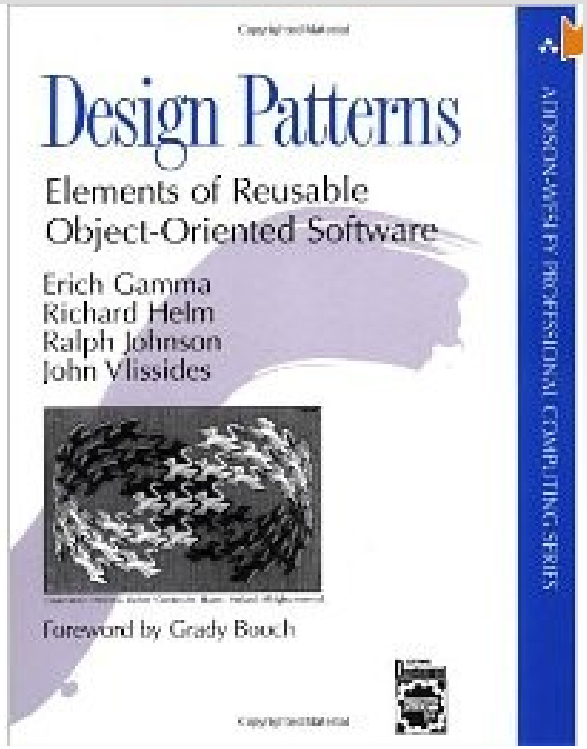
(GoF design patterns)

© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

Návrhové vzory GoF

The Gang-of-Four: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns: Elements of Reusable Object Oriented Software (1995).



Vlissidies pronunciation (dialogue from the internet discussion with his long-time colleague):

Q: And I'd like to make sure I pronounce his name correctly.

A: The weird part is I've known John for years and I'm not even sure :-)

The pronunciation I generally use is VLIH-Suh-dees. I've heard others use VLIH-SEE-DEES...

Návrhový vzor

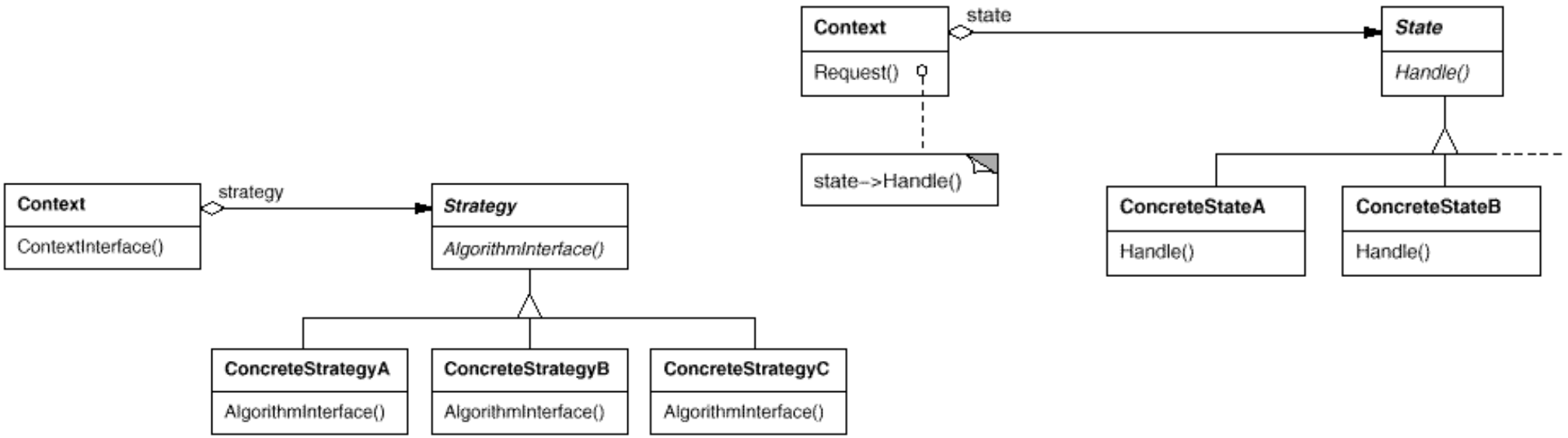
- Popisuje opakovaně se objevující návrhovou strukturu
  - abstrahuje od konkrétních návrhů
  - identifikuje třídy, spolupráce, zodpovědnosti
  - aplikovatelnost, kompromisy, důsledky
- Vzory nejsou návrhy
  - Metamodely, které musí být instanciovány a současně je nutné
    - vyhodnotit kompromisy a důsledky
    - učinit návrhová a implementační rozhodnutí
    - implementovat, kombinovat s jinými vzory
- Společný slovník pro návrh
  - “tady použijeme vzor Observer”
    - zvýšená rychlost návrhu, kultura
  - sdílený slovník
    - Uvnitř/mezi týmy, řízení na různých úrovních
- Příklady
  - Strategie: algoritmy jako objekty
  - Composite: rekurzivní struktury

Vzory vs. důvěra v návrh

- obecná nezkušenost s objekty
  - je můj návrh v pořádku?
- vzory rodí důvěru
  - vždy můžete svést vinu na “Gang of Four”
  - ponechávají prostor pro kreativitu
- většina lidí zná vzory
  - částečně, bez plného pochopení,
  - připouštějí, že ostatní mají podobné návrhy
  - vzory se vylepšují s jejich používáním

Společné problémy

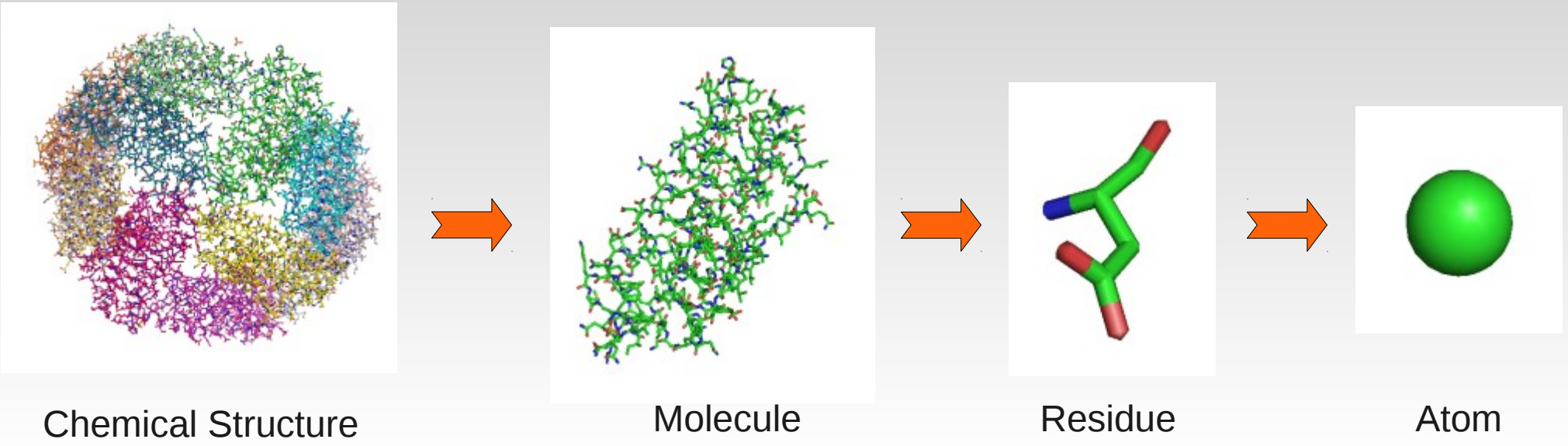
- Krocení přílišného nadšení
  - „vítězíte“, pokud jste použili většinu vzorů
  - řešení nesprávného problému
  - související výdaje a cena
  - vše řeší poslední vzor, který jste se právě naučili
- Struktura místo cíle
  - všechno je Strategie
  - vzory používají podobné konstrukce

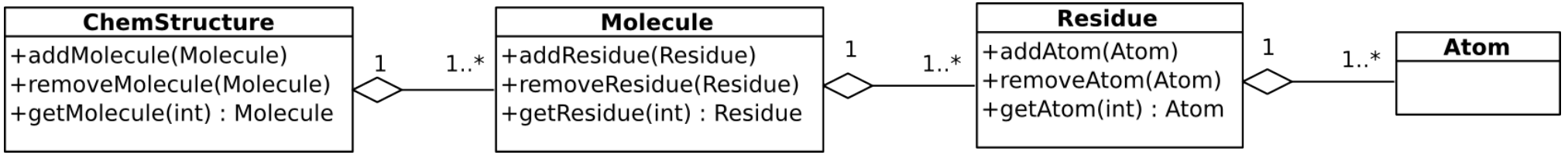


Typy GoF vzorů

- Tvořící vzory (*creational p.*):
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Strukturální vzory (*structural p.*):
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- Vzory chování (*behavioral p.*):
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

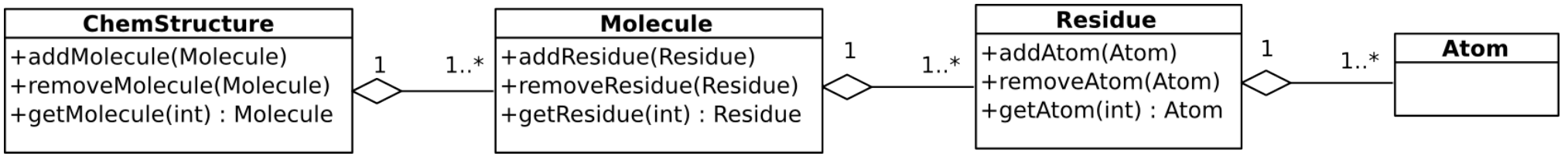
Základní model



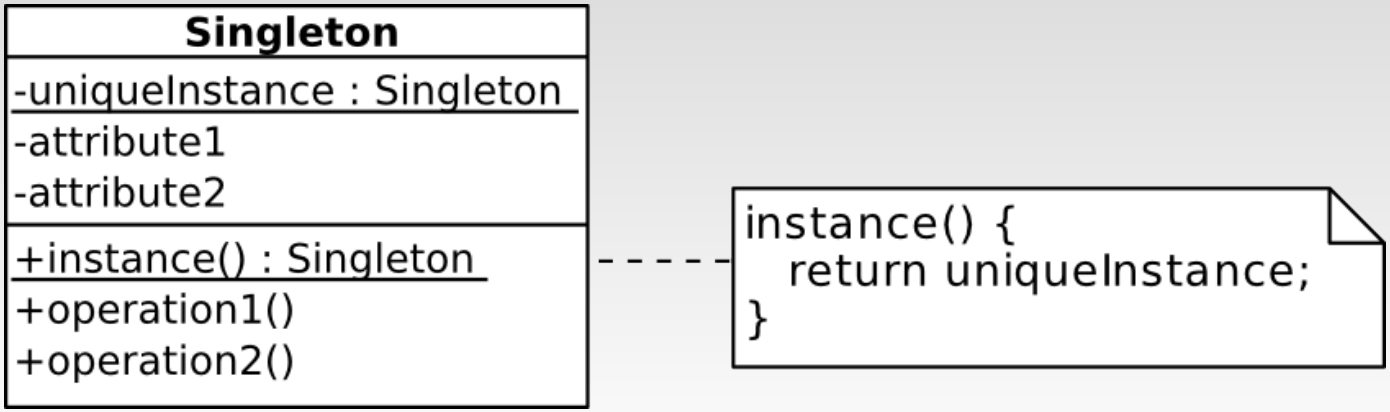


Základní model (pokr.)

- Každý objekt musí být v běžícím systému odkazován (referencován)
  - Každá třída musí být asociována s jinou třídou, NEBO
  - V systému existuje jen několik málo instancí (často jediná), které mají veřejně zjistitelnou adresu. To řeší vzor Singleton.
- Problém:
  - I chemických struktur může být v jednom okamžiku v systému více.



Singleton – vzor




```
public class Singleton {
    private static Singleton uniqueInstance;

    public static Singleton instance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
};
```

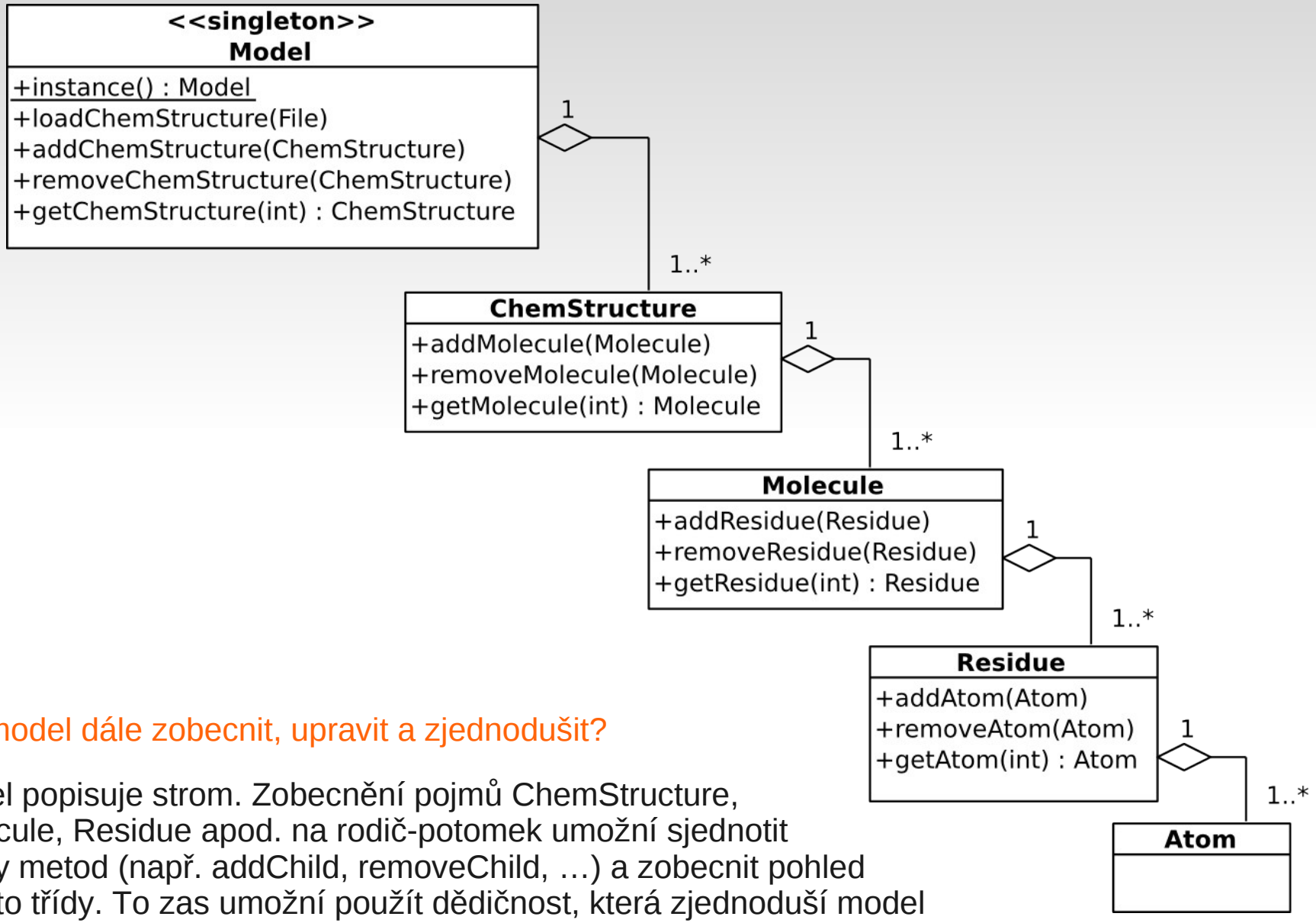
Ukázka volání metod singletonu:

```
// První volání, instance vznikne v paměti
Singleton.instance().operation1();
…
// Další volání stejné nebo jiné metody již
// proběhne na existující jediné instanci
Singleton.instance().operation1();
```

Singleton – charakteristiky

- Aplikovatelnost
  - Chceme, aby existovala právě jedna instance třídy, která by byla přístupná dalším třídám ve známém přístupovém bodě
- Důsledky
  - Řízený přístup k jediné instanci.
  - Redukce jmenného prostoru
    - Náhrada mnoha globálních proměnných jedním singletonem
  - Povoluje úpravu operací
    - Chování singletonu může být změněno jeho podtřídou. Konfigurace aplikace pro použití podtřídy může být dokonce provedena za běhu.
  - Povoluje změnu počtu instancí (úprava vzoru)
  - Pružnější, než pomocí operací třídy
    - Téhož chování lze dosáhnout pomocí statických metod, u kterých ale obvykle chybí polymorfismus. Stejně tak je obtížné pracovat s více instancemi.

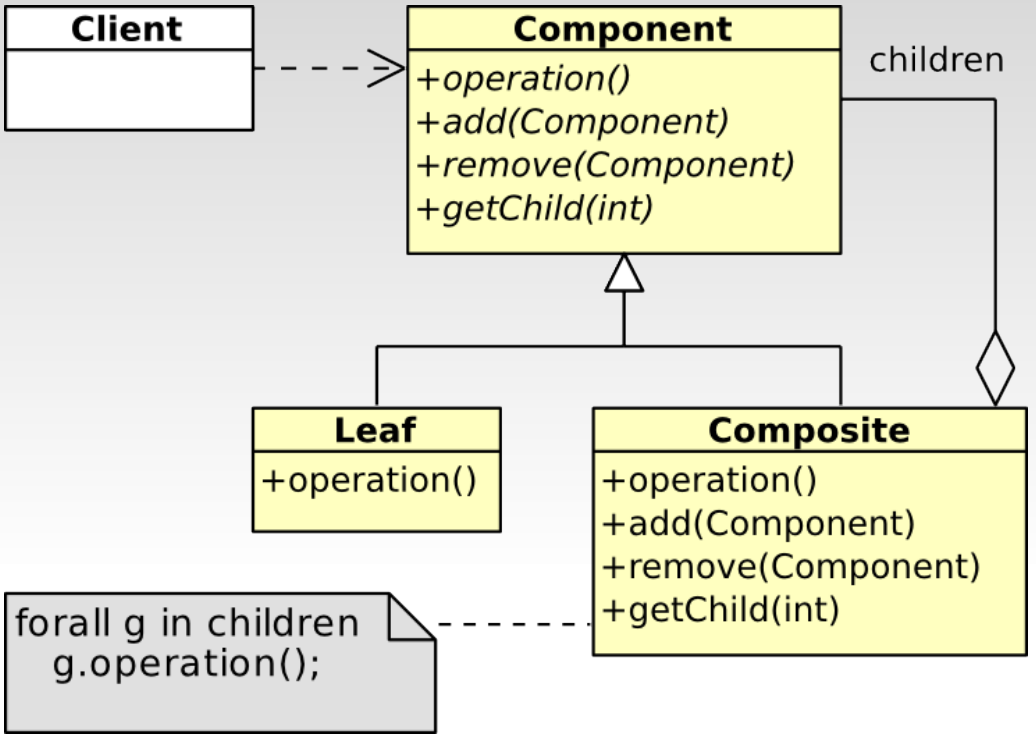
Singleton – příklad aplikace vzoru

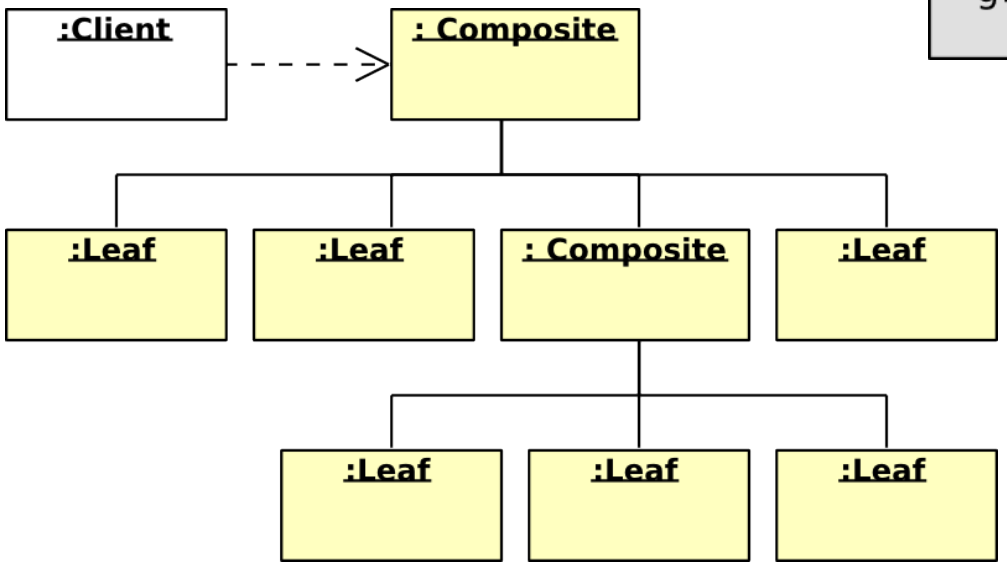


Jak model dále zobecnit, upravit a zjednodušit?

Model popisuje strom. Zobecnění pojmů ChemStructure, Molecule, Residue apod. na rodič-potomek umožní sjednotit názvy metod (např. addChild, removeChild, …) a zobecnit pohled na tyto třídy. To zas umožní použít dědičnost, která zjednoduší model a zredukuje duplicitní kód.

Composite – vzor



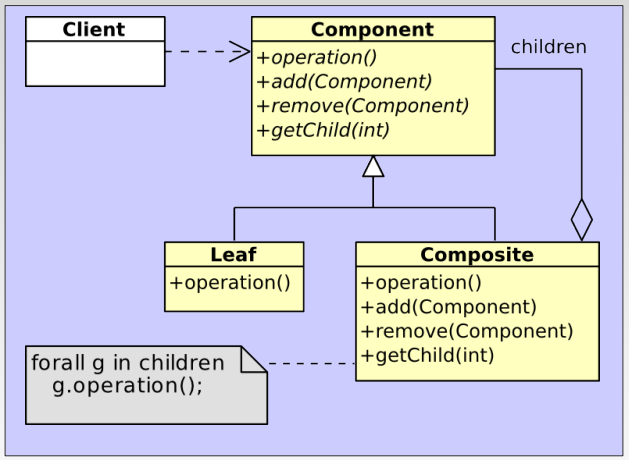


Composite – charakteristiky

- Aplikovatelnost
  - reprezentace hierarchií ***celek - část***
  - klienti mohou ignorovat rozdíly mezi sestavami objektů a individuálními objekty
- Důsledky
  - definuje hierarchie tříd složené z primitivních objektů a složených objektů
  - zjednodušuje klienta, umožňuje mu, aby pracoval stejně s objekty i s jejich kompozicemi
  - usnadňuje přidávání nových druhů komponentů
  - návrh se stává obecnějším
  - Nevýhoda: je obtížné omezit přípustné komponenty v sestavě

Composite – příklad aplikace vzoru

Jak skrýt implementaci seznamu potomků (teď indexované pole) a jak umožnit různé způsoby procházení a/nebo různé implementace v podtřídách?



<<composite-component>>
ModelElement
+add(ModelElement)
+remove(ModelElement)
+get(int) : ModelElement
+setColor(Color)

subElements
1..*
superElement

<<composite-leaf>>
Atom
-color : Color
+setColor(Color)

<<composite-composite>>
GroupElement
+add(ModelElement)
+remove(ModelElement)
+get(int) : ModelElement
+setColor(Color)

setColor() se aplikuje na všechny potomky

<<singleton>>
<<composite-composite>>
Model
+instance() : Model
+loadChemStructure(File)

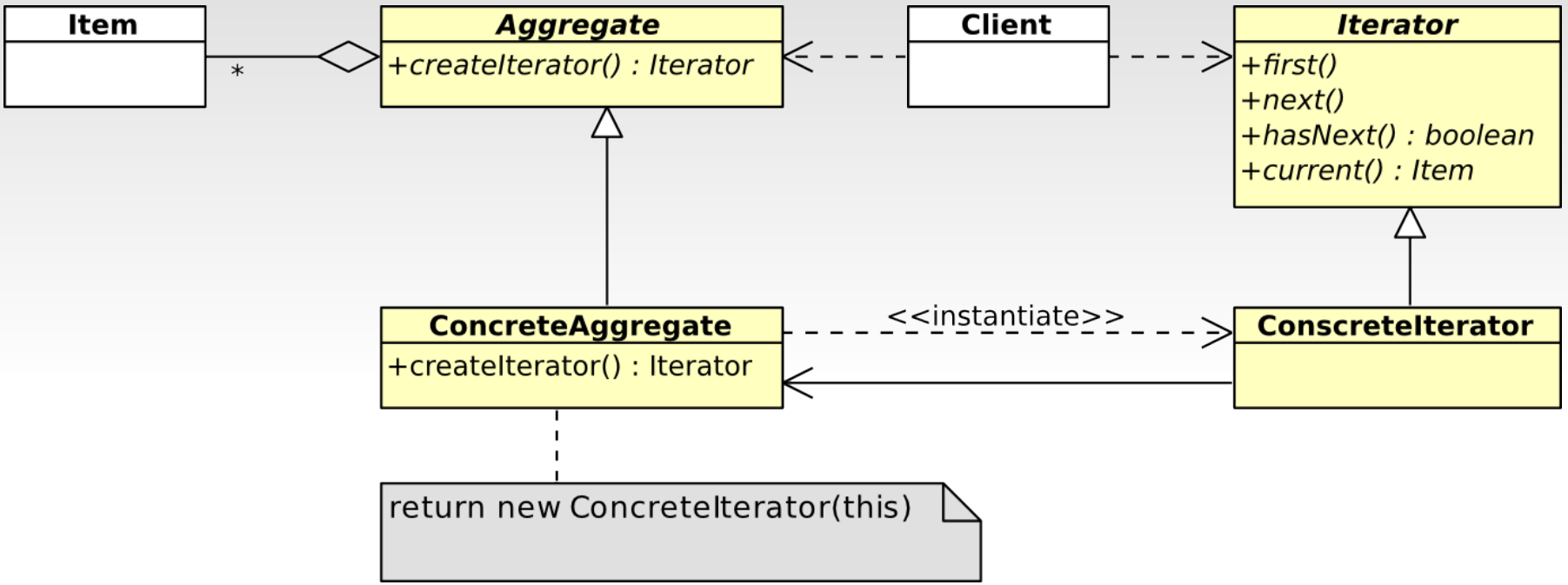
<<composite-composite>>
Molecule

<<composite-composite>>
ChemStructure

<<composite-composite>>
Residue

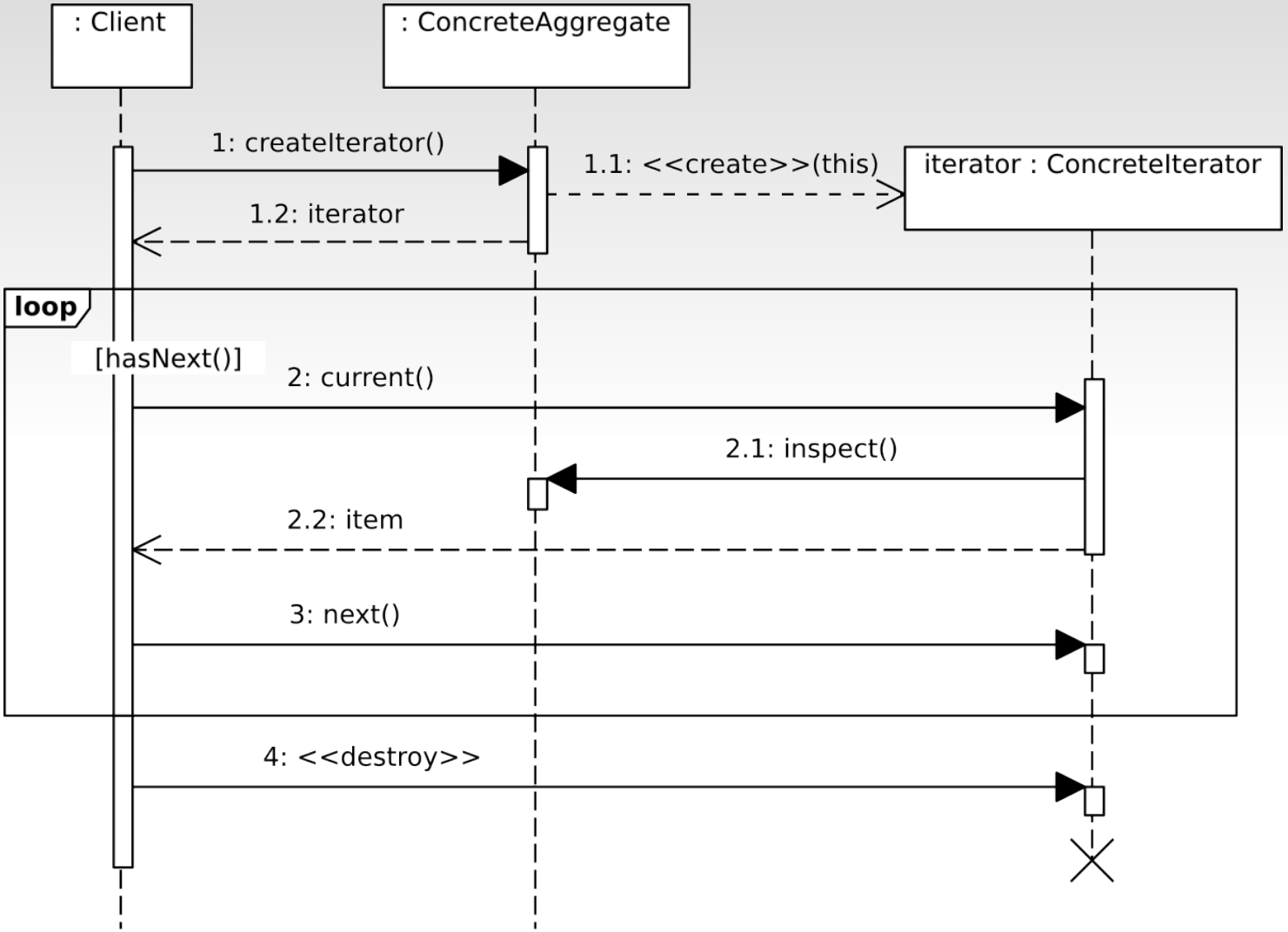
Vzory se často kombinují

Iterator – vzor



Viz. např. java.util.Collection a java.util.Iterator

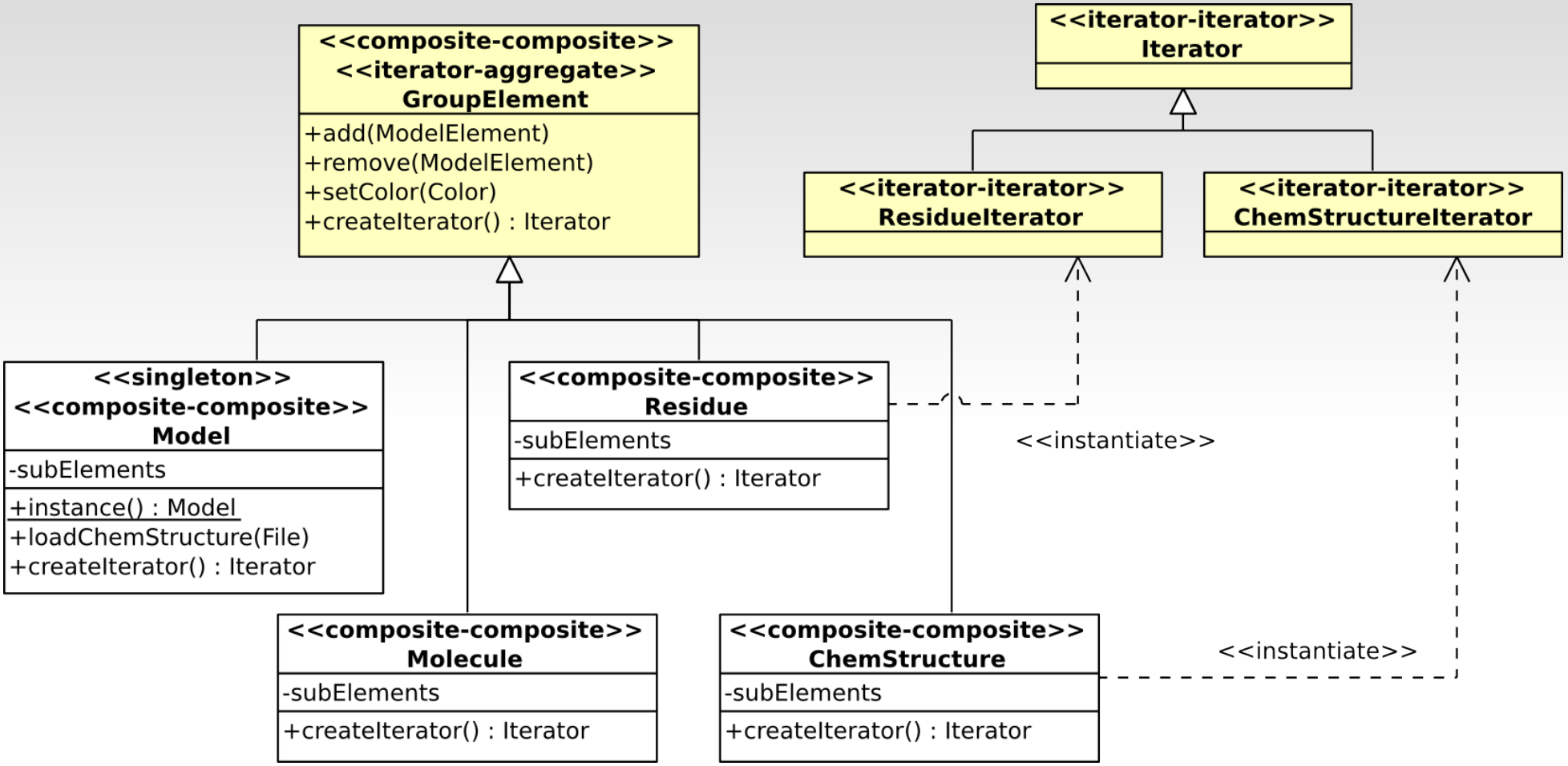
Iterator – vzor (pokr.)



Iterator – charakteristiky

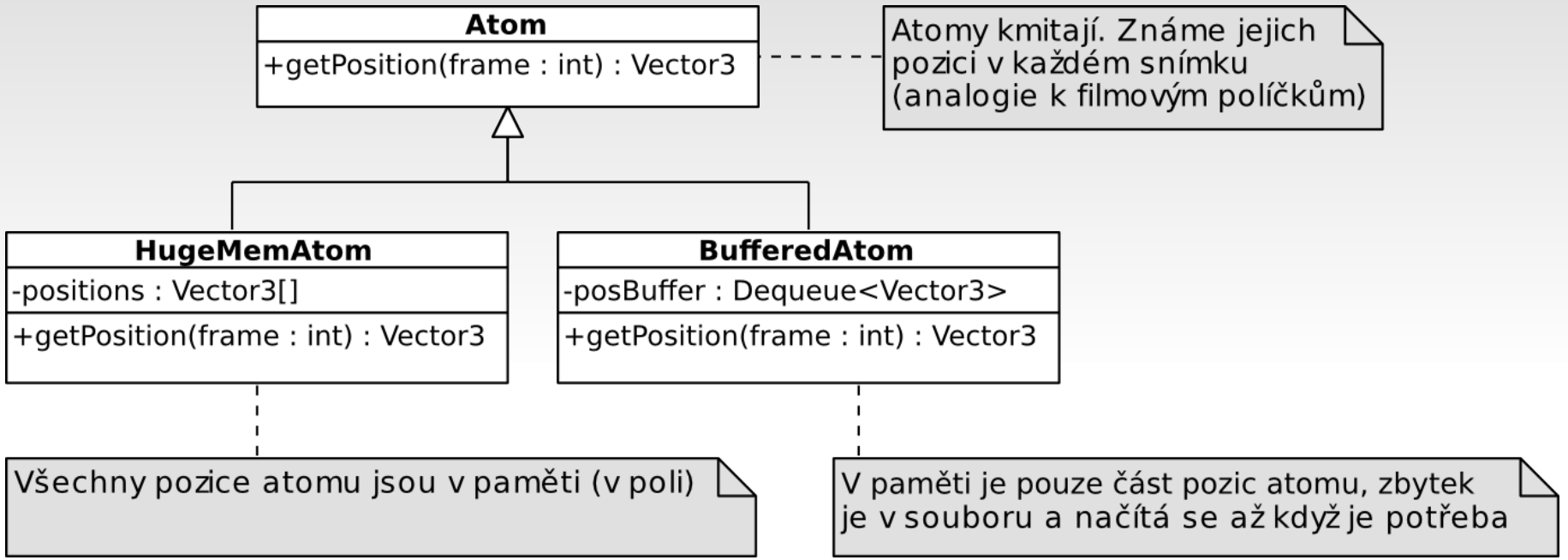
- Aplikovatelnost
  - přístup k obsahu agregovaného objektu bez vystavení jeho vnitřní reprezentace
  - podpora násobných průchodů agregovaným objektem
  - poskytnutí jednotného rozhraní pro procházení odlišných agregovaných struktur (podpora polymorfní iterace)
- Důsledky
  - Podporuje variace průchodů agregátem.
  - Zjednodušuje rozhraní agregátu.
  - Může být spuštěno více souběžných průchodů agregátem (stav procházení je uložen v iterátoru).

Iterator – příklad aplikace vzoru



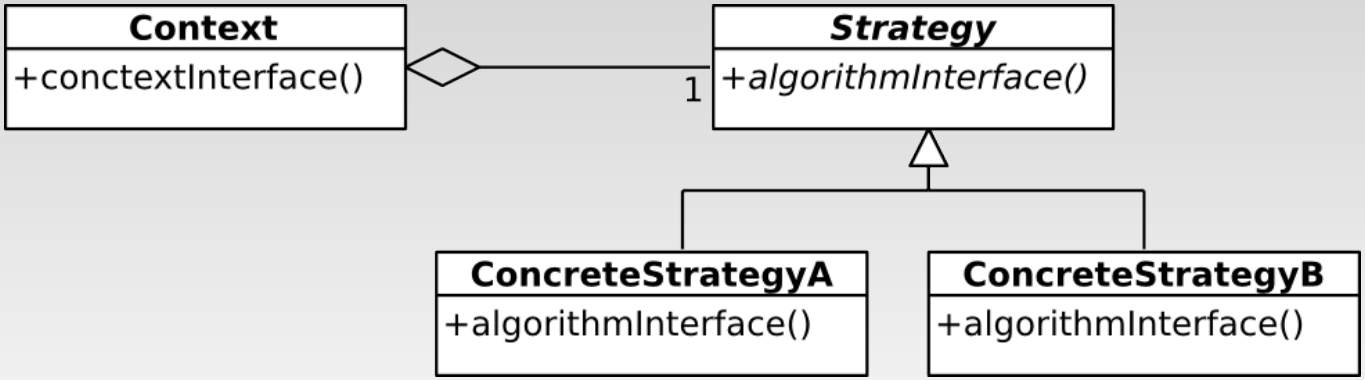
Pozn.: Své potomky nyní neagreguje GroupElement, ale jednotlivé podtřídy.

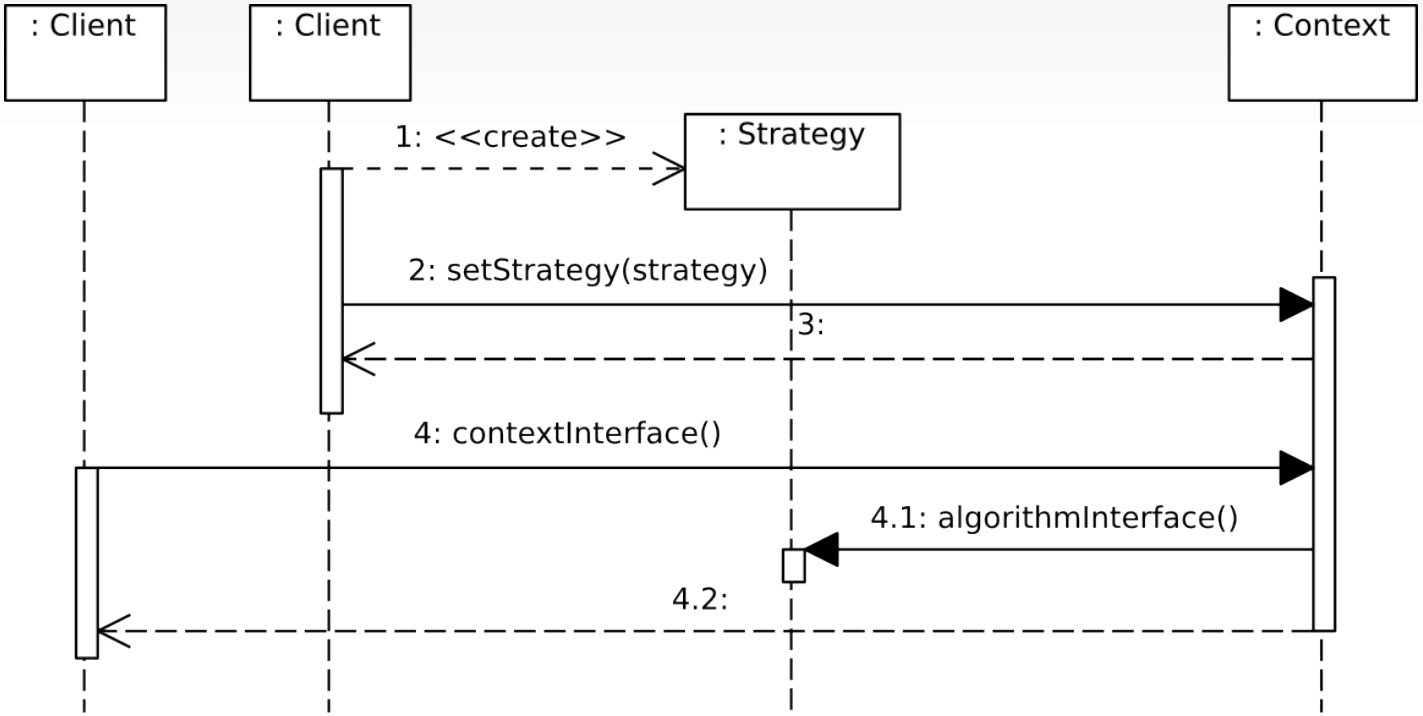
Další geneze příkladu



Podtříd může být mnoho, přitom mají společné rozhraní a liší se jen chováním.
Podtřídy tedy můžeme vnímat jako různé **strategie** výpočtu pozice v daném framu.

Strategy – vzor



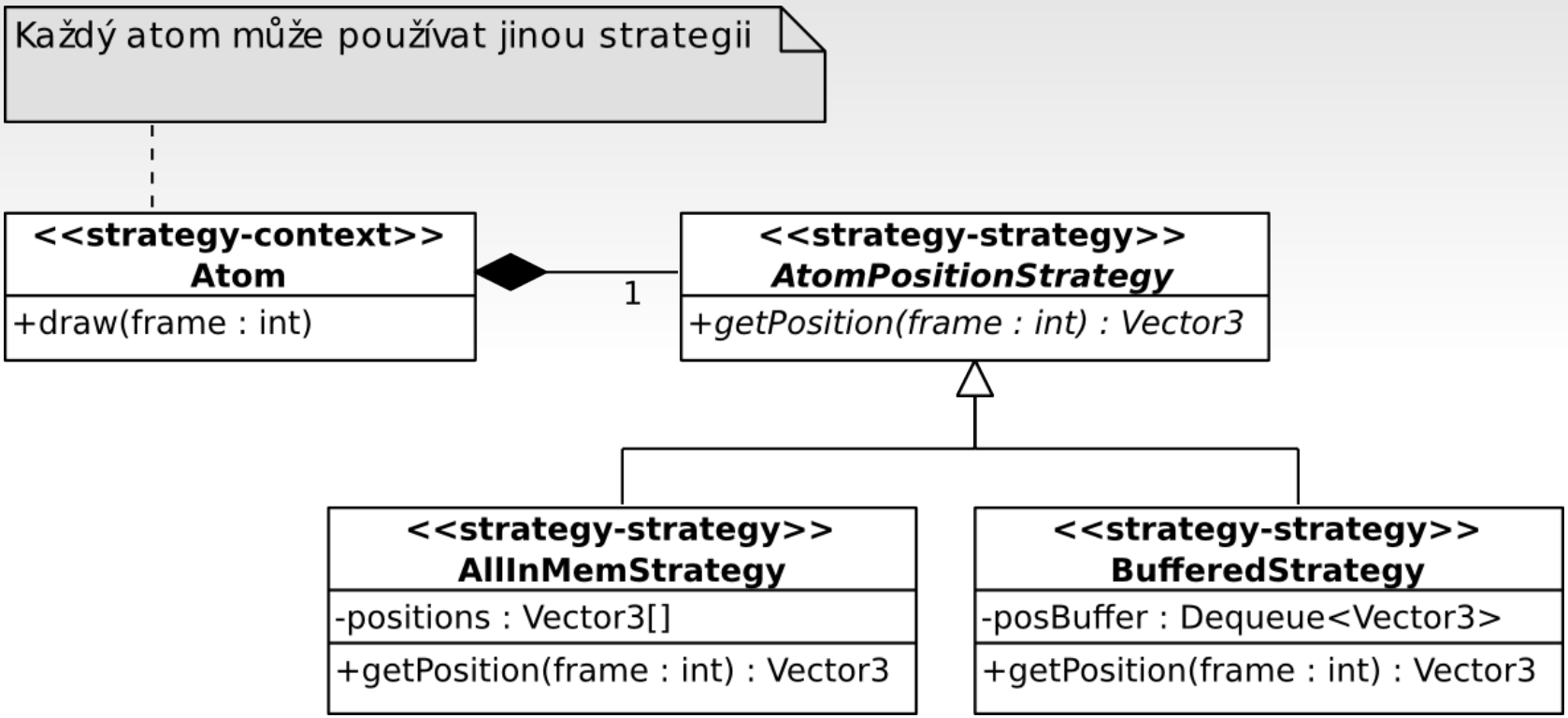


Např.: Context = okno textového editoru, Strategy = algoritmus zalamování řádků.

Strategy – charakteristiky

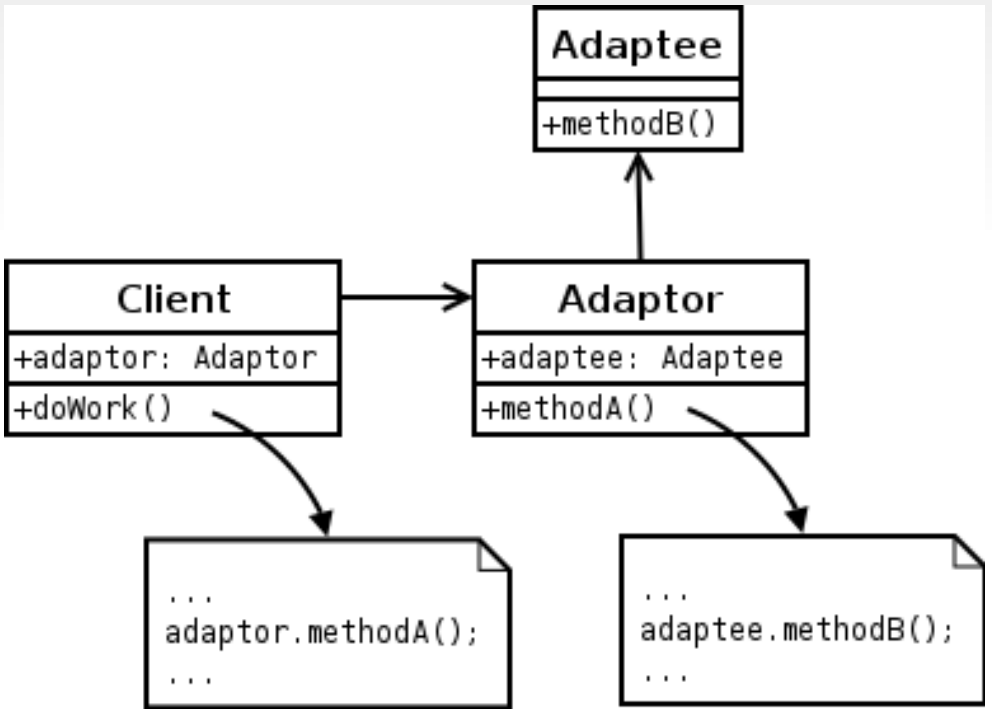
- Aplikovatelnost
  - Mnoho příbuzných tříd (třídy se stejnými metodami) se liší pouze svých chováním (implementací metod). Strategie umožňuje mít pouze jednu třídu, která má „nakonfigurované“ jedno chování.
  - Potřebujeme různé varianty jednoho algoritmu.
  - ...
- Důsledky
  - Vznikají „rodiny“ podobných algoritmů. Vhodné pro znovupoužití.
  - Alternativa k podtřídám.
  - Eliminují se sekvence podmíněných výrazů (switch-case).
  - Různé implementace stejného chování.
  - Klient musí znát rozdíly mezi strategiemi, protože on určuje, která z nich se použije.
  - Komunikační režie.
  - Větší počet objektů.

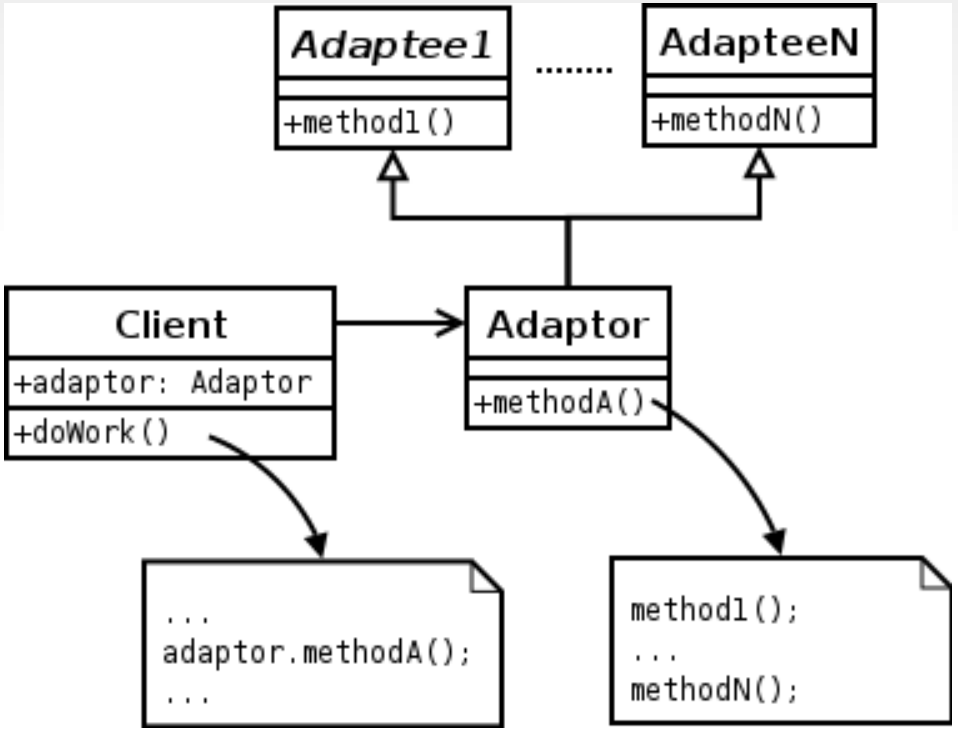
Strategy – příklad aplikace vzoru



Adapter

- Jinak také: wrapper
- Konverze rozhraní do potřebné podoby
- Dva způsoby: adapter objektu (vlevo) a adapter třídy (vpravo)





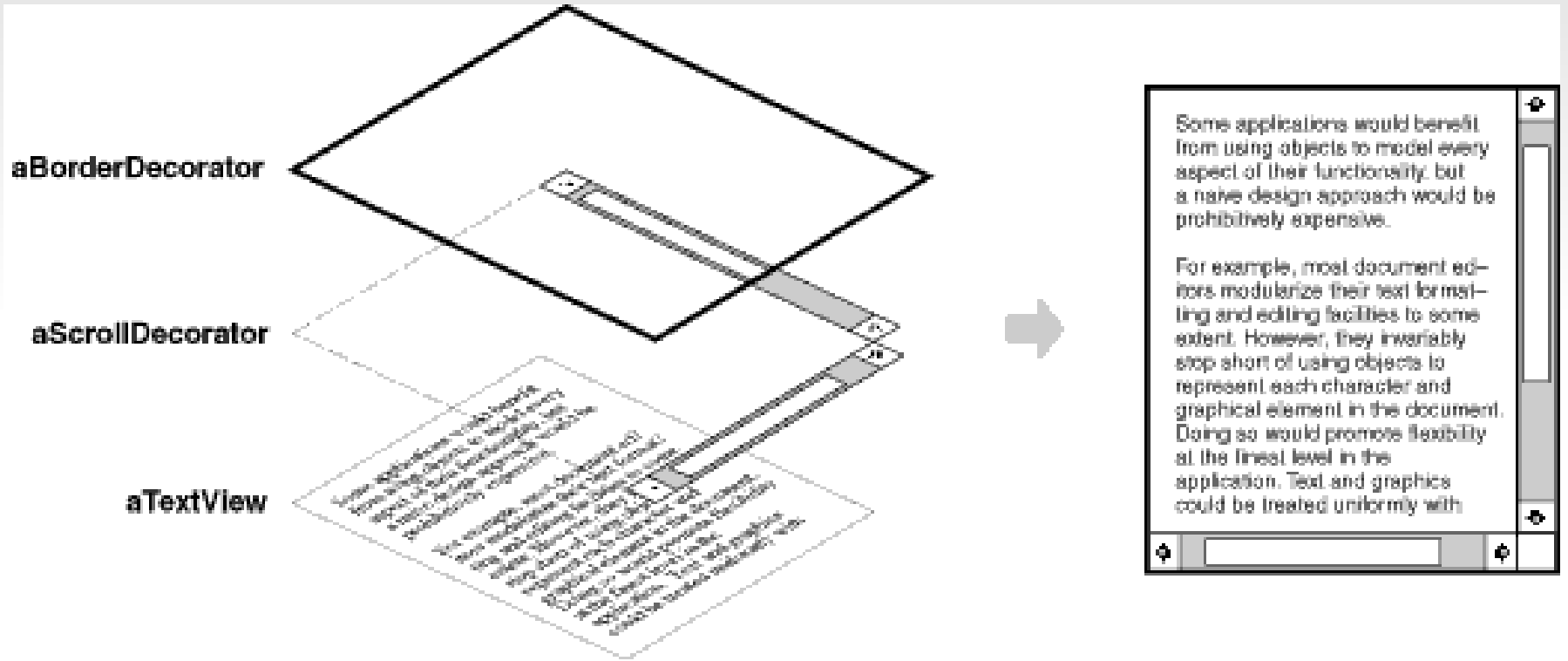
Adapter - charakteristiky

- Aplikovatelnost
  - chceme použít existující třídu a její rozhraní neodpovídá tomu, které potřebujeme, nebo
  - chceme vytvořit znovupoužitelnou třídu, která spolupracuje s nepředvídanými nebo nevztaženými třídami, které nemusí mít kompatibilní rozhraní, nebo
  - ***(pouze objektový adapter)*** potřebujeme použít několik existujících podtříd, ale není praktické je modifikovat mechanismem podtříd (adaptujeme rozhraní jejich rodičovské třídy)

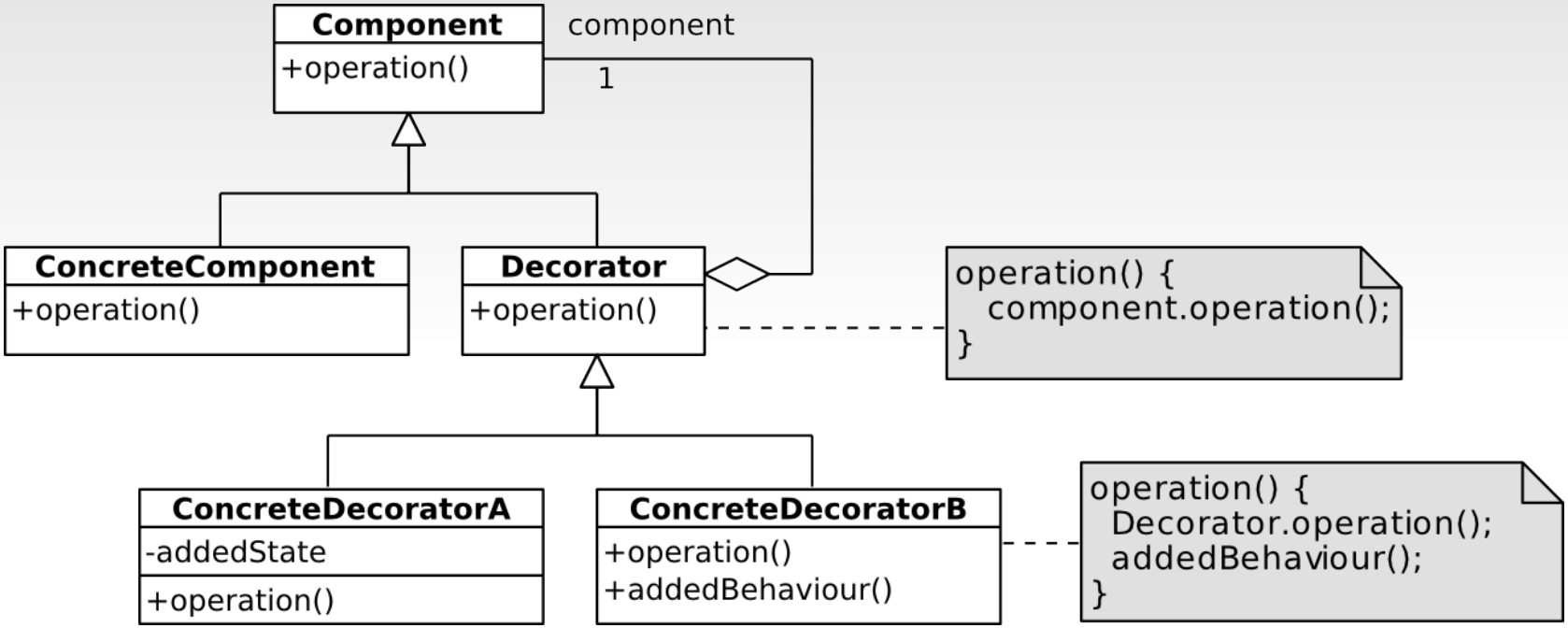
Adapter - charakteristiky

- Důsledky (***adapter třídy***)
  - Přizpůsobujeme Adaptovaného k Cíli připojením se ke konkrétní třídě Adapter. Adapter *třídy* nebude funkční, pokud chceme adaptovat třídu a všechny její podtřídy
  - Umožňuje, aby Adapter změnil chování Adaptovaného, protože Adapter je podtřídou Adaptovaného
  - Zavádí pouze jeden objekt a není tedy nutné použít nepřímé ukazatele k Adaptovanému
- Důsledky (***adapter objektu***)
  - Umožňuje jednomu Adapteru pracovat s mnoha Adaptovanými - tj. nejen s Adaptovaným, ale i s jeho podtřídami
  - Chování Adaptovaného lze změnit obtížněji. Vyžaduje vytvoření podtřídy a odkaz Adapteru na podtřídy Adaptovaného

Decorator – motivace



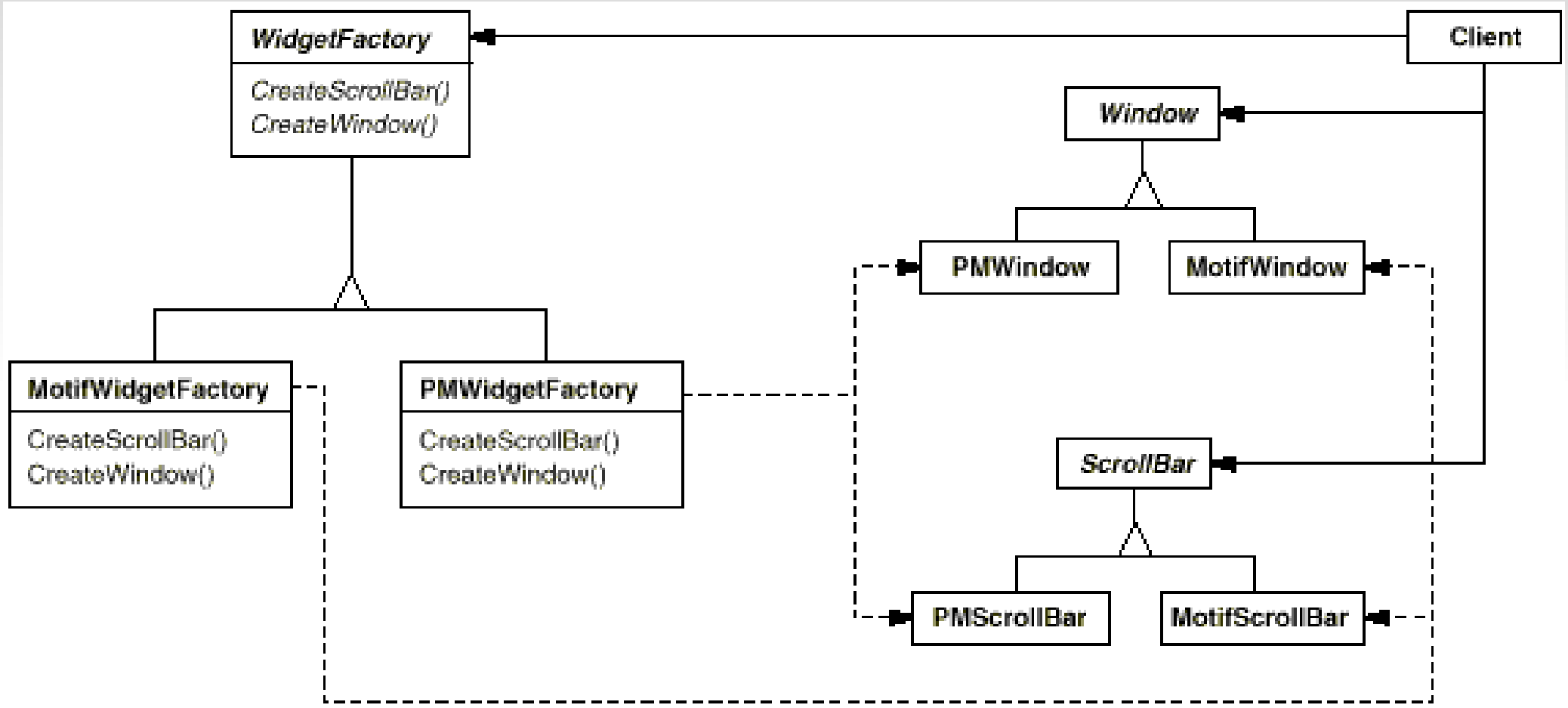
Decorator – vzor



Decorator – charakteristika

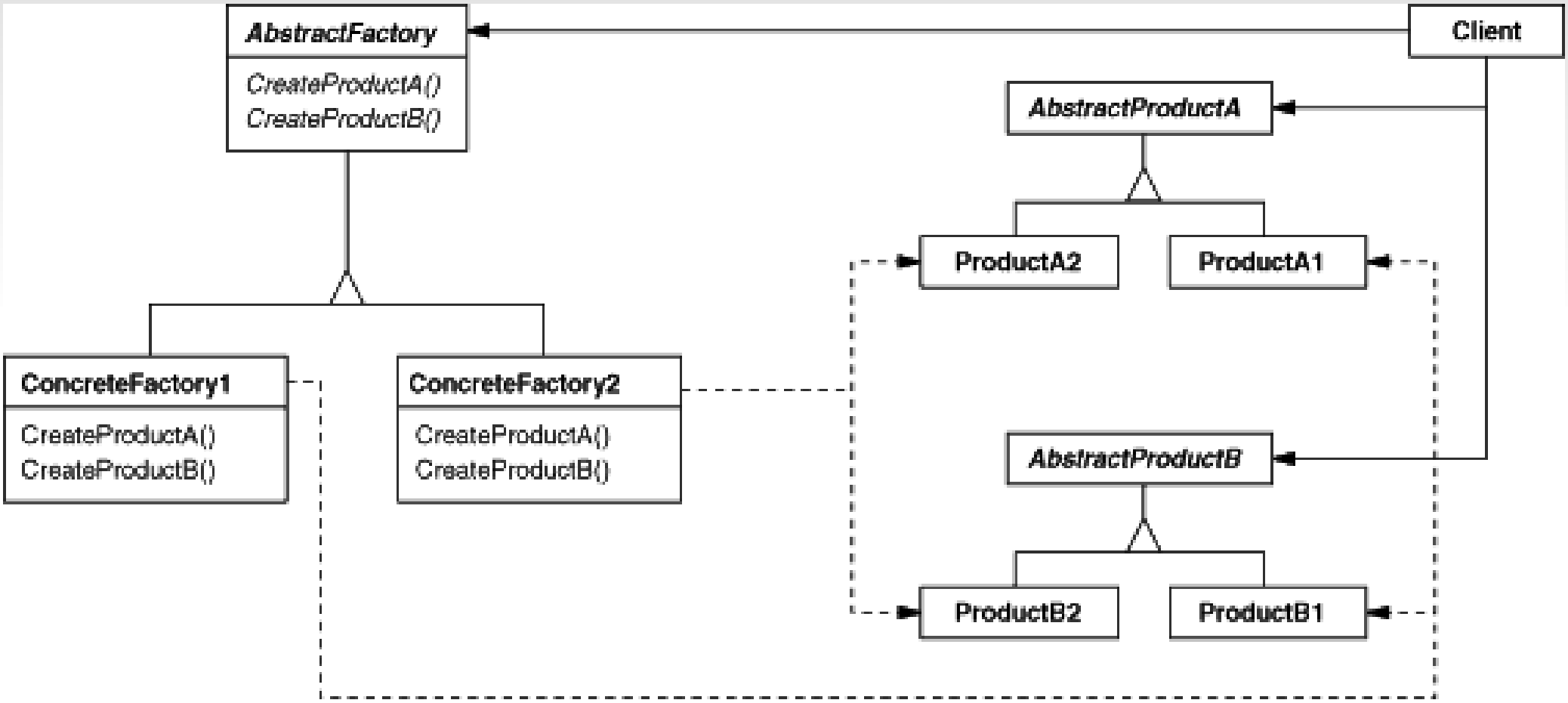
- Podobné vzory
  - **Adapter:** Decorator mění pouze odpovědnost (chování), nikoliv rozhraní.
  - **Composite:** Na Decorator se můžeme dívat jako na degenerovaný Composite s jedinou komponentou. Decorator ale přidává další odpovědnosti, není určen pouze k agregaci.
  - **Strategy:** Decorator umožňuje měnit kůži, Strategy umožňuje měnit vnitřnosti. Jsou to dvě alternativní cesty pro změnu objektu (liší se pohledem na věc).
- Která standardní a často používaná část API jazyka Java (probírá se i v PB162) je navržena pomocí vzoru Decorator?
  - java.io, viz např. BufferedReader

Abstract Factory - motivace



Motif = GUI pro UNIX
PM = Presentation Manager = GUI pro OS/2

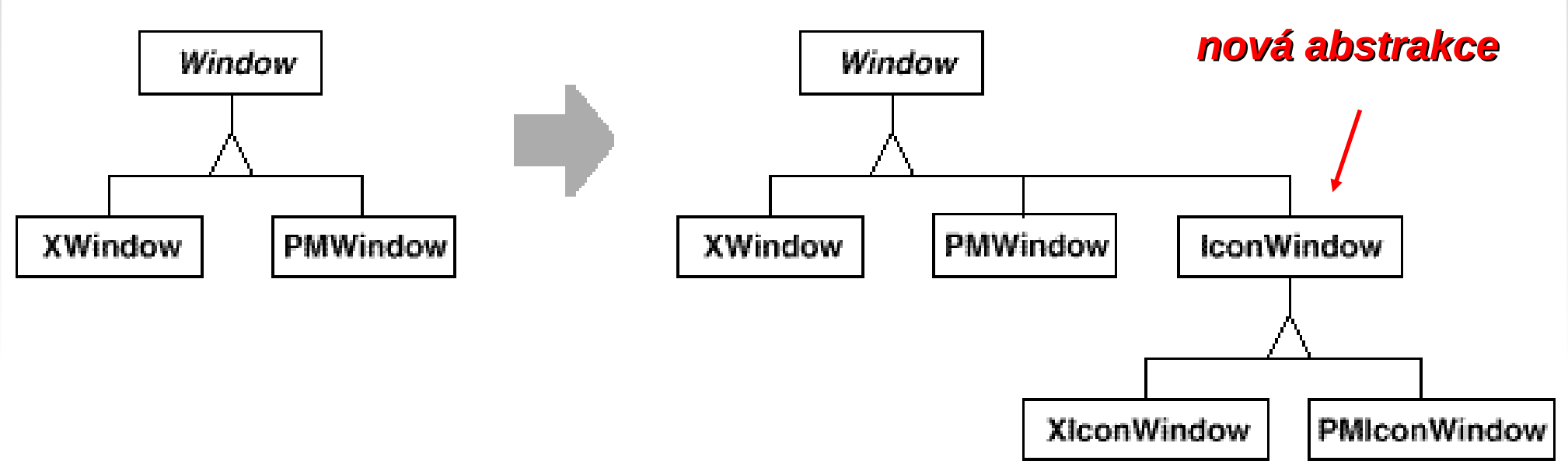
Abstract Factory - vzor



Abstract Factory - charakteristiky

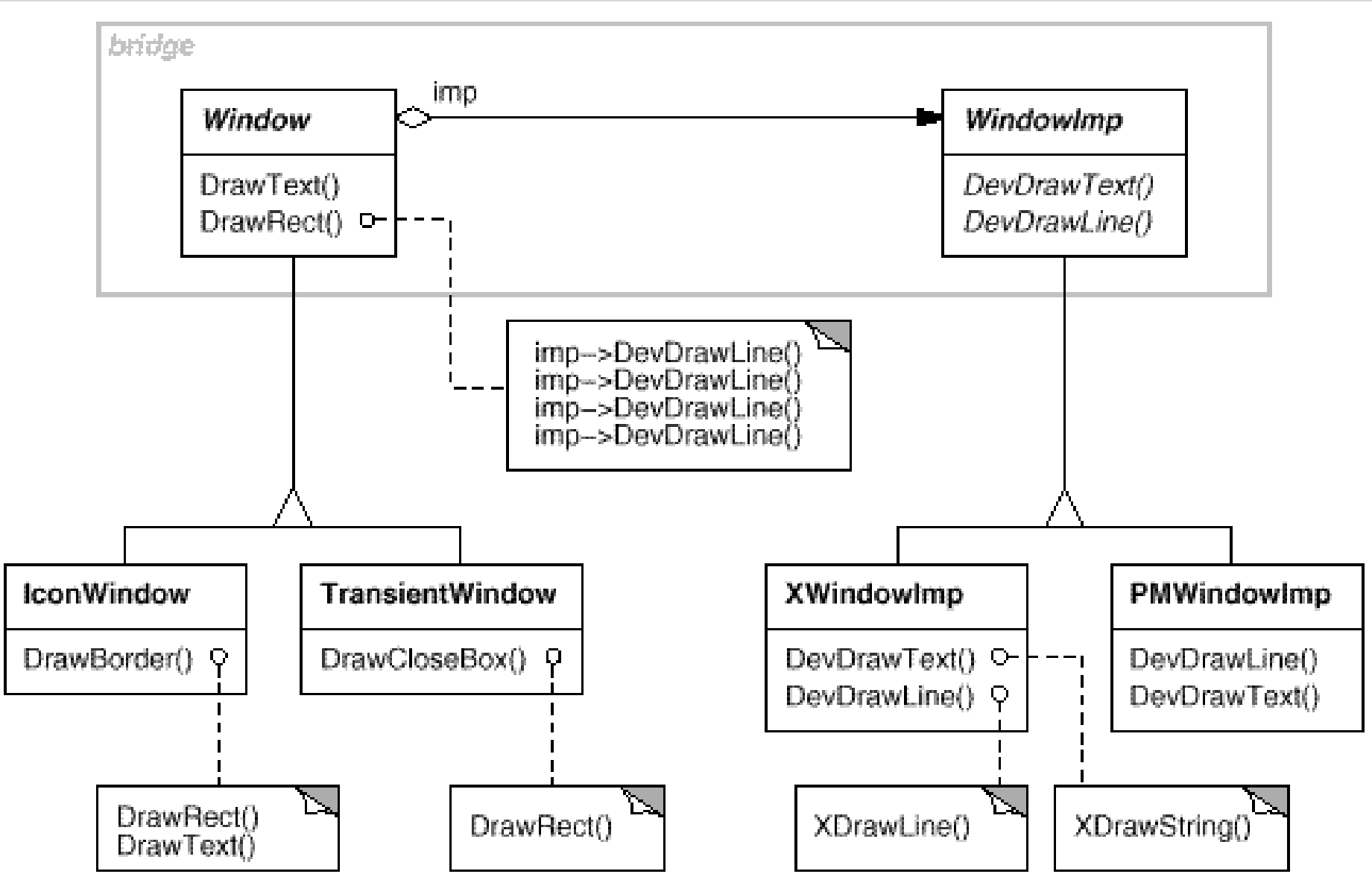
- Aplikovatelnost
  - Systém nemá být závislý na tom, jak jsou výrobky vytvářeny, sestavovány a reprezentovány
  - Systém bude konfigurován pro jednu z mnoha rodin výrobků
  - Rodina souvisejících výrobků bude používána společně a toto je potřeba zajistit (vynutit)
  - Vytváříme objektovou knihovnu výrobků a chceme zveřejnit pouze jejich rozhraní, ne implementaci
- Důsledky
  - Izolace konkrétních tříd
    - produkty jsou implementovány vně klienta, ten používá jen obecné rozhraní.
  - Snadná záměna rodin výrobků
  - Podpora pro konzistenci výrobků
  - Podpora nových druhů výrobků je obtížná.

Bridge - mitivace



oddělení abstrakce od implementace ....

Bridge – motivace

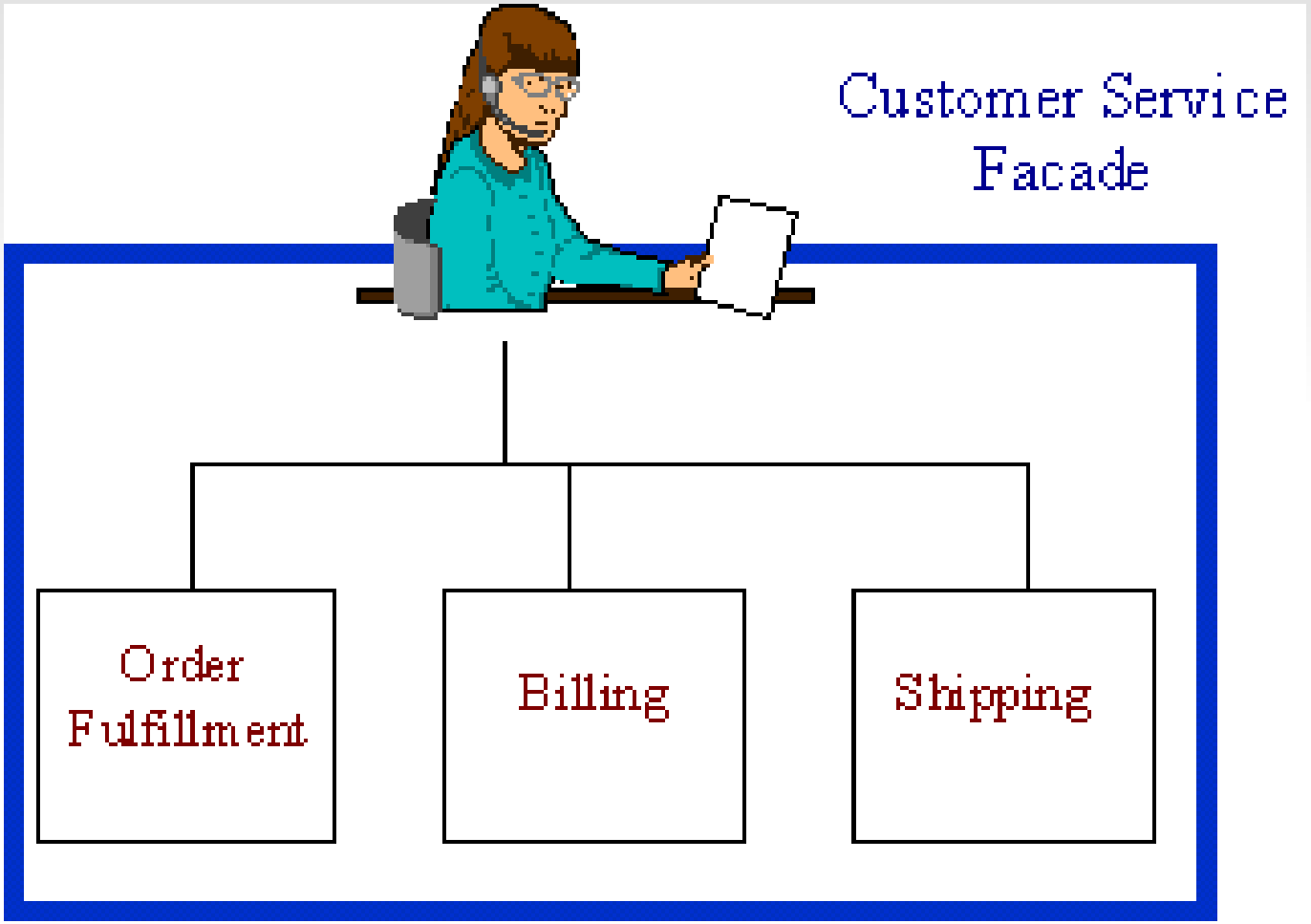


Smysl: Oddělení abstrakce od implementace.

Bridge – charakteristika

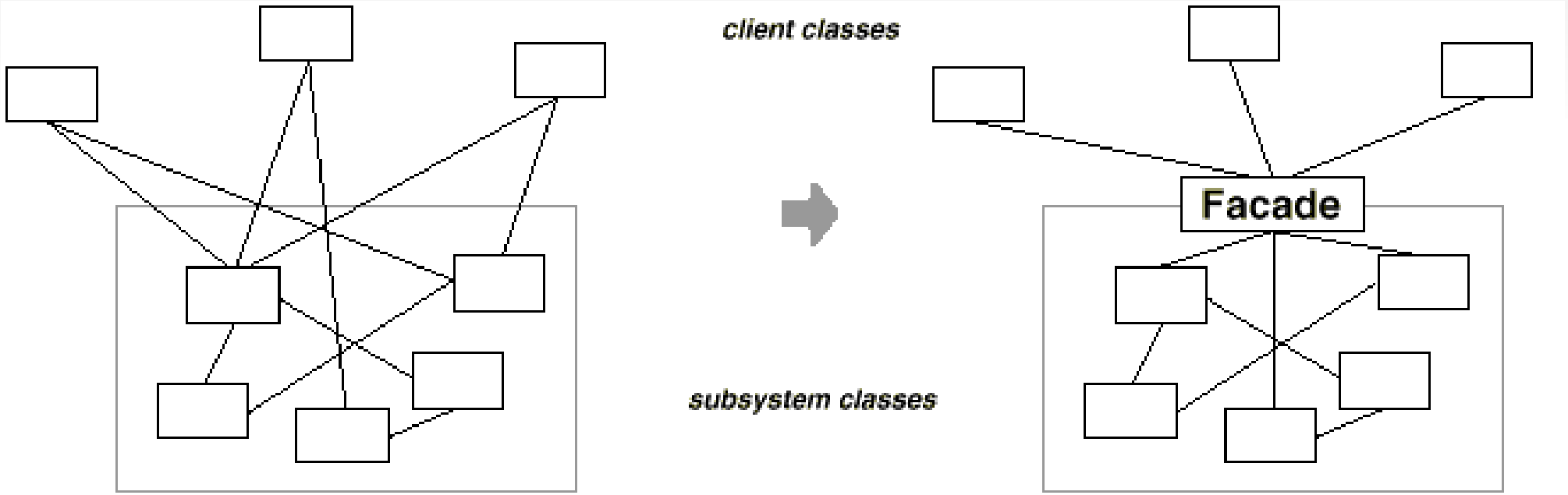
- *Abstract Factory* může vytvářet a konfigurovat jednotlivé *Bridge.*
- V *Bridge* se objevují prvky *objektového adaptéru*. Adapter ale slouží zejména k tomu, aby dvě nekompatibilní třídy spolu spolupracovali. Bridge je používán na začátku návrhu s cílem oddělit abstrakci od implementace.
- Která standardní a často používaná část API jazyka Java (probírá se i v PB162) je navržena pomocí vzoru Decorator?
  - java.io, viz např. OutputStreamWriter (propojuje hierarchii Writer s OutputStream)

Facade (fasáda) - motivace

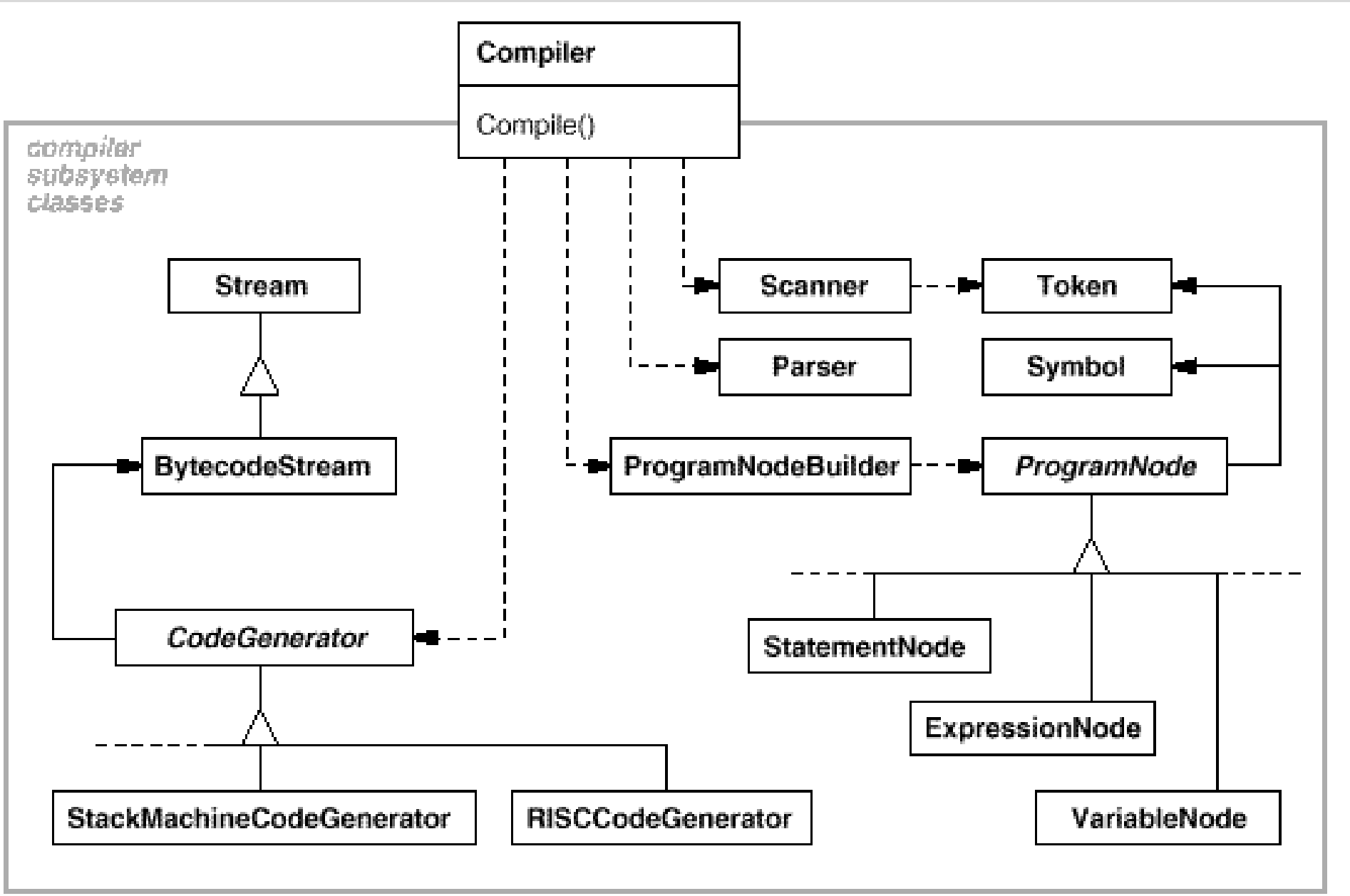


Facade - motivace

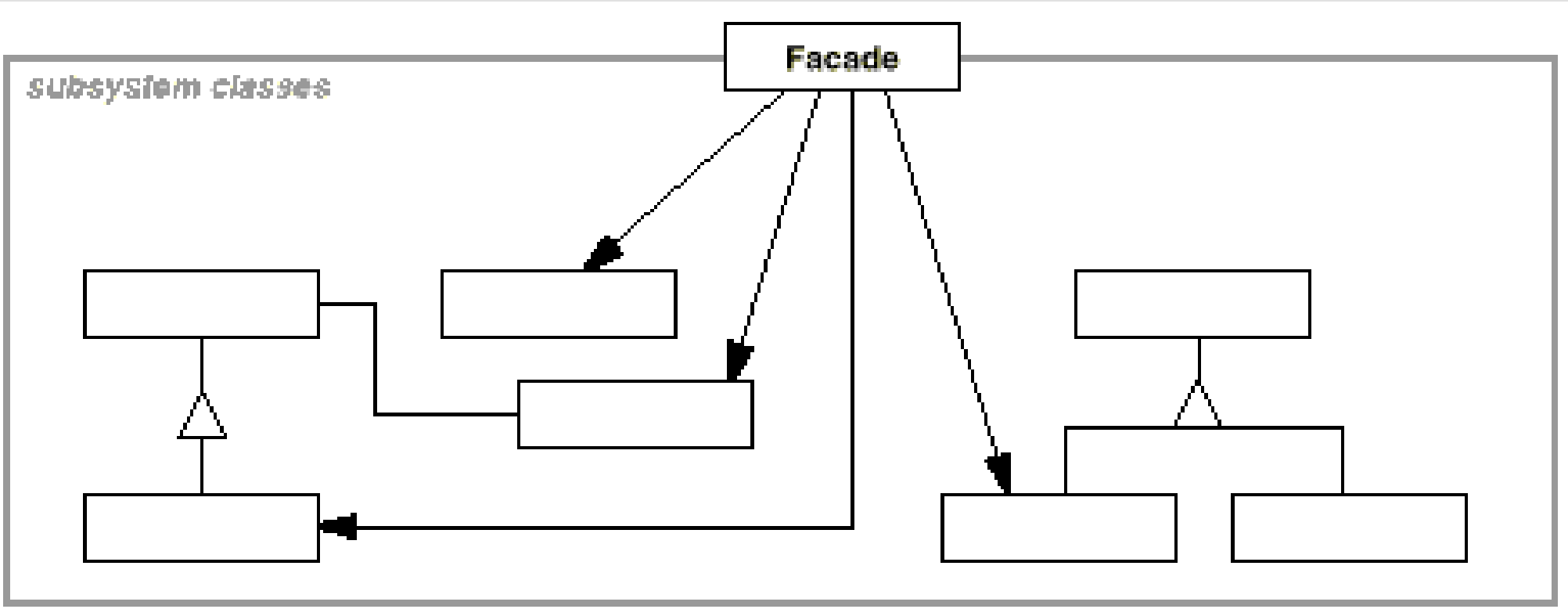
Záměr: Poskytnout sjednocené rozhraní k množině rozhraní v subsystému. Fasáda definuje rozhraní na vyšší úrovni a usnadňuje použití subsytému.



Facade - motivace



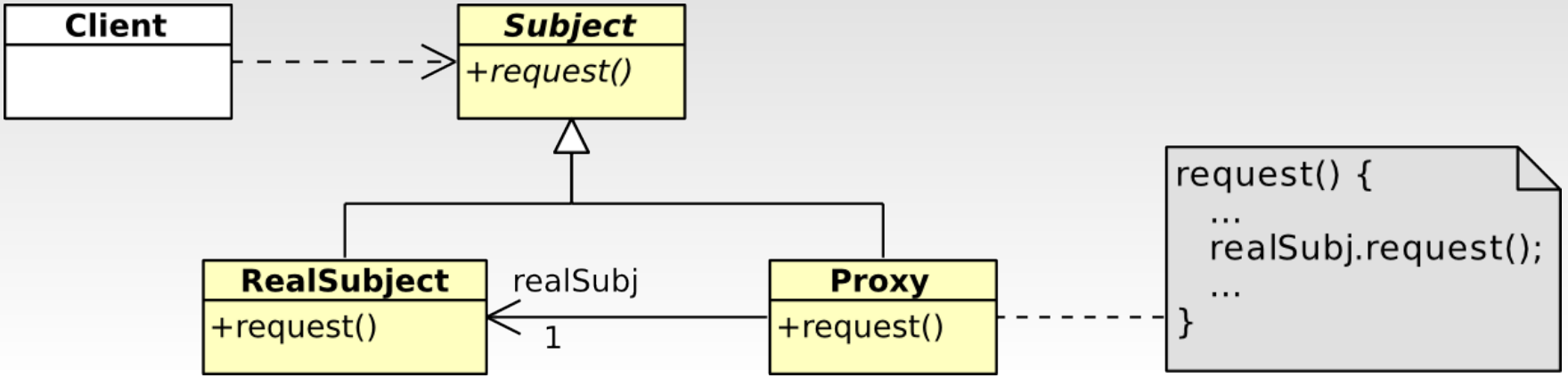
Facade - vzor

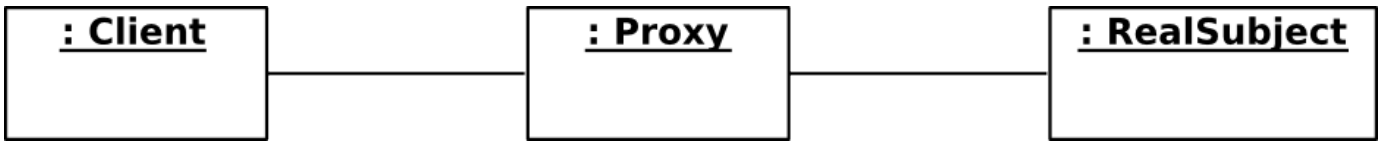


Facade – příklady použití

- Simple Logging Facade for Java (SLF4J)
  - http://www.slf4j.org/
- User Interface Facades

Proxy – vzor

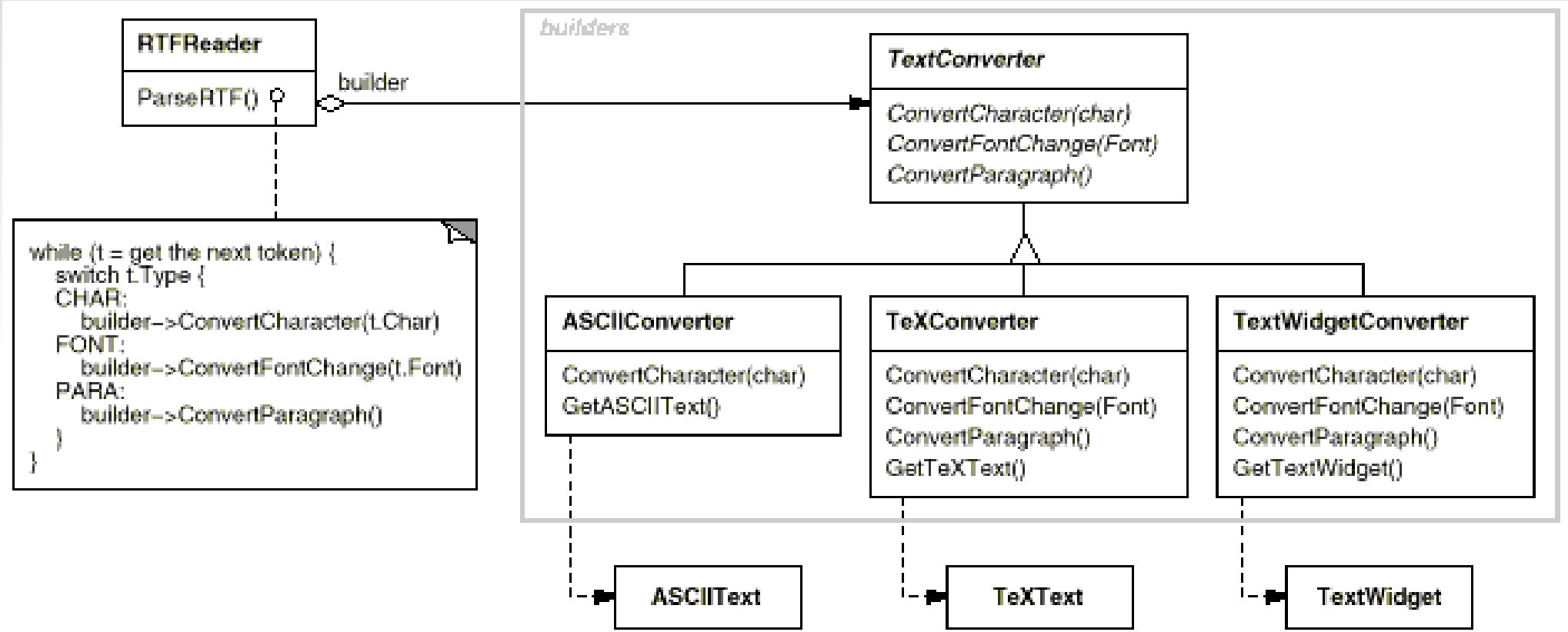

Client
Subject
+request()
RealSubject
+request()
realSubj
1
Proxy
+request()
request() {
...
realSubj.request();
...
}


: Client
: Proxy
: RealSubject

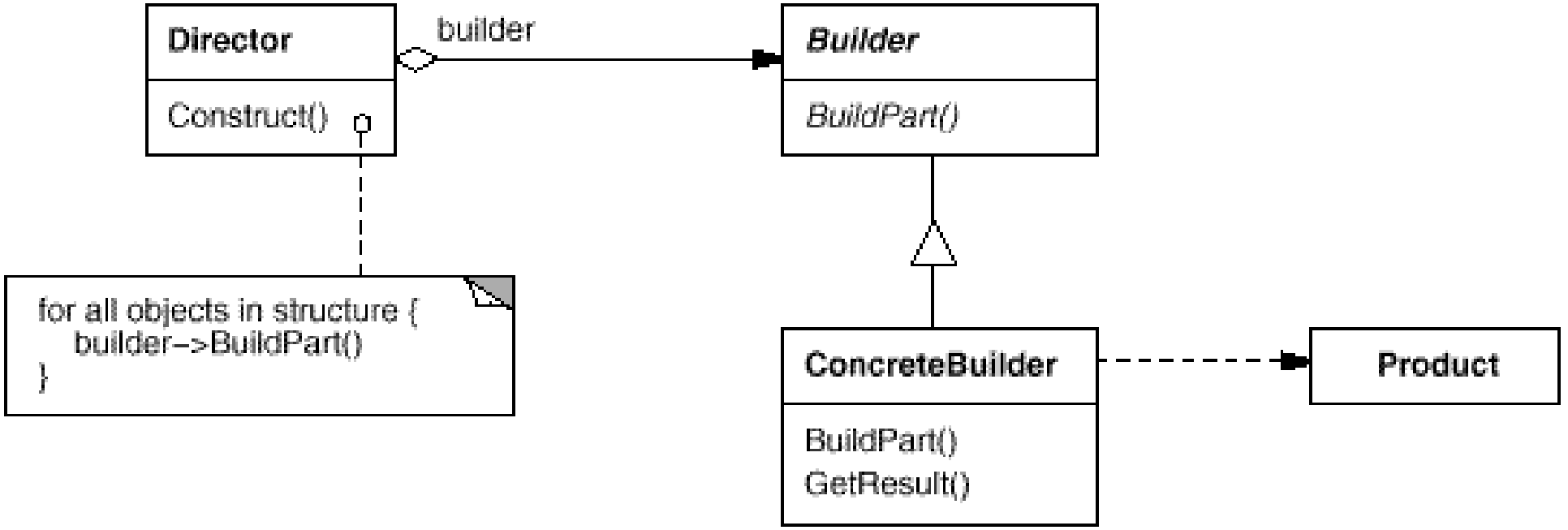
Proxy – charakteristiky

- Aplikovatelnost
  - **Vzdálená proxy** představuje lokální kopii objektu.
  - **Virtuální proxy** vytváří „drahé“ objekty až když jsou potřeba.
  - **Ochranná proxy** kontroluje přístup k objektu.
  - **Chytrý odkaz** nahrazuje holý ukazatel na objekt a provádí dodatečné akce při přístupu k objektu (např. počítání referencí na objekt z důvodů autodestrukce objektu).

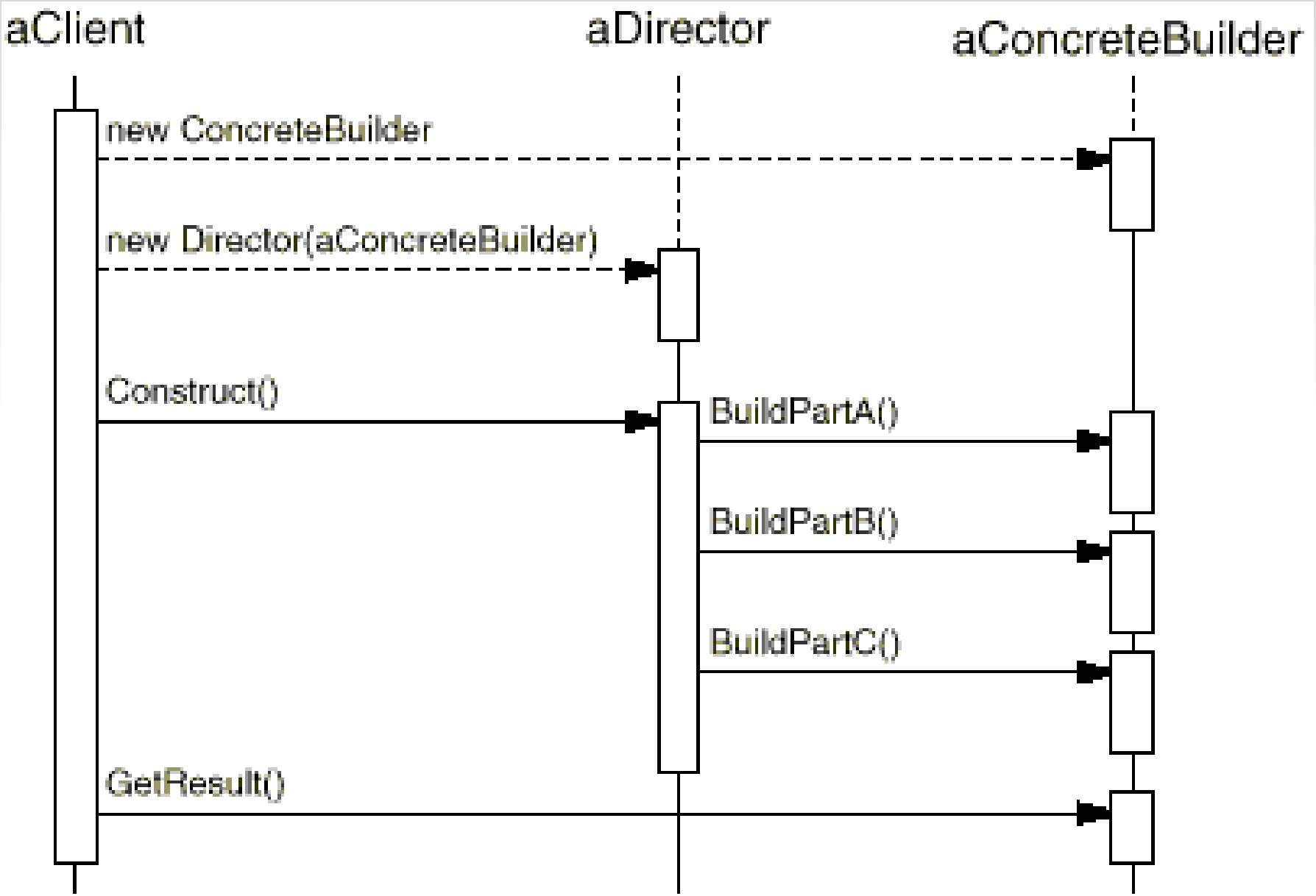
Builder - motivace



Builder - vzor



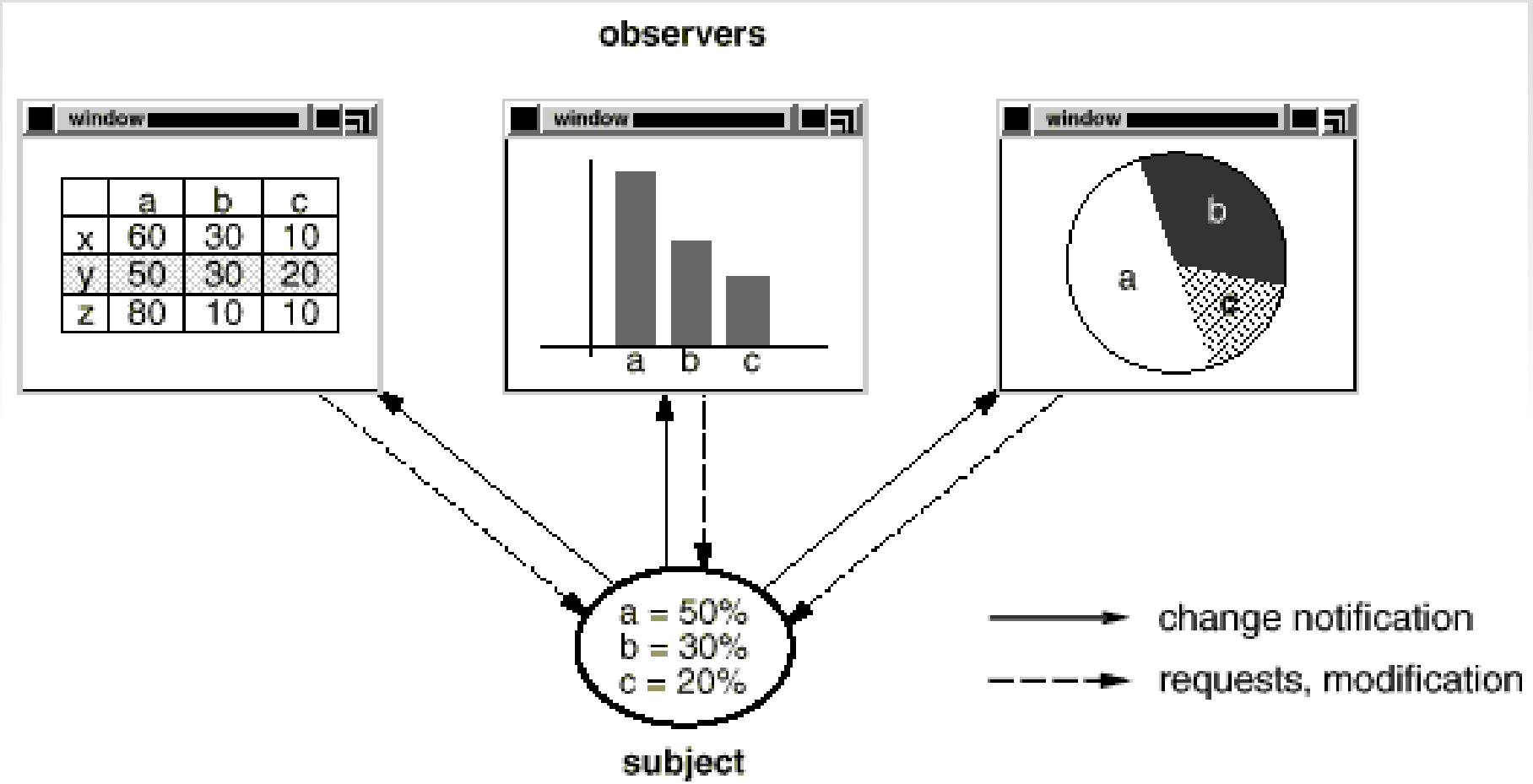
Builder - spolupráce



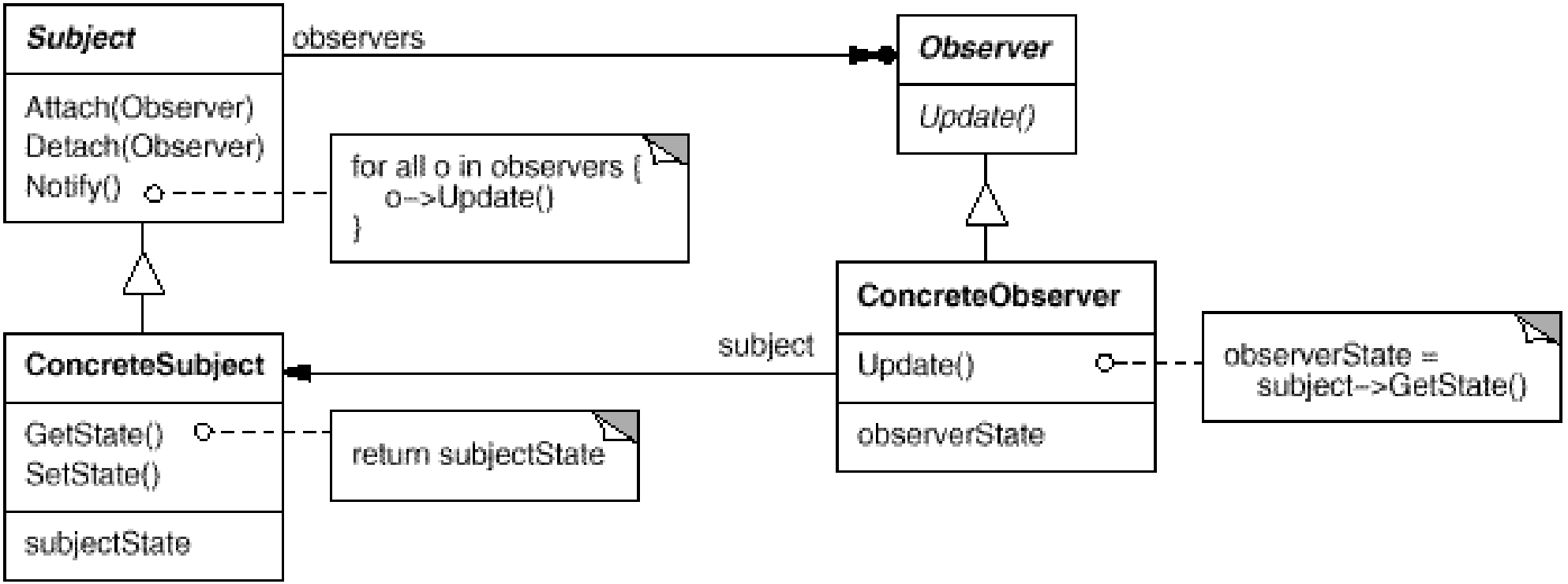
Builder - charakteristiky

- Aplikovatelnost
  - Algoritmus pro tvorbu složitého objektu by neměl záviset na částech, které tvoří objekt, a na tom, jak jsou sestaveny
  - Konstrukční proces musí umožnit různé reprezentace konstruovaného objektu
- Důsledky
  - Je možné měnit interní reprezentaci produktu
  - Kód pro konstrukci a reprezentaci produktu je izolován
    - => *SGMLReader* používající konvertory z příkladu pro převod SGML dokumentů
  - Poskytuje jemnější kontrolu konstrukčního procesu
    - => vhodné zejména pro produkty se složitější strukturou

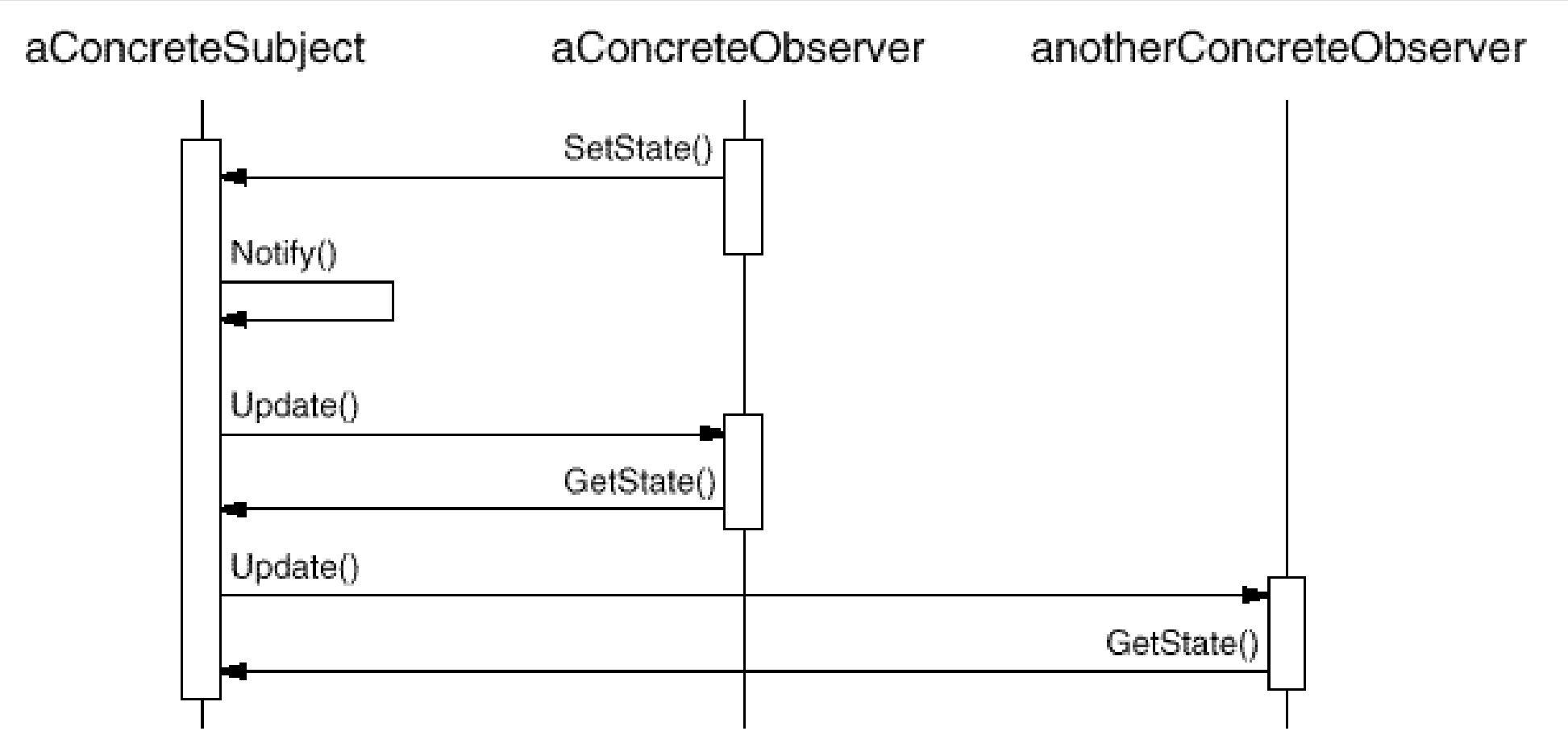
Observer - motivace



Observer - vzor



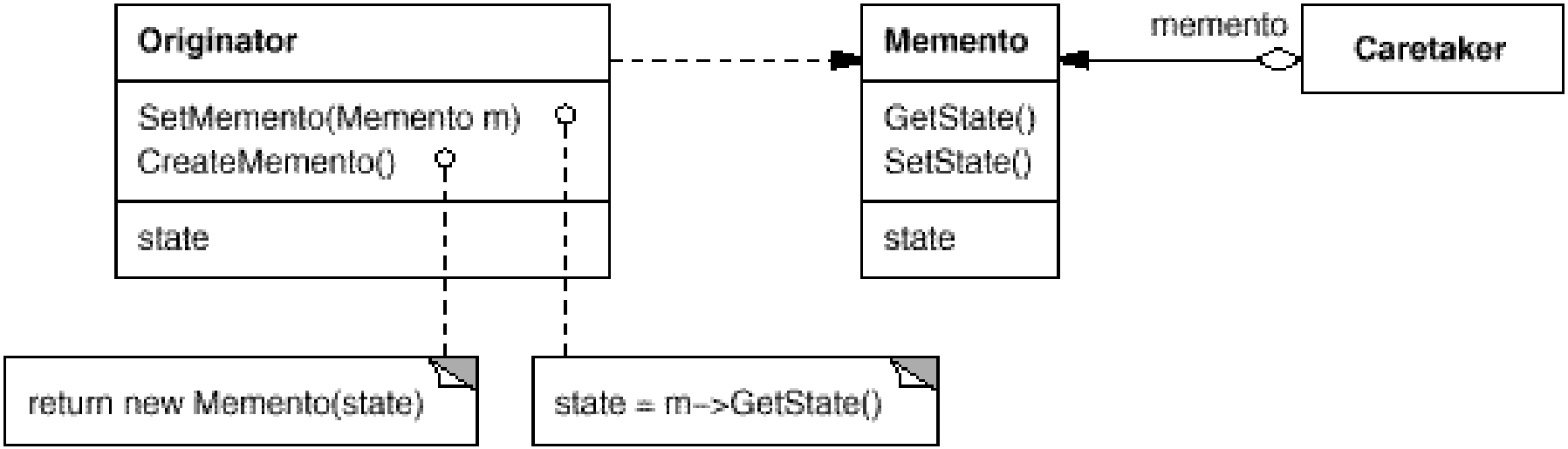
Observer - spolupráce



Observer - charakteristiky

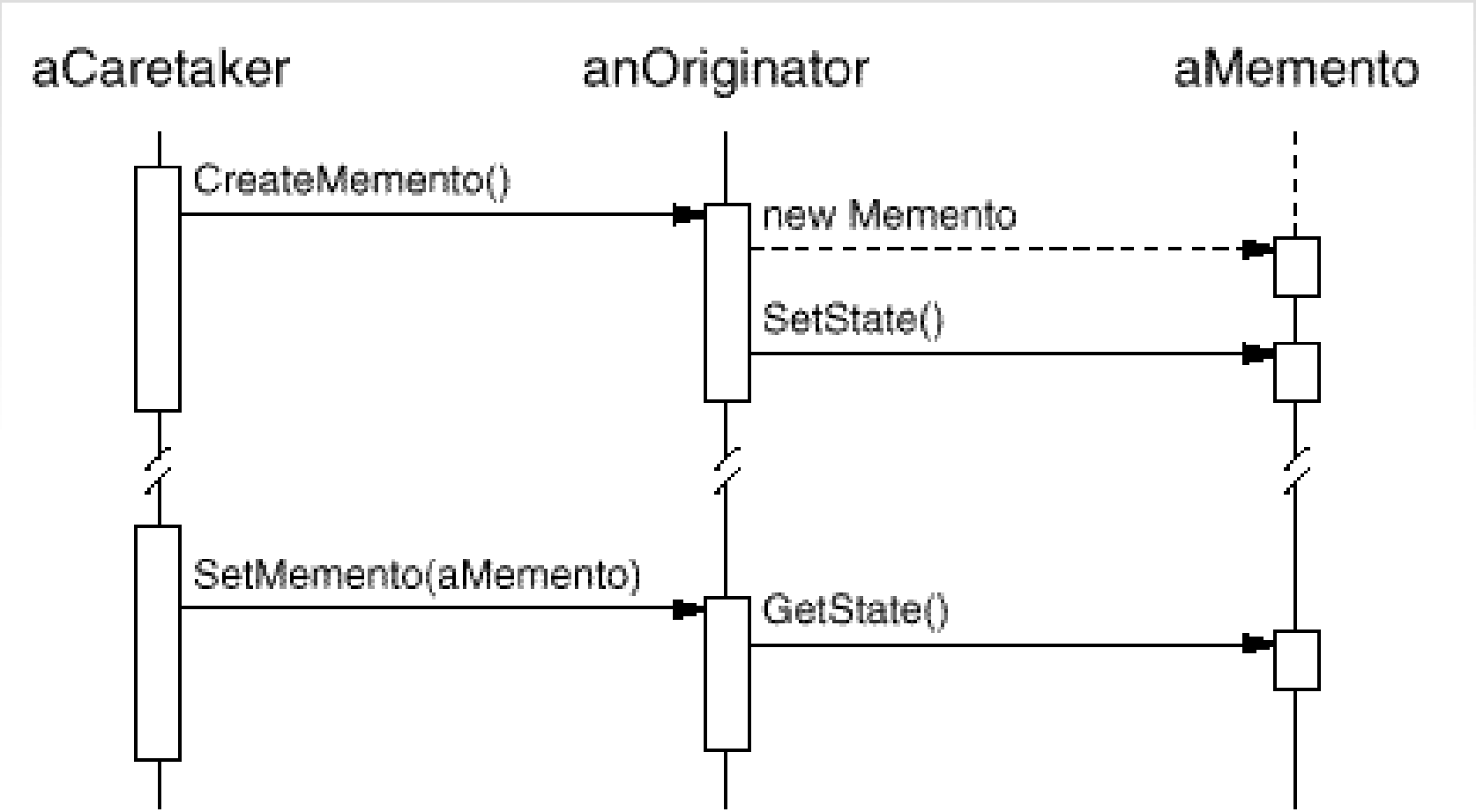
- Aplikovatelnost
  - Pokud má abstrakce dva aspekty, které na sobě závisí. Zapouzdření těchto aspektů do separátních tříd je umožňuje měnit a znovupoužít nezávisle.
  - Pokud změna v jednom objektu vyžaduje změnu v dalších objektech a přitom předem nevíme, kolik objektů bude potřeba změnit.
  - Pokud je objekt schopen informovat jiné objekty o svých změnách bez toho, aby něco o těchto objektech musel předpokládat.
- Důsledky
  - Abstraktní spojení mezi subjektem a pozorovatelem. Subjekt má jen seznam pozorovatelů splňujících základní rozhraní třídy Observer, nic víc o konkrétních pozorovatelích neví.
  - Podpora násobné komunikace (broadcast communication).
  - Neočekávané updaty. Protože pozorovatel neví nic o dalších pozorovatelích, neví nic ani o dopadech updatu vyvolaného na subjektu. To může vést k časově náročným operacím.

Memento - vzor



zachycení určitého stavu objektu pro pozdější použití (např. obnovení)

Memento - spolupráce



Memento - charakteristiky

- Aplikovatelnost
  - snímek (části) objektu musí být uchován pro pozdější obnovení stavu
  - přímé rozhraní pro získání stavu by odhalilo implementační detaily a porušilo zapouzdření objektu
- Důsledky
  - Zachování hranic zapouzdření
  - Zjednodušení „původce“ mementa.
  - Použití mement může být nákladné.
  - Definice úzkých a širokých rozhraní (jazykové problémy omezení přístupu k mementu výhradně pro původce).
  - Skryté náklady pro zpracování mement.