Seminář Java

Návrhové vzory

Radek Kočí

Fakulta informačních technologií VUT

Duben 2009

Znovupoužitelnost

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)

Znovupoužitelnost

Skládání

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

Znovupoužitelnost

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.
```

Znovupoužitelnost

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.
```

Znovupoužitelnost

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

Znovupoužitelnost

```
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
```

Návrhové vzory (design patterns)

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 (design patterns)

Návrhové vzory

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

Návrhový vzor

- nazývá, zabstraktňuje a identifikuje klíčové aspekty běžné návrhové struktury
- popisuje komunikující objekty a třídy upravené k řešení obecného návrhového problému
- vzor je šablona pro řešení, nikoli implemeptace problému!
  ”Každý vzor popisuje problém, který se neustále vyskytuje, a jádro řešení daného problému. Umožňuje toto řešení používat mnohokrát, aniž bychom to dělali dvakrát úplně stejným způsobem.”

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

Návrhový vzor

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

Typy vzorů

Vzory se mohou týkat

- tříd
  - zabývají se vztahy mezi třídami a podtřídami (vztah je fixován)
- objektů
  - zabývaní 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 třídy
  - zabývá se způsoby rozdělení povinností mezi objekty či třídy

Základní návrhové vzory

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)

Základní návrhové vzory

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

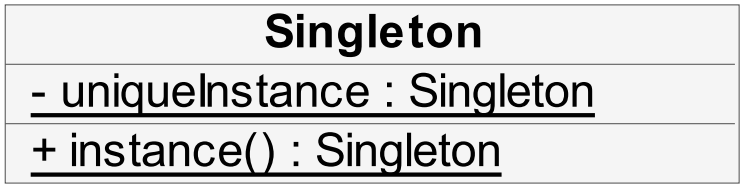
- nutnost mít pouze jednu instanci (např. tiskové fronty)
- při pokusu o vytvoření nové instance se vrátí již existující

Důsledky

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

Jedináček (Singleton)

Struktura




```
public class Singleton {
    protected Singleton inst;

    private Singleton() {}

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

Abstraktní továrna (Abstract Factory)

Účel

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

Motivace

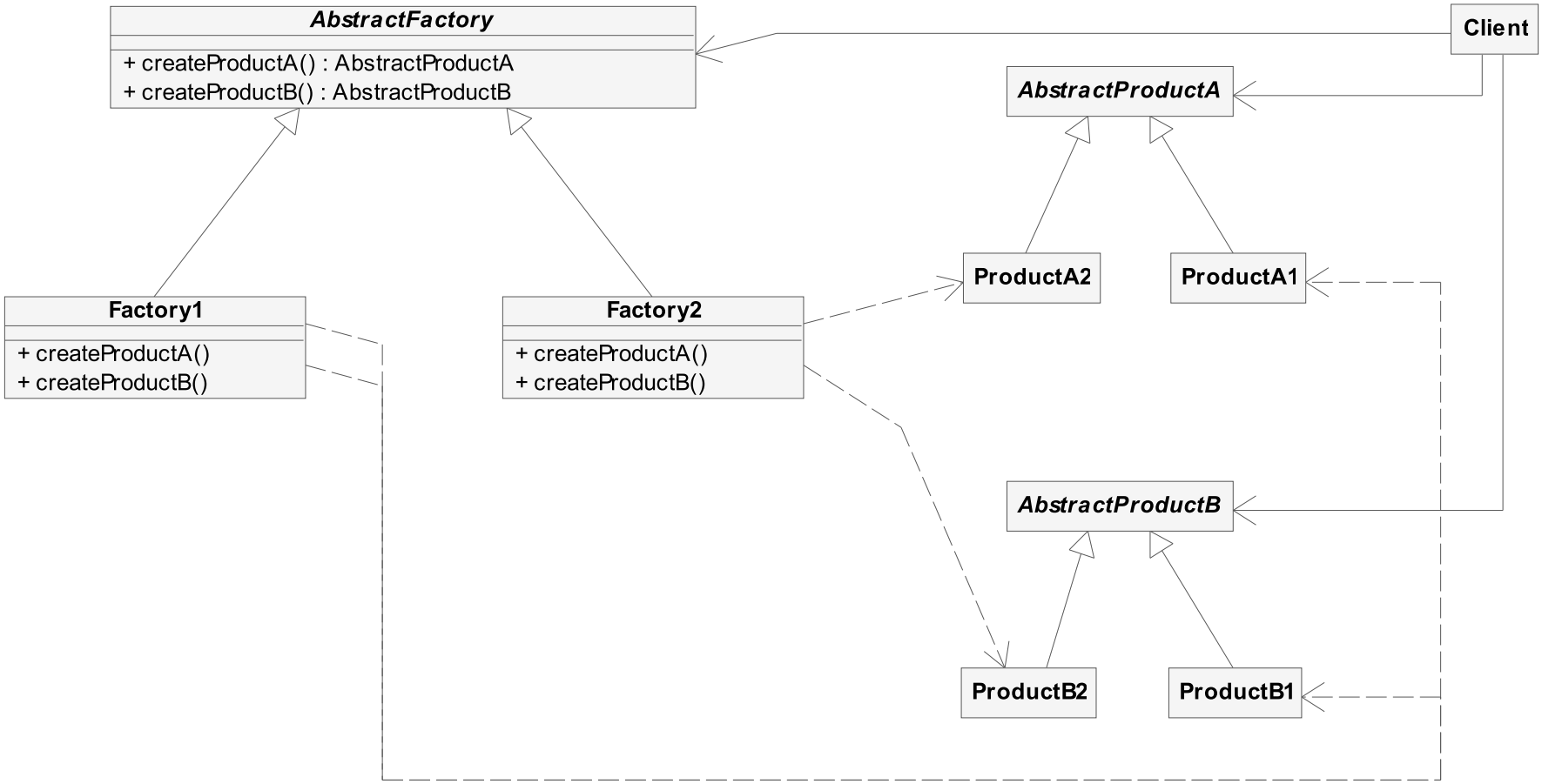
- např. změna vzhledu sady grafických nástrojů

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)

```
// abstract product
public interface Wall { ... }

// abstract factory
public abstract class MazeFactory {
    public abstract Wall makeWall();
}

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

Abstraktní továrna (Abstract Factory)

```
public class StdWall implements Wall { ... }

public class StdMazeFactory extends MazeFactory {
    public Wall makeWall() {
        return new StdWall();
    }
}

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

Abstraktní továrna (Abstract Factory)

```
public class SpecialWall implements Wall { ... }

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

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

Command

Účel

- zapouzdření požadavků nebo operací
- vzor chování

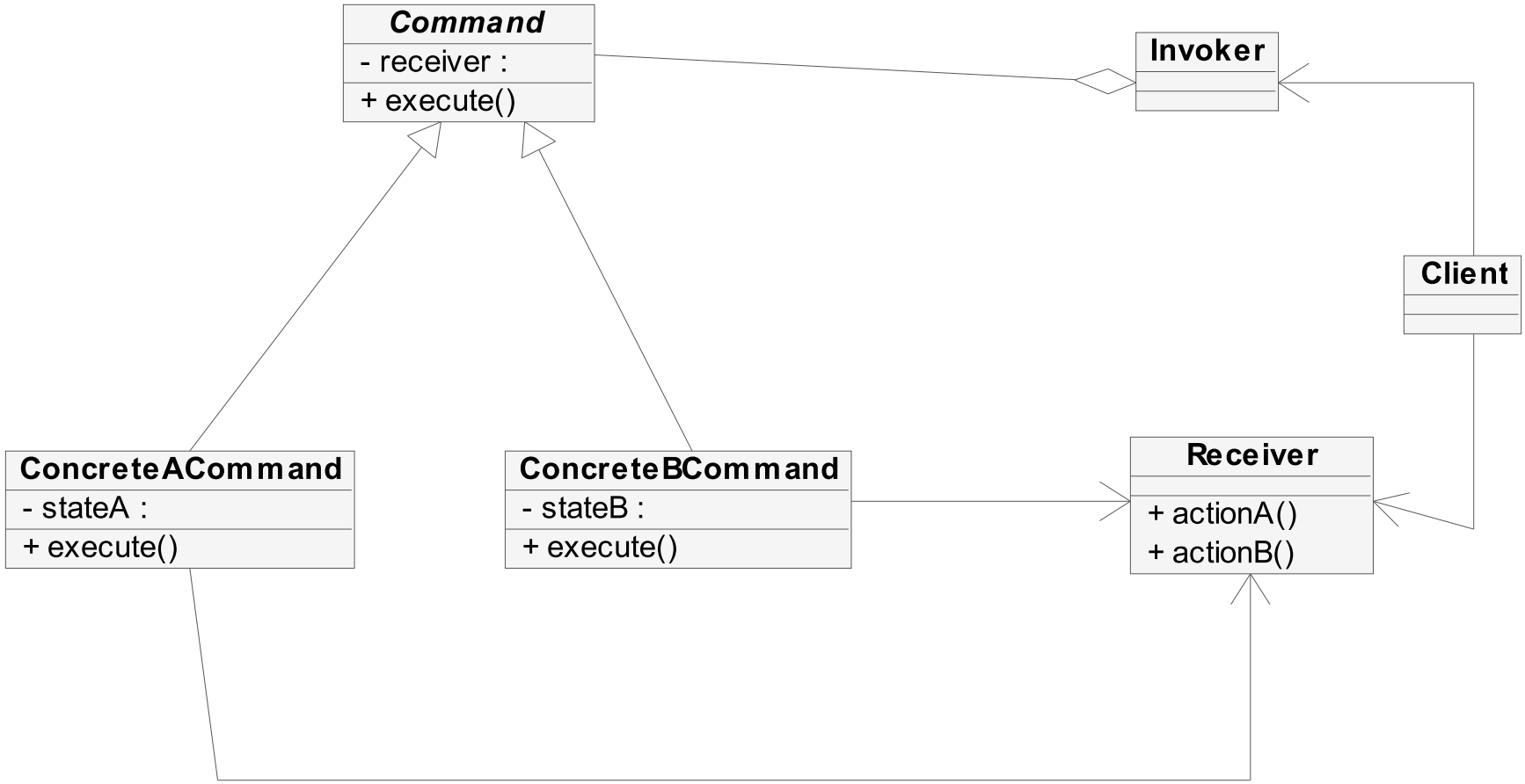
Motivace

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

Důsledky

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

Struktura


Command
- receiver :
+ execute()
Invoker
Client
ConcreteACommand
- stateA :
+ execute()
ConcreteBCommand
- stateB :
+ execute()
Receiver
+ actionA()
+ actionB()

Command

Návrhové vzory

Zdroje

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Návrh programů pomocí vzorů
  - popis 23 základních vzorů
- `wikipedia.org`