Úvod do softwarového inženýrství

IUS 2024/2025

5. přednáška

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

14. a 18. října 2024

Téma přednášky

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

Diagramy jazyka UML 2.0

Diagramy interakce

- Sekvenční diagram (*Sequence Diagram*)
- Diagram komunikace (*Communication Diagram*)
- Diagram přehledu interakcí (*Interaction Overview Diagram*)
- Diagram časování (*Timing Diagram*)

Diagramy interakce

Diagramy interakce

- popisují spolupráci objektů
- typicky modelují chování jednoho případu užití

ne cely system ale pouze specifickou cast jeho

Čára života

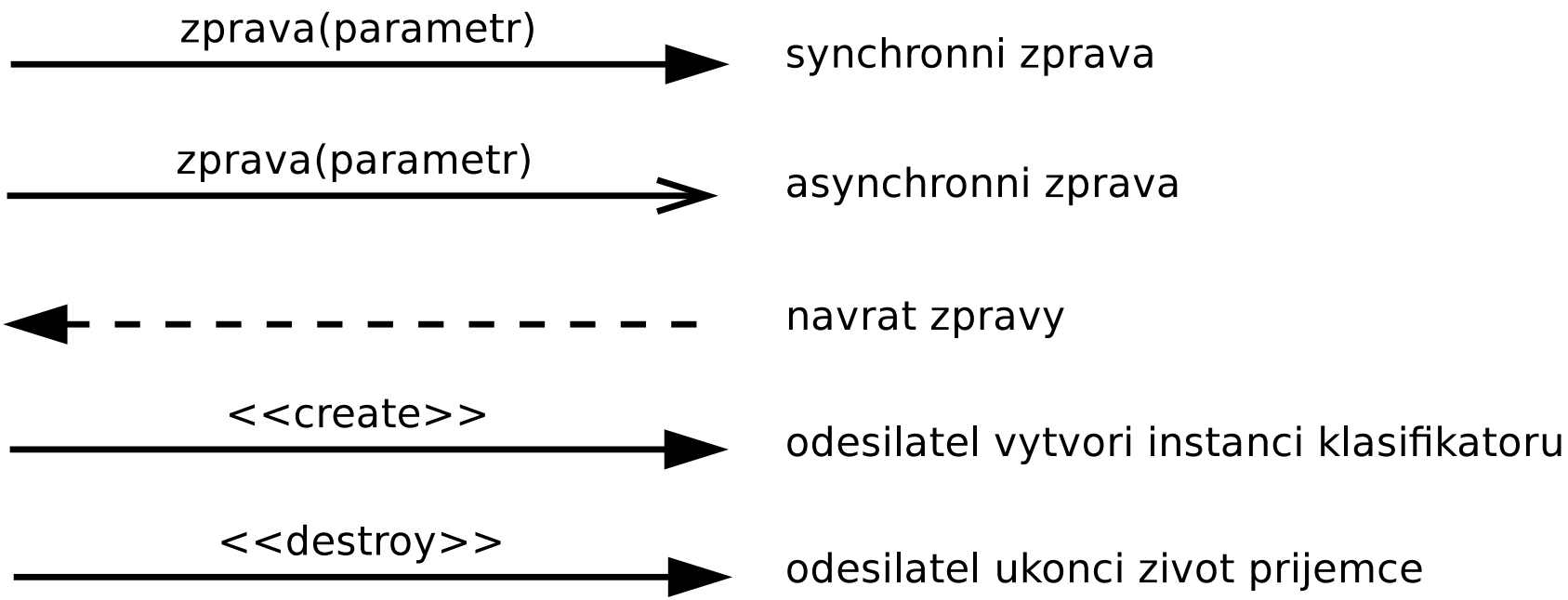
- zastupuje jednoho účastníka interakce (objekt)
- označení: `nazev[selektor]:typ`
  - `nazev` – identifikátor čáry života (objektu)
  - `selektor` – podmínka pro výběr určité instance
  - `typ` – klasifikátor, jehož je čára života instancí

honzuvUcet [id = '15'] : Ucet

Diagramy interakce

Zprávy

- komunikace mezi účastníky interakce
- typy zpráv



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

Prisne navazane na diagram trid ze ktereho se casto vychazi. Jak sekvencni diagram tak diagram komunikace musi byt stejne.

Naucit se prechod mezi diagramem komunikace a diagramem sekvencnim na ZKOUSKU

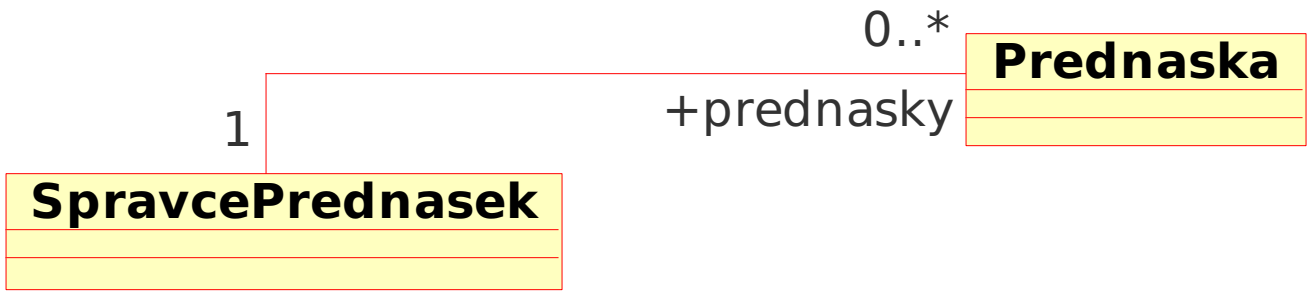
Sekvenční diagram

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

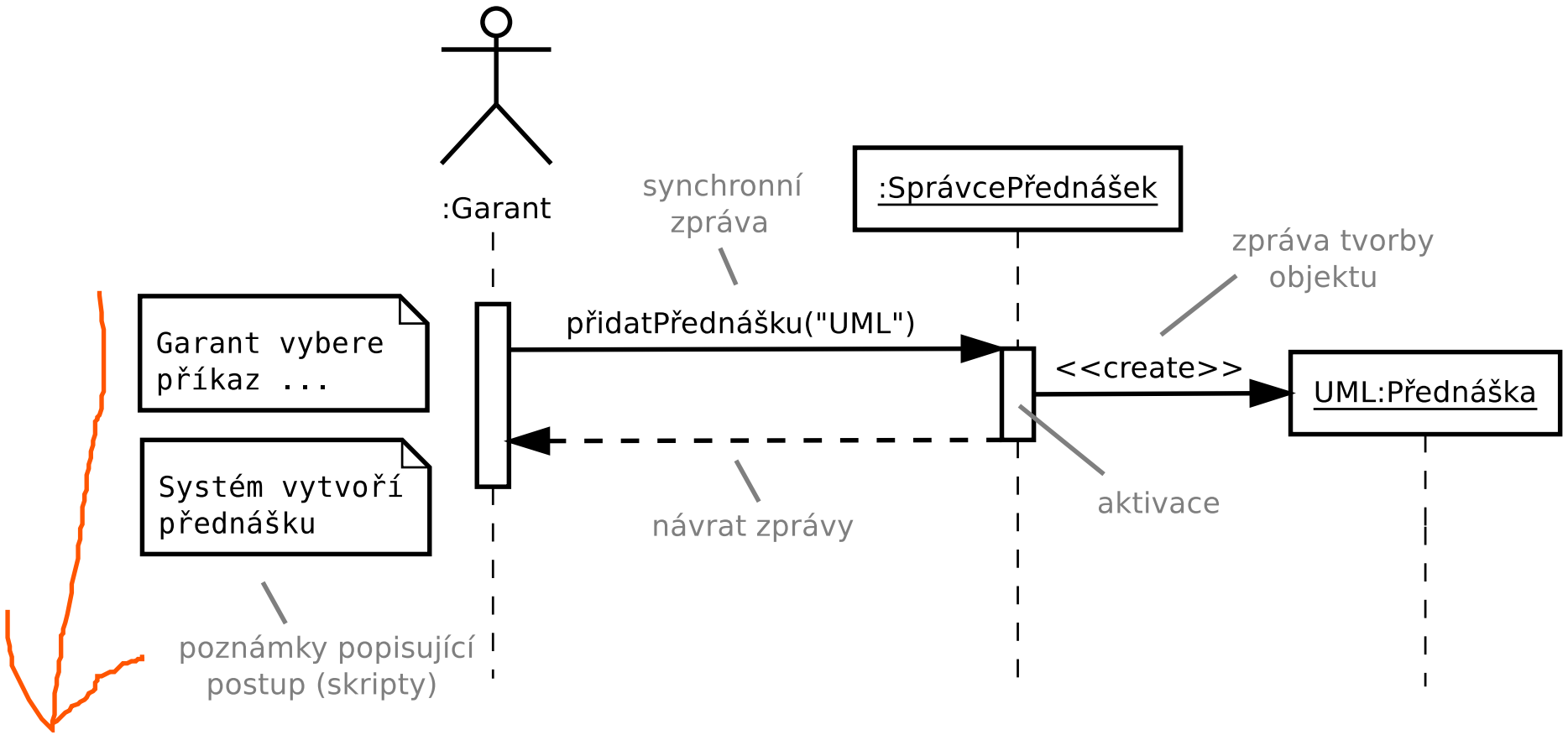
Případ užití: Přidat přednášku
ID: UC11
Účastníci: Garant
Vstupní podmínky: 1. Garant je přihlášen do systému.
Tok událostí: 1. Garant zadá příkaz "přidat přednášku". 2. Systém přijme název nové přednášky. 3. Systém vytvoří novou přednášku.
Následné podmínky: 1. Nová přednáška byla přidána do systému.

Sekvenční diagram

Na základě počáteční analýzy případu užití vytvoříme diagram tříd



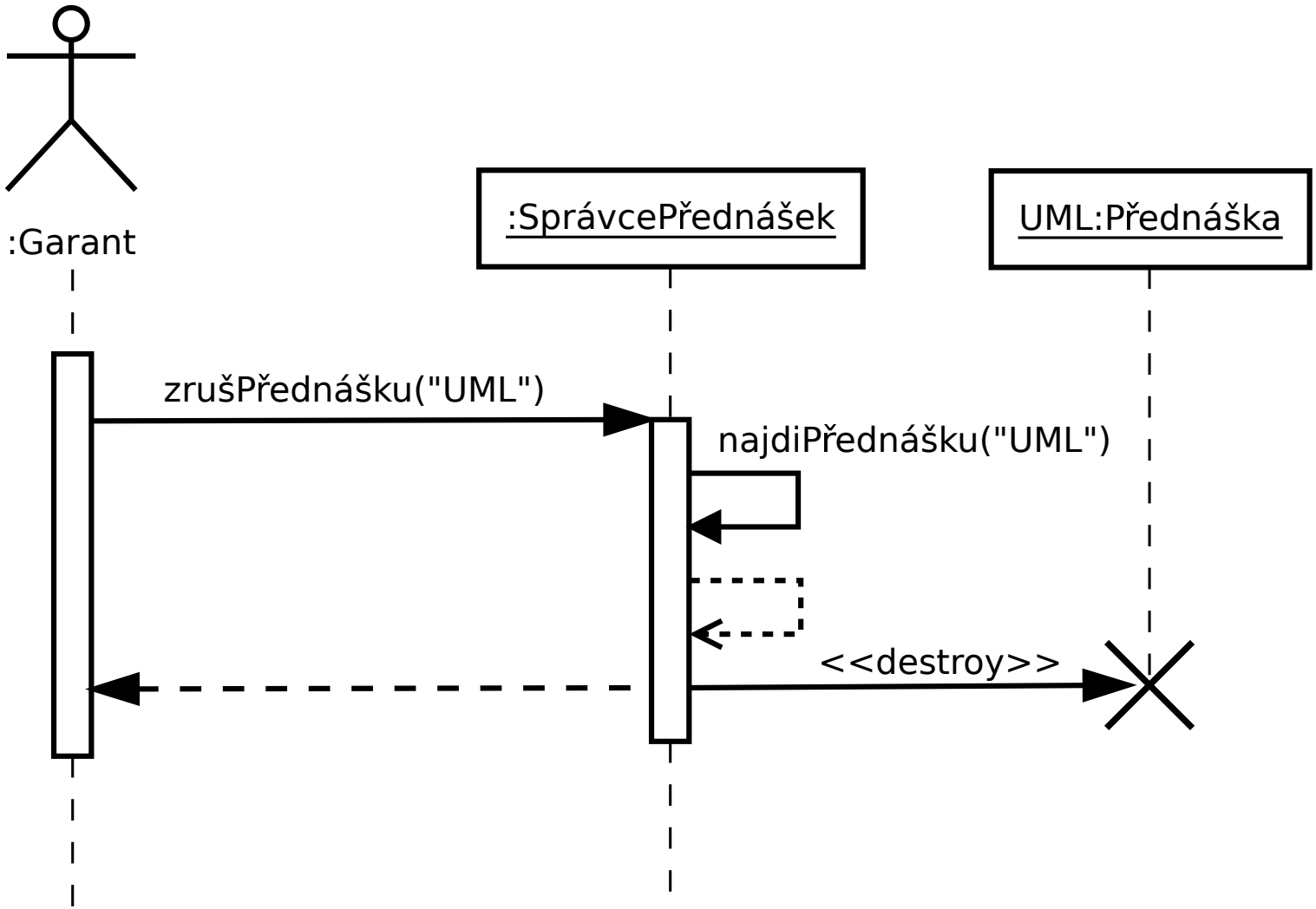
Sekvenční diagram



Casto tady figuruje system jako objekt

cara zivota je z hora dolu a u kazdeho je zobrazena zivotnost pomoci velikosti obdelniku.

Sekvenční diagram



Sekvenční diagram

Rozšíření sekvenčních diagramů (omezení, zobrazení stavů)

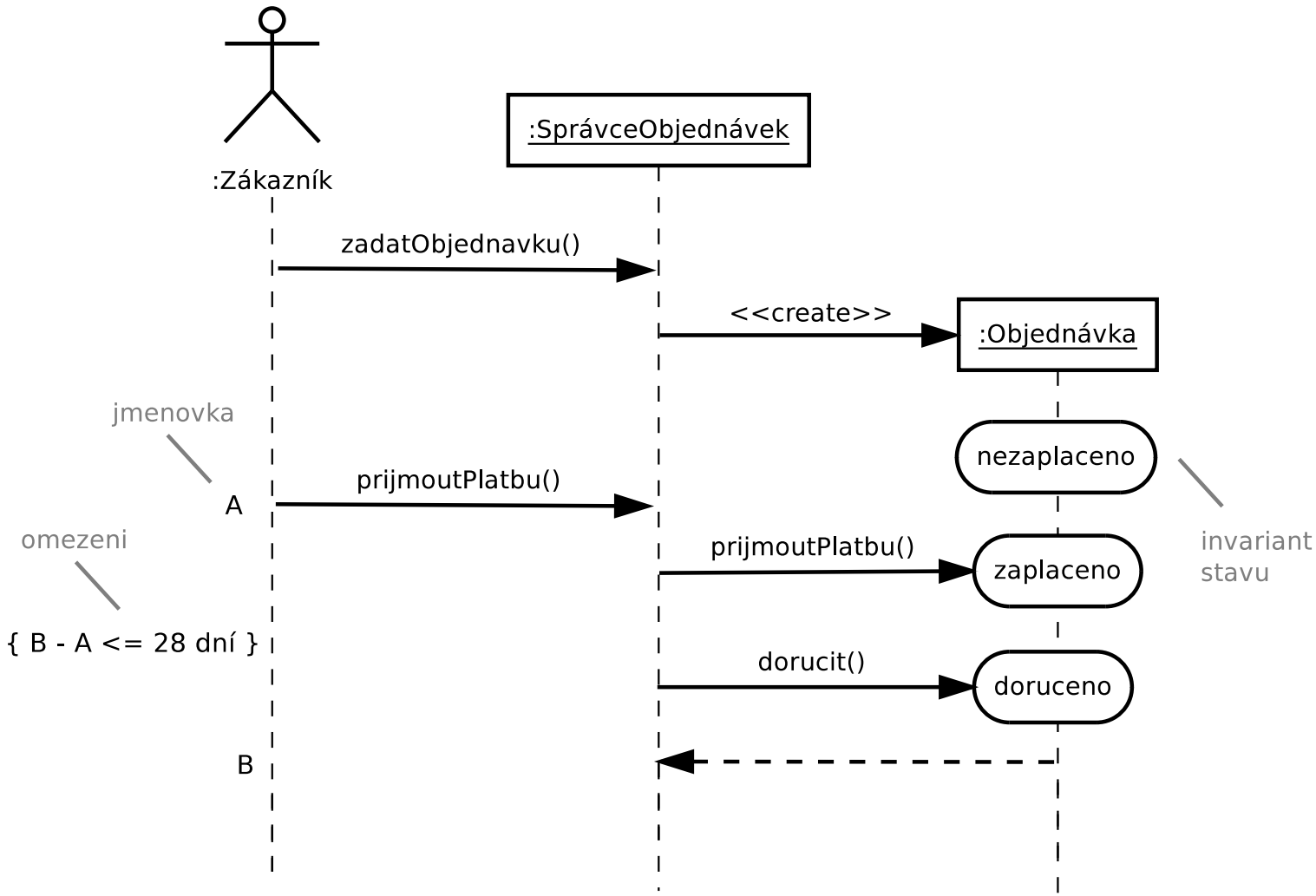
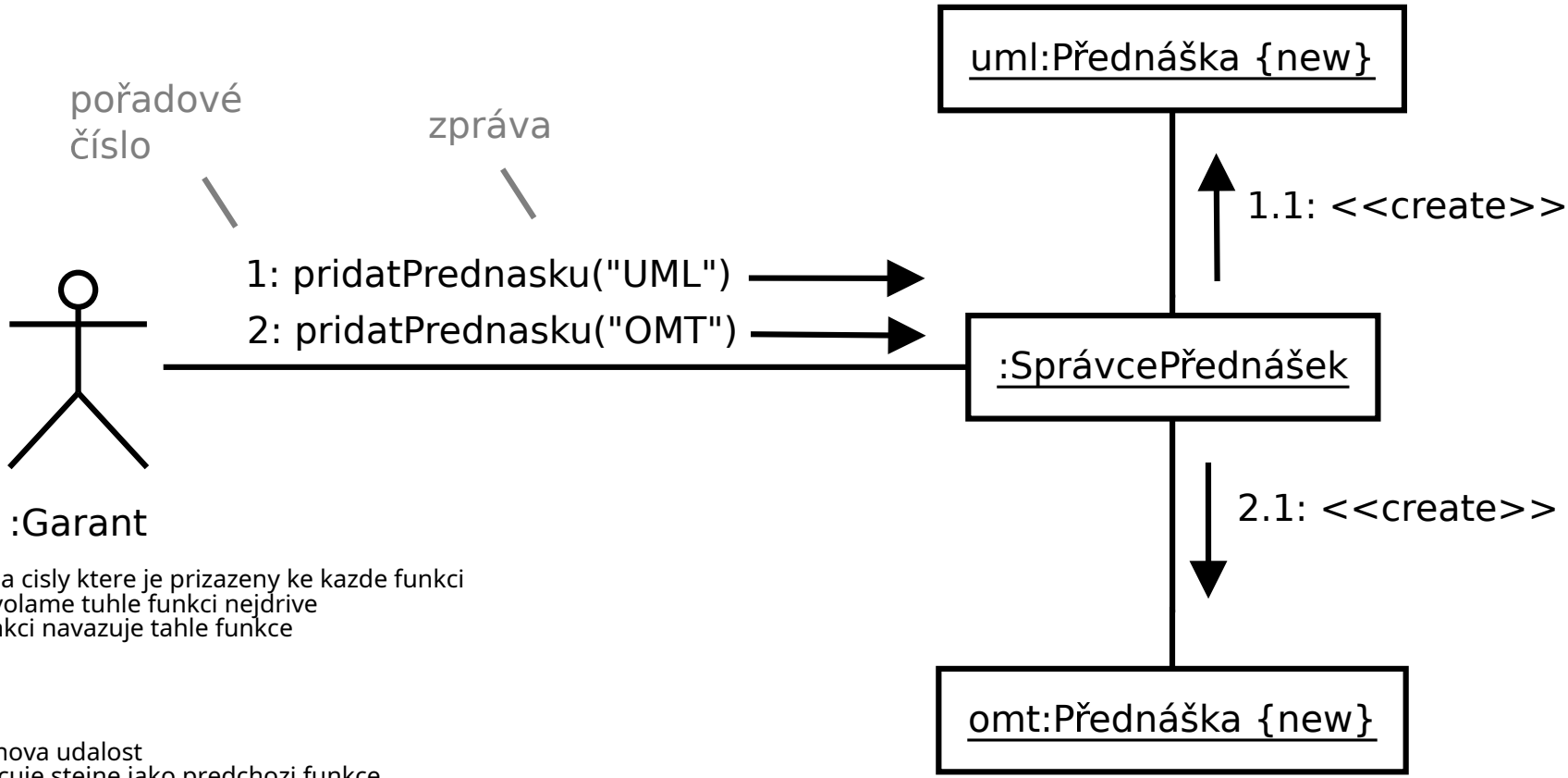


Diagram komunikace

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



Zivotnost je tada cisly ktere je prizazeny ke kazde funkci
1: pridat..... zavolame tuhle funkci nejdrive
1:1 na prvni funkci navazuje tahle funkce
.
.
.
.
2: je vicemene nova udalost
2:1 ktera pokracuje stejne jako predchozi funkce

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: Garant, Student
Vstupní podmínky: 1. Garant je přihlášen do systému.
Tok událostí: 1. Garant 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**.

Diagram komunikace

Rozšíříme diagram tříd

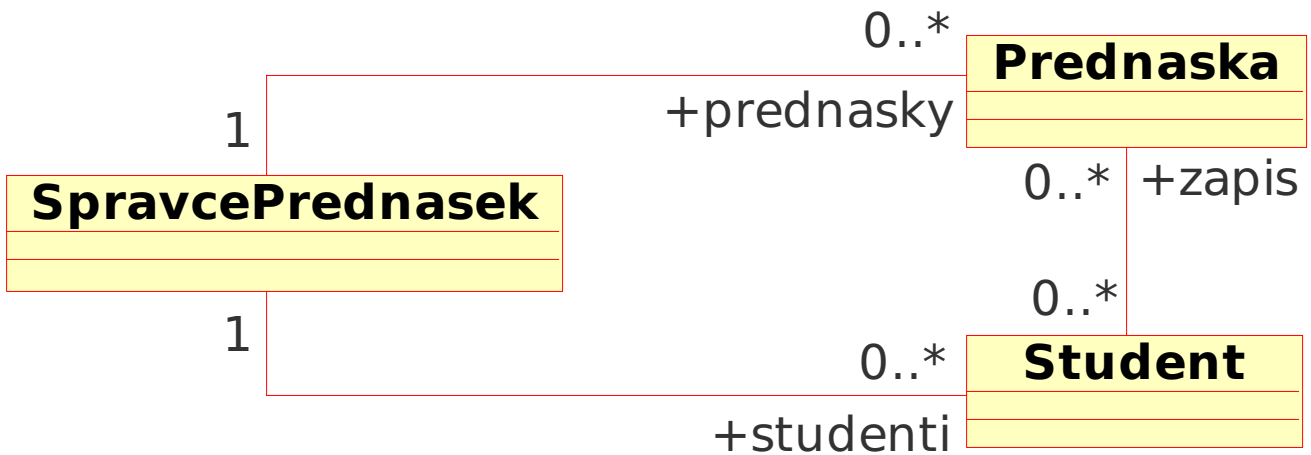
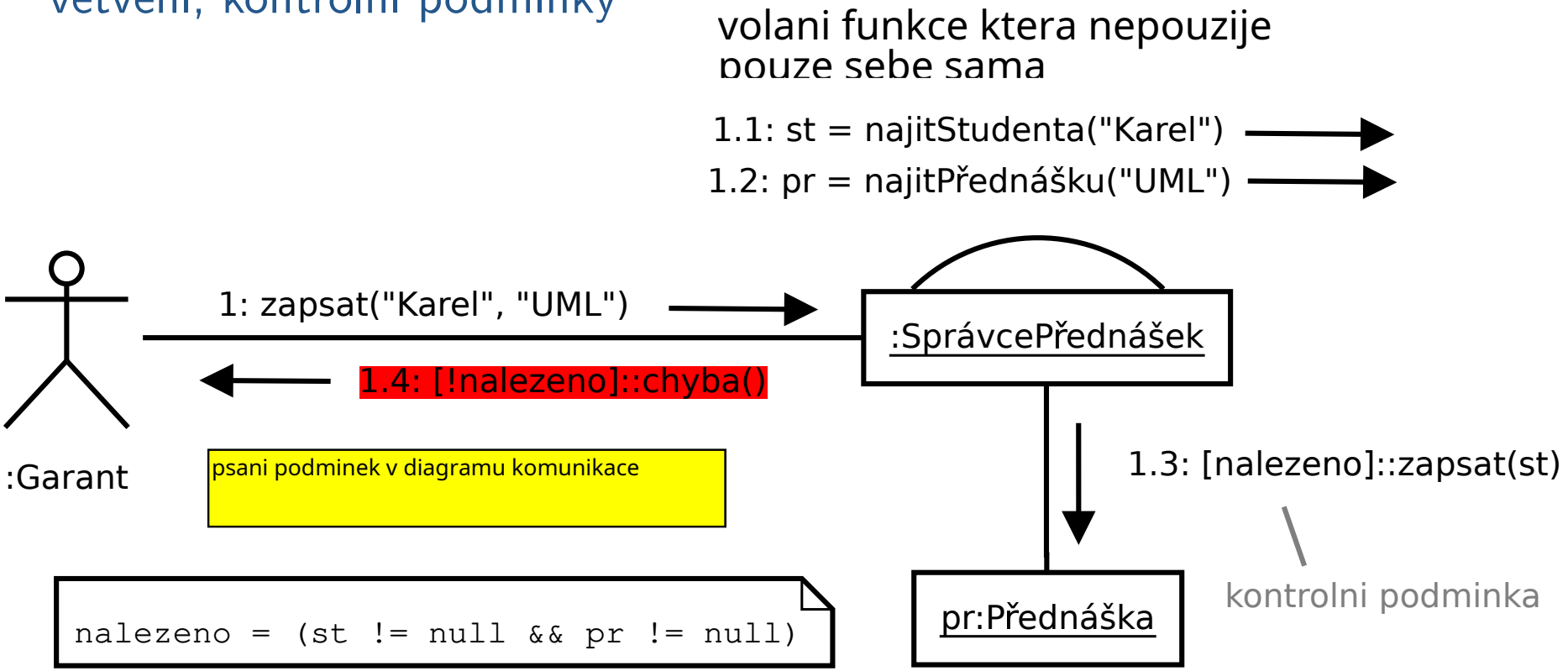


Diagram komunikace

- větvení, kontrolní podmínky



UML v etapách vývoje softwaru

Analytické modely

- zaměřují se na otázku *co*, neodpovídá detailně na otázku *jak*
- zobrazují důležité koncepty (objekty, vztahy, . . .) z problémové domény
  - třídy *Zákazník*, *Košík*, . . .
  - třída pro přístup k databázím patří do řešení (návrhu)

Tvorba analytických modelů

- doménový model (analytické třídy)
- diagramy případů užití
- **specifikace případů užití**
  - diagramy aktivit
  - stavové diagramy
- **realizace případů užití**
  - modelují interakce (zasílání zpráv) konceptuálních objektů
  - diagramy interakce

Neresi jak to udelat ale spis co udelat... proste se v teto fazi nebudeme zabyvat algoritmy a dalsi implementaci samotneho systemu
resime hlavne jak poskladat system aby odpovidal pozadavkum ale neresime je "naprogramovani"

Jak si predstavujeme interakci mezi objekty v systemu v jakem case atd..

UML v etapách vývoje softwaru

Návrhové modely

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

Tvorba návrhových modelů

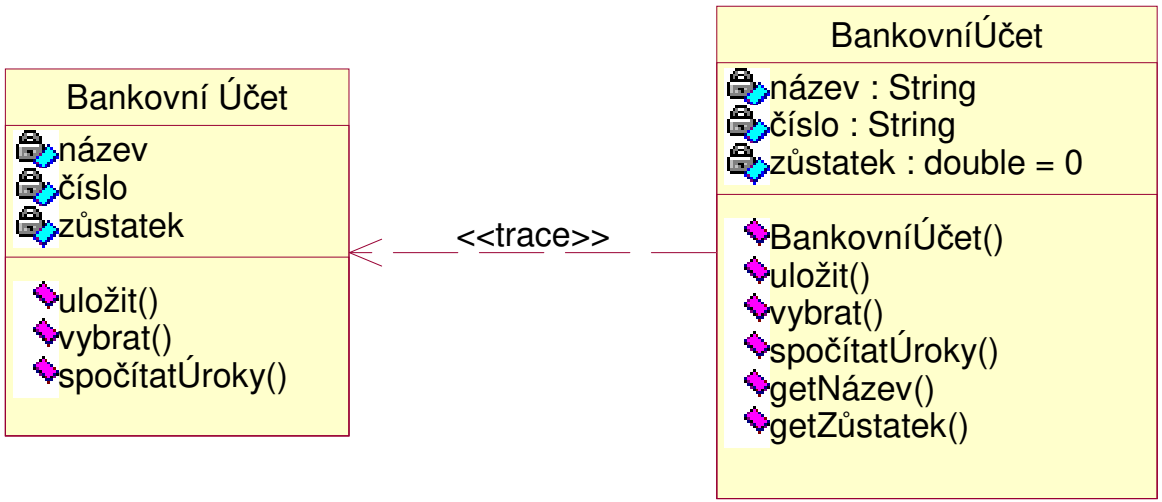
- upřesňování analytických diagramů
  - návrhové třídy
  - realizace případů užití
  - ...

Tahle faze je cela o implementaci systemu a jeho casti samosebou to byva podrobnejsi klademe si otazku JAK

Návrhové třídy

Návrhové třídy

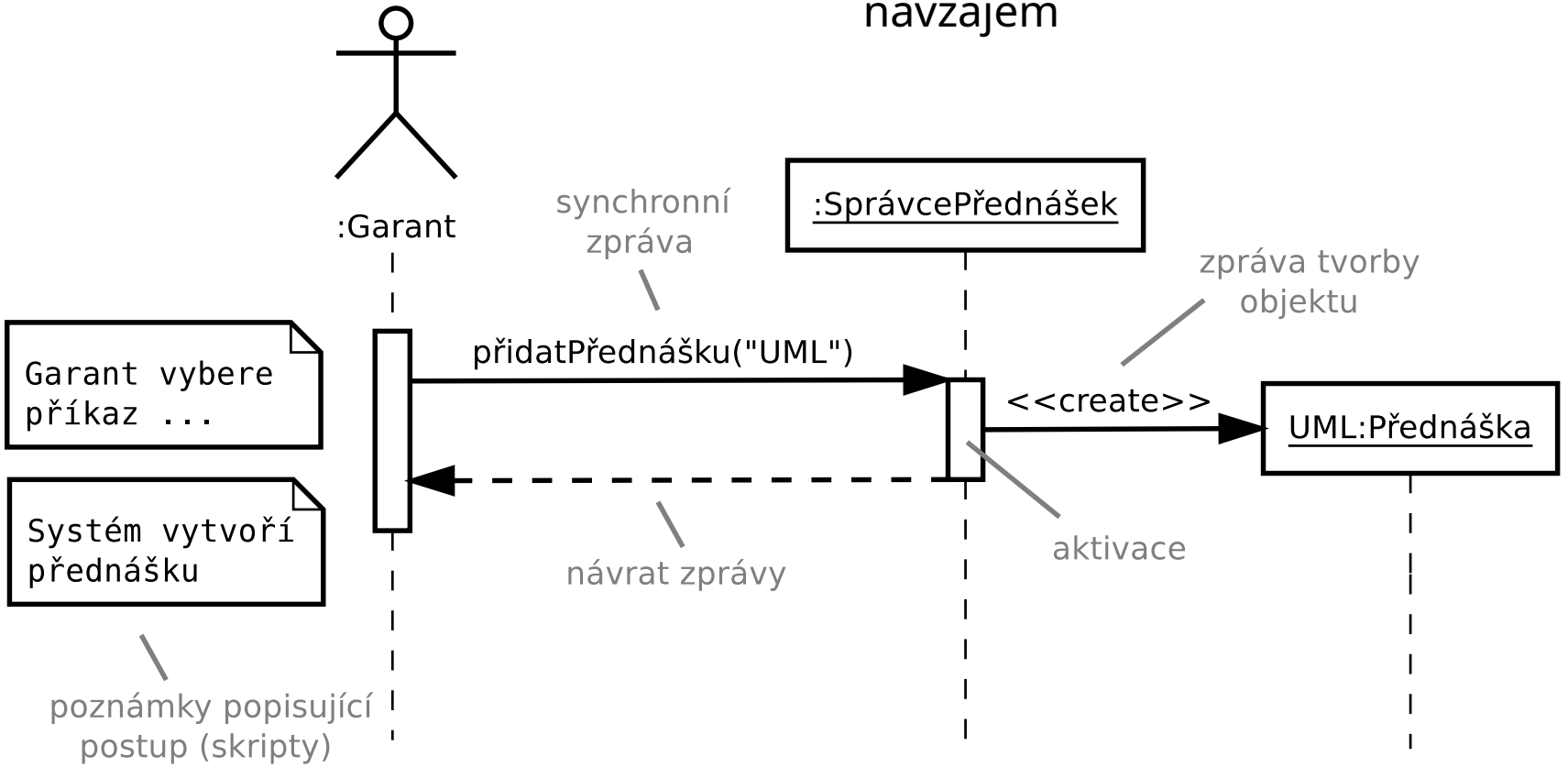
- specifikace návrhových tříd je na takovém stupni, že je lze přímo implementovat
- upřesňování analytických tříd
- využití tříd z doménového řešení
  knihovny, vrstva aplikačního serveru, GUI, ...



Upřesnění modelu v etapě návrhu

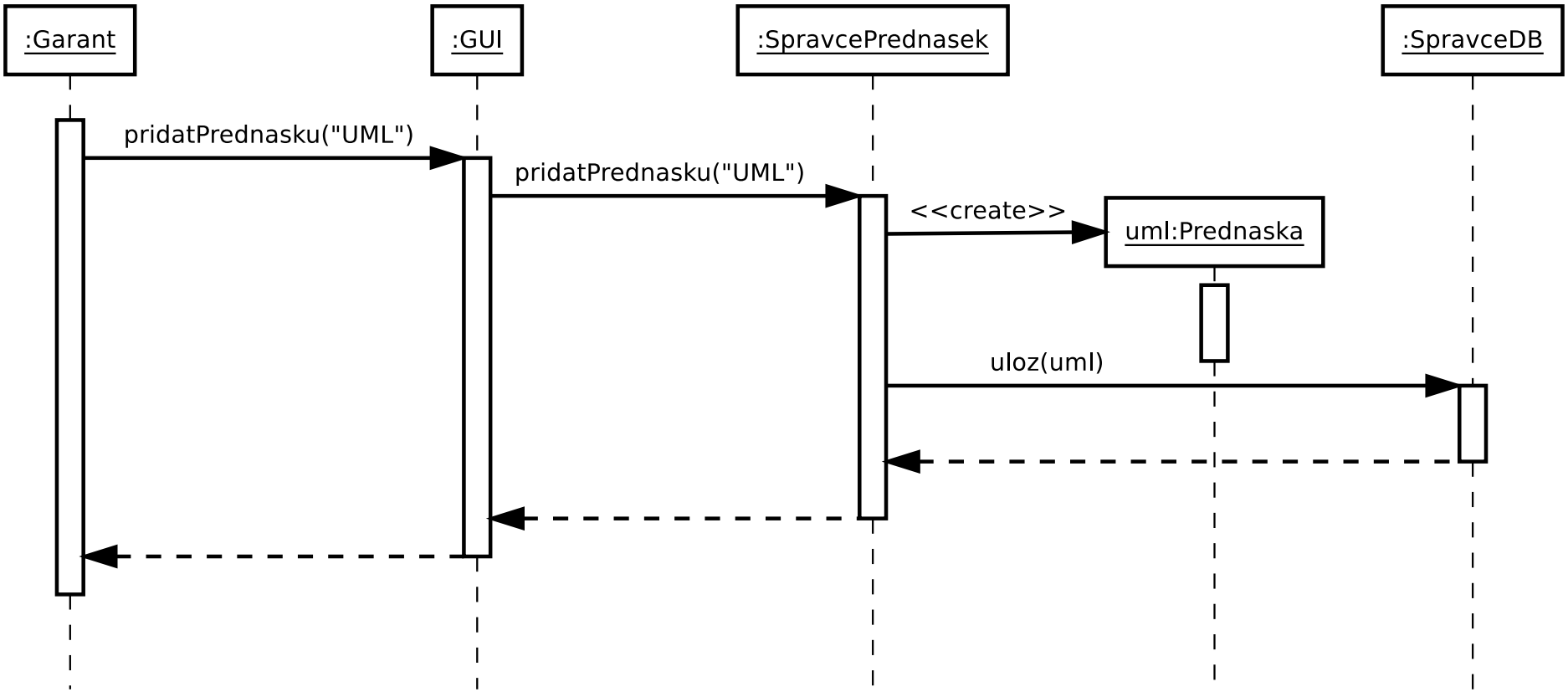
Analytický model interakce

Jak prejde z jedne casti do druhe tak diky vecem a poznatku kterych jsme nabyli v te casti musime specifikovat i tu prvni cast
idealne by ty dve casti se meli doplnovat navzajem



Upřesnění modelu v etapě návrhu

Návrhový model interakce



Upřesnění analytických relací

Asociace

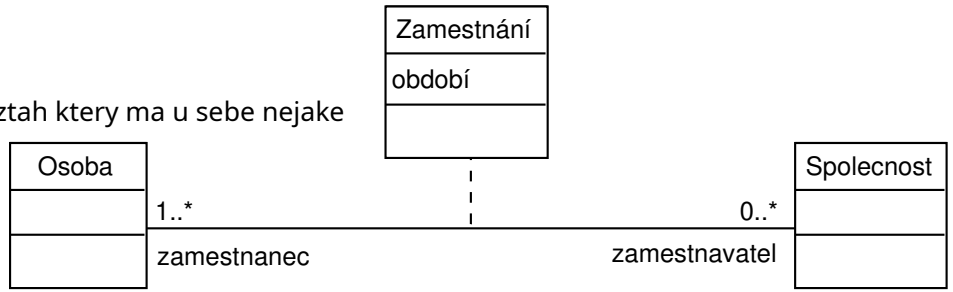
- agregace vs. kompozice
  - upřesnění vztahu celek/část
- asociace povýšená na třídu
  - návrh asociačních tříd
- asociace typu 1:N
  - realizace (nejčastěji kolekce)

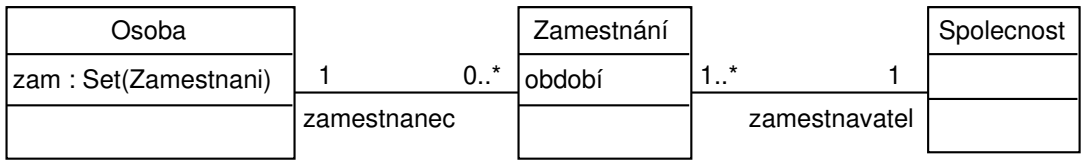
Agregace je vztah kdy je cast systemu volneji navazana na system a muze byt pouzita samostane a je schopna prezit system

Kompozice je naopak totalni diktatura casti nemuzou byti pouzity bez systemu a neprezijou jeho konce

Asociace typu 1:N je vztah kdy je jeden dominatni a ostatni davame do jedne skupina protoze maji podobne vlastnosti a potom vyjadrujme vztah dominatni ku skupine misto dominatni ku jednotlivci

Asociace povysena na tridu znamena kdyz mame vztah ktery ma u sebe nejake atributy a je dulezity povysujemeho na objekt





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

vytvorime si nareci

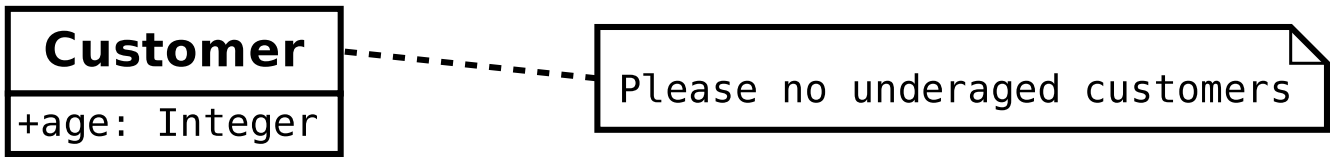
Označené hodnoty (tagged values)

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

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



tim ze v UML nejsme schopni vyjadrit vsechno v diagramech casti se musi popisovat slovne coz nam nevyhovuje protoze je zadouci aby to bylo jednoznacne coz slovni popis neni

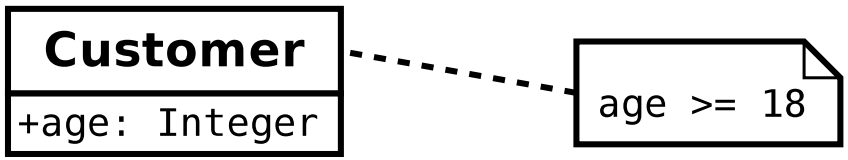
Jazyk UML – omezení a dotazy nad modely

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

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



- $\Rightarrow$ **jednoznačný jazyk** $\Rightarrow$ **OCL**

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*

Object Constraint Language

Object Constraint Language (OCL)

- definuje omezení, podmínky a dotazy nad UML modely
  $\Rightarrow$ zpřesňování modelů
- je *formální deklarativní* jazyk navržený pro návrháře
  $\Rightarrow$ 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
- ...

Object Constraint Language

Typy omezení

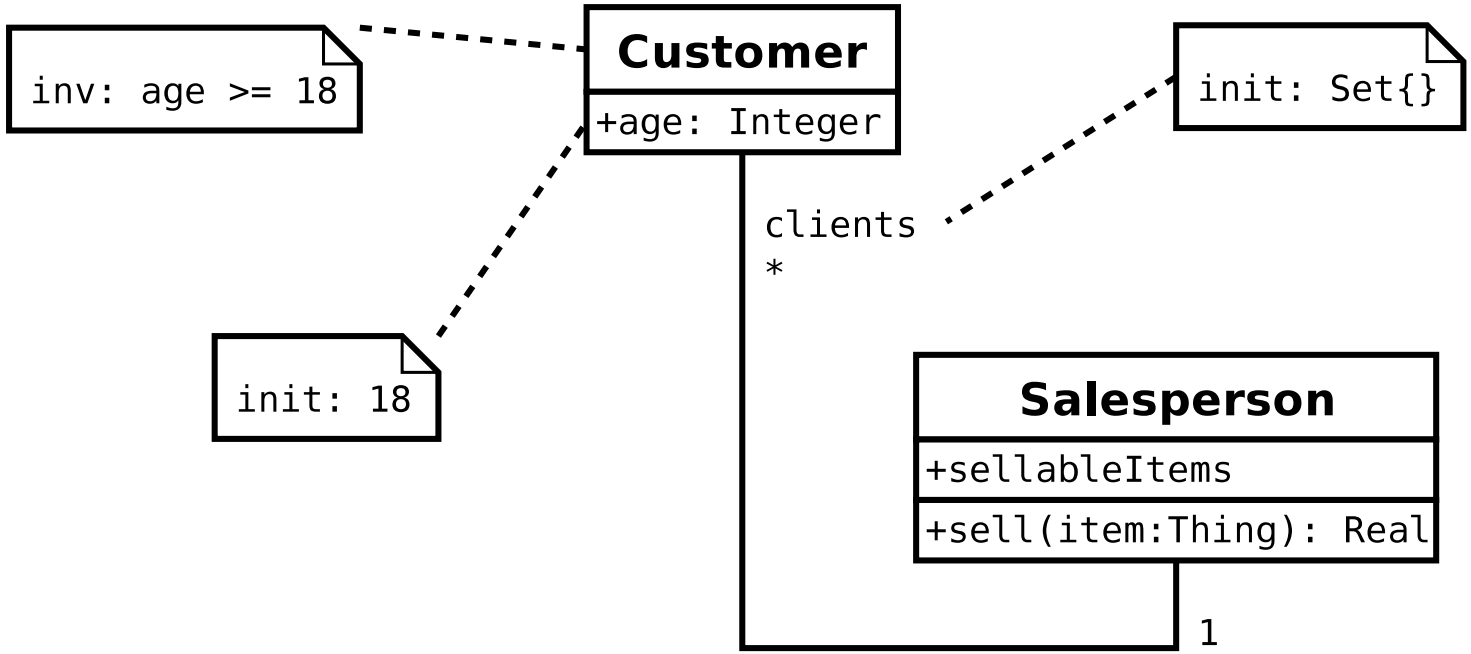
- *invariant* – podmínka, která musí být vždy splněna všemi instancemi
- *precondition* – omezení, které musí být pravdivé před provedením operace
- *postcondition* – omezení, které musí být pravdivé těsně po ukončení operace
- *guard* – omezení, které musí být pravdivé před provedením přechodu mezi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

Základní knihovna

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

Invariant zajistuje konzistenci mezi objekty
vsechny objekty customer maji musi mit vyssi vek nez 18 treba
Precondition urcity vztah je pravdivy pred prubehem operace aby operace jde provest
Postondition urcity vztah je pravdivy po dobehu operace aby bylo zajisteno ze vsechno probehlo spravne
Guard je zpustene pred prechodem mezi stavy tak aby bylo zajistenost stavoveho automatu

Object Constraint Language



Object Constraint Language

Ukázky

- invarianty atributů

  Jakoliv objekt Customer musi mit vysi vek nez 18

```
context Customer inv:
    age >= 18
```

- kolekce objektů (Salesperson $^{1}\rightarrow^{N}$ 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

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)
```

Znovupoužitelnost

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, . . .)

Je to takovy navod ale ne reseni na naz problem
pouze nas to ma navest spravnym smerem

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.*

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

vztah je zafixovan uz pri navrhu nejde s tim menit

Meni se za behu jelikoz se meni stavy samotnych objektu.

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

Jednota tvorby procesu aby byla zajistena efektivita

zabyvase skladbou samotne tridy a zajistuje aby meli vsechny spavnou strukturu

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

tovarna :) na vytvareni objektu jedne rodiny bez urcite specifikace ale stejne jako rodiny by spolu meli umet spolupracovat a komunikovat a zajistovat jednotu

Motivace

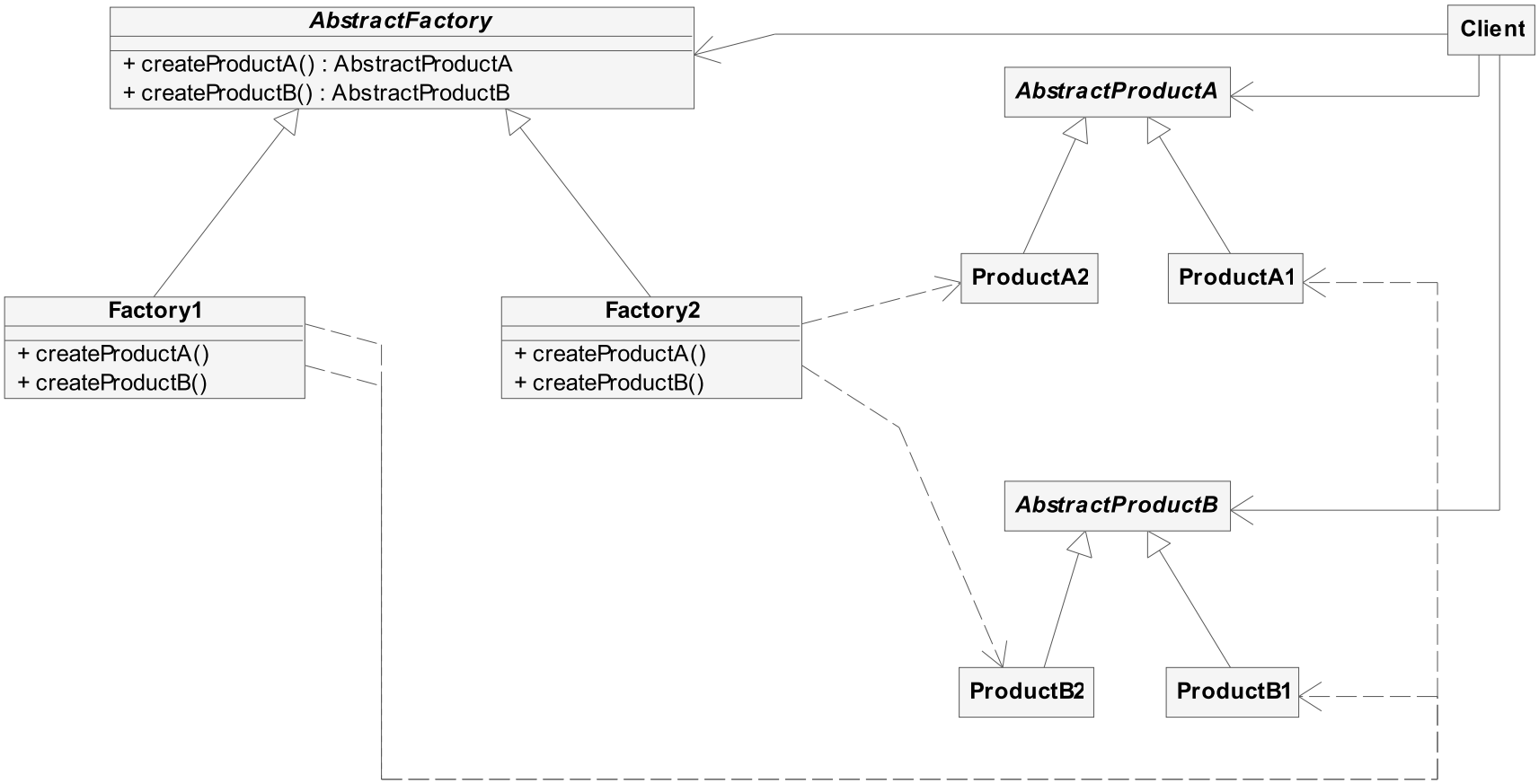
- např. změna vzhledu sady grafických nástrojů

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: 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

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
  ⇒ *nemožnost dynamické změny*
- vytvořit flexibilní kód ⇒ návrhový vzor *Abstraktní továrna*

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

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) { ... }
}
```

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);
```

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);
```

Command

tim ze je prikaz schovan jako objekt jsme schopniho jentak poslat a protejsi strana je schopna ho rozbalit a spracovat pomoci sveho protokolu a take nam je umoznena vvetsi volnost v manipulaci s prikaz

Účel

- zapouzdření požadavků nebo operací
- vzor chování

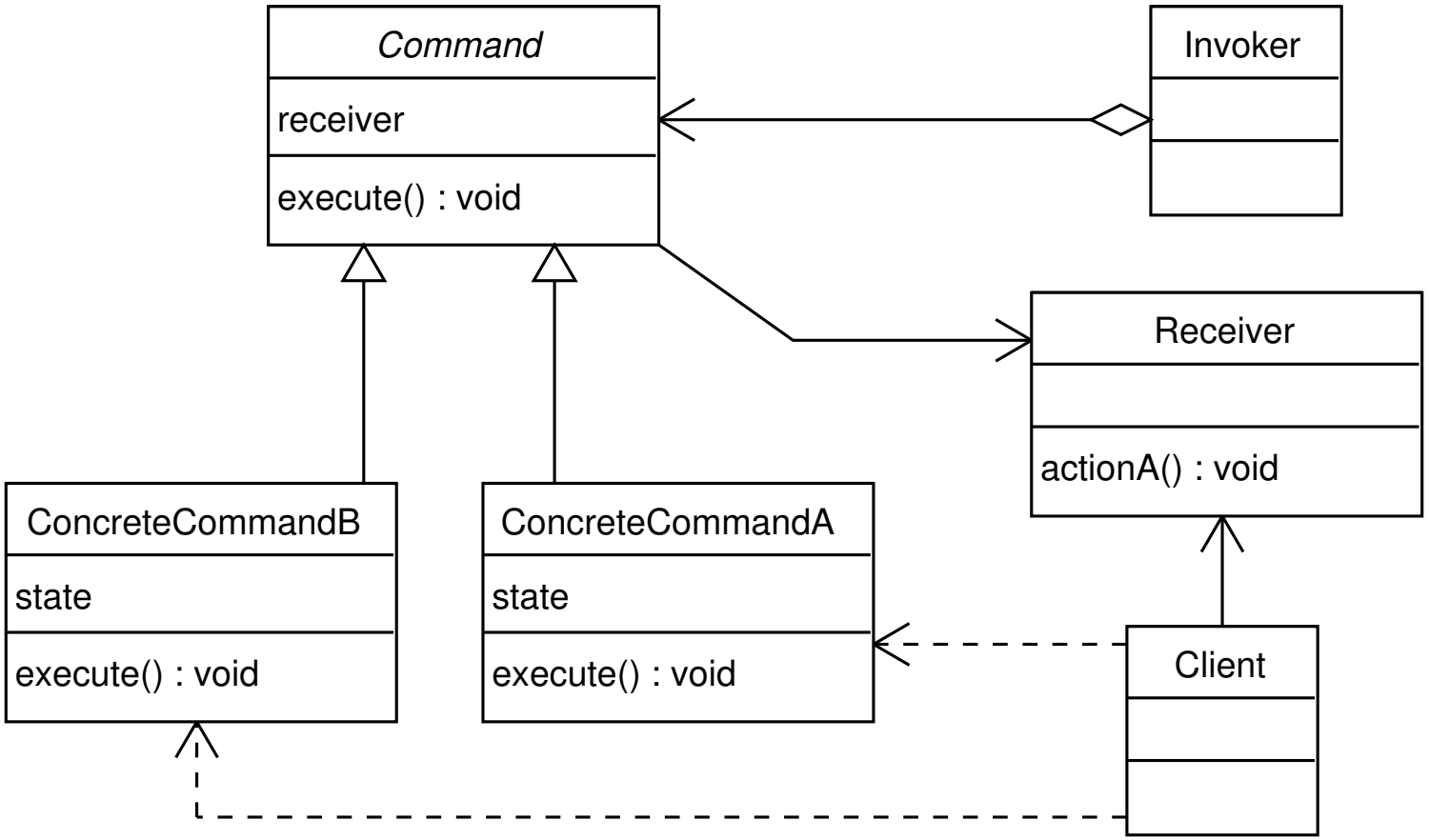
Motivace

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

Důsledky

- reprezentuje jeden provedený příkaz
- umožňuje uchovávat stav klienta před provedením příkazu
- ...

Command – Struktura



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();
}
```

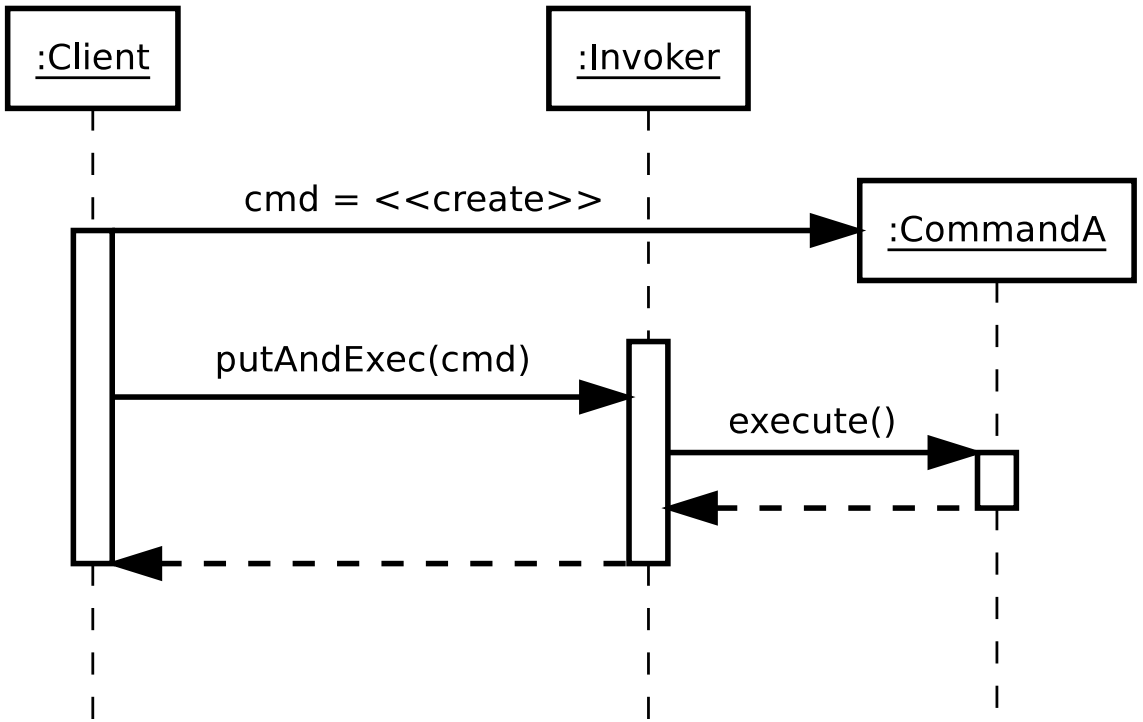
CommandA:

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

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

Command – Příklad

Ukázka sekvenčního diagramu pro popis chování.



Studijní koutek – Sociální bezpečí

Podoby sexuálního obtěžování na VŠ

- znásilnění či pokus o něj
- vynucování sexuálního chování za protihodnotu
  např. lepší podmínky u zkoušky
- nevyžádané opakované sexuální návrhy
- nevyžádané e-maily, fotografie nebo zprávy sexuální povahy
- nevyžádané dotyky
- nevhodné komentáře vůči jednotlivci nebo skupině na základě pohlaví, genderu nebo sexuality
- nevhodné vtipy o sexu a obscénní gesta

Více na

`https://www.vut.cz/vut/podpora-zamestnancu/socialni-bezpeci/studujici`

Studijní koutek – Na koho se obrátit?

- krizová situace
  - Linka první psychické pomoci – 116 123
  - Krizové centrum FN Bohunice – 532 232 078
- trestný čin – Policie ČR
- přestupek – Úřad MČ Brno-Královo Pole
- porušení etického kodexu VUT – Etická komise VUT
- Koordinátorka pro sociální bezpečí na VUT – Ing. Bohdana Šlégrová
- Kontaktní skupina pro sociální bezpečí FIT
  - `https://www.fit.vut.cz/study/social-safety/`
  - Jednotlivé případy neřeší všichni členové skupiny.
  - Obvykle děkan, tajemník, pověřenec pro sociální bezpečí (Křena) a kontaktovaná osoba.

Studijní koutek – Šetření a postihy

- trestný čin – Policie ČR
- přestupek – Úřad MČ Brno-Královo Pole
- porušení etického kodexu VUT – Etická komise VUT
- disciplinární přestupek – Disciplinární komise FIT
- porušení Zákoníku práce
  - porušení pracovní povinnosti zvlášť hrubým způsobem
    okamžité zrušení pracovního poměru
  - závažné porušení pracovní povinnosti
    výpověď
  - méně závažné porušení pracovní povinnosti
    upozornění a výpověď až při opakování