

# Seminář Java

## Návrhové vzory

Radek Kočí

Fakulta informačních technologií VUT

Duben 2008

## Dědičnost

- implementace třídy pomocí jiné (již existující)
- znovupoužitelnost bílé skříňky
- výhody a nevýhody
  - přímočaré použití, jednodušší úprava metod
  - statické
  - těsná vazba s nadřazenou třídou (problémy s modifikací)

## Skládání

- nová funkce = poskládání již existujících objektů
- znovupoužitelnost černé skříňky
- výhody a nevýhody
  - dynamické
  - objekty se používají přes rozhraní
  - objekty lze za běhu zaměňovat (stejně typy)
  - menší, jednodušší a přehlednější návrh

## Skládání

```
class A
{
    foo() { self.m(); }
    m() { print("Object A doing the job"); }
}
```

```
class B
{
    A a;
    foo() { a.foo(); }
    m() { print("Object B doing the job"); }
}
```

B b;

b.foo() => Object A doing the job.

## Skládání

```
class A
{
    foo() { self.m(); }
    m() { print("Object A doing the job"); }
}
```

```
class B
{
    A a;
    foo() { a.foo(); }
    m() { print("Object B doing the job"); }
}
```

B b;

b.foo() => Object B doing the job.

## Parametrizované typy (generické programování)

- *templates* v C++ (viz Standard Template Library – STL)
- *generics* v Java 5
- definují parametrizované typy
- má význam u staticky typovaných jazyků

```
template <typename T>
T max(T x, T y)
{
    if (x < y)
        return y;
    else
        return x;
}
```

`max(3.0, 5.5);`    => T je typu double

## Objektově orientovaný návrh a programování

- *znovupoužitelnost?*
  - zajištění znovupoužitelnosti  $\Rightarrow$  obecný návrh
  - zajištění aplikovatelnosti na řešený problém  $\Rightarrow$  specifický návrh
  - *spor*
- ... *přesto*
  - proč nevyužít řešení, které již fungovalo
  - taková řešení jsou výsledkem mnoha pokusů a používání
  - $\Rightarrow$  vzory pro řešení stejných typů problémů



## Návrhové vzory

- základní sada řešení důležitých a stále se opakujících návrhů
- usnadňují znovupoužitelnost
- umožňují efektivní návrh (výběr vhodných alternativ, dokumentace, ...)

”Každý vzor popisuje problém, který se v našem prostředí neustále vyskytuje. Potom popisuje jádro řešení daného problému tak, že nám umožňuje toto řešení používat třeba milionkrát, aniž bychom to dělali dvakrát stejným způsobem.” — Christopher Alexander

---

## Návrhový vzor

- nazývá, abstraktňuje a identifikuje klíčové aspekty běžné návrhové struktury
- popisuje komunikující objekty a třídy upravené k řešení obecného návrhového problému
- vzor je šablona pro řešení, nikoli implementace problému!

*Některé vzory si konkurují, některé vzory mohou používat pro svou implementaci jiné vzory*

## Prvky návrhového vzoru

- název
  - krátký popis (identifikace) návrhového problému
- problém
  - popis, kdy se má vzor používat (vysvětlení problému, podmínky pro smysluplé použití vzoru, ...)
- řešení
  - popis prvků návrhu, vztahů, povinností a spolupráce
  - nepopisuje konkrétní návrh, obsahuje abstraktní popis problému a obecné uspořádání prvků pro jeho řešení
- důsledky
  - výsledky a kompromisy (vliv na rozšiřitelnost, přenositelnost, ...)
  - důležité pro hodnocení návrhových alternativ – náklady a výhody použití vzoru

## Vzory se mohou týkat

- tříd
  - zabývají se vztahy mezi třídami a podtřídami (vztah je fixován)
- objektů
  - zabývání se vztahy mezi objekty, jsou dynamičtější

## Základní rozdělení vzorů

- tvořivý
  - zabývá se procesem tvorby objektů
- strukturální
  - zabývá se skladbou tříd či objektů
- chování
  - zabývá se způsoby vzájemné interakce mezi objekty či třídami
  - zabývá se způsoby rozdělení povinností mezi objekty či třídami

## Tvořivý

- Tovární metoda (Factory method)
- Abstraktní továrna (Abstract Factory)
- Jedináček (Singleton)
- Prototyp (Prototype)
- Stavitel (Builder)

## Strukturální

- Adaptér – třída (Adapter)
- Adaptér – objekt (Adapter)
- Dekorátor (Decorator)
- Fasáda (Facade)
- Most (Bridge)
- Muší váha (Flyweight)
- Skladba (Composite)
- Zástupce (Proxy)

## Chování

- Interpret (Interpreter)
- Šablonová metoda
- Iterátor (Iterator)
- Návštěvník (Visitor)
- Obnovitel (Memento)
- Pozorovatel (Observer)
- Prostředník (Mediator)
- Příkaz (Command)
- Řetěz odpovědnosti (Chain of Responsibility)
- Stav (State)
- Strategie (Strategy)

# Jedináček (Singleton)

## Účel

- jedna třída může mít pouze jednu instanci
- tvořivý vzor – objekty

## Motivace

- nutnost mít pouze jednu instanci (např. tiskové fronty)
- při pokusu o vytvoření nové instance se vrátí již existující

## Důsledky

- řízený přístup k jediné instanci
- zdokonalování operací (dědičnost)
- usnadňuje změnu v návrhu (variabilní počet instancí)
- tvárnější než třídní (statické) operace (nelze více než jednu instanci, C++ neumožňuje polymorfní překrytí statických metod, ...)

# Jedináček (Singleton)

## Struktura

<b>Singleton</b>
<u>- <code>uniqueInstance</code> : Singleton</u>
<u>+ <code>instance()</code> : Singleton</u>

```
public class Singleton {
    protected Singleton inst;

    private Singleton() {}

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



## Účel

- vytváření příbuzných nebo závislých objektů bez specifikace konkrétní třídy
- tvořivý vzor – objekty

## Motivace

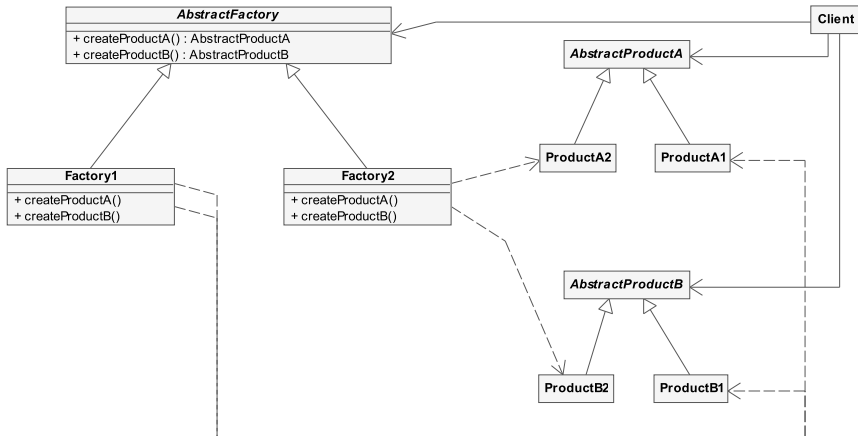
- např. změna vzhledu sady grafických nástrojů

## Důsledky

- izoluje konkrétní třídy – klient pracuje pouze s rozhraním
- usnadňuje výměnu produktových řad (např. změna vzhledu, ...)
- podpora zcela nových produktových řad je obtížnější
- ...

# Abstraktní továrna (Abstract Factory)

## Struktura



# Abstraktní továrna (Abstract Factory)

```
public class MazeFactory {  
    public Wall make Wall() { return new Wall(); }  
}
```

```
public class MazeGame {  
    public Maze createMaze(MazeFactory factory) {  
        Wall wall = factory.makeWall();  
        ...  
    }  
}
```

```
MazeGame game = new MazeGame();  
MazeFactory factory = new MazeFactory();  
game.createMaze(factory);
```

# Abstraktní továrna (Abstract Factory)

```
public class SpecMazeFactory extends MazeFactory {  
    public Wall makeWall() {  
        return new SpecialWall();  
    }  
}
```

```
SpecMazeFactory specFactory = new SpecMazeFactory()  
game.createMaze(specFactory);
```

## Účel

- zapouzdření požadavků nebo operací
- vzor chování

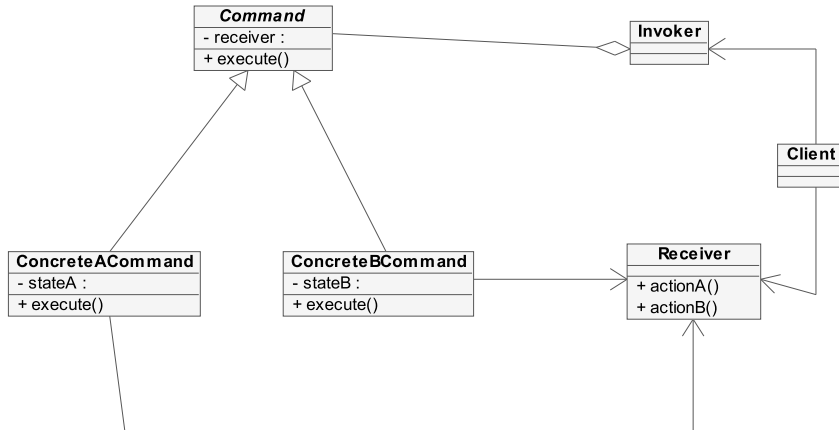
## Motivace

- zaslání požadavku na obecné úrovni, aniž známe konkrétní protokol
- podpora *undo* operací

## Důsledky

- reprezentuje jeden provedený příkaz
- umožňuje uchovávat předchozí stav klienta
- ...

## Struktura





## Zdroje

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Návrh programů pomocí vzorů
  - popis 23 základních vzorů
- [wikipedia.org](http://wikipedia.org)
- [www.patternlanguage.com](http://www.patternlanguage.com)