

# UML

## Unifikovaný modelovací jazyk Teorie a Praxe

---

Karel Richta  
katedra počítačů  
ČVUT FEL Praha  
richta@fel.cvut.cz

Jiří Svačina  
Unicorn a. s.  
Praha  
jiri.svacina@unicorn.cz

# Už jste někdy stavěli psí boudu?

---



Úkol pro jednotlivce  
Vyžaduje

Minimum modelování  
Jednoduchý proces  
Jednoduché nástroje

# Co takhle rodinný dům?



Dosažitelné včas pomocí týmového úsilí  
Vyžaduje  
Modelování  
Dobře definovaný proces  
Sofistikované nástroje

# Obsah tutoriálu

---

- V první části tutoriálu bude prezentována syntaxe UML. V této části by se měl posluchač dozvědět základní informace o tom, jak vytvářet diagramy v UML.
- Ve druhé části se budeme zabývat specifikačním jazykem OCL (Object Constraint Language), který slouží např. pro vyjádření komplikovanějších integritních omezení.
- Třetí část tutoriálu je zaměřena na praktické využití UML, tj. bude zde uveden příklad projektu - aplikace, která bude dokumentována pomocí prostředků UML.

Část I.:

UML – syntaxe a sémantika



# Obsah

---

- Evoluce OO přístupu
- Co obsahuje UML
- Diagramy UML

# Nejprve OO odhalili programátoři

---

- **Simula'67** (1962-67, poprvé zde byly uplatněny OO koncepty jako třída, objekt, dědění a dynamicky vázané metody)
- **Smalltalk** (1972-80, první jazyk založený zcela na konceptu komunikujících objektů, novinkou je též virtuální stroj jako interpret "byte code")
- **C++** (1983, 1998 - ISO, OO rozšíření C, novinkou je zpracování výjimek)
- **Eiffel** (1986, novinkou jsou kontroly invariantů, generické parametry, design-by-contract)
- Atd.

# Pak se do toho vložili metodici

---

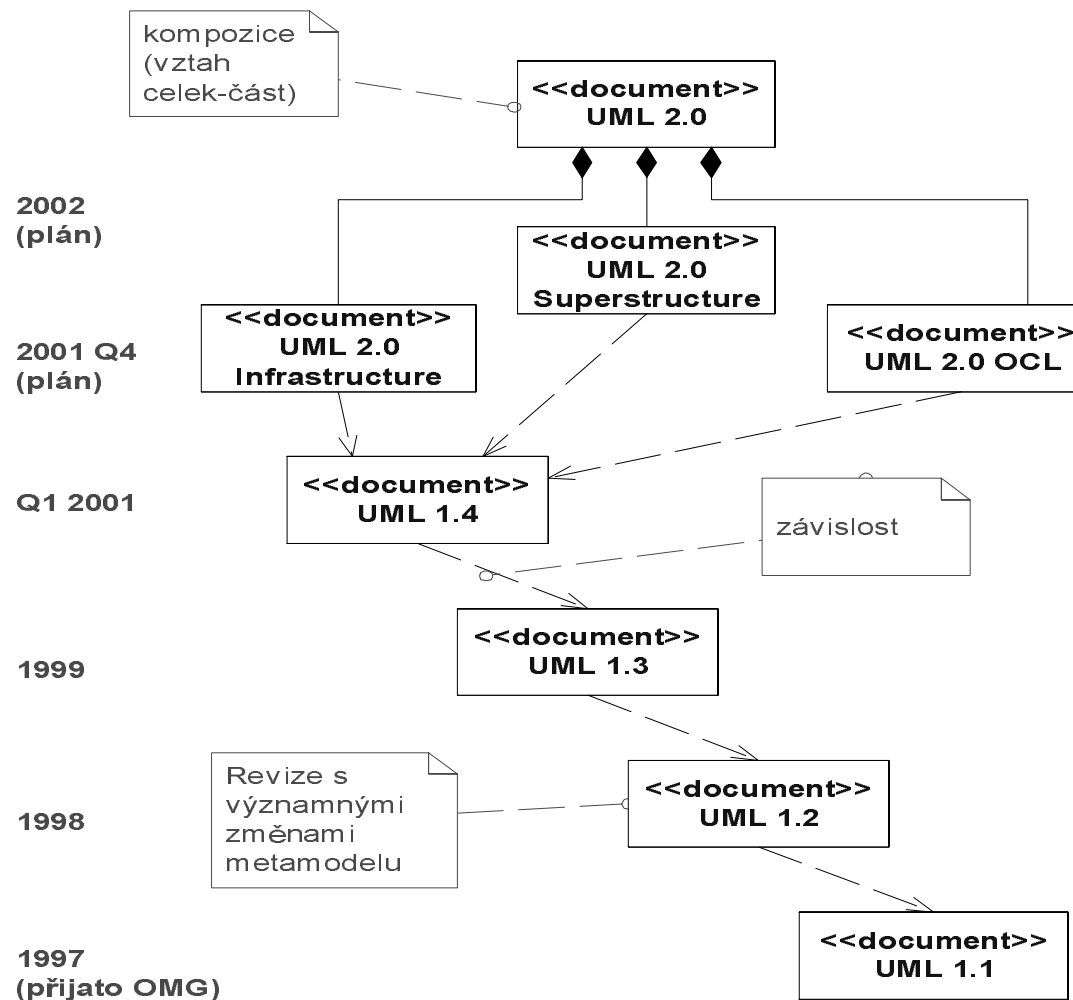
- 1. generace OO metodik:
  - Rumbaugh - OMT (diagramy tříd)
  - Coad/Yourdon - OOAD
  - Martin - OOIE
  - Booch - OOAD (diagramy komponent)
  - Wirfs/Brock - RDD (model odpovědností)
  - Jacobson - OOSE (model jednání - use cases)
- Každá metodika přinesla něco nového, byla v něčem lepší, ale zase něco postrádala ⇒ snaha o kombinaci výhod

# A zůstali u toho

---

- 2. generace OO metodik (kombinace výhod metodik 1.generace):
  - Coleman - Fusion (kompletní cyklus)
  - Rumbaugh/Booch - UM (Unified Method 0.8)
- Zůstaly rozdíly v prezentačních technikách, každá metodika používá svou notaci ⇒ snaha o unifikaci
- 3. generace (unifikace prezentačních technik):
  - Rumbaugh/Booch/Jacobson - UML Unified Modeling Language 1.0 - 1996)

# Historie UML

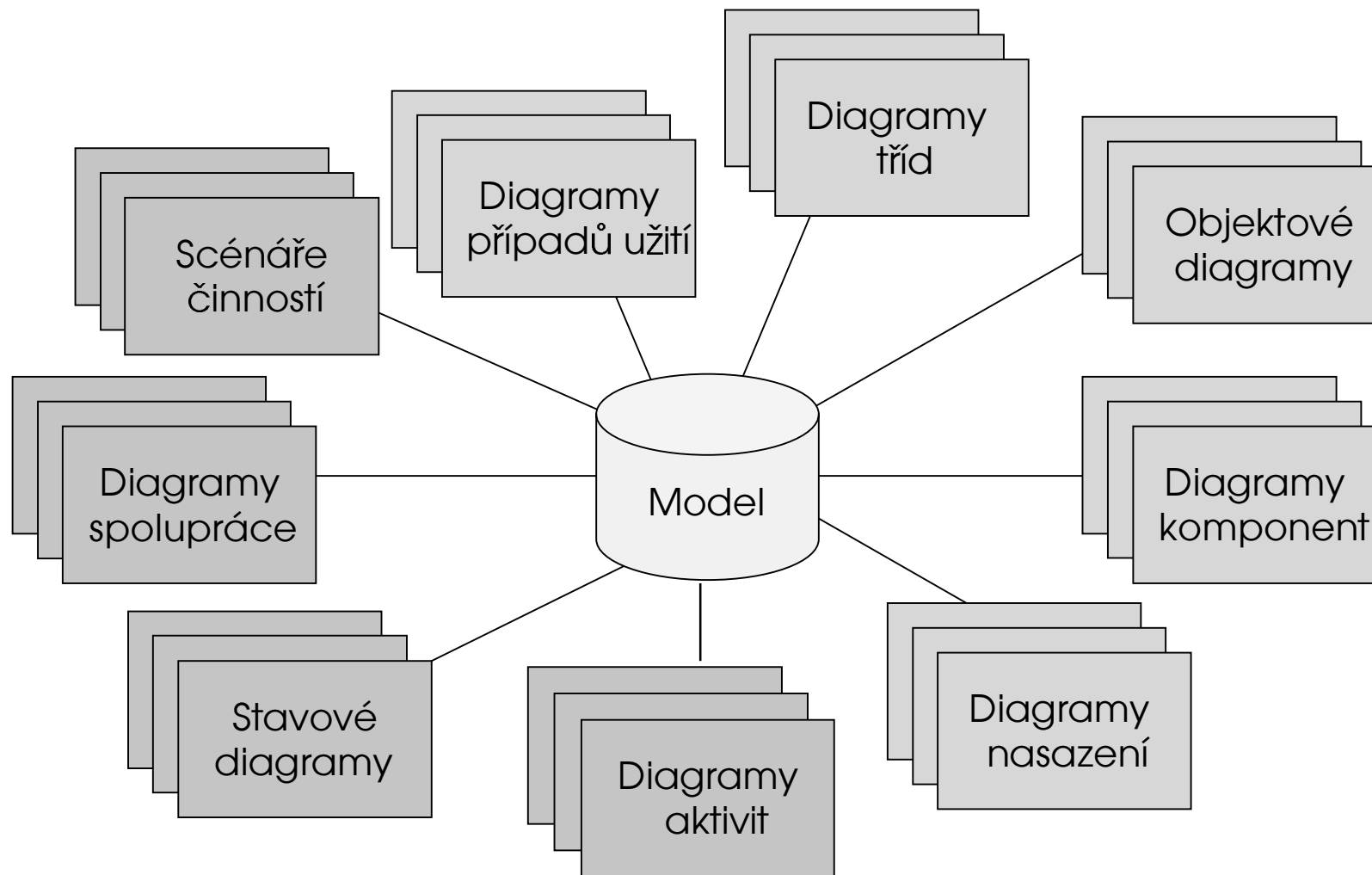


# Co to je UML?

---

- **UML** (Unified Modeling Language) je „standard **OMG** (Object Management Group) pro záznam, vizualizaci a dokumentaci artefaktů systémů s převážně softwarovou charakteristikou“.
- **Definice UML zahrnuje:**
  - Notaci UML (syntaxe)
  - Metamodel UML (sémantika)
  - OCL (Object Constraint Language)
  - Specifikaci převodu do výměnných formátů (CORBA IDL, XML DTD)

# Notace UML (diagramy)



# Diagramy UML

---

- **Diagramy tříd** popisují statickou strukturu systému, znázorňují model systému od konceptuální úrovně až po implementaci
- **Model jednání** dokumentuje možné případy užití systému, požadavky na funkčnost, kterou má systém podporovat
- **Scénáře činností** popisují průběh určité činnosti v systému
- **Diagramy spolupráce** zachycují komunikaci spolupracujících objektů
- **Stavové diagramy** popisují dynamické chování objektu nebo systému
- **Diagramy aktivit** popisují průběh procesu či posloupnost činností
- **Diagramy komponent** popisují rozdělení výsledného systému na funkční celky (komponenty) a definují náplň jednotlivých komponent
- **Diagramy nasazení** popisují umístění funkčních celků (komponent) na výpočetních uzlech softwarového systému

# Směr pohledu na systém

---

- **Statickou strukturu** systému vyjadřují diagramy tříd, diagramy komponent a diagram nasazení
- **Funkční stránku** popisují model jednání, scénáře činností, diagramy spolupráce a diagramy aktivit
- **Dynamickou stránku** dokumentují scénáře činností, diagramy spolupráce, stavové diagramy a diagramy aktivit

# Použití diagramů (orientační)

---

- Analýza -
  - Diagramy tříd, model jednání, scénáře činností, stavové diagramy, diagramy aktivit
- Návrh -
  - Diagramy tříd, scénáře činností, diagramy spolupráce, diagramy aktivit, diagramy komponent, diagramy nasazení
- Implementace -
  - Diagramy tříd, diagramy komponent, diagramy nasazení

# Lexikální elementy UML

---

Čtyři základní druhy elementů:

- **Řetězec** (posloupnost znaků, znaková sada není omezena, v některých případech doporučeno ASCII)
- **Ikona** (grafický symbol zastupující element, neobsahující žádné složky)
- **2D-symbol** (grafický symbol zastupující element, který má obsah, případně rozdělený na oblasti)
- **Spojka** (path - posloupnost úseček, které navazují, mohou mít zvýraněny koncové body a mohou incidovat s hranicí 2D-symbolů - slouží k vyznačení propojení)

# Identifikace elementů UML

---

- Elementy potřebujeme označovat - potřebujeme jména objektů, tříd, atributů, metod, ...
- **Jméno** = identifikátor
  - `josefNovak`, `josef_novak`, `josef novak`
- **Kvalifikované jméno** - jméno se vztahuje k nějakému kontextu (scope), explicitně:
  - `Ucetnictvi::Objednavka`
- **Návěští** - řetězec přidružený ke grafickému symbolu
- **Klíčové slovo**
  - `<<keyword>>`

# Výrazy v UML

---

- Výrazy v UML jsou řetězce sestavené podle jistých pravidel (syntaxe výrazů není striktní, je možné používat OCL)
- Výraz představuje formuli, kterou musí být možno v daném jazyce vyhodnotit (má sémantiku)
- Definice UML zahrnuje definici OCL (Object Constraint Language), který je použit pro definici sémantiky UML (metamodel UML)

# Navigace v OCL

---

## **objekt.selektor**

- označuje položku daného objektu (atribut, jméno role)

## **objekt.selektor [ kvalifikátor ]**

- označuje objekt vybraný podle kvalifikátoru (zobecněné pole)

## **sada -> select ( logická podmínka )**

- označuje objekty vybrané z dané sady podle logické podmínky:

**Firma.zamestnanci->select(funkce="ucetni")**

# Rozšiřitelnost UML

---

- UML se snaží být univerzální notací pro všestranné použití od obchodního modelování po detailní návrh systémů pro práci v reálném čase
- Notace, která by vyčerpávajícím způsobem pokryla tak široké spektrum potřeb, by ale byla velmi komplikovaná a složitá
- UML proto definuje pouze základní výrazové prostředky (UML core) a umožňuje rozšíření notace dle konkrétních potřeb

# Nástroje rozšiřitelnosti UML

---

- Standardní sémantiku UML je možno doplnit standardními omezeními, která umožňují různou interpretaci elementů:
  - Příznaky (tags) – umožňují přidat k prvku diagramu další informace
  - Stereotypy (stereotypes) – umožňují klasifikovat elementy
  - Omezení (constraints) – umožňují specifikovat omezení pro elementy

# Příznaky (tags)

---

- Umožňují přidat k prvku diagramu další informace ve formě dvojice

**{ název atributu = hodnota }**

- Název atributu může být libovolný
- Příznaky se zapisují do složených závorek, připojují se k názvu prvku diagramu a typicky je lze použít například ke specifikaci verze, autora, či implementačních poznámek

**{ autor="Josef Novák", verze=1.2 }**

# Stereotypy (stereotypes)

---

- Umožňují klasifikovat elementy diagramů (a tím vyjádřit další sémantiku)
- Zapisují se jako klíčová slova (mohou se případně zobrazit jako ikony, což se příliš nedoporučuje - porušuje to princip jednotnosti)
- Příklady:
  - `<<database>>` (označení komponenty)
  - `<<entity>>` (označení třídy reprezentující data)
- Existují standardní stereotypy, např.:
  - `<<include>>`, `<<extend>>`

# Omezení

---

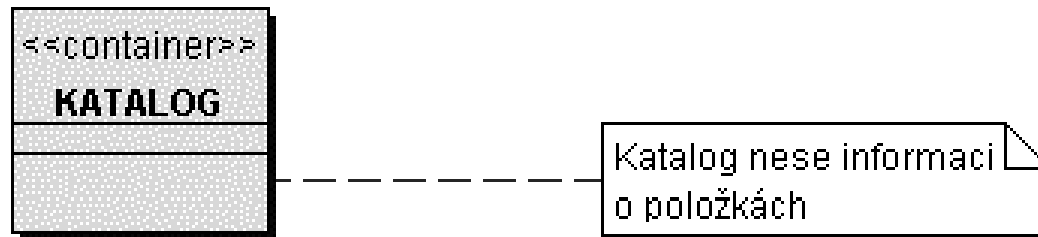
- Umožňují specifikovat požadavky na sémantiku prvků - lze pomocí nich formulovat různá omezení v modelu
- Zapisují se jako výraz uzavřený do složených závorek
- Příklad:  

```
{ počet_zaznamu < 10000 }
```
- Omezení se může týkat více prvků - pak je spojeno s těmito prvky čárkovanou čarou

# Poznámky

---

- Poznámka obsahuje libovolný text
- Zobrazuje se jako obdélník s „přehnutým rohem“
- Může být přidružena k elementu
- Může obsahovat stereotyp (např. **<<constraint>>** - pak se jedná o specifikaci omezení)

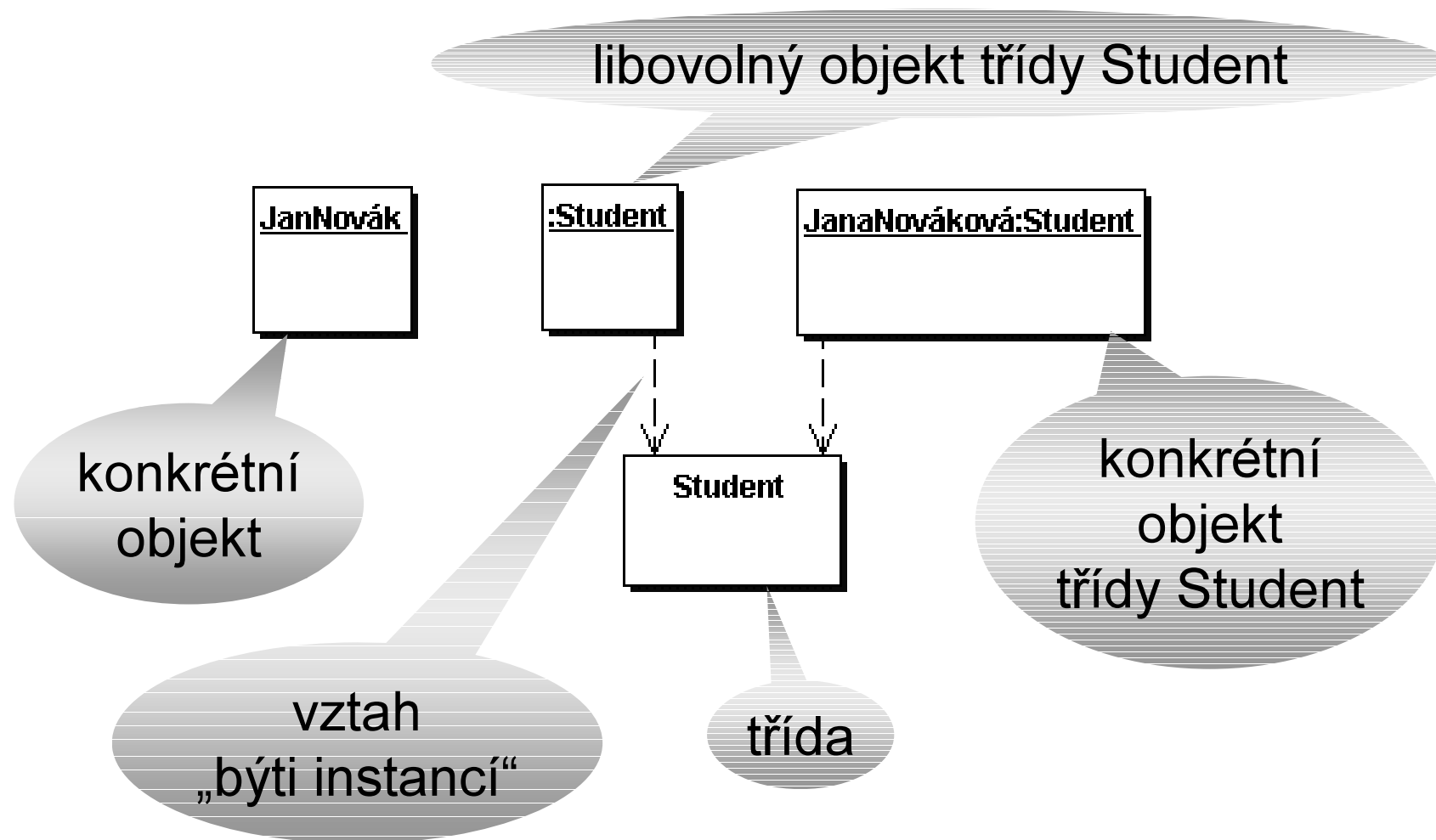


# Diagramy objektů

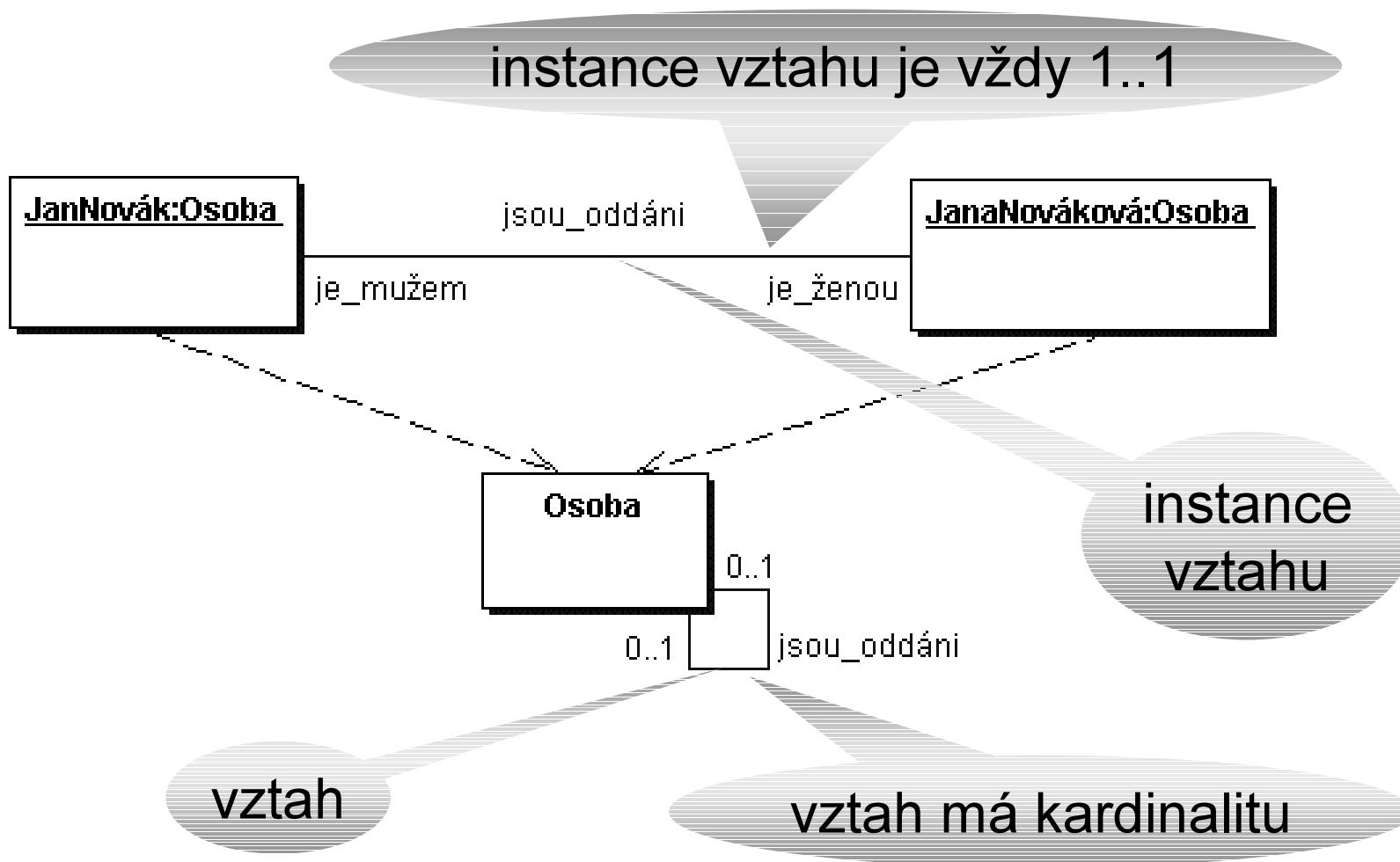
---

- **Objekt** je pojem, abstrakce, nebo věc s dobře definovanými hranicemi a významem
- Každý objekt má tři charakteristiky: identitu, stav a chování.
  - **Stav** objektu je jedna z možných situací, ve kterých se objekt může nacházet. Stav objektu se může měnit a je definován sadou vlastností (atributů) a vztahy.
  - **Chování** určuje, jak objekt reaguje na žádosti jiných objektů a vyjadřuje vše, co může objekt dělat. Chování je implementováno sadou operací (metod).
  - **Identita** znamená, že každý objekt je jedinečný.

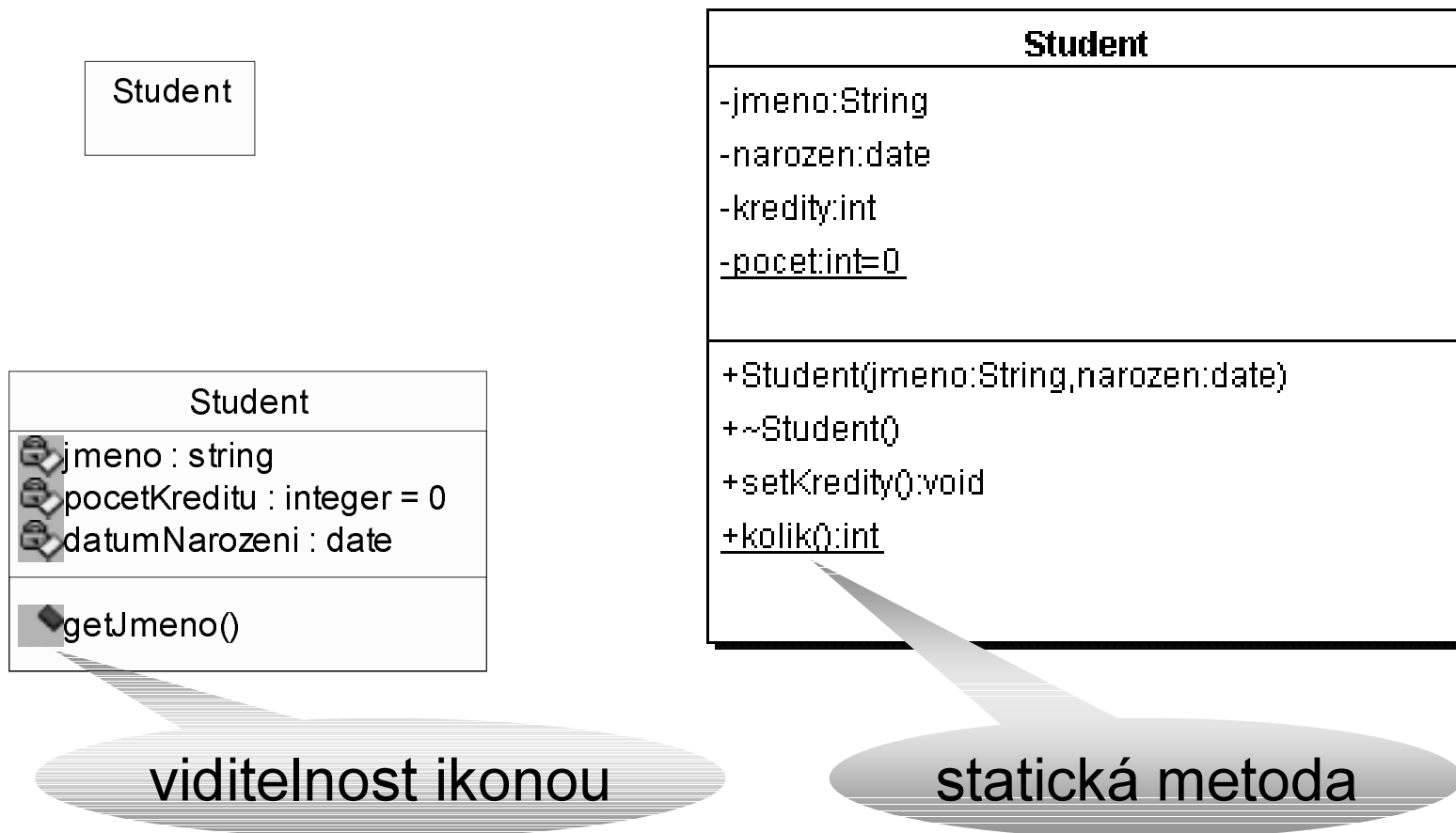
# Notace objektů



# Vztahy mezi objekty



# Diagramy tříd



# Notace tříd

---

Pro definici **atributů** se používá notace:

```
viditelnost jméno [ násobnost ] : typ  
= počáteční hodnota { omezení, vlastnost }
```

Pro definici **operací** (metod) se používá notace:

```
viditelnost jméno(druh jméno: typ, ...) : typ  
{ omezení, vlastnost }
```

Viditelnost: + ... public # ... protected - ... private

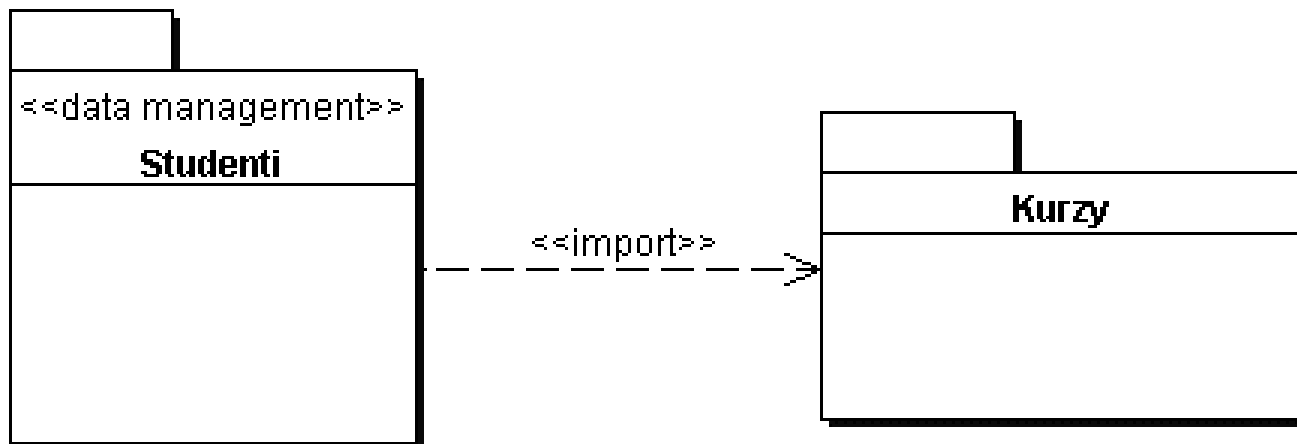
Násobnost: [ a..b ] ( [ 0..1 ] může být NULL)

Druh: in, out, inout (implicitně in)

# Balíky (packages)

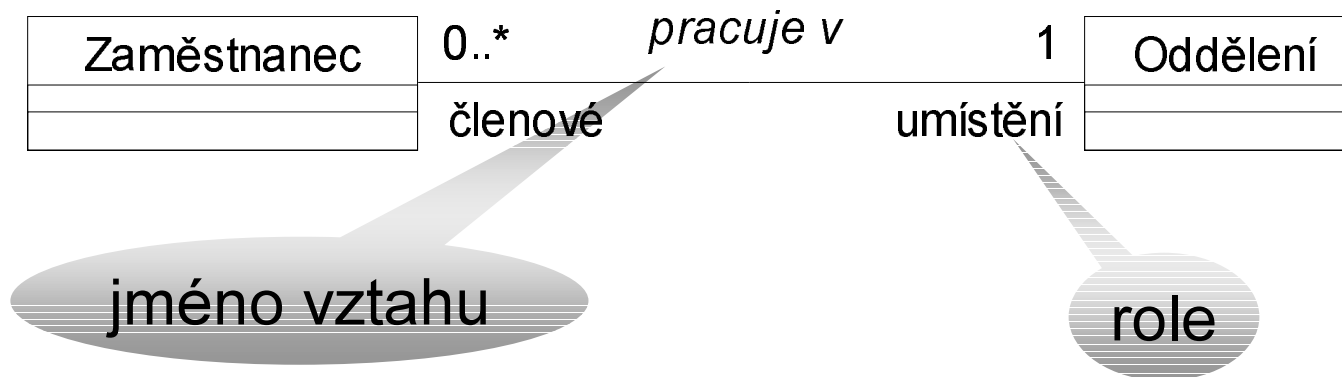
---

Logická skupina elementů



# Vztahy mezi třídami

- **Vztah (asociace)** - vyznačení možného vztahu mezi objekty
- **Role** – označení konce vztahu, které říká, jakou roli třída (objekt) ve vztahu hraje



# Kardinalita a volitelnost vztahů

---

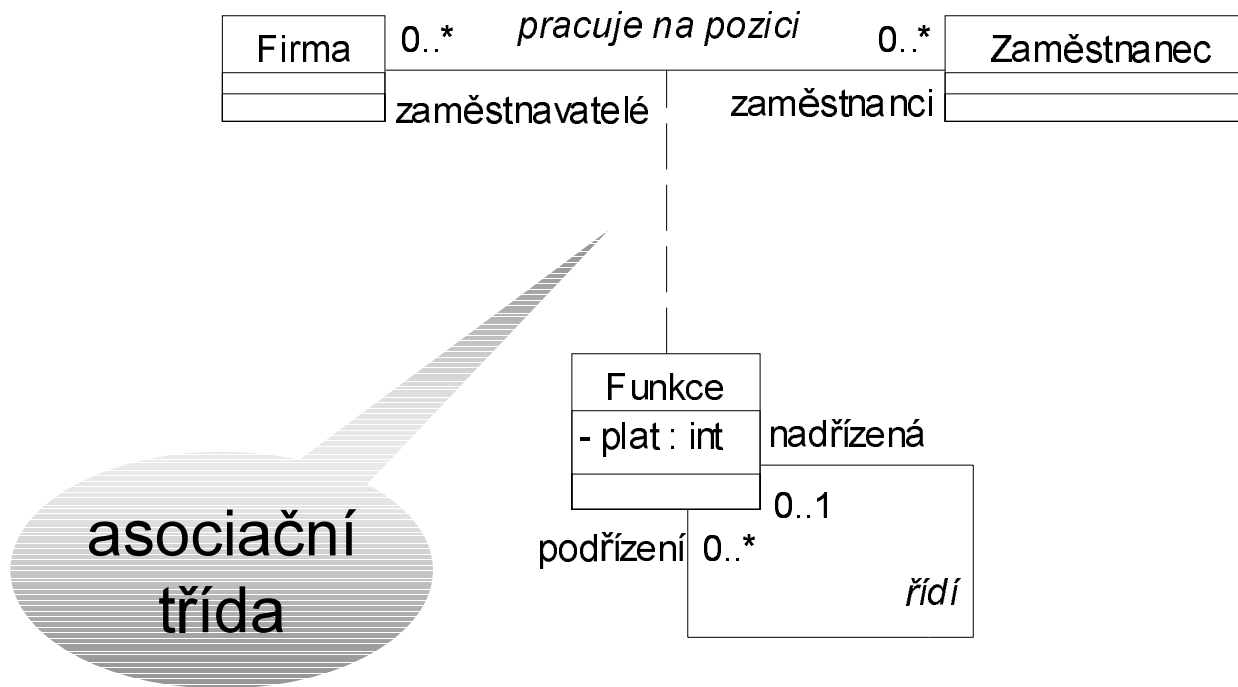
Používá se notace:

**N..M**

kde N a M může být číslo nebo \*

- Vztah nemá atributy, ale může být popsán asociační (přidruženou) třídou

# Příklad asociační třídy

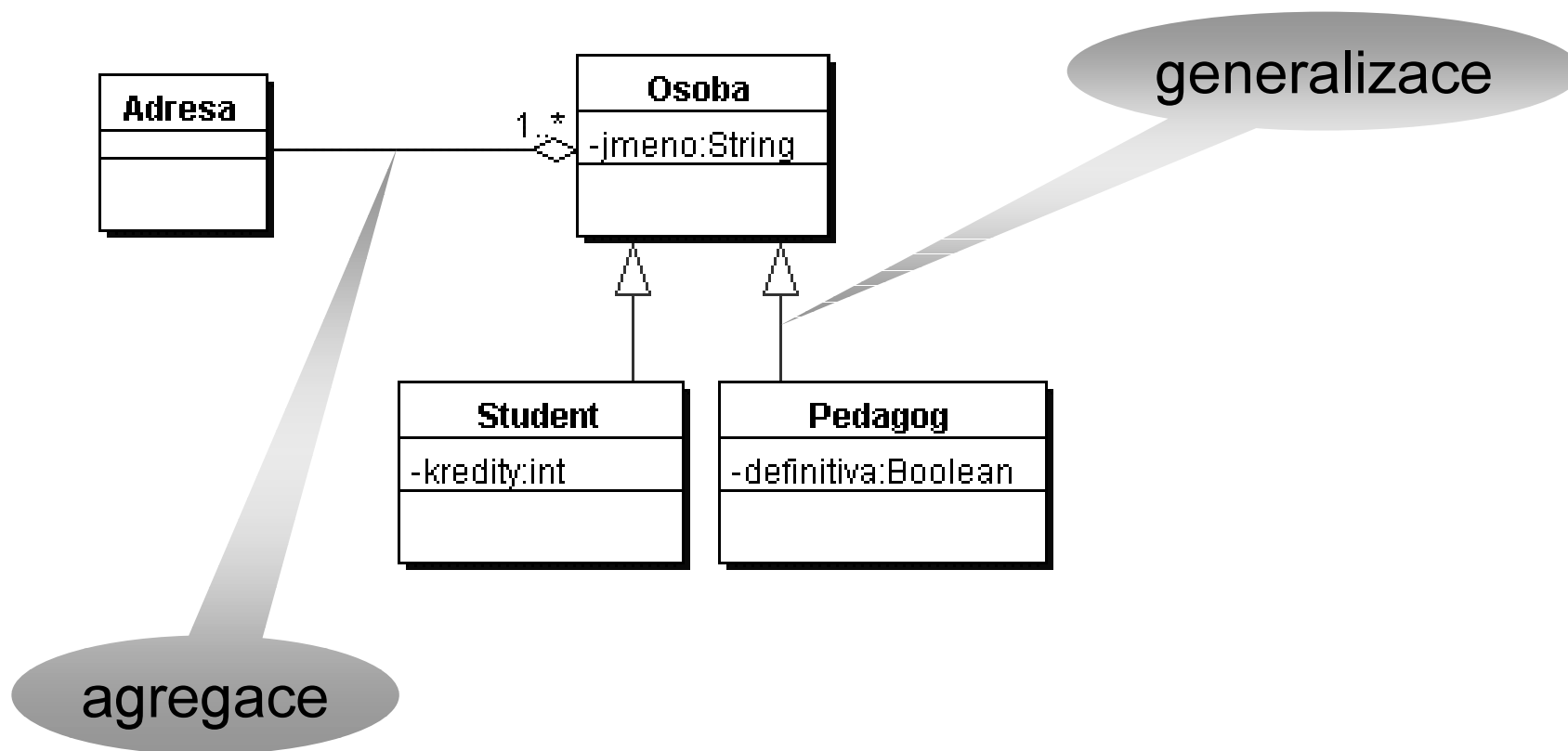


# Agregace a generalizace

---

- **Agregace** (*aggregation*) – druh vztahu, kdy jedna třída je součástí jiné třídy - vztah typu celek/část
- **Kompozice** (*composition*) – silnější druh agregace - u kompozice je část přímo závislá na svém celku, zaniká se smazáním celku a nemůže být součástí více než jednoho celku
- **Generalizace** (*generalization*) – druh vztahu, kdy jedna třída je zobecněním vlastností jiné třídy (jiných tříd) - vztah typu nadtyp/podtyp, generalizace/specializace

# Příklad agregace a generalizace



# Další možnosti

---

- Kvalifikované vztahy
- Navigované vztahy
- Odvozené atributy a vztahy (předznačí se '/')
- Parametrizované třídy (templates)
- Abstraktní třídy
- Interface
- ...

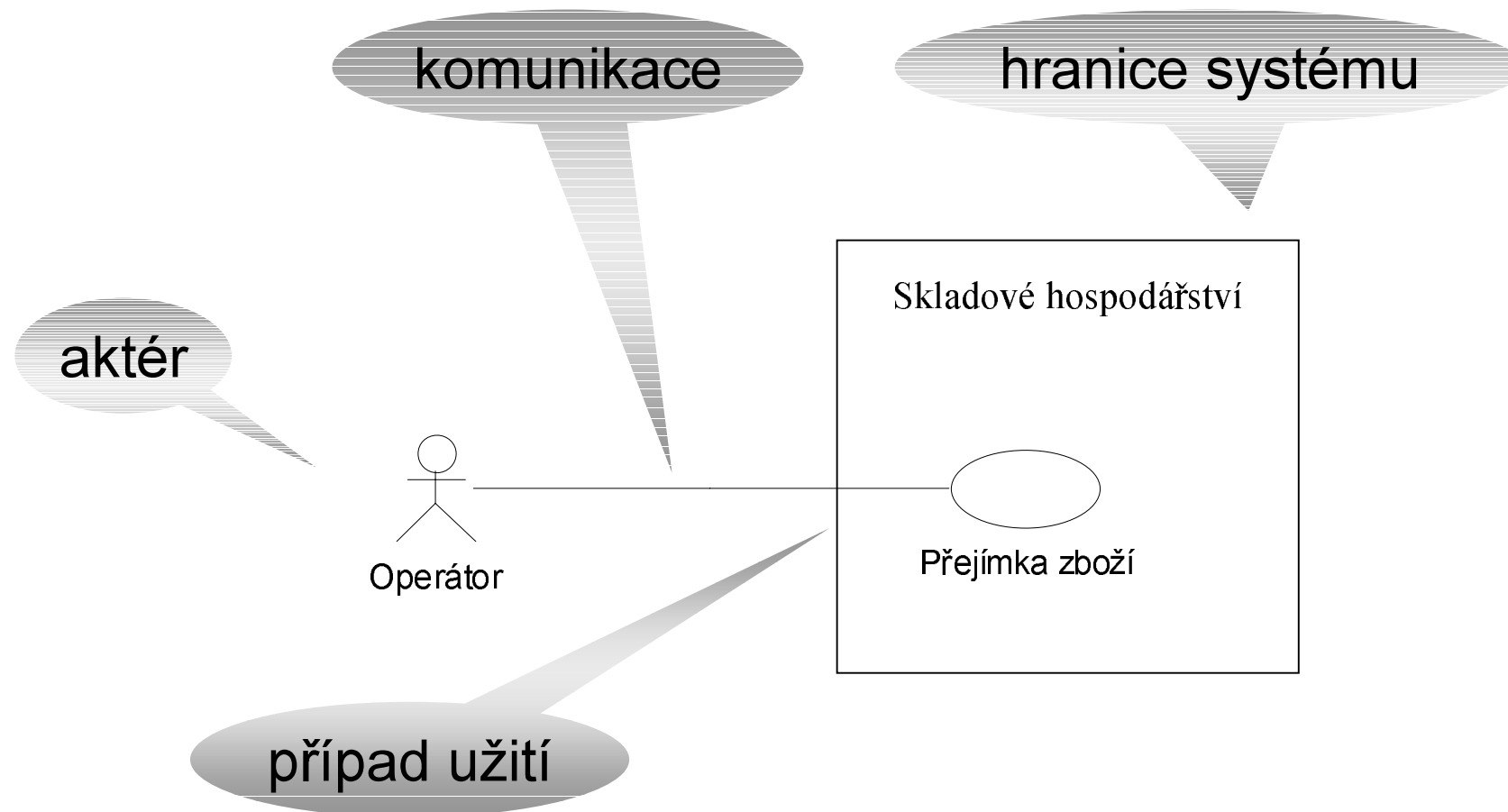
# Model jednání (Use Case Model)

---

Prvky:

- **Aktér (actor)** - role uživatele, jiného systému nebo zařízení
- **Hranice systému (system boundary)** - vymezení hranice systému
- **Případ užití (use case)** - popis funkčnosti, kterou má systém podporovat
- **Komunikace** - vazba mezi aktérem a případem použití (aktér komunikuje se systémem v rámci daného případu užití)

# Notace modelu jednání

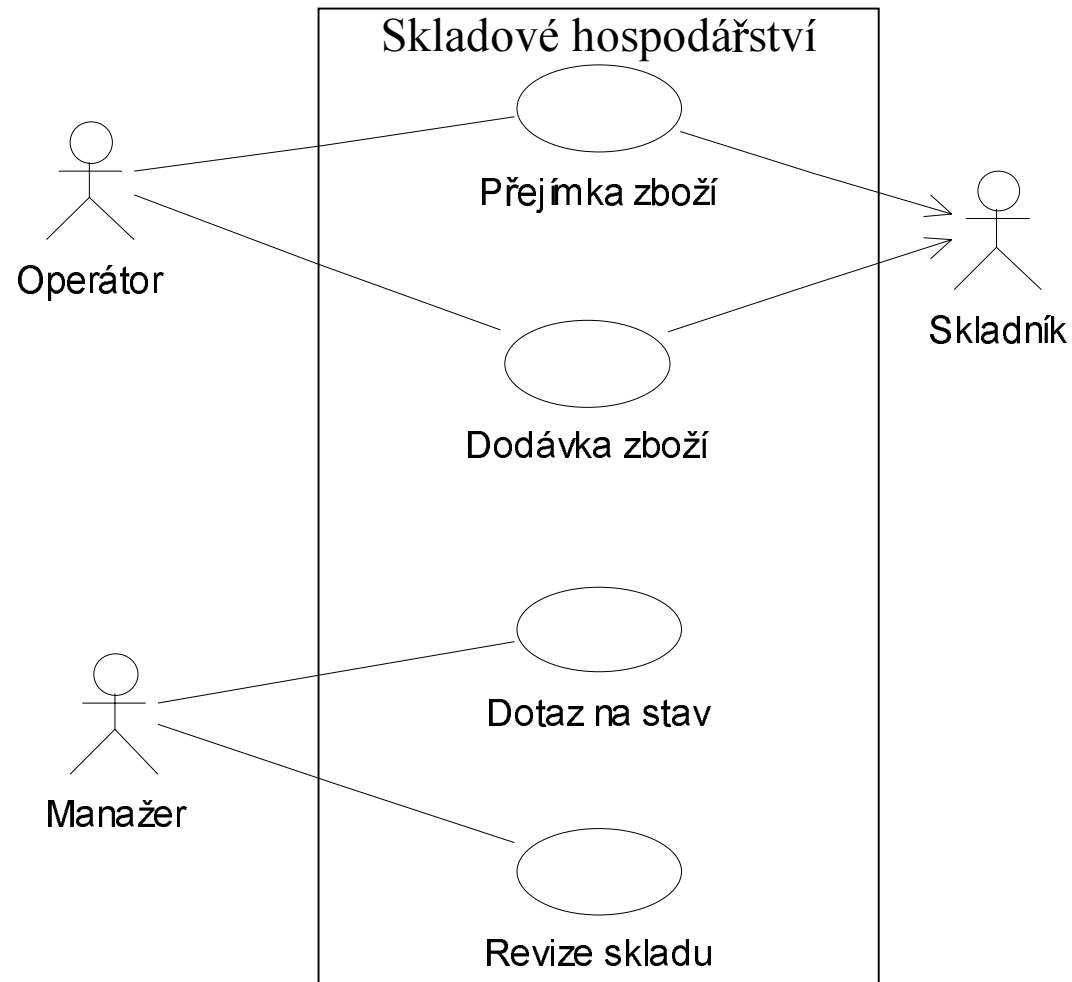


# Varianty v modelu jednání

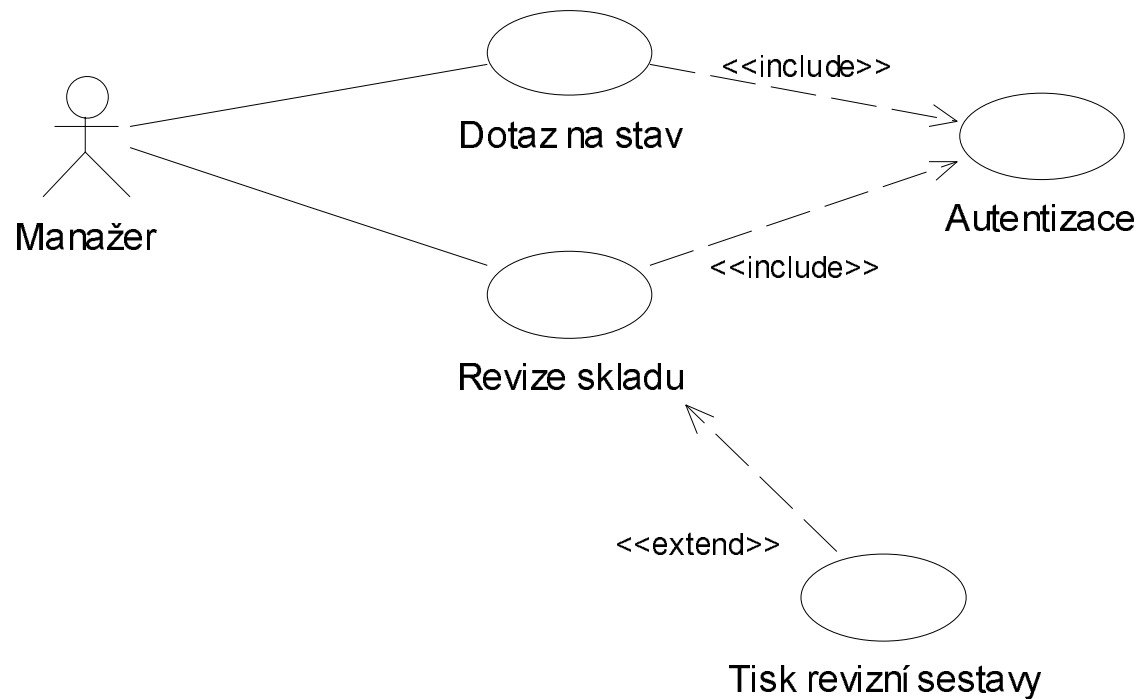
---

- **Sekundární aktér** - uživatelská role nebo spolupracující systém nutná pro činnost systému
- **Orientovaná komunikace** - případ, kdy chceme vyznačit iniciaci interakce
- **Vztahy mezi případy užití** - pokud chceme strukturovat model tak, aby byl lépe udržovatelný
  - **<<include>>** - pokud jeden případ zahrnuje případ jiný (např. autentizace)
  - **<<extend>>** - pokud nějaký případ rozšiřuje chování (je zde možnost volby)

# Příklad modelu jednání



# Vztahy mezi případy užití

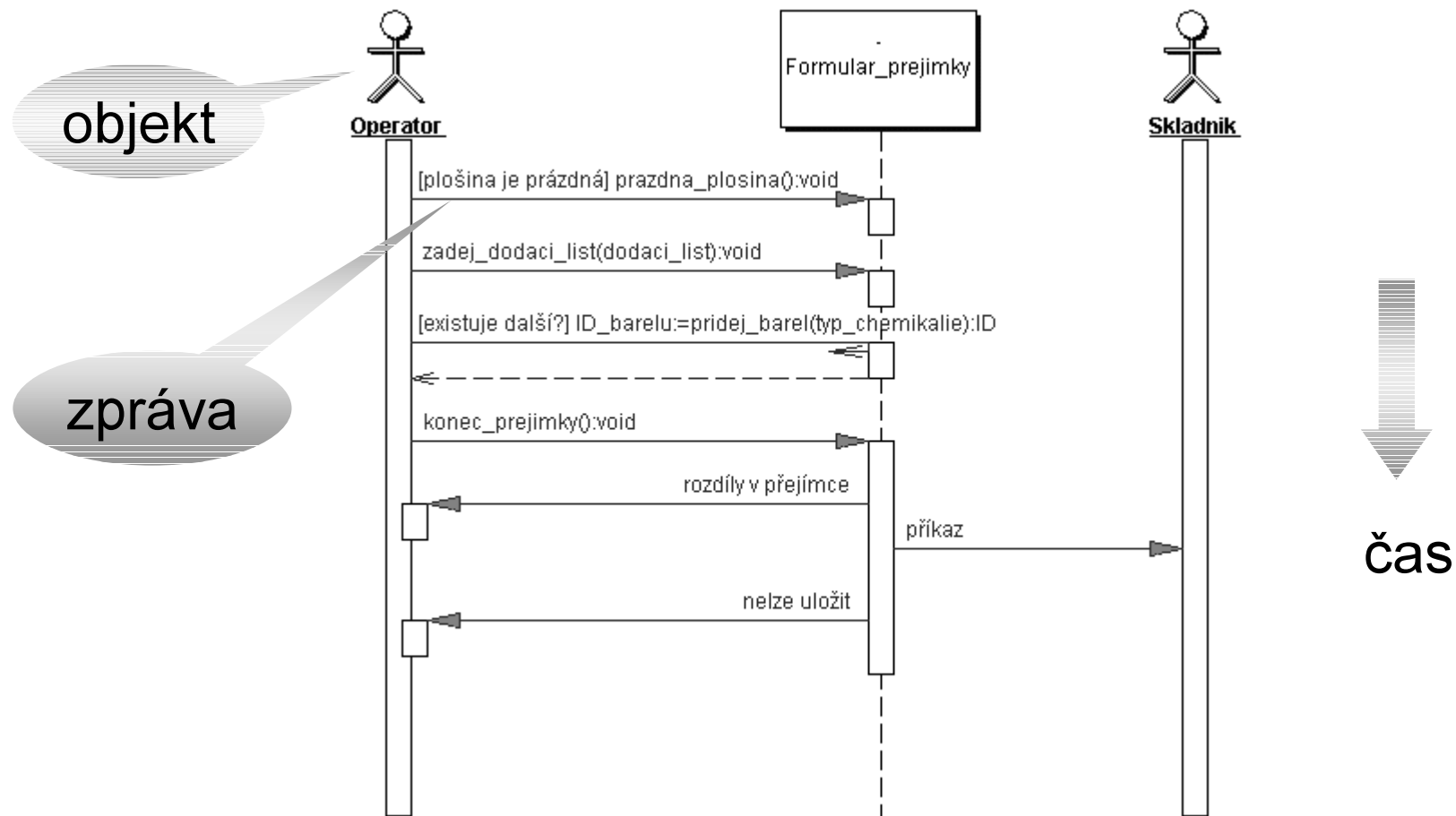


# Scénáře činností

---

- Dokumentují spolupráci prvků na **scénáři** činnosti
- Kladou důraz na **časový** aspekt
- Dokumentují **objekty** a **zprávy**, které si objekty posílají při řešení scénáře
- Umožňují vizualizovat zodpovědnost objektů

# Příklad scénáře činnosti

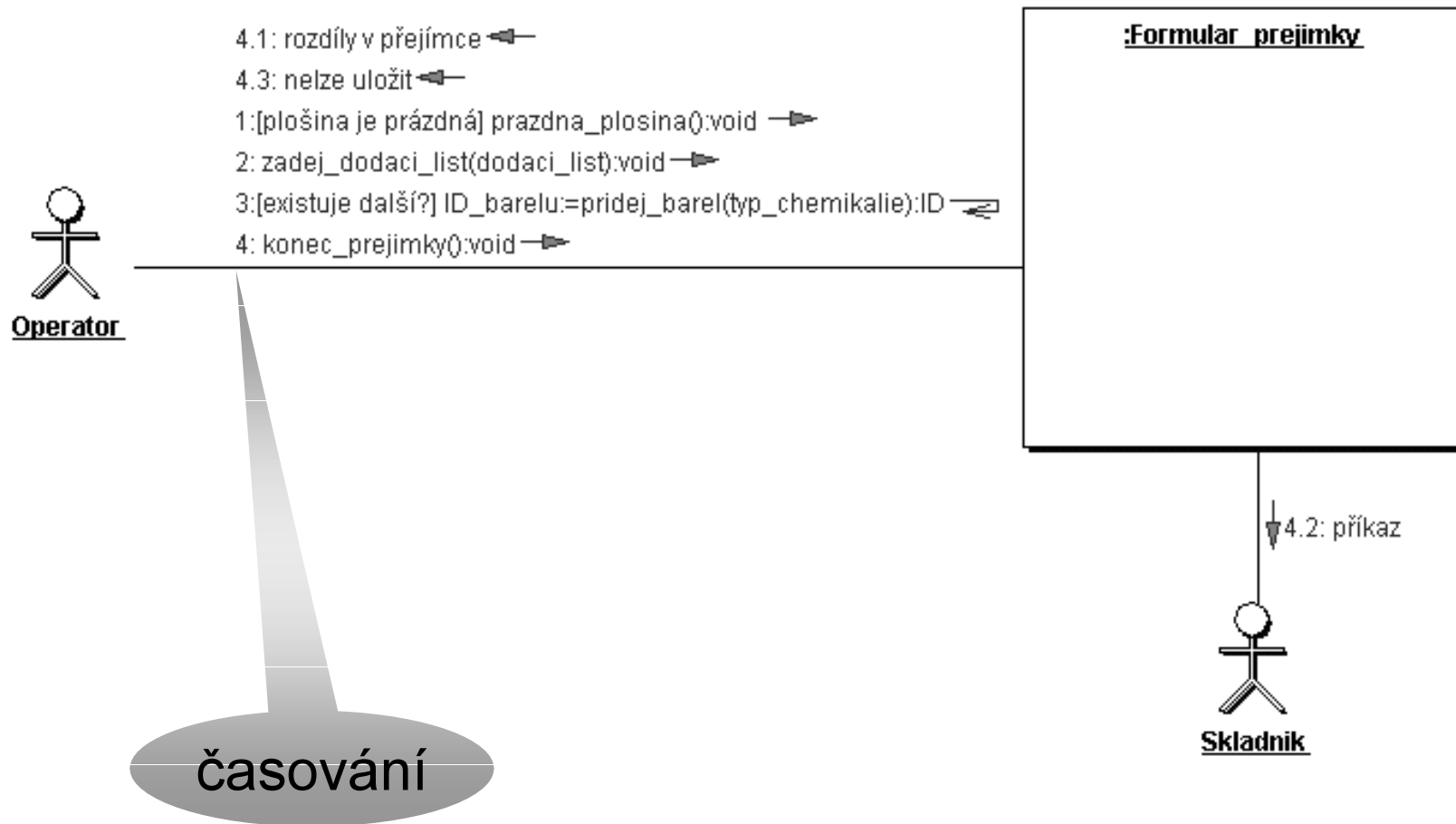


# Diagramy spolupráce (kolaborace)

---

- Podobně jako scénáře činností dokumentují **spolupráci** objektů při řešení úlohy
- Kladou důraz na **komunikační** aspekt (čas je vyjádřen číslováním)
- Dokumentují **objekty** a **zprávy**, které si objekty posílají při řešení problému
- Jsou vhodné pro popis spolupráce objektů při návrhu komunikace (závislosti mezi objekty)

# Příklad diagramu spolupráce

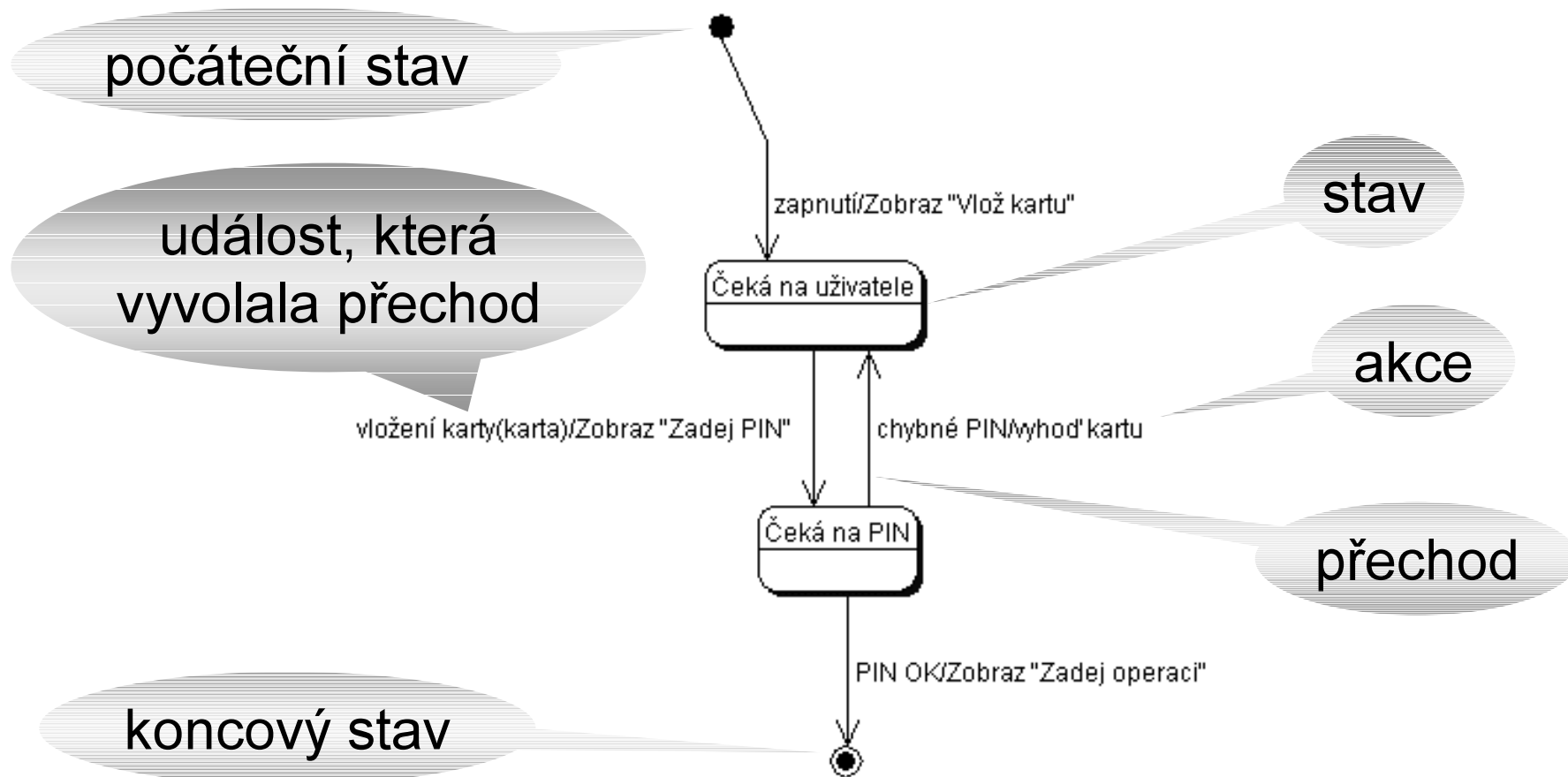


# Stavové diagramy

---

- Slouží k popisu dynamiky systému
- Stavový diagram definuje možné **stavy**, možné **přechody** mezi stavy, **události**, které přechody iniciují, **podmínky** přechodů a **akce**, které s přechody souvisí
- Stavový diagram lze použít pro popis dynamiky objektu (pokud má rozpoznatelné stavy), pro popis metody (pokud známe algoritmus), či pro popis protokolu (včetně protokolu o styku uživatele se systémem)

# Notace stavových diagramů



# Doplňky ke stavovým diagramům

---

- Přejechod může být ohodnocen:

**událost(parameters)[podmínka]/akce^zpráva**

- Každý stav může obsahovat popis akcí pro události vstup, výstup a opakované provádění:

**entry/akce**

**exit/akce**

**do/akce**

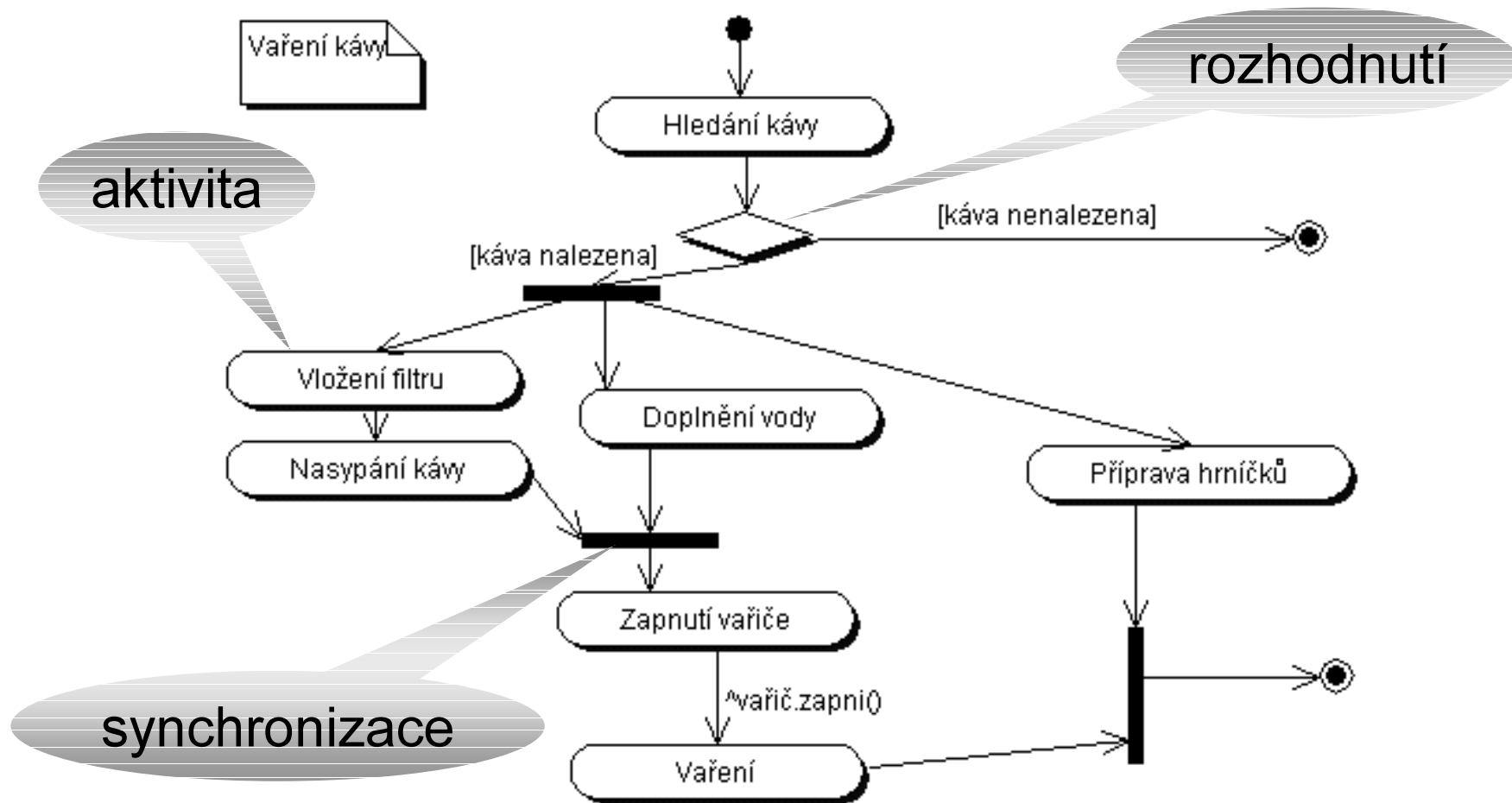
- Stavové diagramy mohou být hierarchické
- Mohou obsahovat synchronizační značky

# Diagramy aktivit

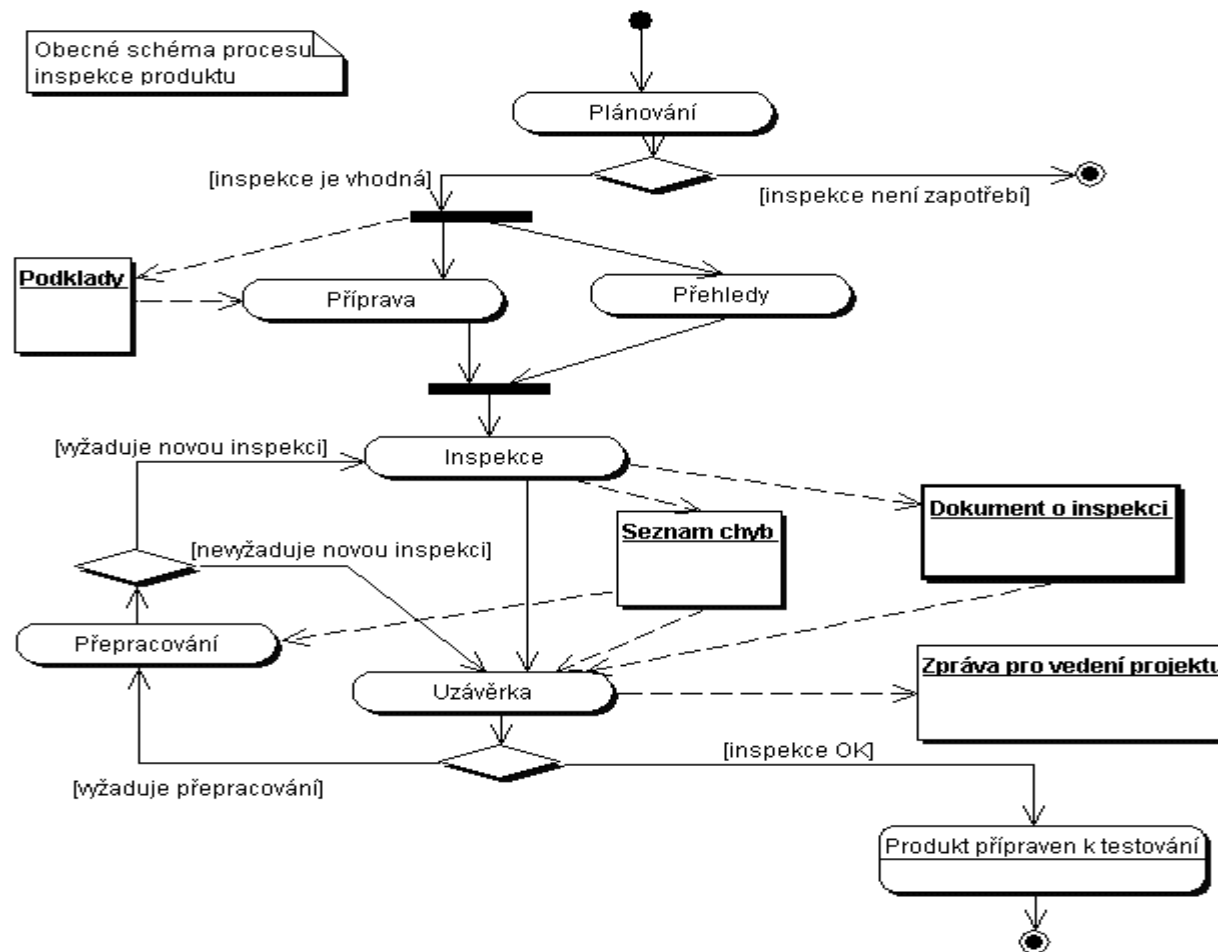
---

- Varianta stavových diagramů, kde stav představuje **aktivitu**, přechody jsou vyvolány dokončením akce (jsou synchronní)
- Popis pracovního postupu, algoritmu, paralelních činností apod.
- Úvaha: Nahrazují do určité míry v UML neexistující diagramy datových toků

# Příklad diagramu aktivity



# Jiný příklad diagramu aktivity

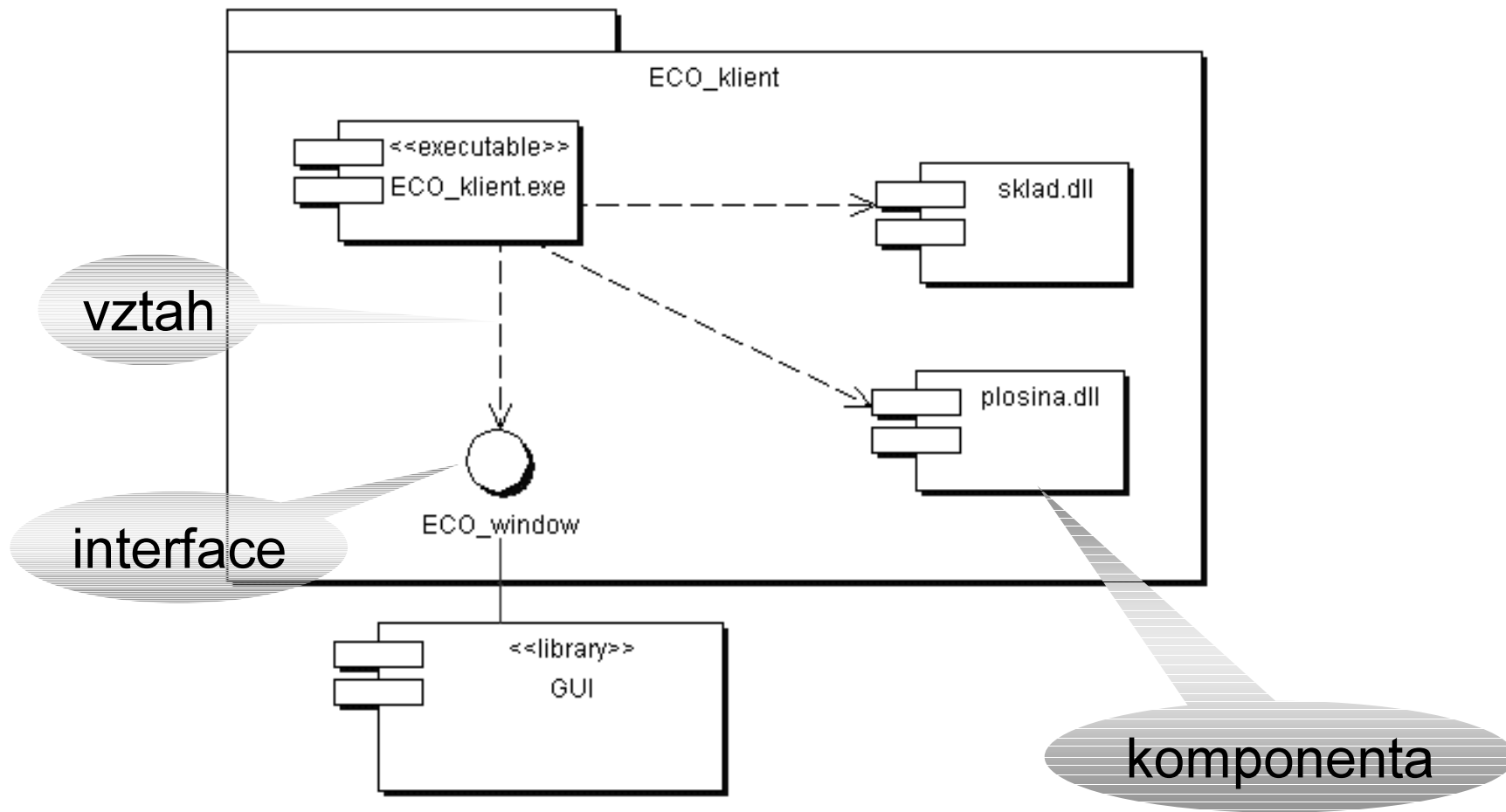


# Diagramy komponent

---

- Vyjadřují (fyzickou) strukturu **komponent** systému
- Popisují typy komponent - instance komponent jsou vyjádřeny v diagramu nasazení
- Komponenty mohou být vnořeny do jiných komponent
- Při vyjadřování **vztahu** mezi komponentami lze znázornit „interface“

# Příklad diagramu komponent

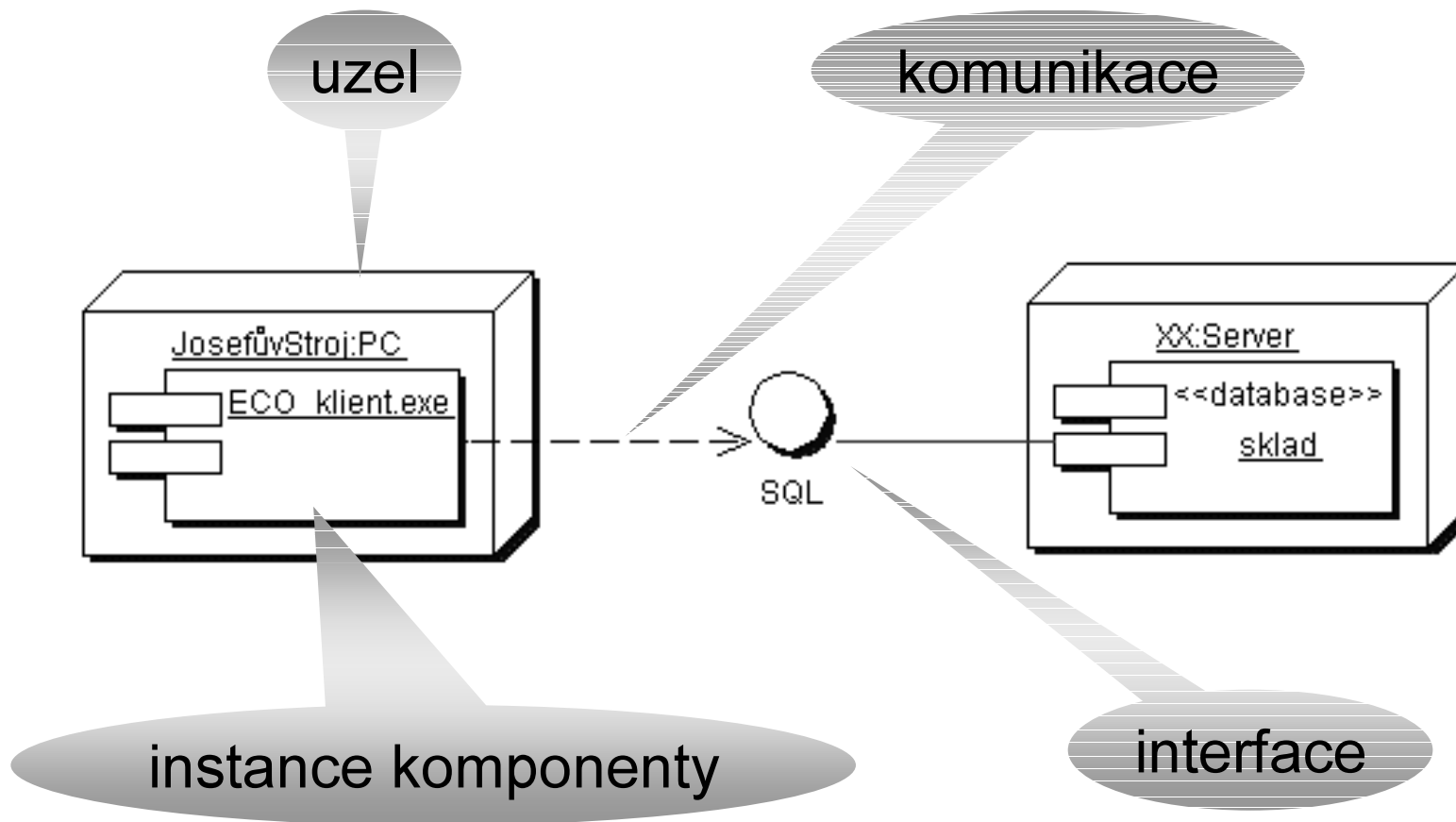


# Diagramy nasazení

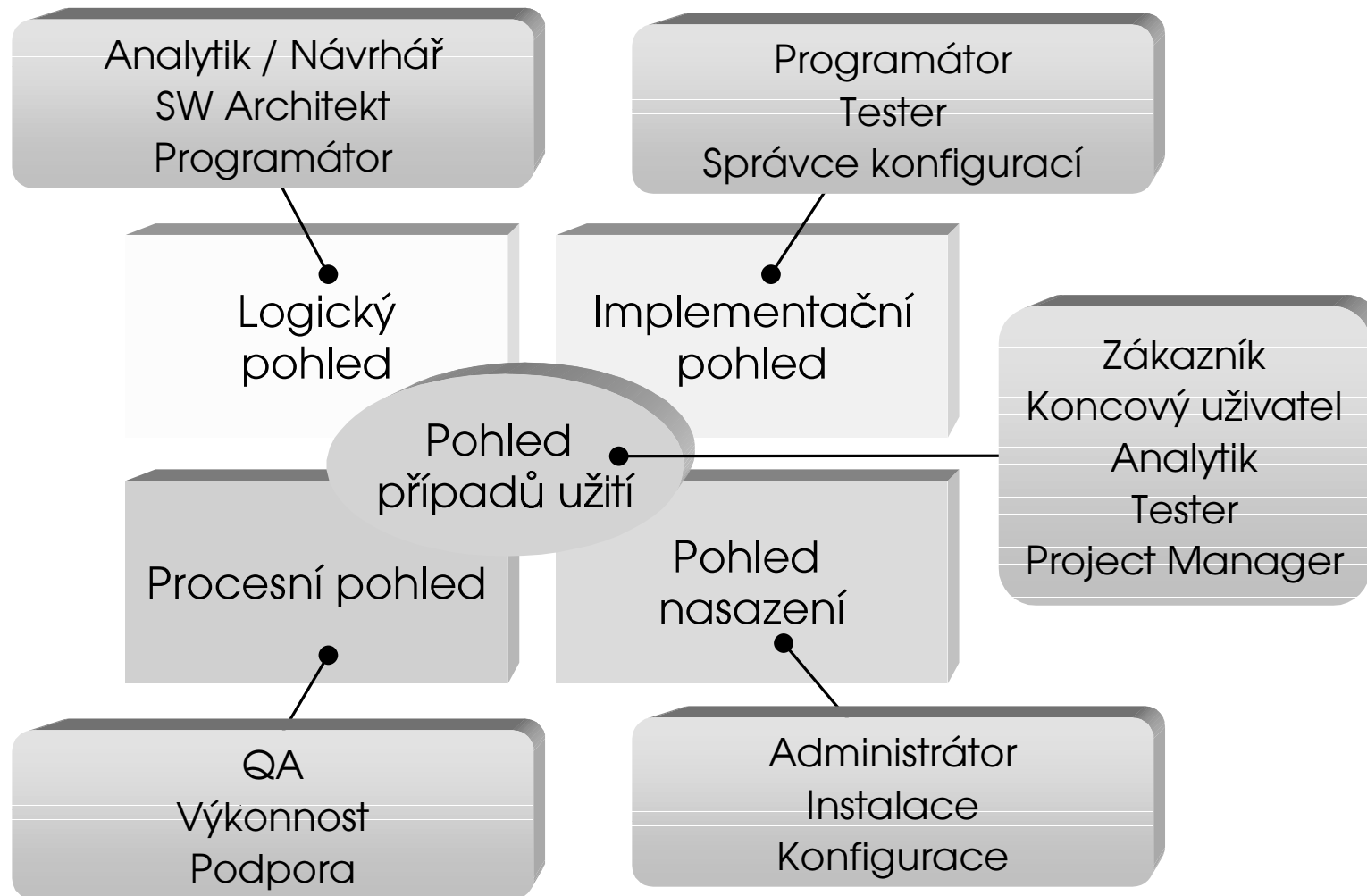
---

- Popisují fyzické rozmístění elementů systému na **uzly** výpočetního systému
- Uzly a elementy jsou značeny obdobně jako objekty a třídy (může být uveden pouze typ, nebo konkrétní instance a typ - podtržena)
- Popisují nutné **vazby** mezi uzly (případně též použitý protokol - „interface“)
- Obsahují pouze **komponenty** potřebné pro běh aplikace - komponenty potřebné pro překlad a sestavení jsou v diagramu komponent

# Příklad diagramu nasazení



# UML - 4+1 Architektura



# Část II.:

## Jazyk OCL



# OCL - Object Constraint Language

---

- Pochází z metodiky Syntropy
- Je čistě funkcionální (bez vedlejších efektů)
- Není to programovací jazyk - je určen pro vyjádření invariantů - je to specifikační jazyk
- Je silně typovaný (každý výraz OCL má definován typ)
- Má předdefinovanou sadu typů

# Kdy použít OCL?

---

- Pro vyjádření integritních omezení v modelu (např. v diagramu tříd)
- Pro vyjádření typových omezení při definici stereotypů
- Pro popis vstupních a výstupních podmínek operací, nebo pro popis operací (metod) ve tvaru:  
$$\text{operace}(x_1, \dots, x_n) = \text{výraz}$$

kde výraz může obsahovat  $x_1, \dots, x_n$
- Jako navigační jazyk

# Jak vypadá zápis v OCL?

---

**context** <jméno> [**inv**|**pre**|**post**] : <výraz>

- klíčové slovo **context** slouží pro definici kontextu pro výraz v OCL, např. zápis:

**context Student inv** : <výraz>

- označuje, že uvedený výraz se vztahuje k instanci kontextu (třídy) **Student** (na kterou se lze odkazovat klíčovým slovem **self**)

klíčová slova **inv**, **pre**, **post** zastupují stereotypy  
**<<invariant>>**, **<<precondition>>**,  
**<<postcondition>>**

# Příklad zápisu v OCL

---

**context Student inv: self.kredity > 10**

- tento zápis vyjadřuje, že pro každou instance třídy Student musí mít atribut kredity hodnotu větší než 10

**context s:Student inv: s.kredity > 10**

- označuje totéž

**context s:Student inv platnýStudent :  
s.kredity > 10**

- zavádí pro toto integritní omezení jméno

# Příklad popisu metody

---

```
context Student::pridejKredity(kolik:  
    Integer) : Integer  
pre : kolik > 1 -- nemá smysl přidávat méně  
post : result = self.kredity@pre + kolik
```

# Typy v OCL

---

- Předefinované primitivní typy (s obvyklými operacemi):  
**Integer, Boolean, Real, String**
- Kolekce:  
**Collection, Set, Bag, Sequence**
- S operátory:  
**collect, select, reject, forAll, exists, iterate, include, count, union, intersect, isEmpty, notEmpty, isUnique, ...**

# Lokální objekty (let)

---

```
context Student inv :  
  let Počet : Integer = self.kredity  
  in  
    if postupuje then Počet > 10  
    else Počet <= 10  
    endif
```

- kde „postupuje“ je atribut (odvozený) třídy Student

# Odkazy v OCL

---

```
context Student inv: self.predmety ->
isEmpty
```

- má hodnotu true, pokud student nemá zapsán žádný předmět, jinak false

```
context Student inv : self.predmety ->
pocetKreditu > 3
```

- má hodnotu true, pokud student nemá zapsán žádný předmět, jehož kredity jsou menší, jinak false

```
context Osoba inv : self.zena -> nonEmpty
implies self.zena.pohlavi = #zena
```

- vyjadřuje fakt, že ženou osoby může být pouze žena

# Předefinované operátory

---

```
oclIsTypeOf (t: oclType) : Boolean
oclIsKindOf (t: oclType) : Boolean
oclInState (t: oclState) : Boolean
oclIsNew : Boolean
oclAsType (t: oclType) : instance
                        typu oclType
```

# Část III.:

## UML – příklad



# Literatura

---

- Booch G., Rumbaugh J., Jacobson I.: The Unified Modeling Language User Guide, Addison Wesley Longman, 1999
- Unified Modeling Language Specification, OMG, <http://www.omg.org/>
- <http://www.rational.com/uml/>
- <http://www.uml-forum.com/>

# Závěr

---

- UML je dnes bez problémů použitelné
  - Není složité se notaci UML naučit
  - UML mohou používat analytici, architekti i programátoři
  - Je možné popsat systém z různých úhlů pohledu
- UML je nezbytné pro moderní vývoj software

Díky za pozornost.

