

Návrhové vzory

Filip Zavoral

www.ksi.mff.cuni.cz/teaching/nprg024-web

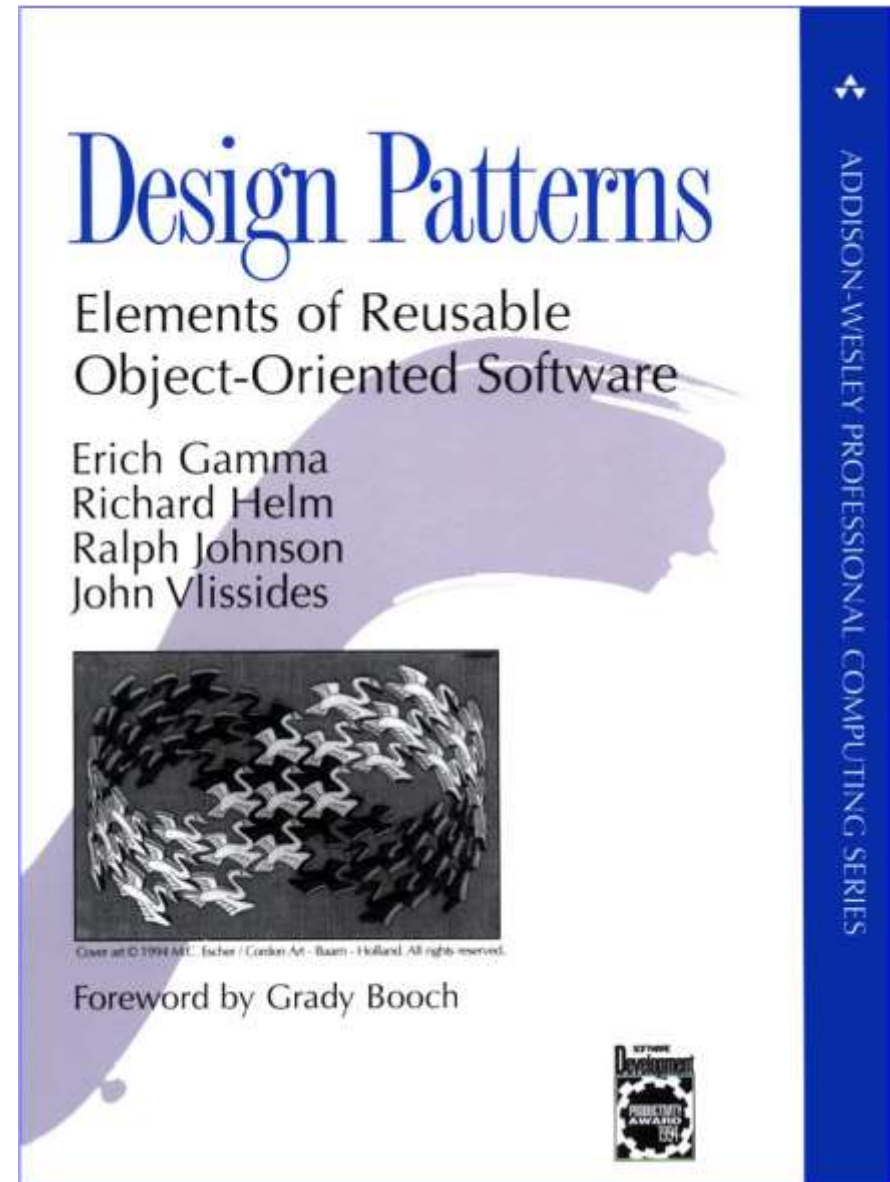


Literatura

- **E. Gamma, R. Helm, R. Johnson, J. Vlissides**
The Gang of Four (GoF)
Design Patterns
Elements of Reusable
Object-Oriented Software
1995

Grada 2003:

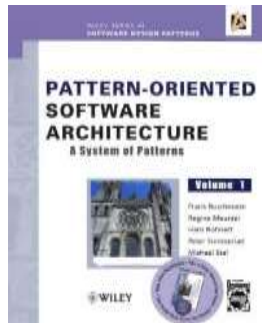
Návrh programů podle vzorů





Literatura

- Design Patterns in *C#, Java, VisualBasic, PHP, ..., ...*
- Pattern Oriented Software Architecture
 - Vol 1: A System of Patterns
 - Vol 2: Patterns for Concurrent and Networked Objects
 - Vol 3: Patterns For Resource Management
 - Vol 4: A Pattern Language for Distributed Computing
 - Vol 5: On Patterns and Pattern Languages



- wikipedia / google: design patterns



Průběh semestru

■ Úvod

- OOP a návrhové vzory
- Podmínky, slajdy, průběh semestru



- Zpracování konkrétních NV, slajdy
- Prezentace
- Interaktivní feedback, hodnocení

■ Shrnutí jednotlivých kategorií NV

■ Rozšiřující NV

- Závěrečný test
- Implementace



4 b.	příprava, slajdy
8 b.	prezentace
2 b.	implementace
10 b.	test

1	22 b.
2	19 b.
3	16 b.



Návrhové vzory

■ OO jazyky - široká paleta technických prostředků

- dědičnost, polymorfismus, šablony, reference, přetěžování, ...
- problém - jak toto všechno efektivně používat
- cíl - udržitelný a rozšiřovatelný 'velký' software
 - rozhraní !!
 - volnější vazby, parametrizace
 - dědičnost x delegace x reference x politiky

■ Návrhový vzor

- pojmenované a popsané řešení typického problému
- principiálně existují již dlouho
 - architektura: 1977 Christopher Alexander - pojem 'Pattern'
 - literatura: tragický hrdina, romantická telenovela, ...



■ Software

- *žádný jiný obor si tolik nelibuje ve vynalézání kola ... stále dokola*



Definice a použití

Návrhový vzor je popis komunikujících tříd a objektů
uzpůsobených k řešení **obecného** problému v **konkrétním** kontextu

■ Relativní komplexnost a obecnost

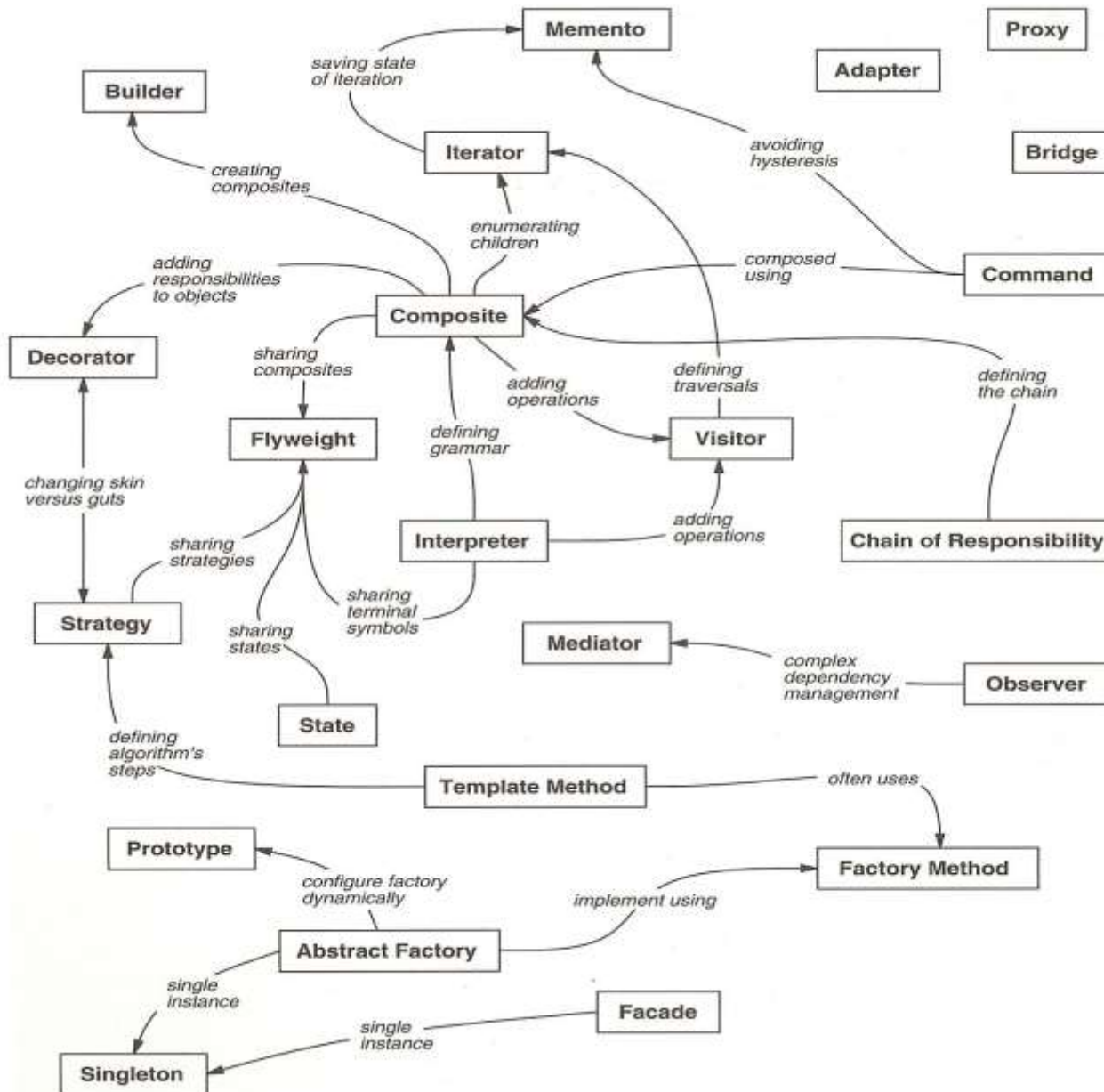
- pro rozsáhlejší systémy
 - předpoklad dlouhé životnosti, údržby a rozšiřování
- při návrhu nových systémů
- při rozsáhlých úpravách

■ Základní prvky

Název	usnadnění komunikace - slovník
Problém	situace, kterou má NV řešit, podmínky použití, kontext
Řešení	statická struktura, dynamika chování
Souvislosti, důsledky	použití, implementace, principu fungování, alternativy
Příklady	konkrétní problém, podmínky, popis implementace, hodnocení
Související vzory	řetězec NV, rozhodování mezi různými NV

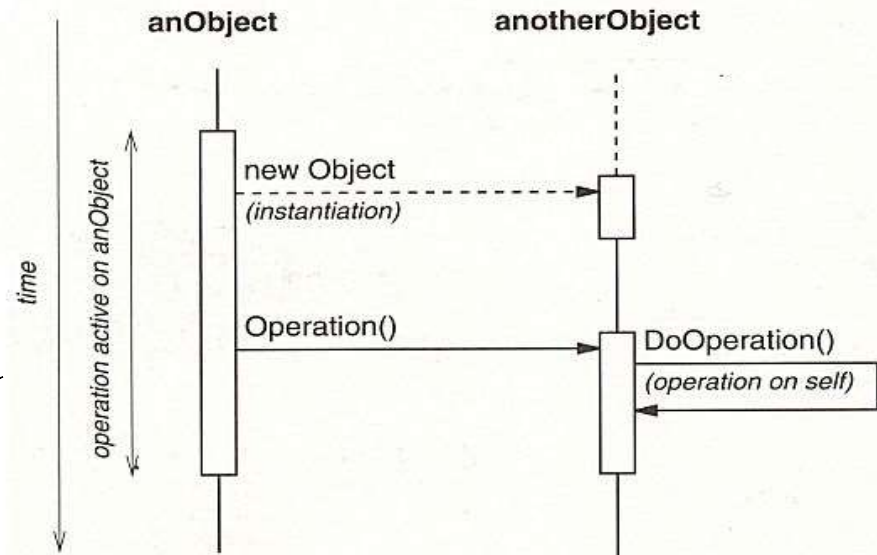
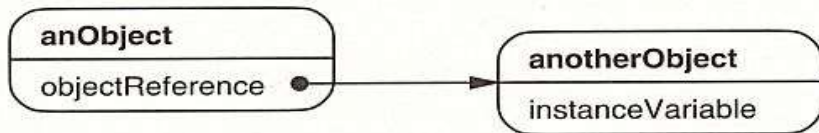
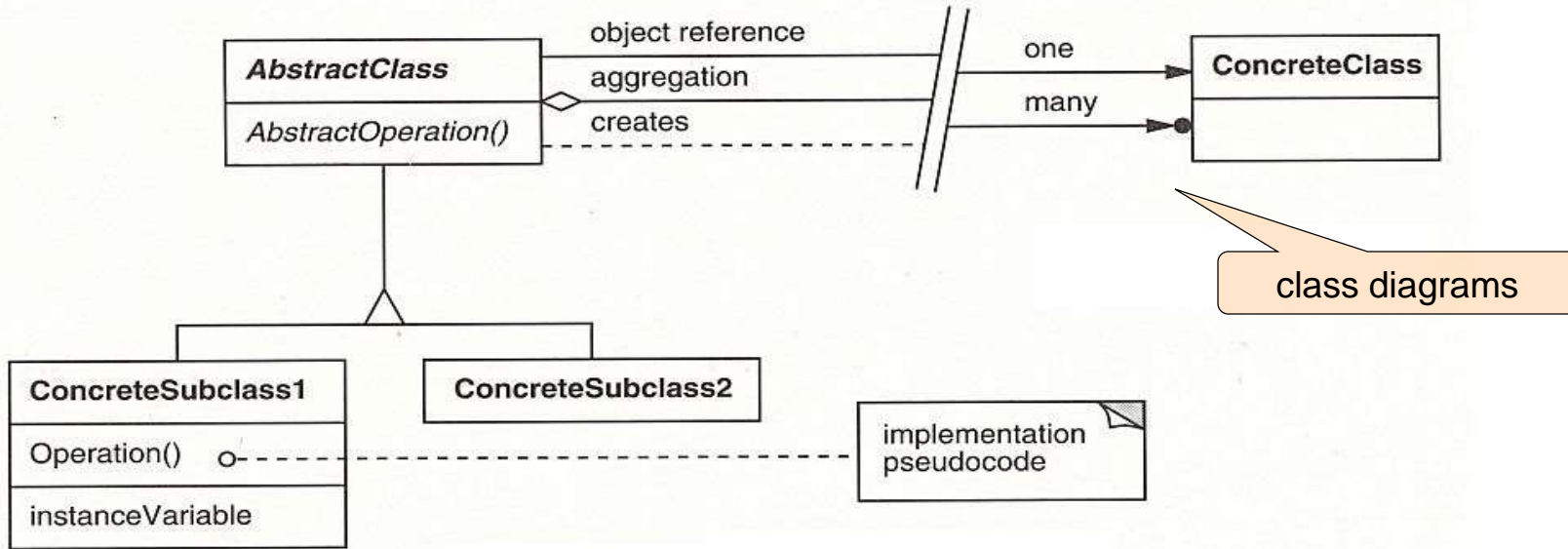


Vztahy mezi NV





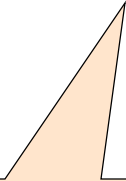
Značení – Object Modeling Technique



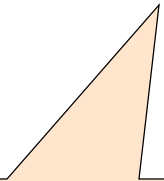


Kategorie základních NV

	Creational <i>Tvořivé vzory</i>	Structural <i>Strukturální vzory</i>	Behavioral <i>Vzory chování</i>
Třída	Factory Method	Adapter	Interpreter Template Method
Objekt	Abstract Factory Builder Prototype Singleton	Bridge Composite Decorator Facade Proxy Flyweight	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor



vytváření objektů



uspořádání tříd a objektů



chování a interakce objektů a tříd



Zpracování NV

■ Stáhněte si loňské slajdy a ostatní materiály

- ❑ slajdy na stránkách předmětu
- ❑ na odkazu *Vzkaz přihlášeným studentům* další materiály a starší prezentace

■ Pokuste se příslušný NV pochopit nejdřív jen ze slajdů

- ❑ jestli vám to nepůjde (*skoro určitě*), není závada na vaší straně
- ❑ slajdy **nejsou** učebnice
- ❑ video z minulých let nestačí

■ Pořádně NV nastudujte

- ❑ nejdřív z GoF - *ignorujte Smalltalk*
- ❑ doplňkové informace z ostatních materiálů
 - *GoF cca 30 let starý!*

■ Praktické použití

- ❑ určitě **nepřebírejte** ty z GoF
 - *velmi zastaralé!*
- ❑ detailně se seznamte s nějakým existujícím použitím
 - **kde, jak** - příklady, fragmenty kódu
 - lze ukázat i vhodné použití ve vlastních projektech

■ Upravte slajdy



Výroba slajdů

■ Hloubka detailu alespoň na úrovni GoF

- ❑ mnohé internetové zdroje popisují NV velmi stručně, typicky bez souvislostí
- ❑ **aktuální** praktické použití

■ Kvalitní slajdy

- ❑ slajdy nejsou kniha - **heslovitě** základní myšlenky, omáčka ústně
 - žádná dlouhá souvětí nebo odstavce, vyhoďte všechna ne-nutná slova
- ❑ **nepoužívejte** **světlé písmo na tmavém pozadí** - hůře čitelné (i pro zdrojáky)
- ❑ zdrojáky nikdy jako obrázek (screenshot)!
 - ne omáčku, jen to podstatné, vhodné callouty na důležité části
- ❑ vhodné postupné 'rozbalování' slajdu (animace)
- ❑ slajdy česky nebo anglicky (**ne slovensky**) v pptx (**ne pdf, odt, ...**)

■ Dodržovat harmonogram!

- ❑ s přípravou začněte **včas**
 - studium a výroba kvalitních slajdů a přednášky trvá dlouho (> **týden!**)

■ Upravte slajdy

- ❑ jako základ lze použít loňské (na stránce předmětu)
- ❑ aktualizujte, doplňte, zkrášte, opravte formátování, design, diakritiku, chyby ...
- ❑ vyzkoušejte zobrazení v **originál Microsoft PowerPointu** !



Prezentace

■ Přednáška - není to 'referativní seminář'

- cílem není 'odvykládat referát', ale **srozumitelně vysvětlit** ostatním NV
- musíte látce rozumět **hlouběji** než o čem budete povídat
 - nemůžete říct '*nevím přesně, co tohle znamená*', '*tomuhle nerozumím*', ...

■ Kvalitní přednášení

- předem si rozmyslete, o čem, co a jak budete povídat
- **nemumlejte, nedrmolte**, přednášejte **nahlas** a **srozumitelně artikulujte**
- přednášejte směrem **k posluchačům**, dívejte se na ně - oční kontakt
- vyndejte ruce z kapes, vyvarujte se výplňových slov a zvuků
 - '*budu povídat eeéé o vzoru eeéé Factory*'

■ Slajdy

- pomůcka pro vás i pro posluchače
 - lze se na ně rychle 'kouknout', ne ale číst celé odstavce
- posluchači umí číst - **neříkejte** jen to, co na slajdech **je**, ale hlavně to, co tam **není**
- zdůrazněte a vysvětlete důležité, klidně přeskočte nedůležité detaily
 - nemusíte předčítat vše, co je na slajdech napsané, např. když jste to už říkali
- komentujte význam fragmentů kódu

■ Hlídejte si čas (20 min) !

Nedodržení pravidel ⇒ 😞



Za co zápočet, hodnocení

■ [4 b.] Příprava slajdů

- být přiřazen v Grupíčku ke konkrétnímu vzoru a termínu
- nastudovat vzor (≠ nabílovat nazpaměť slova na slajdech)
 - vzoru **detailně rozumět** - hlouběji, než budete přednášet
- vyrobit / upravit a **včas** uploadovat slajdy

■ [8 b.] Prezentace

- kvalitně odpřednášet (nikoliv referativně *odmrmlat*)
 - 15-20 minut ve stanoveném **termínu**
- kvalifikovaně zodpovídat dotazy

■ Účast na ostatních přednáškách

- $\geq 50\%$, aktivní zapojení do diskuse
- slovní i 'známkové' hodnocení ostatních

■ [2 b.] Implementace alespoň 1 vzoru

- ... různého od základní verze Singletonu
- v libovolném jazyce / programu, lze i v již hotových projektech
 - reálné použití, nikoliv DÚ na procvičení konkrétního vzoru

■ [10 b.] Zápočtový test

všechny části
povinné




Přiřazení NV a preference termínu

■ Úvodní handshake

- do 26.2. (ale raději hned) do Grupíčku povrzení zájmu
 - "Ano"
 - lze případná časová omezení / preference
 - *"nemůžu 15.4., raději bych dříve, nejdříve od ..."*
- přiřazené vzory a termíny budou zveřejněné v Grupíčku

■ Upload slajdů

- do Grupíčku do  **pondělního rána** (9:00) před termínem
 - v případě ohrožení termínu uploadu alespoň zprávu/e-mail
 - rozhodně nestačí *'jsem myslel, že to přinesu až na vlastní přednášku'* ☠
- úpravy / opravy připomínek upload nejpozději ráno (9:00) před přednáškou



Tvořivé NV

■ Creational Patterns

■ Abstrakce procesu vytváření objektů

- umožňují ovlivnit způsob vytváření objektů a jejich typ a počet
- nevhodné použití *new* - např. typ objektu závisí na parametrech
- větší flexibilita **co** se vytváří, **kdo** to vytváří, **jak** a **kdy** se to vytváří

■ Typické prostředky

- zapouzdření použití konkrétní třídy
- zakrytí vzniku a skládání objektů

■ Tvořivé vzory

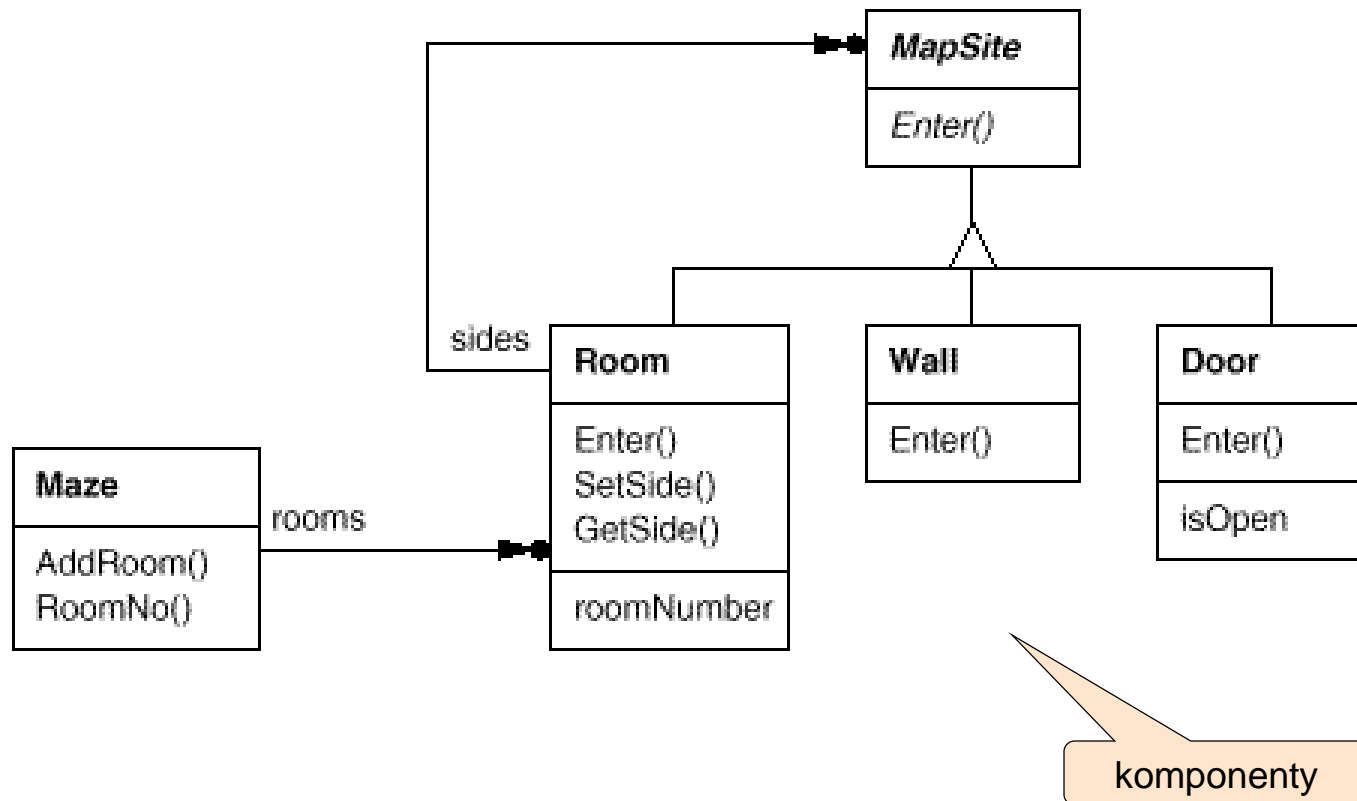
- **Factory Method** - vytváří instance vybrané třídy - virtuální funkce místo *new*
- **Abstract Factory** - vytváří objekty pro vybranou skupinu tříd - tovární třída
- **Builder** - odděluje způsob vytvoření objektu od reprezentace, postupné vytváření
- **Prototype** - umožňuje zkopírovat (klonovat) inicializovanou instanci
- **Singleton** - zaručí pouze jednu instance třídy



Bludiště

■ Jednotný příklad pro tvořivé NV - vytvoření bludiště

- množina místností, místnost zná své sousedy - zeď, jiná místnost nebo dveře





Bludiště – prvotní implementace

```
enum Direction {North, South, East, West};

class MapSite {
public:
    virtual void Enter() = 0;
};

class Room : MapSite {
public:
    Room(int roomNo);

    MapSite GetSide(Direction);
    void SetSide(Direction, MapSite);

    virtual void Enter();
private:
    MapSite sides[4];
    int roomNumber;
};
```

Enter - soused je:
místnost nebo otevřené dveře → projít 😊
zeď nebo zavřené dveře → stát ☹

```
class Wall : MapSite {
public:
    Wall();
    virtual void Enter();
};

class Door : MapSite {
public:
    virtual void Enter();
    Room OtherSideFrom(Room);
private:
    Room room1;
    Room room2;
    bool isOpen;
};

class Maze {
public:
    Maze();
    void AddRoom(Room);
    Room RoomNo(int);
private: // ...
};
```



Bludiště – vytvoření

```
Maze MazeGame::CreateMaze () {  
    Maze aMaze = new Maze;  
    Room r1 = new Room(1);  
    Room r2 = new Room(2);  
    Door theDoor = new Door(r1, r2);  
  
    aMaze.AddRoom(r1);  
    aMaze.AddRoom(r2);  
  
    r1.SetSide(North, new Wall);  
    r1.SetSide(East, theDoor);  
    r1.SetSide(South, new Wall);  
    r1.SetSide(West, new Wall);  
  
    r2.SetSide(North, new Wall);  
    r2.SetSide(East, new Wall);  
    r2.SetSide(South, new Wall);  
    r2.SetSide(West, theDoor);  
  
    return aMaze;  
}
```

vytvoření bludiště se 2 místnostmi

místnosti a dveře mezi nimi

hranice místností

- poměrně komplikované
- neflexibilní !!
 - změna tvaru - změna metody
 - změna chování - nutnost přepsání
 - DoorNeedingSpell, SpecialRoom
- hard coded 💣



Možná vylepšení pomocí tvořivých NV

Zvýšení flexibility - odstranění explicitních referencí na konkrétní třídy

■ Factory Method

- ❑ `CreateMaze` při vytváření komponent volá virtuální funkci místo konstruktoru
- ❑ potomek `MazeGame` může změnou virtuální funkce vytvářet instance jiných tříd

■ Abstract Factory

- ❑ `CreateMaze` dostane parametr objekt pro vytváření komponent
- ❑ možnost změny instanciovanych tříd předáním jiného parametru

■ Builder

- ❑ `CreateMaze` dostane parametr objekt s operacemi pro přidávání komponent
- ❑ pomocí dědičnost lze změnit jednotlivé vytvářené části nebo způsob vytváření

■ Prototype

- ❑ `CreateMaze` dostane parametry prototypy objektů které se umí klonovat
- ❑ možnost změny předáním jiných (podděných) parametrů

■ Singleton

- ❑ zaručí jedinečnost instance bludiště a přístup k ní bez potřeby globálních dat



Tvořivé NV – shrnutí

■ Singleton

- jednoduché použití ...
 - pokud není nutné řešit závislosti, zámky, destrukci objektů, ...
- různé varianty pro vzájemně provázané objekty
 - komplikované a spíš nevhodné použití

■ Factory Method

- flexibilita za relativně malou cenu
 - obvyklá základní metoda, virtual constructor

■ Abstract Factory, Builder, Prototype

- flexibilnější, složitější
 - použít až při zjištění potřeby větší flexibility
- Abstract Factory
 - konzistentní objekty více souvisejících tříd
- Builder
 - postupné skládání, izolace od vnitřní reprezentace
- Prototype
 - nové objekty se vytvářejí kopírováním vzorových objektů



Tvořivé NV – srovnání

```
Maze* CreateMaze() {  
    Maze* aMaze = MakeMaze();  
    Room* r1 = MakeRoom(1);  
    Room* r2 = MakeRoom(2);  
    Door* theDoor = MakeDoor(r1, r2);  
}
```

```
Maze* CreateMaze( MazeFactory& f) {  
    Maze* aMaze = f.MakeMaze();  
    Room* r1 = f.MakeRoom(1);  
    Room* r2 = f.MakeRoom(2);  
    Door* door = f.MakeDoor(r1, r2);  
}
```

Factory Method

Abstract Factory

```
Maze* CreateMaze( MazeBuilder& builder) {  
    return builder.BuildMaze()  
        .BuildRoom(1)  
        .BuildRoom(2)  
        .BuildDoor(1, 2)  
        .GetMaze();  
}
```

Builder

Prototype

```
Wall* MazeProtoFactory::MakeWall() {  
    return protoWall->Clone();  
}  
Door* MazeProtoFactory::MakeDoor( Room* r1, Room *r2) {  
    Door* door = protoDoor->Clone();  
    door->Initialize(r1, r2);  
    return door;  
}
```

```
MazeProtoFactory pf { aMaze, aWall, aRoom, aDoor};
```



Strukturální NV

■ Jak jsou třídy a objekty složeny do větších struktur

□ Adapter

- přizpůsobení rozhraní třídy na rozhraní jiné třídy

□ Bridge

- odděluje abstrakci od implementace
- předchází nárůstu počtu tříd při přidávání implementací

□ Facade

- definuje jedno společné rozhraní pro subsystém

□ Proxy

- zástupce objektu, kontrola přístupu k objektu

□ Decorator

- rozšiřuje objekt o nové vlastnosti
- runtime, transparentní - rozšiřovaný objekt nic neví

□ Composite

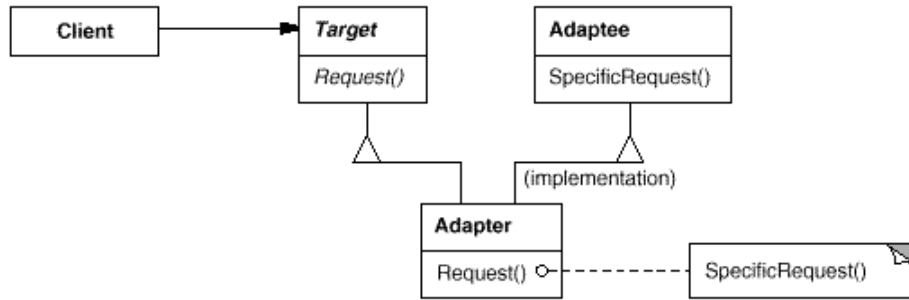
- hierarchie tříd složená z primitivních a složených objektů
- jednotné operace na všech objektech

□ Flyweight

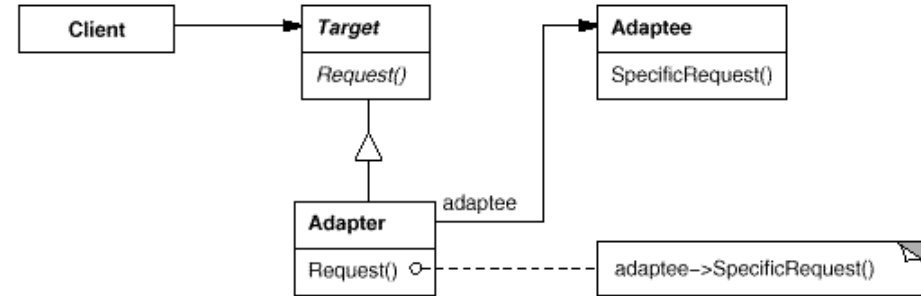
- podpora velkého počtu jednoduchých objektů



Adapter vs. Bridge

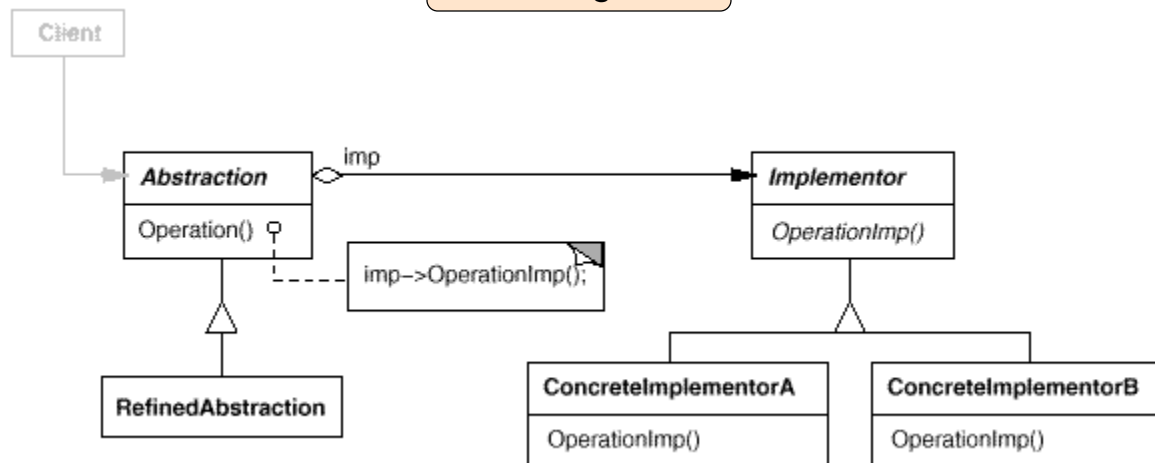


Class Adapter



Object Adapter

Bridge





Adapter vs. Bridge

■ Společné vlastnosti

- flexibilita - stupeň indirekce vůči jiným objektům, zasílání zpráv přes jiné rozhraní

■ Základní rozdíl - účel

□ Adapter

- vyřešení nekompatibilit mezi dvěma existujícími rozhraními
- jak zajistit aby dvě nezávislé třídy mohly spolupracovat bez reimplementace

□ Bridge

- poskytuje relativně stabilní rozhraní pro potenciálně velký počet implementací
- zachovává rozhraní i při dalším vývoji a změně implementačních tříd

■ Důsledek: časté použití při různých fázích vývojového cyklu

□ Adapter

- použití **PO** návrhu
 - pozdější potřeba spolupráce dvou nekompatibilních tříd

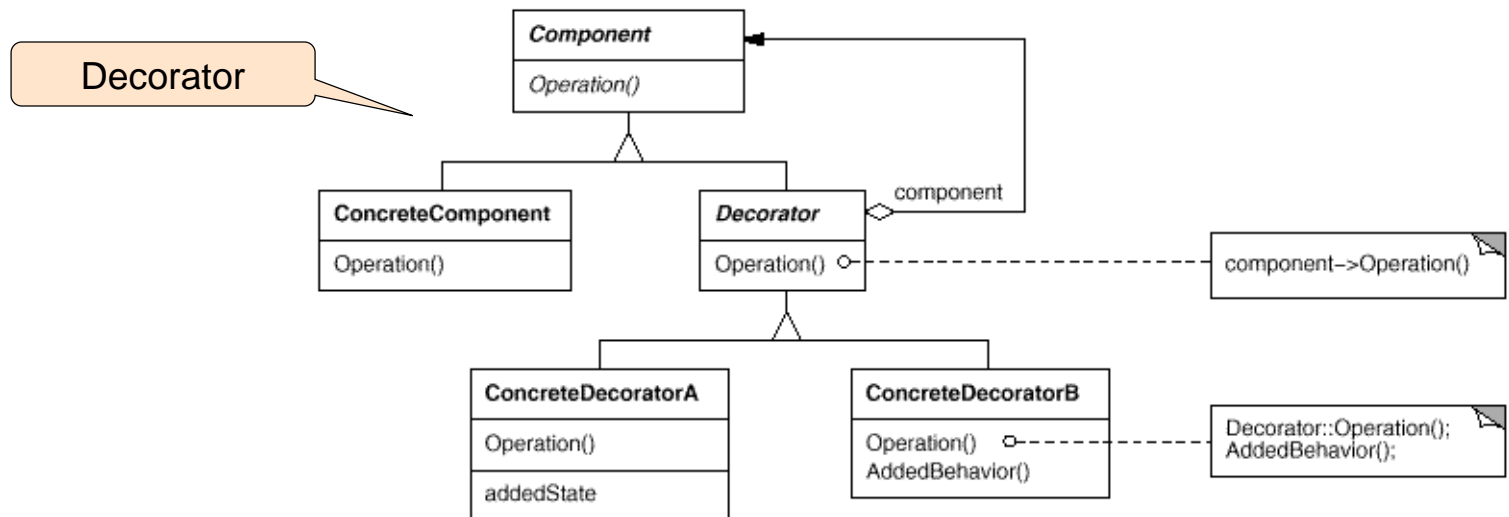
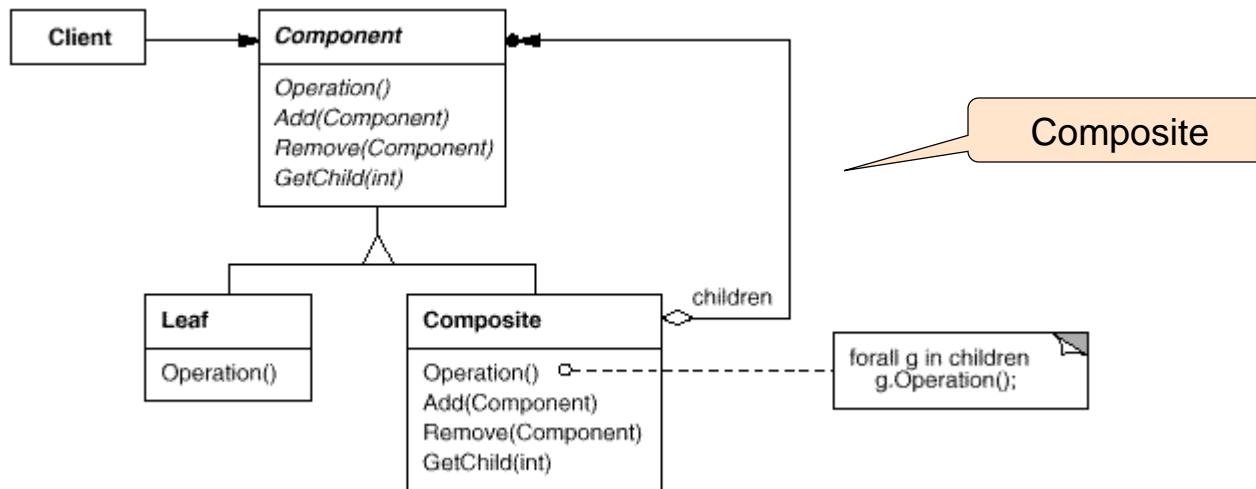
□ Bridge

- použití **PŘI** návrhu
 - abstrakce může mít více implementací, které se můžou dále rozvíjet

- ... řeší jiný problém



Composite vs. Decorator





Composite vs. Decorator

■ Composite a Decorator - podobná struktura

- rekurzivní struktura
- různé účely

- Decorator
 - funkčnost
 - zabraňuje explozi odvozených tříd při přidávání kombinací funkčnosti

- Composite
 - reprezentace
 - objekty různého druhu se zpracovávají jednotně



NV chování (Behaviorální NV)

■ Behavioral design patterns

- rozdělení funkčnosti a zodpovědnosti mezi objekty
- komunikace mezi objekty
- složitější struktura provádění kódu
- abstrakce runtime technických detailů

■ Behavioral class patterns

- použití dědičnosti pro rozložení chování mezi třídy
- **Template method**
 - abstraktní definice algoritmu po jednotlivých krocích
- **Interpreter**
 - reprezentace gramatiky jako hierarchie tříd

■ Behavioral object patterns

- spolupráce mezi skupinami objektů pro dosažení funkčnosti



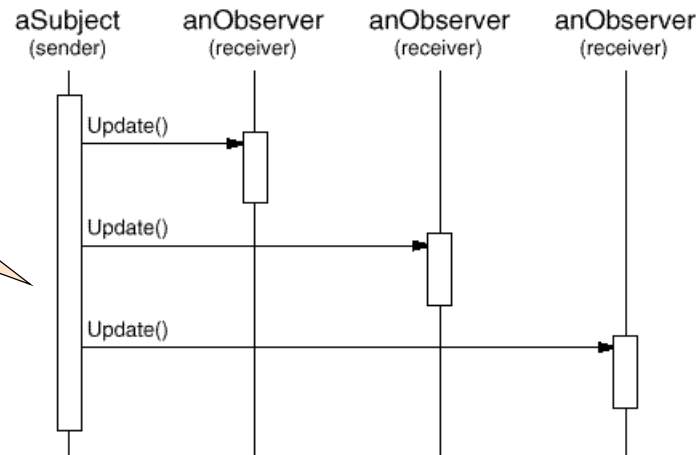
NV chování

- **Behavioral object patterns**
- **Objektová kompozice místo dědičnosti**
 - **Mediator**
 - odstraňuje nutnost referencí na všechny spolupracující objekty
 - **Chain of Responsibility**
 - zasílání zpráv neznámým objektům přes zřetězené objekty
 - **Observer**
 - definování závislosti objektu k více objektům, šíření události k závislým objektům
- **Zapouzdření chování objektu a řízení přístupu**
 - **Strategy**
 - zapouzdření funkčnosti algoritmu do objektu, možnost jejich záměny
 - **Command**
 - zapouzdření požadavku na funkci, oddělení požadavku a vykonání funkce
 - **State**
 - zapouzdření stavu, možnost změny chování objektu při změně stavu
 - **Visitor**
 - zapouzdření chování, které by jinak bylo rozloženo mezi více tříd
 - **Memento**
 - abstrakce uchování obnovitelného stavu
 - **Iterator**
 - abstrakce procházení agregovaných objektů

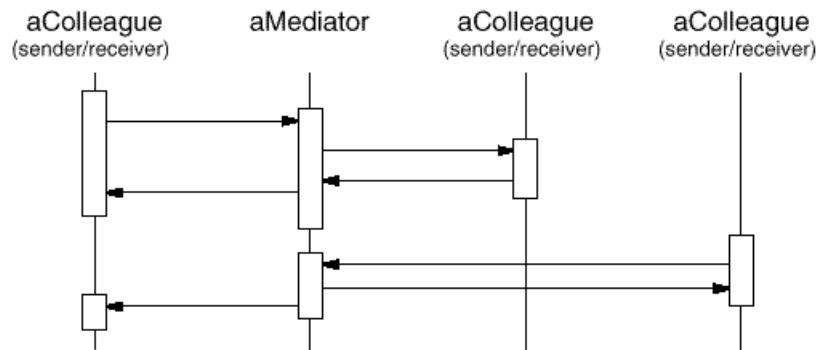


Vztahy mezi odesílateli a příjemci zpráv

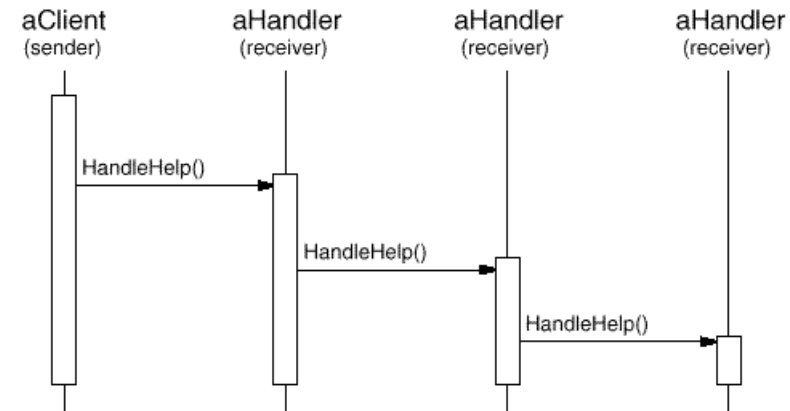
Observer
neznámý počet
příjemců



Mediator
centralizace komunikace
přes prostředníka



Chain of Responsibility
neznámá struktura příjemců i
v okamžiku volání

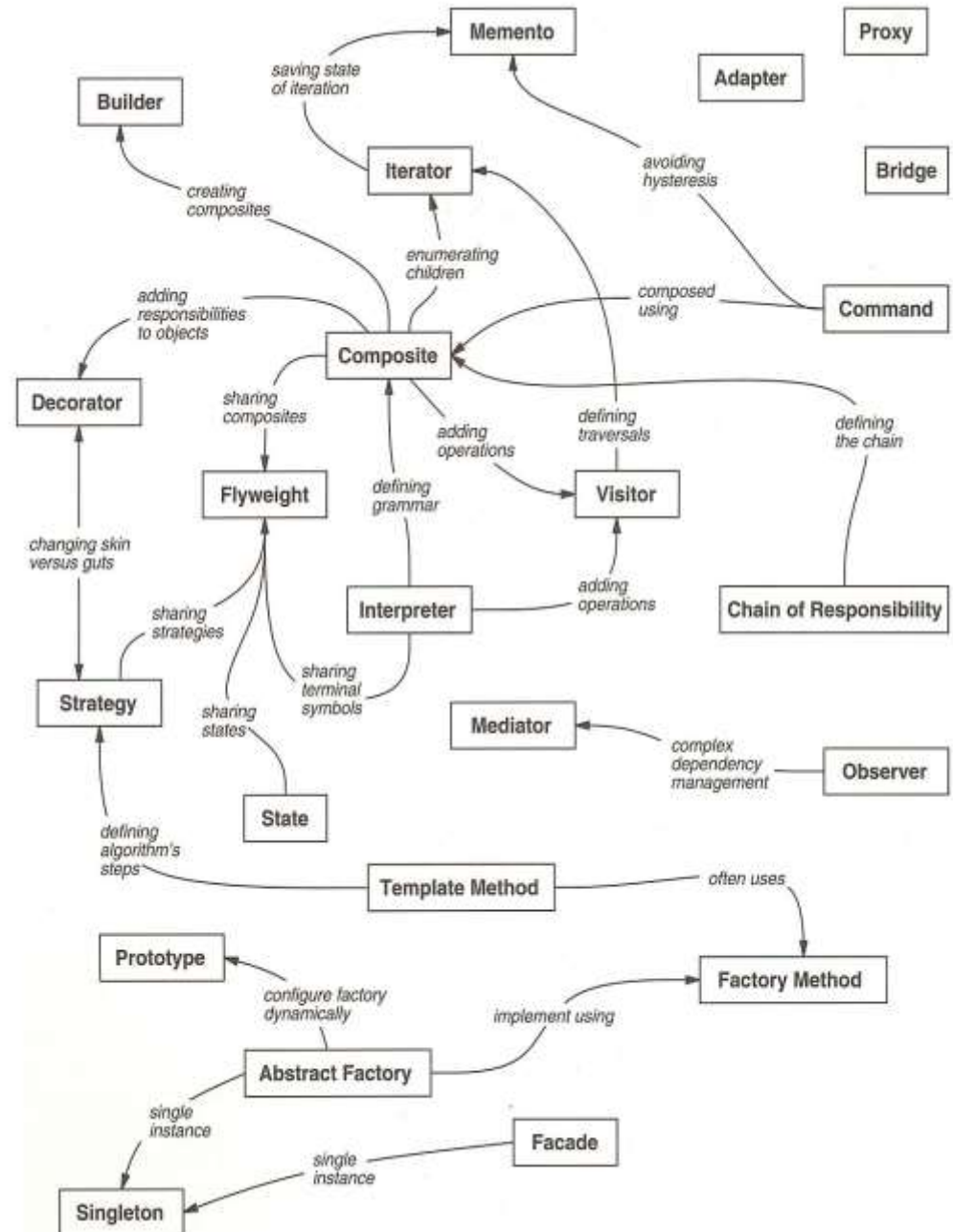




Závěr

■ Shrnutí

- ❑ 'žádná velká věda'
 - ... jak pro koho
- ❑ slovník!
- ❑ implementace bez vymýšlení kola
 - ... a 'bez chyb'
- ❑ kompozice NV
- ❑ mnoho dalších rozšiřujících vzorů
 - často cíleně zaměřených





Pattern Oriented Software Architecture

Vol. 1 - A System of Patterns

2. Architectural Patterns

■ 2.2 From Mud to Structure

- Layers, Pipes and Filters, Blackboard

■ 2.3 Distributed Systems

- Broker

■ 2.4 Interactive Systems

- Model-View-Controller, PAC

■ 2.5 Adaptable Systems

- Microkernel, Reflection

3. Design Patterns

■ 3.2 Structural Decomposition

- Whole-Part

■ 3.3 Organization of Work

- Master-Slave

■ 3.4 Access Control

- Proxy

■ 3.5 Management

- Command Processor, View Handler

■ 3.6 Communication

- Forwarder-Receiver, Client-Dispatcher-Server
- Publisher-Subscriber

Vol. 2 - Patterns for Concurrent and Networked Objects

■ 2. Service Access and Configuration Patterns

- Wrapper Facade
- Component Configurator
- Interceptor
- Extension Interface

■ 3. Event Handling Patterns

- Reactor, Proactor
- Asynchronous Completion Token
- Acceptor-Connector

■ 4. Synchronization Patterns

- Scoped Locking
- Strategized Locking
- Thread-Safe Interface

■ 5. Concurrency Patterns

- Active / Monitor Object
- Half-Sync/Half-Async
- Leader/Followers
- Thread-Specific Storage

Vol. 3 - Patterns for Resource Management

- Lookup
- Lazy / Eager / Partial Acquisition
- Caching / Pooling
- Coordinator
- Leasing / Evictor



Not used

■ Harmonogram závěrečných seminářů

- 25.4. ⊗ studium, otázky
- 2.5. Q&A
- 9.5. písemka
- 16.5. ⊗ (SZZ)
- *další termíny písemky během zkouškového období*