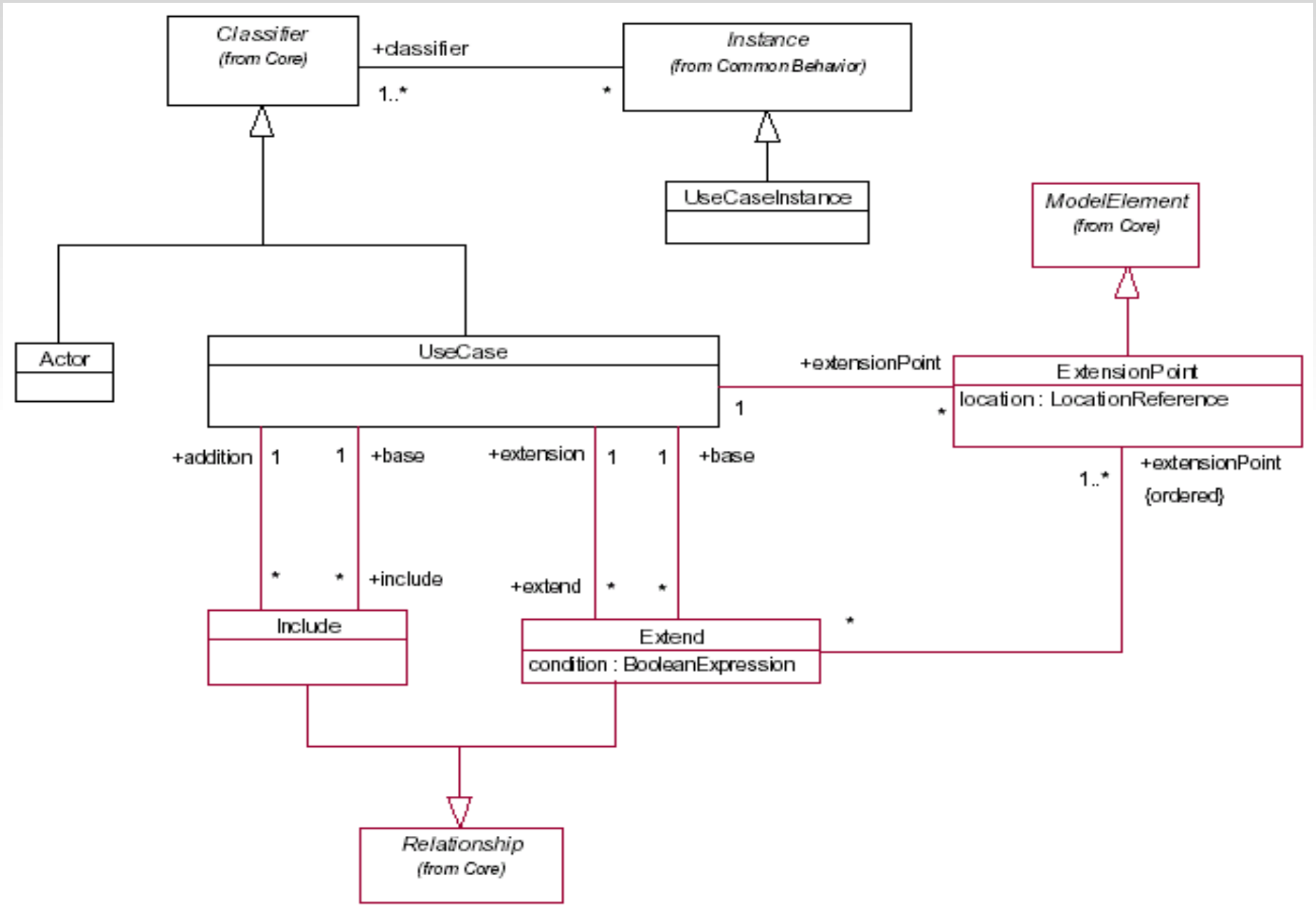
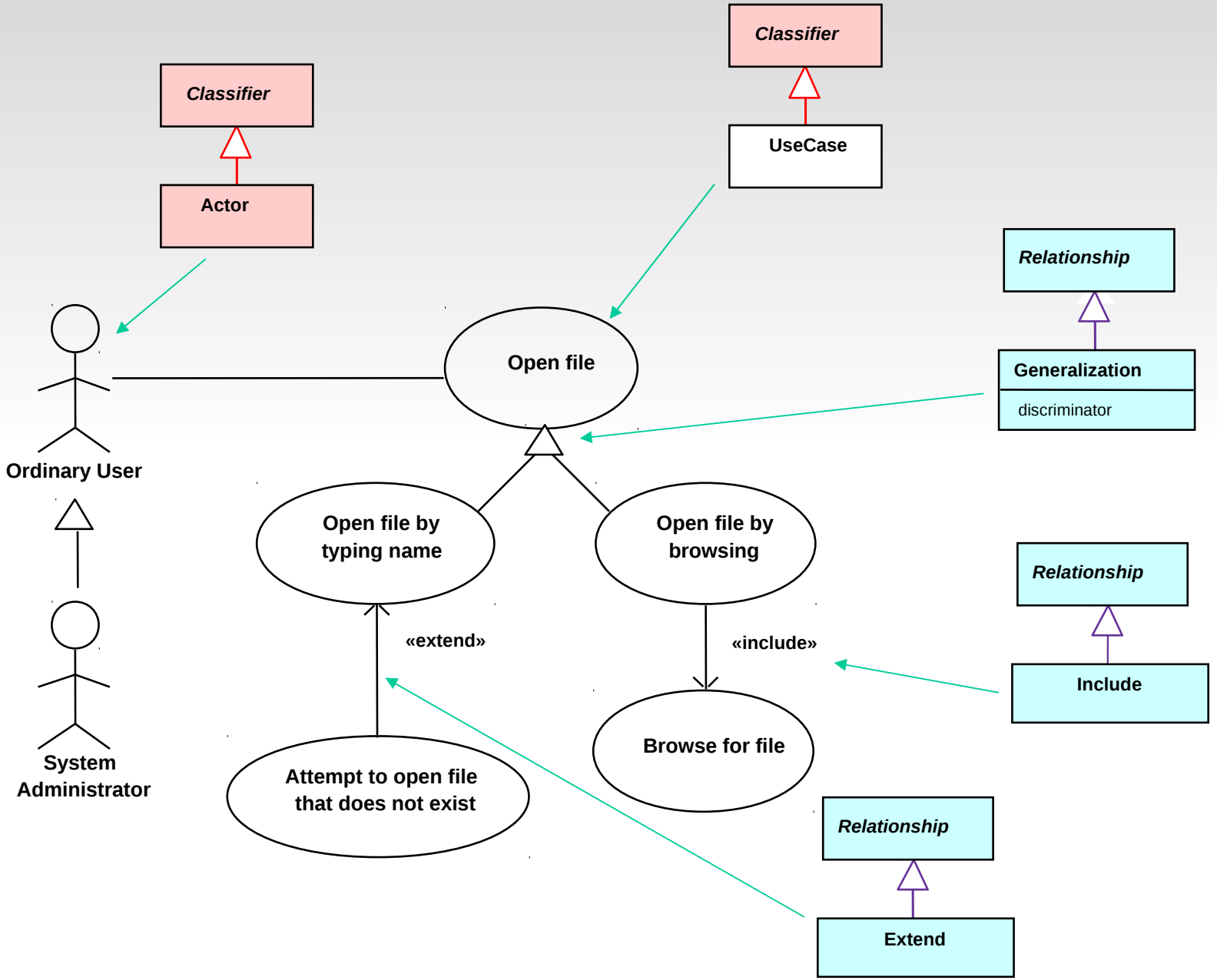
Object Constraint Language – OCL

© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

Metamodel: Př. pro UC diagram



Mapování UC modelu na metamodel



Co je OCL?

- OCL je *formální deklarativní jazyk* pro vyjádření
  - omezujících podmínek (constaints)
  - dotazy na objekty (object queries)

- Omezení:
  - Invariantní podmínka, která musí platit pro modelovaný systém.

- Omezení nemají *postranní efekty*
  - Jejich vyhodnocení nemůže změnit stav běžícího systému.
  - Žádný výraz, který lze zapsat v OCL, nepovoluje postranní efekty.
    - » Např. nelze přiřadit hodnotu atributu
    - » Výrazy mohou pouze vrátit hodnotu
  - OCL nelze použít jako programovací jazyk

Omezení

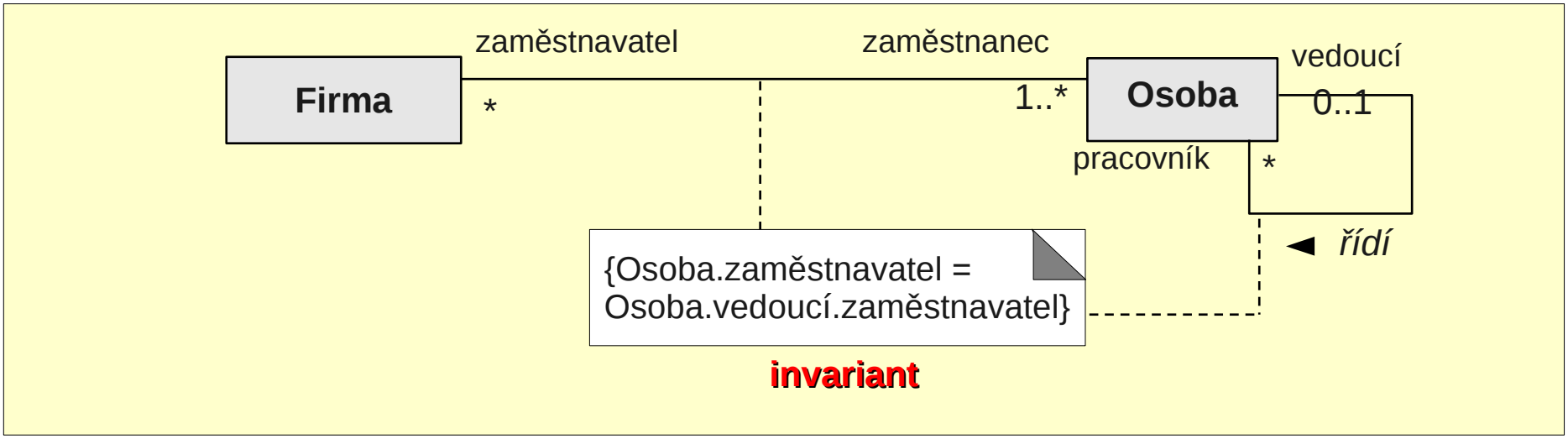
- Omezení jsou aplikována na instance v metamodelu
  - tj. pro modelování objektů, které jsou instancemi Class, Attribute, AssociationEnd, Association, ...
- Bez použití OCL by bylo nutné popsat omezení přirozeným jazykem
  - což obvykle vede k nejednoznačnostem
- Vyhodnocení OCL omezení je „okamžité“
  - během vyhodnocení se nemohou měnit stavy objektů.

Formální specifikační jazyky

- Formální specifikační jazyky slouží pro modelování, ne pro programování
  - Některé prvky v OCL nemusí být proveditelné.
    - » Např. operace „allInstances“
  - Implementační aspekty (např. datové struktury pro implementaci asociace) nemohou být odkazovány
- OCL používá (stejně jako jiné f.s.j.) matematické koncepty z
  - logika
  - teorie množin
- OCL začal jako modelovací jazyk obchodních případů v pojišťovací divizi IBM

Použití OCL v UML modelech

- Specifikace invariantů na třídách a typech v modelu tříd
- Specifikace typového invariantu pro stereotypy
- Popis pre- a post- podmínek u operací a metod
- Popis stráží
- Jako navigační jazyk
- Specifikace omezení u operací



Poznámky v OCL

- následují za dvojicí --

  -- toto je poznámka

  Většina dále uvedených příkladů pochází z UML 1.5 specifikace:
  http://www.omg.org/technology/documents/formal/uml.htm

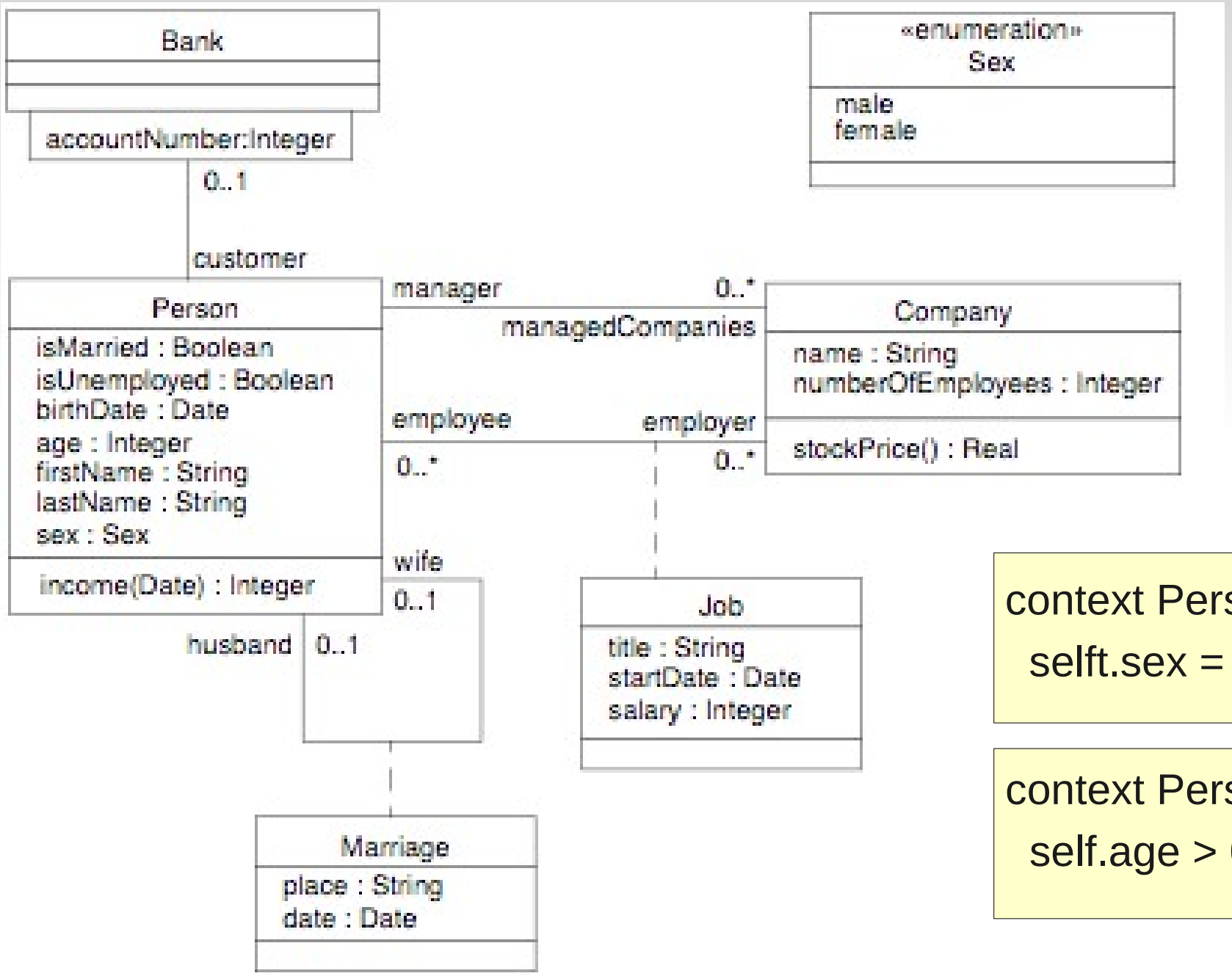
OCL je *silně typovaný* jazyk

- Správně vytvořené OCL výrazy musí splňovat typová pravidla
  - např. nelze porovnávat Integer a String.
- Každý Classifier, který je použit v UML modelu, se stává OCL typem
  - např. každá vytvořená třída
- OCL zahrnuje množinu doplňujících předdefinovaných typů

Základní typy v OCL

- Boolean -> true, false
  - Ops: and, or, xor, not, implies, if-then-else
- Integer -> 1, -5, 2, 34, 26524, ...
  - Ops: *, +, -, /, abs()
- Real -> 1.5, 3.14, ...
  - Ops: *, +, -, /, floor()
- String -> 'To be or not to be...'
  - Ops: toUpper(), concat()

Příklady invariantů

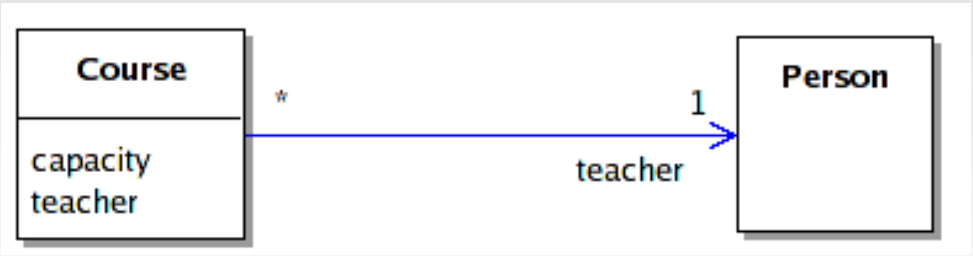


context Person inv:
selft.sex = Sex::male

context Person inv:
self.age > 0

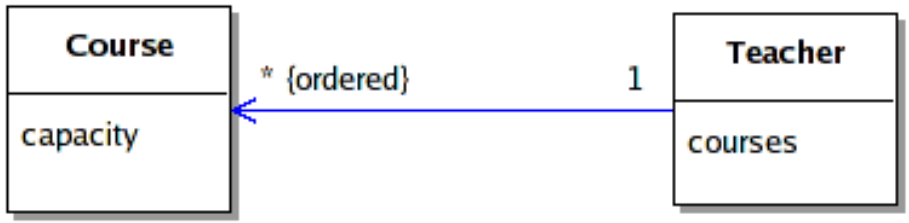
Kolekce a navigace v OCL výrazech

- Pokud self je třída *C* s atributem *atr*
  - pak **self.atr** se vyhodnotí jako **objekt** uložený v *atr*.



- co vrátí *self.capacity* pro konext *Course*?
- co vrátí *self.teacher* pro konext *Course*?

- Pokud *C* má násobnou asociaci 1..N *atr* ke třídě D
  - pak **self.atr** se vyhodnotí jako množina **Set,** jejíž prvky jsou typu *D*.
  - jestliže *atr* je {ordered}, je výsledkem **Sequence**
- Pokud *D* má navíc atribut *atrD*
  - pak výraz **self.atr.atrD** se vyhodnotí jako množina všech *atrD* náležících do *D*



- co vrátí *self.courses* pro konext *Teacher*?
- co vrátí *self.courses->first().capacity* ?
- co vrátí *self.courses.capacity* ?
- co říká invariant *self.courses->size() > 0*
- co říká invariant *max(self.courses.capacity) < 150*

Příklady výrazů

- context Company
- inv: self.manager.isUnemployed = false
- -- self.manager se vyhodnotí jako Person
- inv: self.employee->notEmpty()
- -- self.employee se vyhodnotí jako Set

- Note: ->notEmpty() vyvolání OCL vestavěné funkce Collection
  - později

Předdefinované OCL funkce

- oclIsTypeOf(t : OclType) : Boolean
  – true pokud typ „self“ a „t“ jsou shodné.

context Person

inv: self.oclIsTypeOf(Person) -- is true

inv: self.oclIsTypeOf(Company) -- is false

- oclIsKindOf(t : OclType) : Boolean
  – true pokud „t“ je buď přímo typ nebo jeden z nadtypů objektu.

context Employee

inv: self.oclIsTypeOf(Person) -- is true

OCL vestavěné funkce pro kolekce

- Kolekce lze definovat pomocí navigace nebo přímo:
  - Set { 1 , 2 , 5 , 88 }
  - Set { 'apple' , 'orange', 'strawberry' }
  - Sequence { 1, 3, 45, 2, 3 }
  - Sequence { 'ape', 'nut' }
- Notace *->function()* označuje vyvolání vestavěněné funkce OCL v kolekci
  - odlišné od '.' , která se používá k
    - » přístupu k vlastnosti a navigaci
    - » včetně vyvolání dotazovacích funkcí definovaných v modelu

Důležité funkce „Collection“ v OCL

- **aCollection->isEmpty(), ->notEmpty**
- **aCollection->size()**
- **aCollection->includes(anObject)**
- **aCollectionOfNumbers->sum()**

- **aCollection->exists(booleanExpression)**
  - vrací true, jestliže booleanExpression je true pro nějaký prvek kolekce

Select a reject: výběr subkolekcí

- **context Company inv:**
- ** self.employee->select(age > 50)->notEmpty()**
- ** -- select(age > 50) vybírá lidi nad 50**

- **context Company inv:**
- ** self.employee->select(p | p.age > 50)->notEmpty()**

- **->reject() vybírá doplňkovou podmnožinu**

Collect: generování paralelních kolekcí

- self.employee->collect(birthdate)

- vytvoří kolekci shodné velikosti jako množina zaměstnanců, která obsahuje data narození
  - výběr již může obsahovat duplikáty
  - ekvivalent *projekcí* v SQL

forAll: vyhodnotí výraz na každém prvku kolekce

- **collection->forAll(v : Type | bool-expr-with-v)**
- **collection->forAll(v | boolean-expression-with-v)**
- **collection->forAll(boolean-expression)**

- následující je pravda, pokud se každý zaměstnanec jmenuje Jack

- **inv: self.employee->forAll(forename = 'Jack')**
- **inv: self.employee->forAll(p | p.forename = 'Jack')**
- **inv: self.employee->forAll(p : Person | p.forename = 'Jack')**

Další funkce pro kolekce

- c1->includesAll(c2)
  - True if every element of c2 is found in c1
- c1->excludesAll(c2)
  - True if no element of c2 is found in c1
- For sets:
- s1->intersection(s2)
  - The set of those elements found s1 and also in s2
- s1->union(s2)
  - The set of those elements found in either s1 or s2
- s1->excluding(x)
  - The set s1 with object x omitted.
- For sequences
- seq->first()

Logické implikace

- **context Person inv:**
- ** self.wife->notEmpty() implies**
- **   self.wife.age >= 18**
- **   and**
- **   self.husband->notEmpty() implies**
- **   self.husband.age >= 18**

forAll se dvěma proměnnými

- Všechny dvojice *Kartézského součinu* zaměstnanců

- **context Company inv:**
- **self.employee->forAll(e1, e2 : Person |**
- **e1 <> e2 implies e1.forename <> e2.forename)**

- shodné vyjádření
- **self.employee->forAll(e1 | self.employee->forAll (e2 |**
- **e1 <> e2 implies e1.forename <> e2.forename)))**

„Let“ výrazy

```
context Person inv:
  let income : Integer =
                        self.job.salary->sum()
  let hasTitle(t : String) : Boolean =
                        self.job->exists(title = t) in
  if isUnemployed then
    self.income < 100
  else
    self.income >= 100 and self.hasTitle(‘manager’)
  endif
```

Definice hodnoty pro použití v jiných výrazech

- ‘def’
- **context Person def:**
- **let income : Integer =**
  self.job.salary->sum()
- **let hasTitle(t : String) : Boolean =**
  self.job->exists(title = t)

Příklady invariantu třídy

- následující výrazy vyjadřují to samé
  - numberOfEmployees > 50
  - self.numberOfEmployees > 50
  - context Company inv:
    self.numberOfEmployees > 50
  - context c : Company inv:
    c.numberOfEmployees > 50
  - context c : Company inv enoughEmployees:
    c.numberOfEmployees > 50

Příklady pre- a post-podmínek

- **context Person::income(d : Date) : Integer**
- ** post: result = 5000**

- **context Typename::operationName(**
- ** param1 : Integer): Integer**
- ** pre parameterOk: param1 > 5**
- ** post resultOk: result < 0**

Použití @pre v post-podmínkách

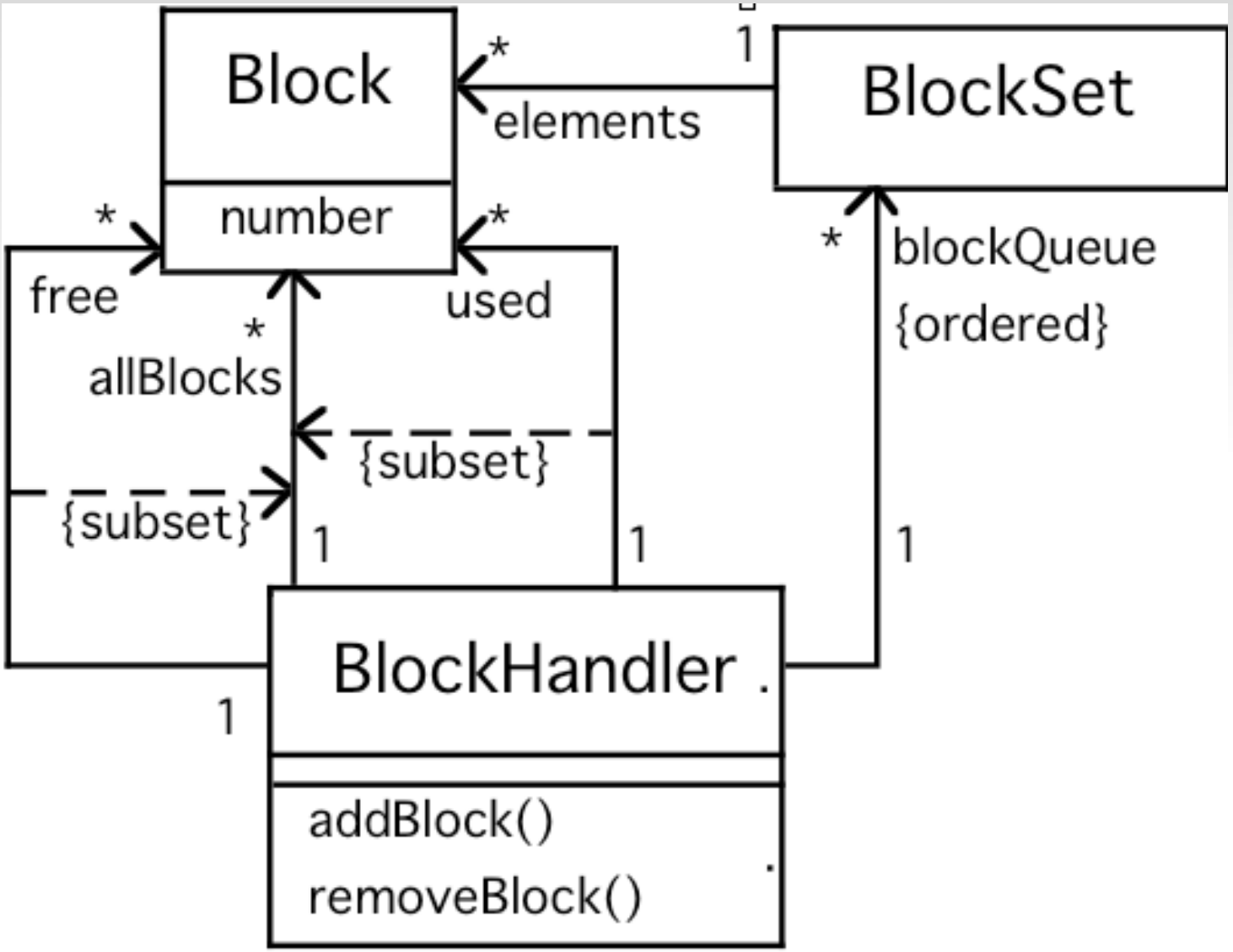
- zápis post-podmínky vyjadřující, co se změnilo ve vztahu k pre-podmínce
  - použití **property@pre** pro odkaz na hodnotu **property** před provedením

- např.
- **context Company::hireEmployee(p : Person)**
- **pre : not employee->includes(p)**
- **post: employees->includes(p) and**
- **stockprice() = stockprice@pre() + 10**

Precedentní pravidla OCL výrazů

- Precedentní uspořádání operací od nejvyšší priority:
  - @pre
  - operace ‘.’ a ‘->’
  - unární ‘not’ a unární ‘-’
  - ‘*’ a ‘/’
  - ‘+’ a binární ‘-’
  - ‘if-then-else-endif’
  - ‘<’, ‘>’, ‘<=’, ‘>=’
  - ‘=’, ‘<>’
  - ‘and’, ‘or’, ‘xor’
  - ‘implies’
- Závorky ‘(’ a ‘)’ mohou změnit precedenci.

Příklad „Zpracování bloků“



Příklady OCL výrazů „Zpracování bloků“

- žádný blok nebude současně označen jako použitý a nepoužitý.

- **context BlockHandler inv:**
- **(self.used->intersection(self.free)) ->isEmpty()**

- Všechny množiny bloků uložených ve frontě budou podmnožinami právě používaných bloků

- **context BlockHandler inv:**
- **blockQueue->forAll(aBlockSet |**
- **used->includesAll(aBlockSet))**

Příklady (II)

- žádné prvky ve frontě neobsahují stejná čísla bloků

- **context BlockHandler inv:**
- **blockQueue->forAll(blockSet1, blockSet2 |**
- **blockSet1 <> blockSet2 implies**
- **blockSet1.elements.number -> excludesAll(blockSet2.elements.number))**

Příklady (III)

- Kolekce použitých a nepoužitých bloků tvoří kolekci bloků, ze kterých se skládají soubory.

- **context BlockHandler inv:**
- **allBlocks = used->union(free)**

- Kolekce nepoužitých bloků nebude mít žádné duplikáty čísel bloků.

- **context BlockHandler inv:**
- **free->isUnique(aBlock | aBlock.number)**

Příklady (IV)

- Kolekce použitých a nepoužitých bloků tvoří kolekci bloků, ze kterých se skládají soubory.
- **context BlockHandler inv:**
- **allBlocks = used->union(free)**
- Kolekce nepoužitých bloků nebude mít žádné duplikáty čísel bloků.
- **context BlockHandler inv:**
- **free->isUnique(aBlock | aBlock.number)**
- Kolekce použitých bloků nebude mít žádné duplikáty čísel bloků.
- **context BlockHandler inv:**
- **used->isUnique(aBlock | aBlock.number)**

Příklady (V)

- context BlockHandler::removeBlocks()
- pre: blockQueue->size() >0
- post: used = used@pre - blockQueue@pre->first() and
- free = free@pre->union(blockQueue@pre->first()) and
- blockQueue = blockQueue@pre->excluding(blockQueue@pre->first)

- context BlockHandler::addBlocks(aBlockSet :BlockSet)
- pre: used->includesAll(aBlockSet.elements)
- post: (blockQueue.elements = blockQueue.elements@pre
- - aBlockSet.elements) and
- used = used@pre and
- free = free@pre