

# Podstawy inżynierii oprogramowania

---

dr inż. Krystian Wojtkiewicz



Język OCL jako formalny język  
specyfikacji ograniczeń  
systemu

---



# Ograniczenie

---

Semantyczny warunek lub zawężenie nałożone na element modelu.

Restrykcja nałożona na jedną lub więcej wartości (części) modelu lub systemu obiektowego.

## Zalety

Poprawienie jakości dokumentacji

Zwiększa precyzję diagramów (UML)

Lepsze zrozumienie pomiędzy projektantem, klientem i deweloperem

# Ograniczenia na system

---

Niezmiennik dla atrybutów klasy

Niezmiennik dla asocjacji klasy

Określenie warunków początkowych i końcowych realizacji operacji

Ograniczenia dla relacji generalizacji

Precyzowanie sposobów postępowania z kolekcjami



# Wprowadzenie

---

OCL jest formalnym językiem specyfikacji wyrażeń UML

Deklaratywny  
Kontekstowy  
Silnie typizowany  
...prosty

Wyrażenia OCL nie mogą zmieniać stanu systemu

OCL nie jest językiem programowania



# OCL - historia

---

- Pierwotnie opracowany m.in. przez IBM (1995) jako język biznesowo-inżynierski
- Zaadoptowany na potrzeby UML jako składowa formalnej specyfikacji
- Od wersji 1.1 część oficjalnego standardu OMG

Warmer, J., Kleppe, A.: *The Object Constraint Language. Precise Modeling with UML*. Addison-Wesley, 1999

Warmer, J., Kleppe, A.: *The Object Constraint Language Second Edition. Getting Your Models Ready for MDA*. Addison-Wesley, 2003

OMG UML Specification

OMG OCL Specification



# OCL – Object Constraint Language

---

- Aktualna wersja – 2.4
- Data publikacji: luty 2014

Specyfikacja OCL opracowywana jest od 2006 roku przez OMG i utrzymywana w zgodności z UML (2.4.1) oraz z MOF (2.4.1)



# OCL – zakres użycia

---

Specyfikacja zapytań

Specyfikacja niezmienników klas i typów danych

Specyfikacja niezmienników typu dla stereotypów

Specyfikacja warunków wstępnych i końcowych dla operacji i metod

Specyfikacja warunków dozoru

Specyfikacja celu komunikatów i akcji

Specyfikacja ograniczeń dla operacji



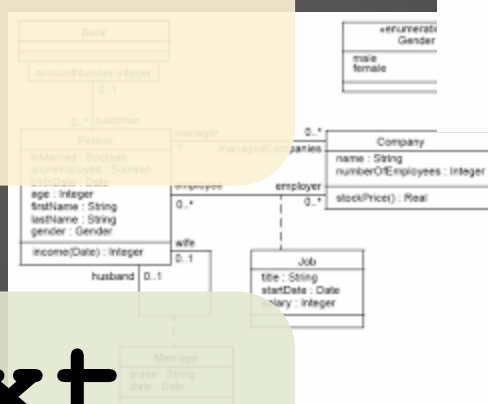
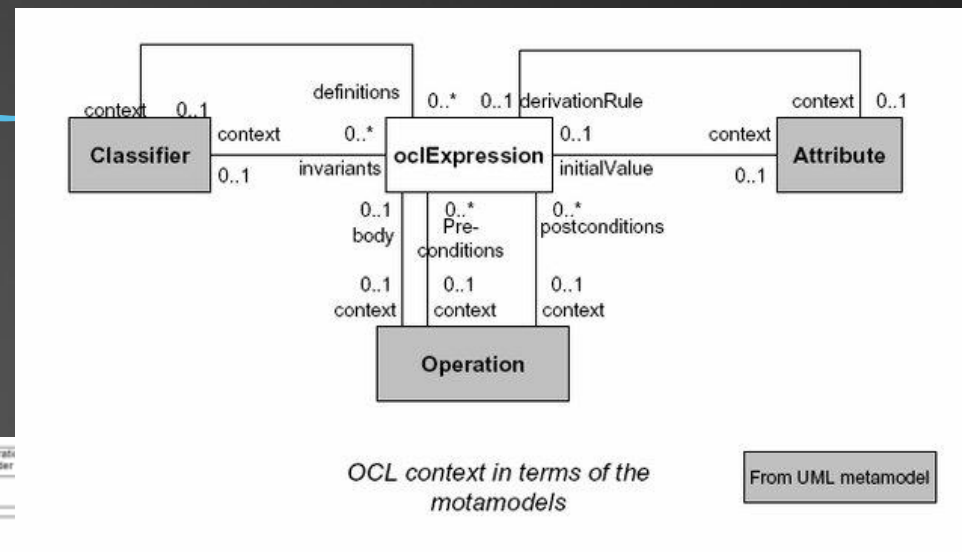
# Przykład

Każde wyrażenie OCL jest pisane w kontekście pewnej instancji konkretnego typu

self

Kontekst wewnątrz wyrażenia można określić przy pomocy deklaracji kontekstu

context  
Company





## Przykład

**Niezmiennik** – rodzaj ograniczenia, który musi być spełniony przez wszystkie instancje danego klasyfikatora w każdej chwili

```
context Company
```

```
inv: self.numberOfEmployees > 50
```

`self` można opuścić w większości przypadków

```
context Company
```

```
inv: numberOfEmployees > 50
```



## Przykład

---

`self` można zamienić na inny symbol

```
context c:Company
```

```
inv: c.numberOfEmployees > 50
```

Ograniczeniu można nadać nazwę

```
context c:Company
```

```
inv: enoughEmployees: self.numberOfEmployees > 50
```



# Przykład

---

Kontekst to operacja lub inna cecha czynnościowa

`self` – instancja typu, do którego należy dana cecha czynnościowa

Wynik operacji określa słowo zarezerwowane `result`

```
context Type::operation(par1: T1, ...): ReturnType
```

```
pre: ...
```

```
post: ...
```



# Przykład

---

## Warunek końcowy

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

## Nazwy dla warunków

```
context ClassA::op(par: T): ReturnType  
  pre: parameterOk: par > 63  
  post: resultOk: result = par + 23
```



## Przykład

---

### Kontekst w obrębie pakietu

```
package A::B
context C
  inv: ...
context C::op1()
  pre: ...
endpackage
```



# Przykład

---

Ograniczenia dla operacji będących zapytaniami (`body condition`)

```
context ClassA::op1(): T  
  body: ...
```

Przykład

```
context Person::getCurrentSpouse():  
  Person  
  pre: self.isMarried = true  
  body: self.marriages -> select(ended =  
    false).spouse
```



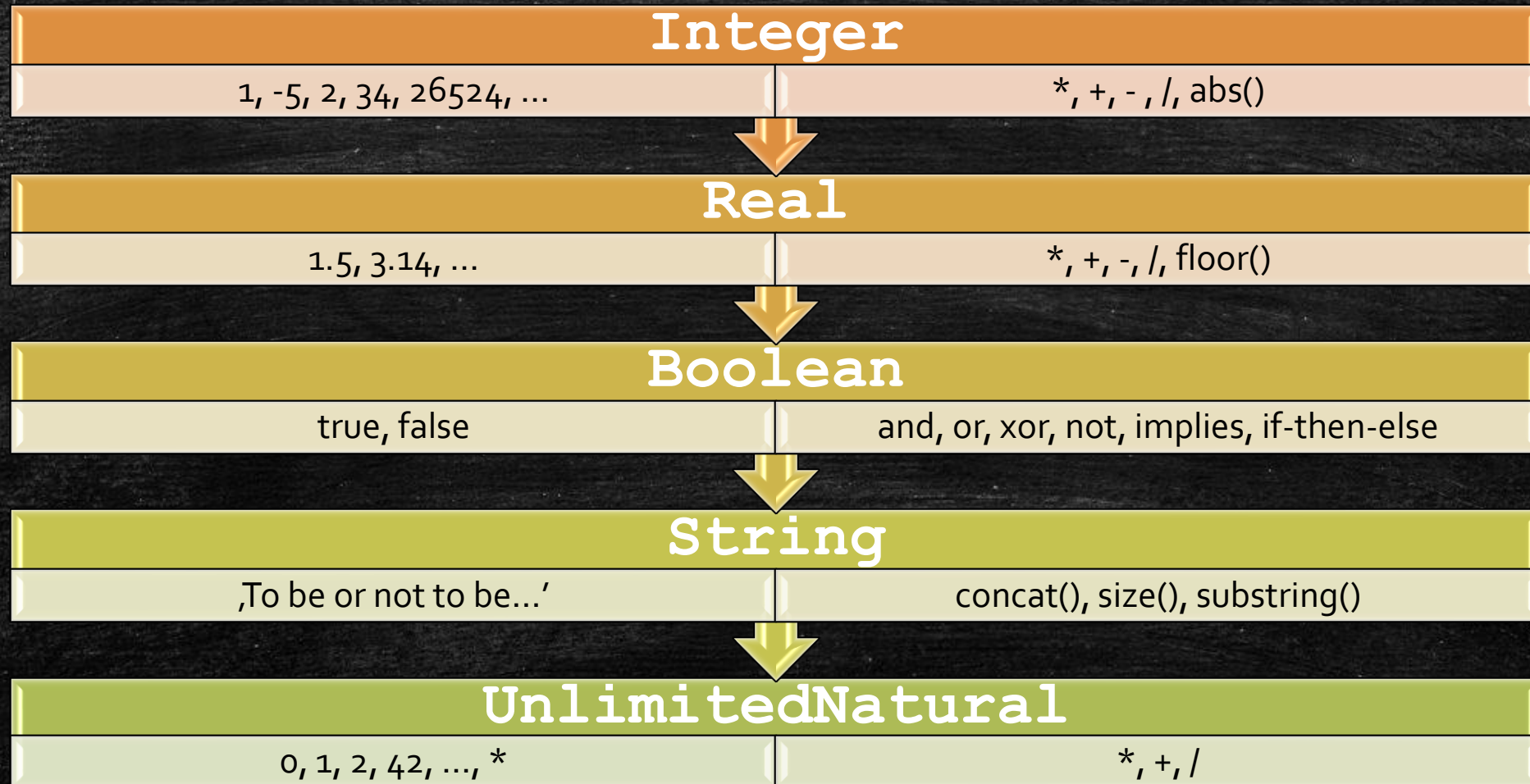
## Przykład

Mogą opisywać wartości początkowe i wyliczane (wyprowadzone) atrybutów i końców powiązań

```
context Person::income: Integer
  init: parents.income -> sum()*1% --
    kieszonkowe
  derive:
    If underAge
    then parents.income -> sum()*1%
    else job.salary
    endif
```



# Podstawowe typy danych i operacje





# Podstawowe typy danych i operacje

---

**oclInvalid**

invalid

**OclVoid**

null  
Invalid

Collection  
Set  
Bag  
Sequence  
Tuple



# Znaki specjalne

`\b` -- backspace

`\t` -- horizontal tab

`\n` -- linefeed

`\f` -- form feed

`\r` -- carriage return

`\"` -- double quote

`\'` -- single quote

`\\` -- backslash

`\xhh` -- #x00 to #xFF

`\uhhhh` -- #x0000 to #xFFFF

**h jest znakiem z systemu szesnastkowego 0 - 9, A - F lub a - f**



## Zmienne o zasięgu pojedynczego wyrażenia

---

```
context Person inv:
```

```
  let income : Integer = self.job.salary->sum() in
```

```
  if isUnemployed then
```

```
    income < 100
```

```
  else
```

```
    income >= 100
```

```
  endif
```



# Definiowanie dodatkowych atrybutów/operacji

---

`<<definition>>`

- Ograniczenia oznaczone tym stereotypem można w wyrażeniach OCL definiować dodatkowe atrybuty i operacje klasyfikatorów
  - Dzięki temu możemy współdzielić zmienne i operacje pomiędzy kilkoma ograniczeniami
- Dodatkowe cechy stają się zwykłymi atrybutami i operacjami klasyfikatora o stereotypie `<<OclHelper>>`

**context** Person

**def:** income : Integer = self.job.salary->sum()

**def:** nickname : String = 'Little Red Rooster'

**def:** hasTitle(t : String) : Boolean = self.job->exists(title = t)



# Zgodność typów

Każdy typ jest zgodny z każdym swoim nadtypem

Relacja zgodności jest przechodnia

Relacja zgodności nie jest symetryczna

Typ kolekcji elementów typu  $T_1$  jest zgodny z typem kolekcji elementów typu  $T_2$  o ile  $T_1$  jest zgodny z  $T_2$

$\text{Collection}(T_1) / \text{Collection}(T_2)$

$\text{Set}(T_1) / \text{Set}(T_2)$

$\text{OrderedSet}(T_1) / \text{OrderedSet}(T_2)$

$\text{Bag}(T_1) / \text{Bag}(T_2)$

$\text{Sequence}(T_1) / \text{Sequence}(T_2)$



# Wartości nieokreślone

---

Wartości nieokreślone są wynikiem niektórych operacji

Zazwyczaj wyrażenie o składowych nieokreślonych ma wartość nieokreśloną

Za wyjątkiem wyrażeń logicznych, w których wartość nieokreślona spełnia reguły logiki trójwartościowej

Można jawnie testować wartość nieokreśloną obiektów

```
ocllsUndefined()
```

```
OclAny
```



# Kolejność wykonywania

literal and variable expressions, "(" and ")", "if-then-else-endif"

"let-in"

@pre

call expressions: "^", "^^", "." and "->"

unary "not" and unary "-"

"\*" and "/"

"+" and binary "-"

"<", ">", "<=", ">="

"=", "<"

"and"

"or"

"xor"

"implies"

"in"





# Operacje kolekcji

---

sprawdza czy kolekcja jest pusta,

```
collection->isEmpty() : Boolean
```

zwraca rozmiar kolekcji (liczba jej elementów),

```
collection->size() : Integer
```

sprawdza czy obiekt object jest w danej kolekcji

```
collection->includes(object : OclAny) : Boolean
```

zwraca sumę wszystkich elementów kolekcji,

```
collection->sum() : T
```

operacje reprezentujące kwantyfikatory ogólne dla kolekcji

```
collection->exists(expr : OclExpression) : Boolean
```

```
collection->forall(expr : OclExpression) : Boolean
```



# Operacje kolekcji

Operacja `select` tworzy nową kolekcję będącą podkolekcją kolekcji, dla której wywołana została operacja

```
collection->collect( expression )  
collection->collect( v : Type | expression-with-v )  
collection->collect( v : Type | expression-with-v )
```

Wynikiem tej operacji jest nowa kolekcja z tymi elementami z kolekcji źródłowej, dla których wyrażenie logiczne w operacji `select` było prawdziwe.

```
collection->select( boolean-expression )  
collection->select( v | boolean-expression-with-v )  
collection->select( v : Type | boolean-expression-with-v )
```

## Przykład

```
Rezerwacja.pozycja->select(k: pozycja.Książka | k.autor ="Pressman")
```



# Iteracja po elementach kolekcji

Operacja ta przebiega całą kolekcję z iteratorem `elem`.

`acc` jest akumulatorem, który może być na początku zainicjowany wartością początkową (`expression`).

Dla każdej wartości iteratora, czyli dla wszystkich elementów kolekcji obliczane jest wyrażenie `expression-with-elem-and-acc`, a jego wynik jest podstawiany do akumulatora.

```
collection->iterate( elem : Type; acc : Type = <expression> |expression-with-elem-and-acc )
```

operacja sumowania wartości kolekcji (elementy typu `Integer`) zdefiniowana jako iteracja

```
collection->iterate(x : Integer; acc : Integer = 0; acc + x)
```



# Ograniczenia OCL

---

Zła składnia (przerost, brak czytelności)

Brak uniwersalności

Niepełna specyfikacja

Redundancja funkcjonalności

**Brak związku z uniwersalnym językiem programowania**

Brak perspektyw w aspekcie baz danych

Niejasny stosunek do wartości zerowych (pustych)

Brak zgodności z UML 2.5.1

Brak optymalizacji w kontekście języka zapytań



**WHAT IF I TOLD YOU**



**ITS NOT OVER YET**

