Návrhové vzory

Jakub Klemsa
Jan Legerský

Objektově orientované programování

klemsjak@fjfi.cvut.cz
jan.legersky@gmail.com

30. října 2012

Návrhové vzory

- návrhový vzor (design pattern) – obecné řešení problému, které se využívá při návrhu programů
- není knihovnou nebo částí zdrojového kódu
- popis řešení problému nebo šablona
- OO návrhové vzory ukazují vztahy a interakce mezi třídami a objekty bez implementace konkrétní třídy
- algoritmy nejsou návrhové vzory (řeší konkrétní problémy a nikoliv problémy návrhu)
- nepocházejí ze softwarového inženýrství, např. architektura

Použití

- opakující se, netriviální situace
- kombinace teoretických a praktických zkušeností
- dva druhy
  - implicitní – nepopsaná řešení odvíjející se od znalostí a zkušeností
  - explicitní – zdokumentované zkušenosti při řešení jednotlivých problémů

Historie

Ve stavební architektuře:
„Každý návrhový vzor popisuje problém, který se vyskytuje znovu a znovu v našem prostředí a popisuje jádro řešení tohoto problému takovým způsobem, že můžeme použít toto řešení mnohokrát, aniž bychom dělali stejnou věc dvakrát.“ (Christopher Alexander, 1964)

- v IT na konferenci OOPSLA (1987, Orlando) – Smalltalk
- idoms (Jim Coplien, 1989) – specifické vzory představující menší úseky kódu
- Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson a John Vlissides) – na ECOOP'91 představil Gamma a Helm několik návrhových vzorů (Composite, Decider, Observer atd.)
- Hillside Group (1993) – konference PLoP (Pattern Languages of Programs)
- Design Patterns: Elements of Reusable Object-Oriented Software – GoF, 1994

Základní rozdělení

- Creational Patterns (vytvářející)
- Structural Patterns (strukturální)
- Behavioral Patterns (chování)

Creational Patterns

- řeší problémy související s vytvářením objektů
- snaha popsat postup výběru třídy nového objektu
- zajištění správného počtu daných objektů
- dynamická rozhodnutí učiněná za běhu programu

Creational Patterns

- Factory Method Pattern
- Abstract Factory Method Pattern
- Builder Pattern
- Prototype Pattern
- Singleton Pattern

Structural Patterns

- zaměřují se na možnosti uspořádání jednotlivých tříd nebo komponent
- zpřehledňují systém
- strukturalizace kódu

Structural Patterns

- Adapter Pattern
- Bridge Pattern
- Composite Pattern
- Decorator Pattern
- Facade Pattern
- Flyweight Pattern
- Proxy Pattern

Behavioral Patterns

- navrhují chování systému
- objektové
- u tříd využívají principu dědičnosti
- u objektů řeší spolupráci mezi objekty a skupinami objektů

Behavioral Patterns

- Chain Of Responsibility Pattern
- Command Pattern
- Interpreter Pattern
- Iterator Pattern
- Mediator Pattern
- Memento Pattern
- Observer Pattern
- State Pattern
- Strategy Pattern
- Template Pattern
- Visitor Pattern

Factory Pattern (Creational)

Zadání problému

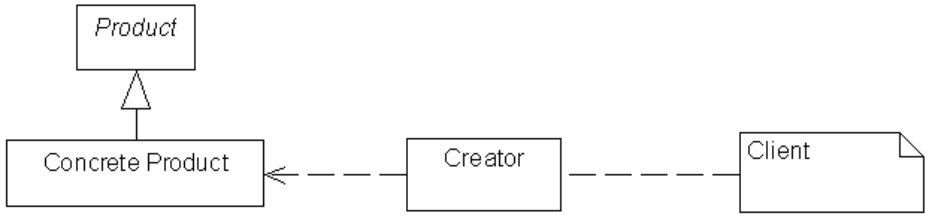
Jakým způsobem rozhodnout o vytvoření instance konkrétní třídy až v průběhu programu?

Podmínky:

- mějme několik tříd se společným předkem
- Factory pattern za běhu programu vytvoří instanci některé z těchto tříd
- princip výběru třídy: pro tuto funkci je definován objekt

Factory Pattern (Creational)

Řešení

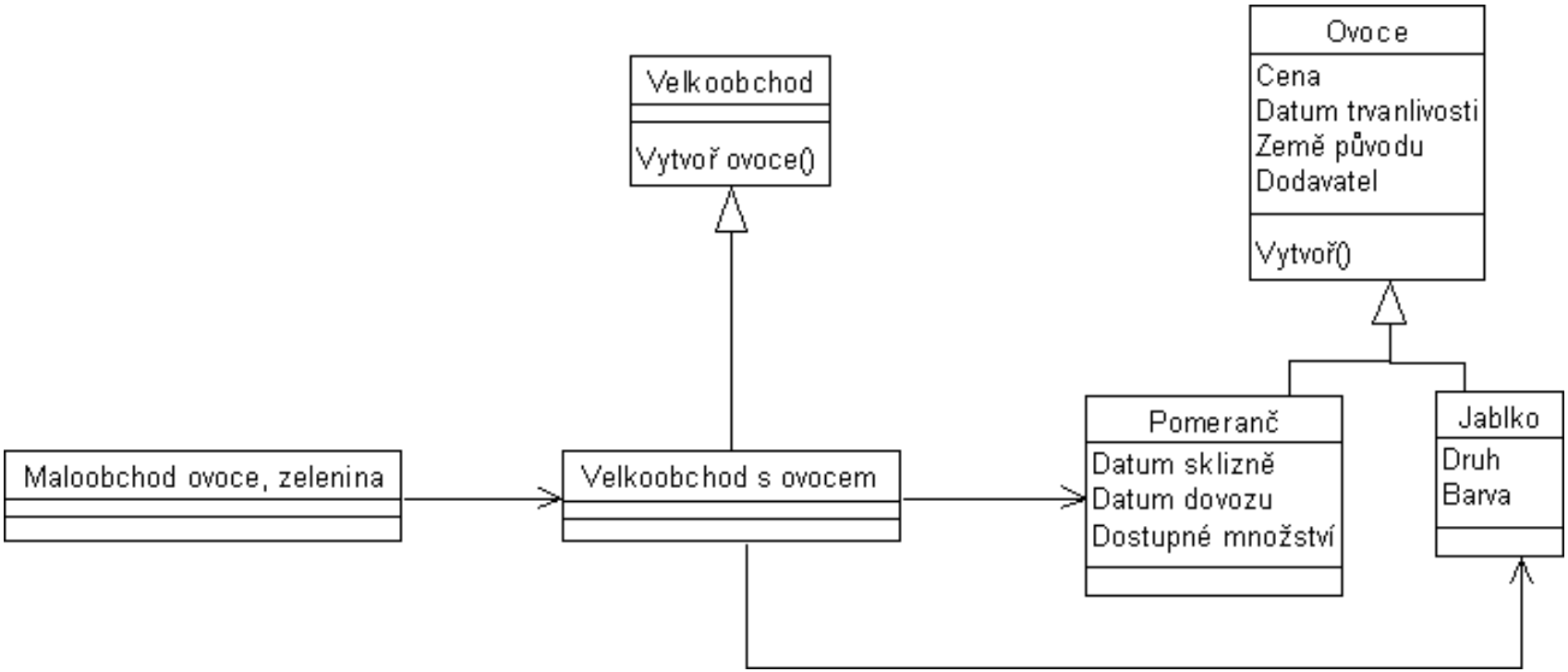


Obrázek: Factory Pattern

- obecná třída Product představuje rozhraní implementované třídami Concrete Product
- klient má pouze referenci na objekt Creator
- vyžádá si instanci třídy Concrete Pruduct, přímo ji neurčuje
- Creator rozhodne, která třída bude vybrána pro vytvoření instance, a vrací ji jako výsledek klientovi
- klient nemusí implementovat logiku výběru třídy
- neví tak, jakou instanci Concrete Product získal

Factory Pattern (Creational)

Příklad



Obrázek: Factory Pattern – příklad

Factory Pattern (Creational)

Důsledky

- uzavření logiky vytváření objektů do samostatné třídy
- klient nevytváří přímo požadovaný objekt
- Factory method rozhodne, jestli vrátí instanci nového objektu nebo použije nevyužívaný objekt
- zvyšuje flexibilitu návrhového vzoru
- v případě změny tříd vytvářených objektů stačí provést změnu pouze na jednom místě
- uplatnění v oblasti frameworků a distribuovaných aplikací (rozhoduje se mezi vytvořením nového objektu nebo využitím nepoužívaného)
- nastává při správě zdrojů – jejich použití je omezením pro výkon systému
- typicky připojení do databáze (nevytváří se nové, použije se vytvořené)

Singleton Pattern alias „*Jedináček*“ (Creational)

Zadání problému

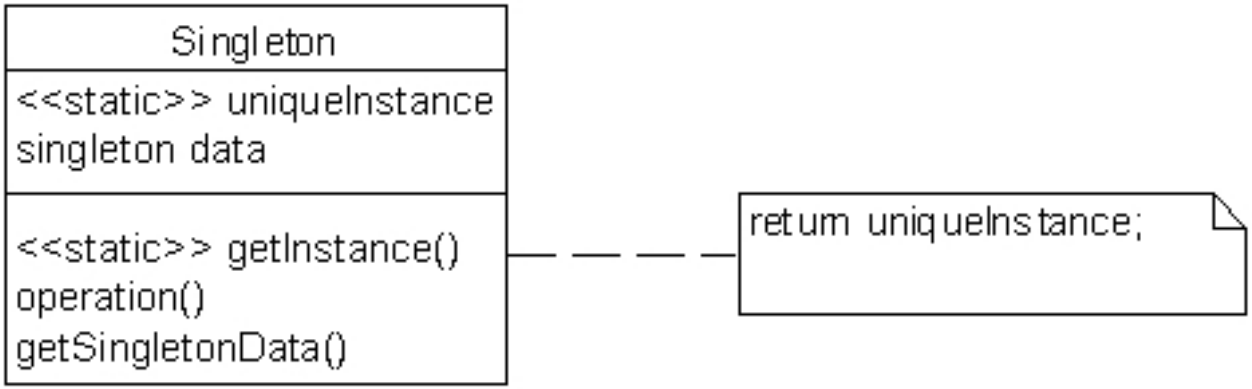
Jak zajistit vytvoření pouze jedné instance daného objektu?

Podmínky:

- Závisí na prostředcích, které nám dává použitý programovací jazyk
- Požadujeme globální přístup k instanci
- Např. pro dialogová okna, schránku, ovladače zařízení...

Singleton Pattern (Creational)

Řešení



Obrázek: Singleton Pattern

- Použití statické metody pro vytvoření instance
- Konstruktor jako Private

Singleton Pattern (Creational)

Ukázka kódu v C++

```
class Singleton {
 private:
  static Singleton* uniqueInstance;
  Singleton() {}
 public:
  static Singleton* getInstance() {
    if (uniqueInstance == NULL)
      uniqueInstance = new Singleton();
    return uniqueInstance;
    }
 };
Singleton* Singleton::uniqueInstance = NULL;
```

Adapter Pattern (Structural)

Zadání problému

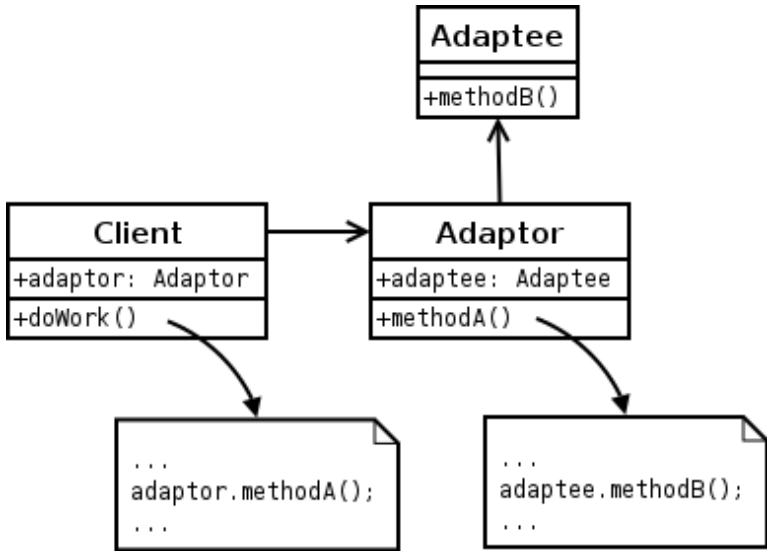
Jak přizpůsobit třídu tak, aby se dala využívat i jiným způsobem?

Podmínky:

- Prostředník mezi požadovaným rozhraním a třídou
- Konverze rozhraní jedné třídy na rozhraní jiné
- Spolupráce tříd

Adapter Pattern (Structural)

Řešení



Obrázek: Adapter Pattern

- Reference na Adaptee v Adapteru
- Delegování požadavků na metody Adaptee
- Může spojovat více objektů
- Nebo: Odvození nové třídy od staré a přidání požadovaného rozhraní

Adapter Pattern (Structural)

Ukázka kódu v C++

```
class Stack{
  private:
  List list;
  public:
  Stack(){
    list = new List;}
  void push(T what){
    list.insertBegin(what);}
  T top(){
    return list.getFirst();}
  T pop(){
    T tmp = list.getFirst();
    list.deleteFirst();
    return T;}
};
```

Iterator Pattern (Behavioral)

Zadání problému

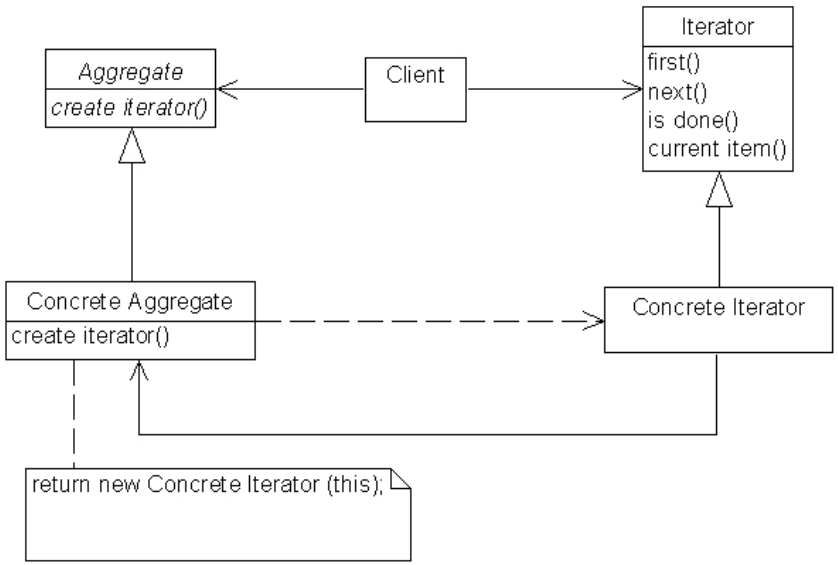
Jak se pohybovat mezi prvky, které jsou sekvenčně uspořádány?

Podmínky:

- Sekvenční pohyp v kolekci objektů
- Bez znalosti implementace
- Nezávisle na použitém kontejneru

Iterator Pattern (Behavioral)

Řešení



Obrázek: Iterator Pattern

- Iterator má metody: First, next, isDone, current...
- Datová struktura vrací iterátor
- Vhodné od abstraktního iterátoru odvodit speciální pro jednotlivé datové struktury

Zdroje


DVOŘÁK, Miloš. `http://objekty.vse.cz/Objekty/Vzory`, VŠE, Praha (2003)


VIRIUS, Miroslav. *Základy algoritmizace*, 2. vydání, ČVUT, Praha (1998)