

# Tipi e costrutti avanzati di OCL

## Lezione 23 di Ingegneria del software

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)  
Facoltà di Scienze Matematiche, Fisiche e Naturali  
Corso di Studi in Informatica applicata, AA 2008-9

### Indice

1. Tipi e costrutti avanzati di OCL
2. tipi di base predefiniti, Boolean
3. i tipi di base Integer e Real
4. il tipo String
5. sintassi delle espressioni di base
6. caratteristiche dei tipi da utente
7. associazioni, tipi enumerativi
8. struttura dei tipi di collezioni
9. operazioni comuni sulle collezioni
0. operazioni varianti su collezioni
- .1. operazioni varianti su collezioni ordinate
2. operazioni iterative sulle collezioni
3. l'operazione iterate
4. costrutti per postcondizioni
5. conformità, casting, tipo universale

## tipi di base predefiniti, Boolean

OCL ha 4 tipi di base predefiniti (i soliti), con relative operazioni:

**Boolean, Integer, Real, String**

le definizioni delle operazioni non sono sempre le solite...

operazioni su valori **Boolean**, con valore di ritorno **Boolean** :

disgiunzione:	$a \text{ or } b$
congiunzione:	$a \text{ and } b$
or esclusivo:	$a \text{ xor } b$
negazione:	$\text{not } a$
eguaglianza:	$a = b$
ineguaglianza:	$a \langle \rangle b$
implicazione:	$a \text{ implies } b$

## i tipi di base Integer e Real

operazioni standard di **Integer** e **Real** :

operazione	notazione	tipo del risultato
eguaglianza	$a = b$	<b>Boolean</b>
ineguaglianza	$a \langle \rangle b$	<b>Boolean</b>
minore	$a < b$	<b>Boolean</b>
maggiore	$a > b$	<b>Boolean</b>
minore o uguale	$a \leq b$	<b>Boolean</b>
maggiore o uguale	$a \geq b$	<b>Boolean</b>
addizione	$a + b$	<b>Integer or Real</b>
sottrazione	$a - b$	<b>Integer or Real</b>
moltiplicazione	$a * b$	<b>Integer or Real</b>
divisione	$a / b$	<b>Real</b>
modulo	$a .\text{mod}(b)$	<b>Integer</b>
divisione intera	$a .\text{div}(b)$	<b>Integer</b>
valore assoluto	$a .\text{abs}()$	<b>Integer or Real</b>
massimo fra due	$a .\text{max}(b)$	<b>Integer or Real</b>
minimo fra due	$a .\text{min}(b)$	<b>Integer or Real</b>
miglior arrotondamento	$a .\text{round}()$	<b>Integer</b>
arrotondamento per difetto	$a .\text{floor}()$	<b>Integer</b>

## il tipo String

sequenza finita di caratteri fra apici:

'ad esempio questa stringa'

operazioni standard di *String*, dove *s*, *t* sono di tipo *String*, e *m*, *n* sono di tipo *Integer*:

operazione	notazione	tipo del risultato
concatenazione	<i>s</i> .concat( <i>t</i> )	<i>String</i>
lunghezza	<i>s</i> .size()	<i>Integer</i>
conversione minuscolo	<i>s</i> .toLowerCase()	<i>String</i>
conversione maiuscolo	<i>s</i> .toUpperCase()	<i>String</i>
sottostringa	<i>s</i> .substring( <i>m</i> , <i>n</i> )	<i>String</i>
eguaglianza	<i>s</i> = <i>t</i>	<i>Boolean</i>
ineguaglianza	<i>s</i> <> <i>t</i>	<i>Boolean</i>

## sintassi delle espressioni di base

per ridurre il numero di parentesi: **precedenza** fra operatori OCL  
in ordine decrescente:

operazioni	operatori
nome di path	::
valore pre- in postcondizione	@pre
operazioni 'dot', 'arrow' e di messaggi	., ->, ^, ^^
operazioni unarie	-, not
moltiplicazione, divisione	*, /
addizione, sottrazione	+, -
operazioni relative di confronto	<, >, <=, >=, <>, =
operazioni booleane	and, or, xor
implicazione booleana	implies

operatori (binari) **infissi**:

+, -, \*, /, <, >, <=, >=, <>, =, and, or, xor, implies

**N.B.:** solo la forma infissa di questi operatori è ammessa

## caratteristiche dei tipi da utente

**tipi definiti dall'utente:** quelli presenti in elementi del modello

le loro *caratteristiche*:

attributi

operazioni

attributi di classe

operazioni di classe

estremi di associazioni

attributi e operazioni sono utilizzabili in espressioni OCL

**N.B.** operazioni: solo *query*

estremi di associazioni: v. appresso

## associazioni, tipi enumerativi

uso di estremi di associazioni (o di nomi di classi associate, per estremi anonimi):

**in espressioni di navigazione**

ciò con "." o "->" a seconda delle molteplicità nel percorso

la navigazione attraverso **classi di associazione** sembra alquanto controversa...

navigazione attraverso relazioni di generalizzazione?

no, non ha senso, non almeno con gli operatori "." e "->"

**associazioni qualificate: navigazione indicizzata**

sintassi: oggetto.navigazione[valoreQual,...]

**tipi enumerativi:**

definiti dall'utente con lo stereotipo <<enumeration>>

sintassi per l'uso in espressioni OCL: Tipo::idValore

## struttura dei tipi di collezioni

un supertipo astratto *Collection*

4 (sotto)tipi concreti: *Set*, *Bag*, *OrderedSet*, *Sequence*

questi ereditano dal supertipo:

operazioni comuni standard

operazioni con significato variante, ridefinito nei sottotipi

*OrderedSet* e *Sequence* lo estendono ulteriormente con altre operazioni varianti, definite solo per essi

i tipi collezione sono polimorfi:

il tipo degli elementi della collezione è generico

specificandolo, si hanno tipi concreti monomorfi:

ad es. *Set(Integer)*, *Bag(Boolean)*, *Sequence(String)*,  
*Set(Studente)*, *Sequence(OrderedSet(Integer))*,  
*Sequence(Sequence(String))*, etc.

## operazioni comuni sulle collezioni

definite dal supertipo astratto *Collection*, dove

*e* è un oggetto del tipo degli elementi nella collezione

*c*, *s* sono collezioni di elementi dello stesso tipo

operazione	tipo del risultato
<i>c</i> ->count( <i>e</i> )	<i>Integer</i>
<i>c</i> ->excludes( <i>e</i> )	<i>Boolean</i>
<i>c</i> ->excludesAll( <i>s</i> )	<i>Boolean</i>
<i>c</i> ->includes( <i>e</i> )	<i>Boolean</i>
<i>c</i> ->includesAll( <i>s</i> )	<i>Boolean</i>
<i>c</i> ->isEmpty()	<i>Boolean</i>
<i>c</i> ->notEmpty()	<i>Boolean</i>
<i>c</i> ->size()	<i>Integer</i>
<i>c</i> ->sum()	<i>Integer</i> o <i>Real</i> o ...

## operazioni varianti su collezioni

operazioni varianti su sottotipi di *Collection*, dove

*e* è un oggetto del tipo degli elementi nella collezione

*c*, *s* sono collezioni "compatibili" di elementi dello stesso tipo

operazione	Set	OrderedSet	Bag	Sequence
=	Y	Y	Y	Y
<>	Y	Y	Y	Y
-	Y	Y	-	-
<i>c</i> ->asBag()	Y	Y	Y	Y
<i>c</i> ->asOrderedSet()	Y	Y	Y	Y
<i>c</i> ->asSequence()	Y	Y	Y	Y
<i>c</i> ->asSet()	Y	Y	Y	Y
<i>c</i> ->excluding( <i>e</i> )	Y	Y	Y	Y
<i>c</i> ->including( <i>e</i> )	Y	Y	Y	Y
<i>c</i> ->flatten()	Y	Y	Y	Y
<i>c</i> ->union( <i>s</i> )	Y	Y	Y	Y
<i>c</i> ->intersection( <i>s</i> )	Y	-	Y	-
<i>c</i> ->symmetricDifference( <i>s</i> )	Y	-	-	-

## operazioni varianti su collezioni ordinate

le seguenti operazioni sono definite solo su collezioni ordinate, dove

*e* è un oggetto del tipo degli elementi nella collezione

*i*, *l* e *u* sono interi positivi (indici di posizione)

operazione	OrderedSet	Sequence
<i>c</i> ->append( <i>e</i> )	Y	Y
<i>c</i> ->at( <i>i</i> )	Y	Y
<i>c</i> ->first()	Y	Y
<i>c</i> ->indexOf( <i>e</i> )	Y	Y
<i>c</i> ->insertAt( <i>i</i> , <i>e</i> )	Y	Y
<i>c</i> ->last()	Y	Y
<i>c</i> ->prepend( <i>e</i> )	Y	Y
<i>c</i> ->subOrderedSet( <i>l</i> , <i>u</i> )	Y	-
<i>c</i> ->subSequence( <i>l</i> , <i>u</i> )	-	Y

## operazioni iterative sulle collezioni

hanno un *body parameter* : un'espressione OCL da valutare su ogni elemento della collezione

possono avere un altro parametro, la *variabile d'iterazione*, riferita all'elemento su cui si valuta il body, e che può occorrere nel body

**sintassi:** `opname(E)`, oppure `opname(v | E)`

nella forma sintattica più semplice, eccole di seguito, dove

*B* è un'espressione di tipo **Boolean**

*E* è un'espressione OCL di qualsiasi tipo

*O* è un'espressione OCL di un tipo sul quale sia definita l'operazione <

`any(B)`

`collect(E)`

`collectNested(E)`

`exists(B)`, `forall(B)`

`isUnique(E)`

`iterate(...)` (v. appresso)

`one(B)`

`select(B)`, `reject(B)`

`sortedBy(O)`

## l'operazione iterate

è la più **generale** delle operazioni iterative

tutte le altre operazioni iterative possono definirsi per il tramite di *iterate*

**sintassi:** `c->iterate(element:Type1; result:Type2 = <expression>  
| <expression-with-element-and-result>)`

**esempi:**

se *c* è di tipo `Collection(Integer)`:

---

```
c->sum() = c->iterate(i:Integer; somma:Integer = 0 | somma + i)
```

---

per ogni collezione *c* di tipo `Collection(t)` ed espressione booleana *B(x)*, con *x:t*

---

```
c->forall(B(x)) = c->iterate(e:t; r:Boolean = true | r and B(e))
```

---

per qualsiasi tipo concreto di collezione *CT* (uno dei 4 sottotipi di `Collection`),  
per ogni collezione *c* di tipo `CT(t)` ed espressione booleana *B(x)*, con *x:t*

---

```
c->select(B(x)) = c->iterate(e:t; r:CT(t) = CT{}  
| if B(e) then r->including(e) else r endif
```

---

## costrutti per postcondizioni

operatore suffisso @pre

variabile predefinita result

operazione booleana oclIsNew() su oggetto

operatore infisso ^ ("hasSent"), booleano:

oggetto^messaggio(...) = true sse l'istanza contestuale ha inviato messaggio(...) a oggetto durante l'esecuzione dell'operazione

operatore infisso ^^ ("messaggi"):

oggetto^^pattern\_messaggio(...) dà la sequenza di messaggi (oggetti del tipo predefinito OclMessage), inviati a oggetto dall'istanza contestuale durante l'esecuzione dell'operazione, che corrispondono al pattern\_messaggio(...)

## conformità, casting, tipo universale

**conversione di tipo:** detta anche *casting*

se  $e$  è di tipo  $t1$ , e  $t2$  è un sottotipo di  $t1$ , allora la conversione di tipo per  $e$  può effettuarsi con la meta-operazione OCL

oclAsType( $t$ :Type) che ha un tipo come parametro:

$e.oclAsType(t2)$

**conformità di tipo:** cf. principio di sostituzione (di Liskov)

se  $t2$  è un sottotipo di  $t1$ , allora è conforme a  $t1$

la relazione di conformità fra tipi è riflessiva e transitiva

se  $t2$  è conforme a  $t1$ , allora  $Collection(t2)$  è conforme a

$Collection(t1)$ , e  $CT(t2)$  è conforme a  $CT(t1)$  per i 4 sottotipi concreti CT di Collection

sottotipi concreti distinti di Collection non sono conformi fra loro

ogni tipo è sottotipo di OclAny : il tipo universale in OCL