Téma přednášky

- **Jazyk UML**
  - Sekvenční diagram
  - Diagram komunikace
  - Analytické vs. návrhové modely
- **Jazyk OCL**
- **Návrhové vzory**
  - Singleton
  - Abstract Factory
  - Command

Úvod do softwarového inženýrství

IUS 2019/2020

5. přednáška

Ing. Radek Kočí, Ph.D.
Ing. Bohuslav Křena, Ph.D.

21. a 25. října 2019

Diagramy jazyka UML 2.0

Diagramy interakce

- Sekvenční diagram (*Sequence Diagram*)
- Diagram komunikace (*Communication Diagram*)
  - původní diagram spolupráce (*Collaboration Diagram*) z UML 1.x
- Diagram přehledu interakcí (*Interaction Overview Diagram*)
- Diagram časování (*Timing Diagram*)

Odpadnutí přednášek

- **Pondělí 28. října 2019** je státní svátek.
  ⇒ Přednáška odpadá bez náhrady.
- Obě přednáškové skupiny musí mít stejný počet přednášek.
  ⇒ Jedna přednáška tak musí odpadnout i páteční skupině.
- Bude to v **pátek 1. listopadu 2019**.
  ⇒ zachování týdenní synchronizace obou skupin
  ⇒ Svátek všech svatých
  ⇒ den pracovního klidu na Slovensku

Diagramy interakce

Základní typy diagramů interakce

- sekvenční diagramy (*Sequence Diagrams*)
  - zdůrazňují časově orientovanou posloupnost předávání zpráv mezi objekty (chronologie zasílání zpráv)
  - bývají přehlednější a srozumitelnější než diagramy komunikace
  - každá čára života (objekt) je zobrazena s časovou osou
- diagramy komunikace (*Communication Diagrams*)
  - zdůrazňují strukturální vztahy mezi objekty
  - výhodné pro rychlé zobrazení komunikace mezi objekty

Diagramy interakce

Diagramy interakce

- popisují spolupráci objektů
- typicky modelují chování jednoho případu užití

Čára života

- zastupuje jednoho účastníka interakce (objekt)
- označení: `nazev[selektor]:typ`
  - `nazev` – identifikátor čáry života (objektu)
  - `selektor` – podmínka pro výběr určité instance
  - `typ` – klasifikátor, jehož je čára života instancí

honzuvUcet [id = '15'] : Ucet

Sekvenční diagram

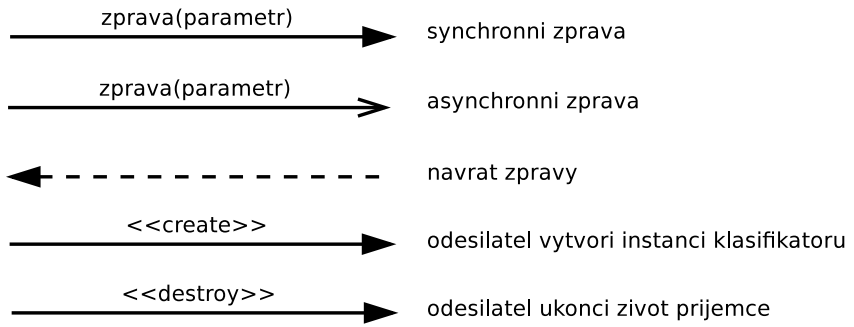
Vyjdeme z následující specifikace případu užití (uvedená specifikace je pouze ilustrativní, v reálném systému by se řešilo jinak):

Případ užití: Přidat přednášku
ID: UC11
Účastníci: Kvestor
Vstupní podmínky: 1. Kvestor je přihlášen do systému.
Tok událostí: 1. Kvestor zadá příkaz "přidat přednášku". 2. Systém přijme název nové přednášky. 3. Systém vytvoří novou přednášku.
Následné podmínky: 1. Nová přednáška byla přidána do systému.

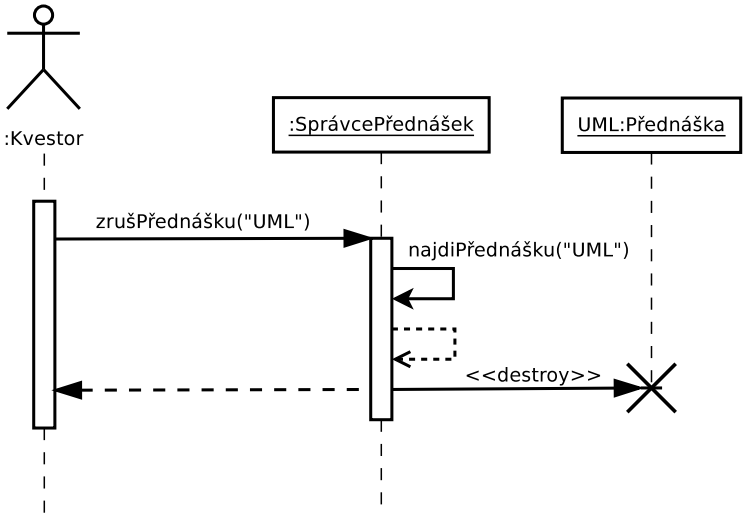
Diagramy interakce

Zprávy

- komunikace mezi účastníky interakce
- typy zpráv

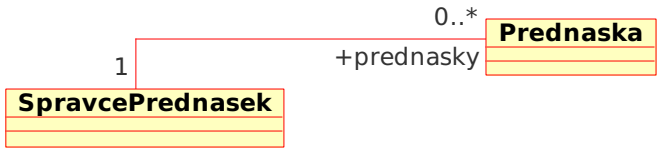


Sekvenční diagram



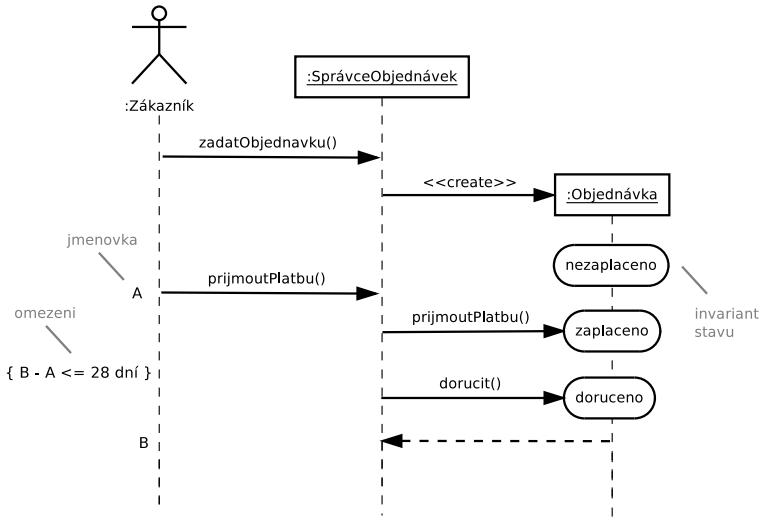
Sekvenční diagram

Na základě počáteční analýzy případu užití vytvoříme diagram tříd



Sekvenční diagram

Rozšíření sekvenčních diagramů (omezení, zobrazení stavů)



Sekvenční diagram

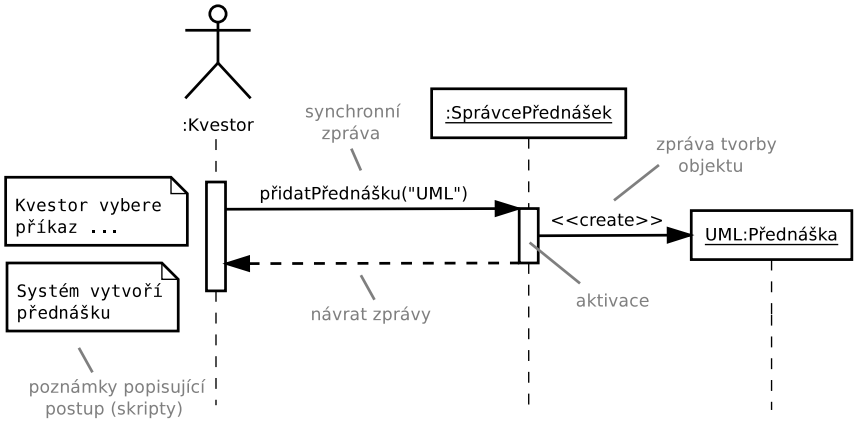


Diagram komunikace

Rozšíříme diagram tříd

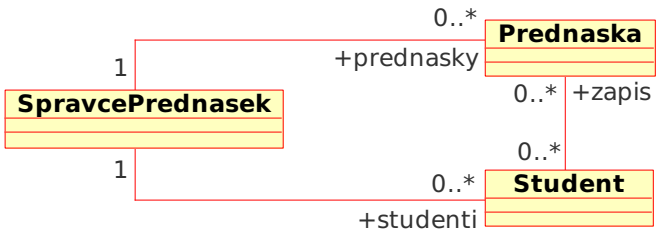


Diagram komunikace

- objekty jsou spojeny linkami (komunikační kanály)
- zprávy jsou řazeny podle hierarchického číslování

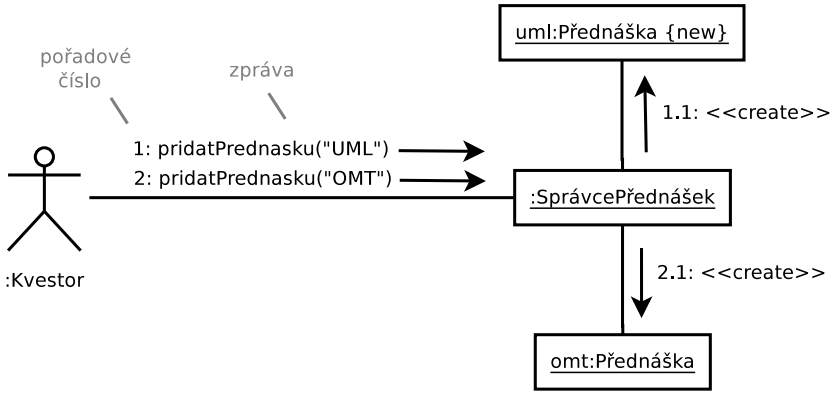


Diagram komunikace

- větvení, kontrolní podmínky

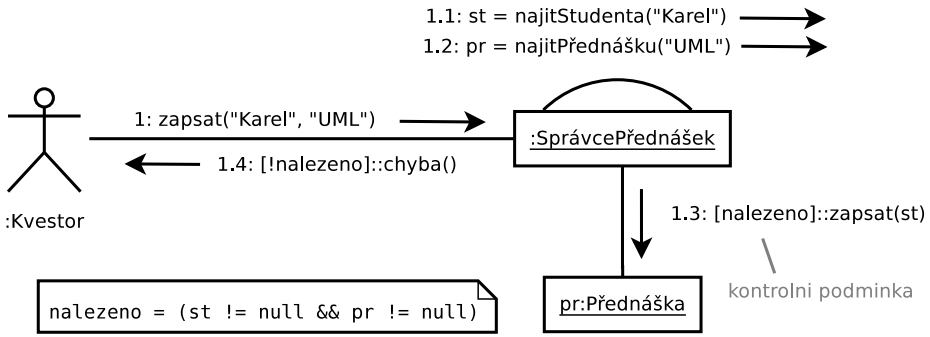


Diagram komunikace

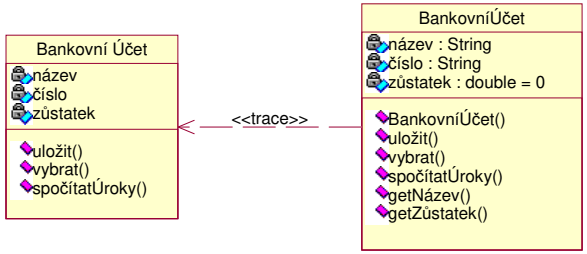
Pro analýzu přidáme následující specifikaci případu užití (uvedená specifikace je pouze ilustrativní, v reálném systému by se řešilo jinak):

Případ užití: Zapsat studenta na přednášku
ID: UC17
Účastníci: Kvestor, Student
Vstupní podmínky: 1. Kvestor je přihlášen do systému.
Tok událostí: 1. Kvestor zadá příkaz "zapsat studenta". 2. Systém vyhledá studenta **S** podle zadaného jména. 3. Systém vyhledá přednášku **P** podle zadaného názvu. 4. Systém zapíše studenta **S** na přednášku **P**.
Následné podmínky: 1. Student **S** je zapsán na přednášku **P**.

Návrhové třídy

Návrhové třídy

- specifikace návrhových tříd je na takovém stupni, že je lze přímo implementovat
- upřesňování analytických tříd
- využití tříd z doménového řešení
  knihovny, vrstva aplikačního serveru, GUI, ...



UML v etapách vývoje softwaru

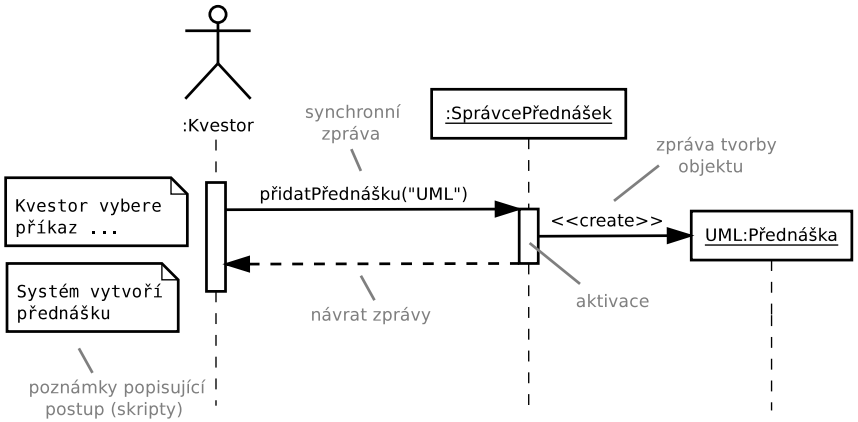
Tvorba analytických modelů

- zaměřuje se na otázku *co*, neodpovídá detailně na otázku *jak* (to je úkolem návrhu)
- zobrazuje důležité věci (objekty, vztahy, ...) z problémové domény
  - třídy *Zákazník*, *Košík*, ...
  - třída pro přístup k databázím patří do řešení (návrhu)

- analytické třídy
- diagramy případů užití
- **specifikace případů užití**
  - diagramy aktivit
  - stavové diagramy
- **realizace případů užití**
  - modelují interakce analytických tříd
  - identifikují zasílané zprávy mezi objekty (instancemi tříd)

Upřesnění modelu v etapě návrhu

Analytický model interakce



UML v etapách vývoje softwaru

Tvorba návrhových modelů

- vychází z výstupů etapy analýzy
- zaměřuje se na otázku *jak*, věnuje se detailům
- specifikace modelů je na takové úrovni, že je lze přímo implementovat

- upřesňování analytických diagramů
  - návrhové třídy
  - realizace případů užití
  - ...

Mechanismy rozšiřitelnosti UML

Omezení (Constraints)

- definují omezující podmínky
- rozšiřují sémantiku elementu (např. OCL)

Stereotypy (Stereotypes)

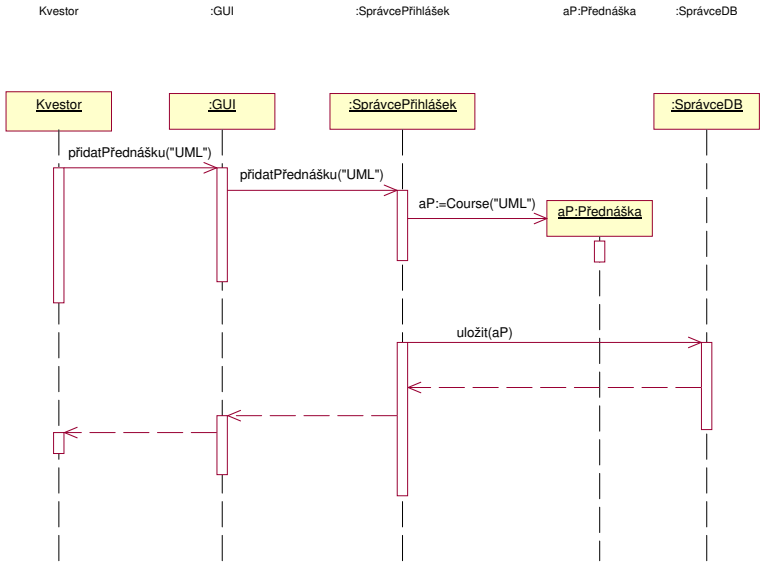
- definuje nový element na základě stávajícího elementu
- stereotyp má svou sémantiku

Označené hodnoty (tagged values)

- {tag1 = hodnota1, tag2 = hodnota2}
- většinou se přidružují k stereotypu, vyjadřují vlastnosti nových elementů

Upřesnění modelu v etapě návrhu

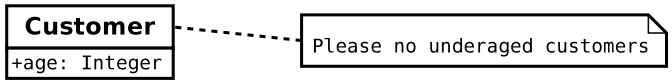
Návrhový model interakce



Jazyk UML – omezení a dotazy nad modely

Modelovací techniky UML

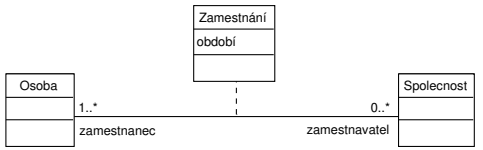
- nedokáží zachytit všechny závislosti mezi elementy graficky
- řeší se poznámkou se slovním popisem
- slovní popis je nedostatečný
  - není vždy jednoznačný, může být různě pochopen
  - komplikuje automatické konverze

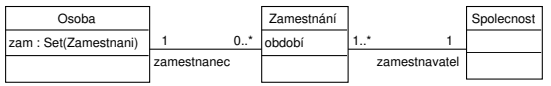


Upřesnění analytických relací

Asociace

- agregace vs. kompozice
  - upřesnění vztahu celek/část
- asociace povýšená na třídu
  - návrh asociačních tříd
- asociace typu 1:N
  - realizace (nejčastěji kolekce)





Object Constraint Language

Object Constraint Language (OCL)

- speciální *formální* jazyk pro UML
- OCL *není* programovací jazyk
- původně vytvořen jako obchodní modelovací jazyk v IBM, 1995
- součástí OMG standardů pro UML (od verze 1.1)
- OCL 2.0 (2006)
- OCL 2.4 (2014)
- `http://www.omg.org/spec/OCL/`
- *Warmer, J., Kleppe, A.: The Object Constraint Language. Getting Your Models Ready For MDA. Addison-Wesley, 2003*

Jazyk UML – omezení a dotazy nad modely

Zdroj: http://www.slideshare.net/jcabot/ocl-tutorial

Object Constraint Language

Object Constraint Language (OCL)

- definuje omezení, podmínky a dotazy nad UML modely
  ⇒ zpřesňování modelů
- je *formální deklarativní* jazyk navržený pro návrháře
  ⇒ nevyžaduje se silný matematický základ
- spojený s dalšími metamodely definovanými OMG
- umožňuje transformace modelů

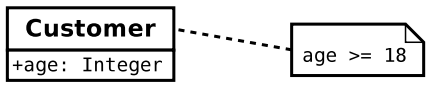
Využití OCL

- specifikace podmínek pro vykonání metod
- specifikace invariantů tříd
- specifikace počátečních hodnot atributů
- specifikace těla operace
- specifikace omezení
- ...

Jazyk UML – omezení a dotazy nad modely

Modelovací techniky UML

- nedokáží zachytit všechny závislosti mezi elementy graficky
- řeší se poznámkou se slovním popisem
- slovní popis je nedostatečný
  - není vždy jednoznačný, může být různě pochopen
  - komplikuje automatické konverze



- ⇒ **jednoznačný jazyk** ⇒ **OCL**

Object Constraint Language

Ukázky

- invarianty atributů
```
context Customer inv:
   age >= 18
```

- kolekce objektů (`Salesperson` $^{1}\!\!\rightarrow^{\mathrm{N}}$ `Customer` / role `clients`)
```
context Salesperson inv:
   clients->size() <= 100 and
   clients->forAll(c: Customer | c.age >= 40)
```

- počáteční hodnoty
```
context Customer::age : Integer
   init: 18

context Salesperson::clients : Set(Customer)
   init: Set
```


Object Constraint Language

Typy omezení

- *invariant* – podmínka, která musí být vždy splněna všemi instancemi
- *precondition* – omezení, které musí být pravdivé před provedením operace
- *postcondition* – omezení, které musí být pravdivé těsně po ukončení operace
- *guard* – omezení, které musí být pravdivé před provedením přechodu mezi stavy

Základní knihovna

- typy: Boolean, Integer, Real, String
- kolekce: Collection, Set, Ordered Set, Bag, Sequence
- operace: and, or, $<$, size, includes, count, ...

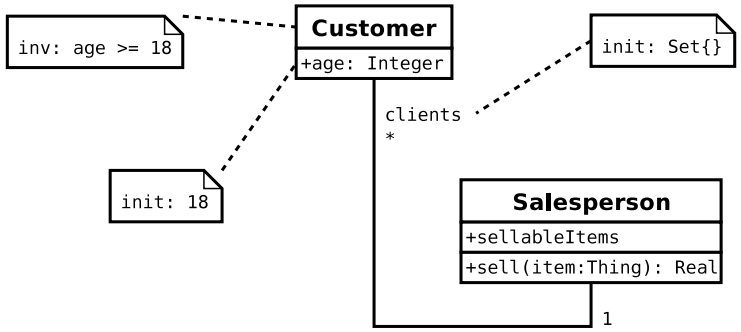
Object Constraint Language

Ukázky

- precondition, postcondition
```
context Salesperson::sell( item: Thing ): Real
    pre: self.sellableItems->includes( item )
    post: not self.sellableItems->includes( item ) and
    result = item.price
```

- podmínky
```
self.clients.select(c : Customer | c.age > 50)
```


Object Constraint Language



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

Znovupoužitelnost

Objektově orientovaný návrh a programování

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

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 nebo třídami
  - zabývá se způsoby rozdělení povinností mezi objekty nebo třídy

Návrhové vzory (Design Patterns)

Návrhové vzory

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

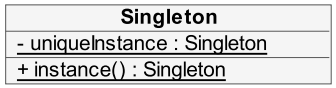
Návrhový vzor

- vzor je šablona pro řešení, nikoli implementace problému!
- každý vzor popisuje problém, který se neustále vyskytuje, a jádro řešení daného problému
- umožňuje jedno řešení používat mnohokrát, aniž bychom to dělali dvakrát stejným způsobem

Zdroje

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Návrh programů pomocí vzorů. *Popisuje 23 základních vzorů.*

Jedináček (Singleton)




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

    private Singleton() {}

    // Tovarni metoda (Factory method)
    public static Singleton instance() {
        if (uniqueInstance == null)
            uniqueInstance = new Singleton();
        return uniqueInstance;
    }
}
```


Základní návrhové vzory

Tvořivý	Strukturální	Chování
Factory method	Adapter (class)	Interpreter
Abstract Factory	Adapter (object)	Iterator
Singleton	Decorator	Visitor
Prototype	Facade	Memento
Builder	Bridge	Observer
	Flyweight	Mediator
	Composite	Command
	Proxy	Chain of Responsibility
		State
		Strategy

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
- . . .

Jedináček (Singleton)

Účel

- 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)
- . . .

Abstraktní továrna: Příklad

Úprava na novou sadu objektů:

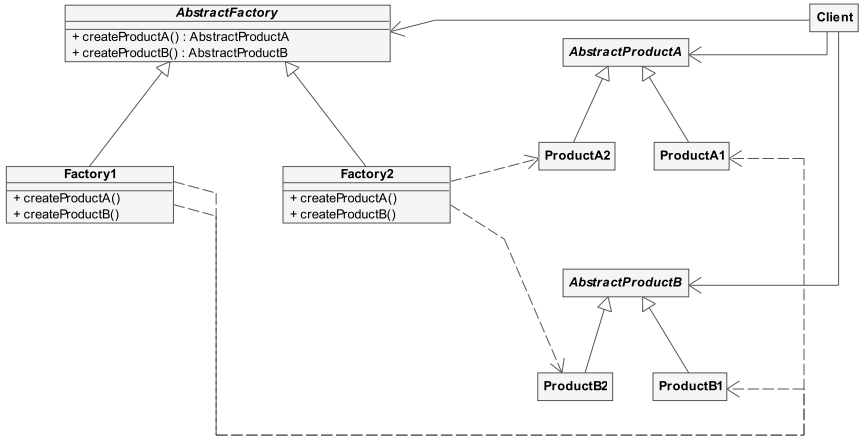
```
public class MazeGame {
    public Maze createNewMaze() {
        SpecWall wall = new SpecWall();
        SpecGate gate = new SpecGate();
        ...
    }
    private doSomething(SpecWall wall) { ... }
}
```

Řešení

- přepis stávajícího kódu ⇒ ztrácíme původní verzi
- kopie stávajícího kódu ⇒ musíme udržovat více verzí
- vytvořit flexibilní kód ⇒ návrhový vzor *Abstraktní továrna*

Abstraktní továrna (Abstract Factory)

Struktura



Abstraktní továrna: Příklad

Vytvoříme abstraktní prvky podle vzoru:

```
// abstraktní produkty
public interface Wall { ... }
public interface Gate { ... }

// abstraktní továrna
public abstract class MazeFactory {
    public abstract Wall createWall();
    public abstract Gate createGate();
}
```


Abstraktní továrna: Příklad

Příklad bludiště, pracuje s objekty zeď a brána:

```
public class MazeGame {
    public Maze createNewMaze() {
        StdWall wall = new StdWall();
        StdGate gate = new StdGate();
        ...
    }
    private doSomething(StdWall wall) { ... }
}
```

Použití

```
MazeGame game = new MazeGame();
game.createNewMaze();
```

Abstraktní továrna: Příklad

Vytvoříme jinou sadu prvků:

```
// konkrétní produkty
public class SpecWall implements Wall { ... }
public class SpecGate implements Gate { ... }

// konkrétní továrna
public class SpecMazeFactory {
    public Wall createWall() { return new SpecWall(); }
    public Gate createGate() { return new SpecGate(); }
}
```

Použijeme konkrétní prvky *bez modifikace kódu bludiště*:

```
MazeGame game = new MazeGame();
MazeFactory factory = new SpecMazeFactory();
game.createNewMaze(factory);
```


Abstraktní továrna: Příklad

Upravíme původní kód podle vzoru:

```
// aplikace vzoru v původním kódu
public class MazeGame() {
    public Maze createNewMaze(MazeFactory factory) {
        // StdWall wall = new StdWall();
        Wall wall = factory.createWall();

        // StdGate gate = new StdGate();
        Gate gate = factory.createGate();
        ...
    }
    private doSomething(Wall wall) { ... }
}
```


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 stav klienta před provedením příkazu
- ...

Abstraktní továrna: Příklad

Vytvoříme konkrétní prvky podle vzoru:

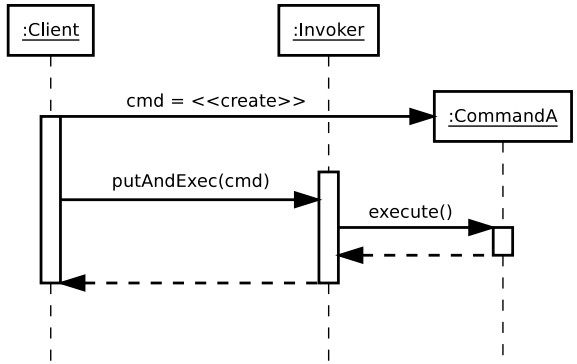
```
// konkrétní produkty
public class StdWall implements Wall { ... }
public class StdGate implements Gate { ... }

// konkrétní továrna
public class StdMazeFactory {
    public Wall createWall() { return new StdWall(); }
    public Gate createGate() { return new StdGate(); }
}
```

Použijeme konkrétní prvky:

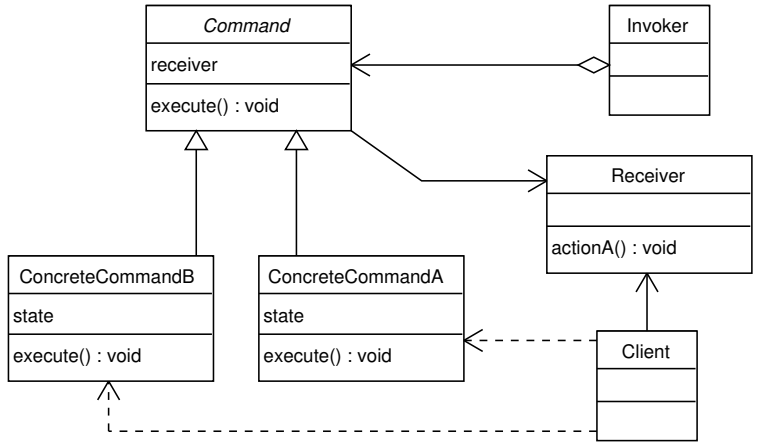
```
MazeGame game = new MazeGame();
MazeFactory factory = new StdMazeFactory();
game.createNewMaze(factory);
```

Command – Příklad

Ukázka sekvenčního diagramu pro popis chování.



Command – Struktura



Studijní koutek – Studium v zahraničí

Získání nových poznatků, kontaktů, poznání odlišného prostředí, sblížení s cizím jazykem, ...

Program Erasmus+

- Studijní pobyt je součástí studia na české univerzitě.
- Výsledky studijního pobytu se uznávají – kreditový systém ECTS.
- Student v zahraničí má stejné podmínky jako místní student.
- Student získává grant ve formě stipendia jako příspěvek na náklady spojené s cestou a pobytem.
- Délka pobytu 90-360 dnů (v rámci jednoho ak. roku).
- Suma pobytů během jednoho stupně studia je nejvýše 360 dnů.
- FIT má nasmlouváno cca 150 míst na 64 univerzitách v 22 zemích: Francie, Německo, Španělsko, Řecko, Finsko, Velká Británie, ...

Další informace a programy

`https://www.vutbr.cz/studenti/staze`

`https://www.fit.vut.cz/study/study-abroad/`

Command – Příklad

Původní operace:

```
t1.remove(ch);
t2.put(ch);
```

⇒

Aplikace vzoru:

```
cmd = new CommandA(t1,t2,ch);
invoker.putAndExec(cmd);
...
invoker.removeAndUndo();
```

Invoker:

```
putAndExec(cmd) {
  stack.push(cmd);
  cmd.execute();
}

removeAndUndo() {
  cmd = stack.pop();
  cmd.undo();
}
```

CommandA:

```
execute() {
  t1.remove(ch);
  t2.put(ch);
}

undo() {
  t2.remove(ch);
  t1.put(ch);
}
```