Progettazione del Software

Giuseppe De Giacomo, Massimo Mecella,
Dipartimento di Informatica e Sistemistica
SAPIENZA Università di Roma

Analisi

Specifica delle Operazioni & OCL

La specifica

Lo schema concettuale viene alla fine corredato da
- una **specifica** delle operazioni di ogni **Classe**
- una **specifica** delle operazioni dei clienti (vedi dopo)

La specifica delle operazioni di una classe ha lo scopo di definire precisamente il comportamento di ogni operazione della classe

La specifica di delle operazioni dei clienti ha lo scopo di definire precisamente il comportamento di ogni operazione dei clienti

Specifica di operazioni di una classe

La specifica delle operazioni di una classe C ha la seguente forma:

InizioSpecificaOperazioniClasse C

Specifica della operazione 1

…

Specifica della operazione N

FineSpecifica

Specifica di un Cliente

La specifica delle operazioni di un cliente si fornisce facendo la lista delle operazioni (una o più) del cliente stesso, e fornendo poi la specifica di ogni operazione. La specifica delle operazioni di un cliente D ha la seguente forma:

InizioSpecificaOperazioniCliente D

Specifica della operazione 1

…

Specifica della operazione N

FineSpecifica

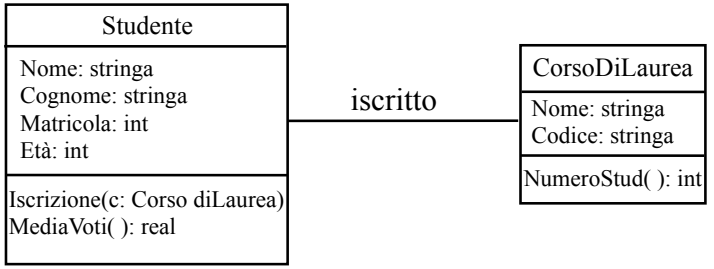
Specifica di una operazione

Che sia una operazione di una classe o una operazione di un cliente, la specifica di una operazione ha la seguente forma:

alfa (X: T1, … , Xn: Tn): T
pre: *condizione*
post: *condizione*

- alfa (X: T1, … , Xn: Tn): T è la **segnatura** dell'operazione (T può mancare)
- pre rappresenta la **precondizione** dell'operazione, cioè l'insieme delle condizioni (a parte quelle già stabilite dalla segnatura) che devono valere **prima** di ogni esecuzione della operazione
- post rappresenta le **postcondizioni** della operazione, cioè l'insieme delle condizioni che devono valere **alla fine** di ogni esecuzione della operazione

Esempio di specifica di una operazione



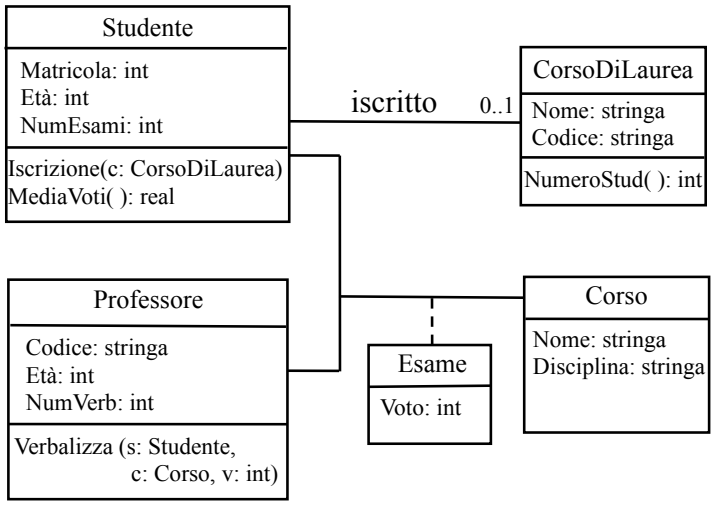
InizioSpecificaOperazioniClasse CorsoDiLaurea
NumeroStud() : int
pre : nessuna
post : **result** è uguale al numero di studenti iscritti nel corso di laurea **self**
FineSpecifica

Precondizioni e postcondizioni

- Nella specifica di una operazione, nella **precondizione** si usa "**self**" per riferirsi all'oggetto di invocazione della operazione
- Nella specifica di una operazione, nella **postcondizione** si usa
  - "**self**" (o a volte "this") per riferirsi all'**oggetto di invocazione** della operazione nello stato corrispondente **alla fine** della esecuzione della operazione
  - "**result**" per riferirsi al **risultato** restituito dalla esecuzione della operazione
  - **"@pre"** per riferirsi al valore della espressione **alfa nello stato corrispondente alla precondizione**

Nota: self denota l'oggetto di invocazione, quindi nel caso di specifica di operazioni dei cliente (dove l'oggetto di invocazione è inesistente) non si può usare.

Esempio di diagramma delle classi



Esempio di specifica di classi

InizioSpecificaOperazioniClasse Professore

Verbalizza(s: Studente, c: Corso, v: int)
pre : s non ha ancora sostenuto l'esame c, e 18 ≤ v ≤ 31
post : self, s e c sono collegati da un link di tipo Esame, con voto v. Inoltre vale che s.NumEsami = s.NumEsami@pre + 1, e self.NumVerb = self.NumVerb@pre + 1

FineSpecifica

InizioSpecificaOperazioniClasse Studente

Iscrizione(c: CorsoDiLaurea)
pre : self non è iscritto ad alcun CorsoDiLaurea
post : self è iscritto al CorsoDiLaurea c,
MediaVoti() : real
pre : self .NumEsami > 0
post : result è la media dei voti degli esami sostenuti da self

FineSpecifica

Frame Problem

- Quando nella post "affermo" qualcosa, cosa succede a tutti gli oggetti su cui non ho affermato nulla ?
- Si assume che gli oggetti su cui non faccio affermazioni siano immutati rispetto allo stato precedente alla esecuzione della operazione, cioè sono come in @pre
- Ma degli oggetti sui cui affermo qualcosa, devo affermare tutto quello che è necessario per specificare correttamente e completamente la loro situazione

Specifica formale

- E' opportuno essere il più rigorosi possibile nel formulare la specifica delle operazioni, per non generare ambiguità.
- Spesso si utilizza una **notazione formale** per la specifica, tipicamente basata sulla teoria degli insiemi e sulla logica del prim'ordine.
- Esistono molti notazioni formali per la specifica. Tra questi noi approfondiremo **OCL - Object Constraints Language**.
- *Ma prima vediamo qualche esempio direttamente espresso usando una notazione mutuata direttamente dai corsi di matematica.*

Specifica mediante una notazione formale

InizioSpecificaOperazioniClasse Professore

Verbalizza(s: Studente, c: Corso, v: int)
pre: ¬(∃ p | p ∈ Professore ∧ <s,p,c> ∈ Esame)
∧ 18 ≤ v ≤ 31

post: Esame = pre(Esame) ∪ {<s,self,c>} ∧
Esame.voto(s,self,c) = v ∧
s.NumEsami = s.NumEsami@pre + 1 ∧
self.NumVerb = self.NumVerb@pre + 1

FineSpecifica

Specifica mediante una notazione formale (2)

InizioSpecificaOperazioniClasse Studente

Iscrizione(c: CorsoDiLaurea)
pre: ¬(∃ c2 | <self,c2> ∈ iscritto)
post: iscritto = iscritto@pre ∪ {<self,c>}

MediaVoti() : real
pre: (∃ c | c ∈ CorsoDiLaurea ∧ <self,c> ∈ iscritto) ∧
self.NumEsami > 0
post: definiamo l'insieme Voti come segue:

Voti = {<c,v > | ∃ prof | <self,prof,c> ∈ Esame ∧ Esame.voto(self,p,c) = v}

$$result = \frac{\sum_{<c,v>\in Voti} v}{self.NumEsami}$$

FineSpecifica

Esempio di specifica di operazioni di un cliente

InizioSpecificaOperazioniCliente Controlli

MediaVoti(s: Studente): real
pre: s.NumEsami > 0
post: result = s.MediaVoti() -- invoca operazione di Studente

NumeroStudenti(c: CorsoDiLaurea): int
pre: nessuna
post: result = c.NumeroStud() -- invoca operazione di CorsoDiLaurea

MediaEsami(c: CorsoDiLaurea): real
pre: c ha almeno uno studente iscritto
post: result è la media del numero di esami sostenuti dagli studenti iscritti al corso di laurea c

FineSpecifica

Specifica mediante una notazione formale

InizioSpecificaOperazioniCliente Controlli

MediaVoti(s: Studente): real
pre: (∃ c | c ∈ CorsoDiLaurea ∧ <s,c> ∈ iscritto) ∧
s.NumEsami > 0
post: result = s.MediaVoti()

NumeroStudenti(c: CorsoDiLaurea): int
pre: true
post: result = c.NumeroStud()

…

FineSpecifica

OCL – Object Constraints Language

- Linguaggio formale per la specifica di "vincoli" (*constraint*) in diagrammi UML
- I vincoli sono asserzioni matematiche (o meglio logiche) sui diagrammi
- Semanticamente OCL è basato sulla teoria degli insiemi e logica del prim'ordine
- Sintatticamente è vicino ad un linguaggio di programmazione

Specifica OCL di una operazione di classe

La specifica di una operazione ha la seguente forma:

```
context <classe>::foo(x1: T1, … , xn: Tn): T
pre: espressione booleana OCL
post: espressione booleana OCL
```

- `context <classe>::foo(…)` In OCL ogni asserzione deve avere un contesto in questo caso l'operazione `foo(…)` della classe `<classe>`
- `foo(x1: T1, … , xn: Tn): T` è la **segnatura** dell'operazione `foo(…)` della classe `<classe>` (`T` manca se l'operazione non restituisce risultati)
- `pre:` rappresenta la **precondizione** dell'operazione, espressa come **una espressione booleana OCL**
- `post:` rappresenta le **postcondizioni** della operazione, espressa come **una espressione booleana OCL**

Uso di `pre` e `post`

- `--` in una **`pre`** significa che non c'è precondizione (cioè la precondizione è semplicemente **`true`**)
- Sia in **`pre`** che in **`post`** si può usare **`self`**, per indicare l'oggetto d'invocazione (quello su cui è invocata l'operazione)
- In **`post`** si possono usare:
  - **`@pre`** per indicare il valore prima dell'invocazione dell'operazione
  - **`result`** per indicare il valore di ritorno
  - **`^`** per indicare invocazione di un'operazione su un oggetto durante l'esecuzione

Sintassi espressioni booleane in OCL

Un'espressione booleana OCL ha la forma

- `E` (espr. booleana atomica, eg. ottenuta,con oper. di confronto ecc.)
- **`not`** ϕ
- ϕ **`and`** θ
- ϕ **`or`** θ
- ϕ **`implies`** θ
- *`<collection>`* **`-> forAll(`**θ**`)`**
- *`<collection>`* **`-> exists(`**θ**`)`**

`forAll()` *ed* **`exists()`** *si applicano solamente a collezioni di oggetti/valori*

Elementi e collezioni

- OCL distingue tra **elementi** del diagramma delle classi UML (oggetti, classi, associazioni, ecc.) e **collezioni** OCL.
- Le collezioni OCL sono collezioni di oggetti/valori tipicamente ottenute come risultato della navigazione attraverso le associazioni, attributi multivalore, ecc.
- Esistono tre differenti tipi di collezioni in OCL
  - Set (non ordinati, elementi unici)
  - Bag (non ordinati, elementi non unici)
  - Sequence (ordinati, elementi non unici)
- Esistono molti operatori predefiniti sulle collezioni OCL. Si utilizzano con la notazione `collection->operation`

Elementi e collezioni

- OCL distingue tra **elementi** del diagramma delle classi UML (oggetti, classi, associazioni, ecc.) e **collezioni** OCL.
- Le collezioni OCL sono collezioni di oggetti/valori tipicamente ottenute come risultato della navigazione attraverso le associazioni, attributi multivalore, ecc.
- Esistono tre differenti tipi di collezioni in OCL
  - Set (non ordinati, elementi unici)
  - Bag (non ordinati, elementi non unici)
  - Sequence (ordinati, elementi non unici)
- Esistono molti operatori predefiniti sulle collezioni OCL. Si utilizzano con la notazione `collection->operation`

Uso di "->" (arrow/freccia) invece di "." (dot/punto)

Operatori di selezione: . e ->

- L'operatore . è l'operatore di selezione sugli elementi UML (e.g., oggetti):

  `self.NumEsami` -- restituisce num esami

  Se applicato ad una collezione, esso si riferisce ai singoli elementi (e.g., oggetti) UML della collezione:

  `self.Esame.voto` -- restituiscre collezione dei voti

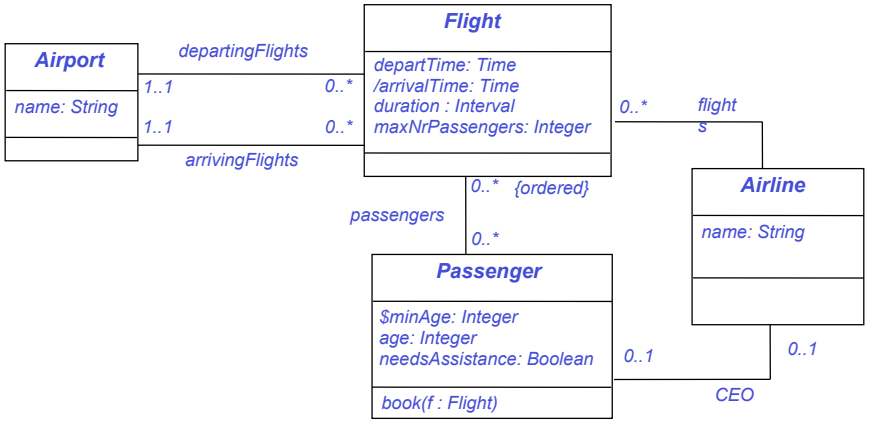
- L'operatore -> è l'operatore di selezione sulle collezioni OCL. Esso serve ad invocare operazioni sulla collezione vista come una unità.

  `self.Esame.voto -> sum()` -- restituisce somma voti

Nota

- Un elemento UML (se è una collezione) può essere usato come una collezione OCL
- Una collezione OCL **con un solo elemento** può essere usata come un elemento: l'elemento in essa contenuto
- Uso della freccia (->) o del punto (.) distingue tra i due utilizzi

Esempio (1)

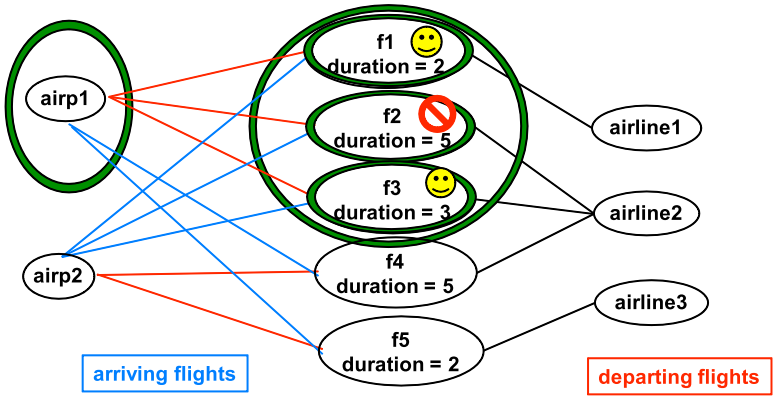


select

- *Sintassi*: `collection->select(elem|expr)`
- *Semantica:* denota il sottoinsieme di tutti gli elementi di `collection` per i quali `expr` è vera
- *Nota*: `elem` può essere omesso se non necessario
  `collection->select(expression)`

Esempio (2)

context Airport inv:
self.departingFlights->select(duration<4)->notEmpty()

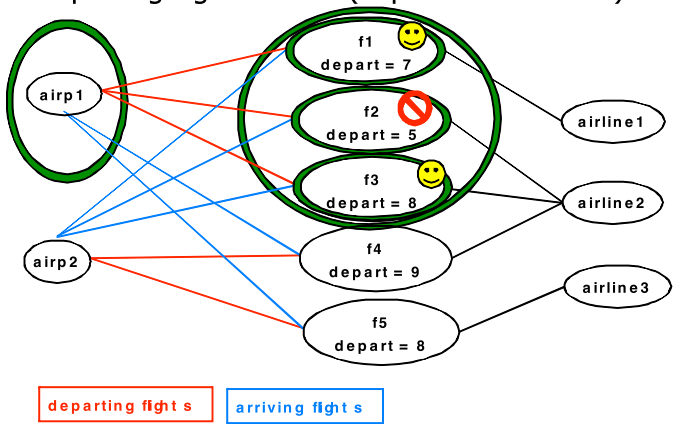


forAll

- *Sintassi*: `collection->forAll(elem|expr)`
- *Semantica*: corrisponde al quantificatore universale ∀: è vero sse tutti gli elementi di `collection` soddisfano `expr`
- *Nota*: `elem` può essere omesso se non necessario
  `collection->forAll(expr)`

Esempio (3)

context Airport inv:
self.departingFlights->forAll(departTime.hour>6)

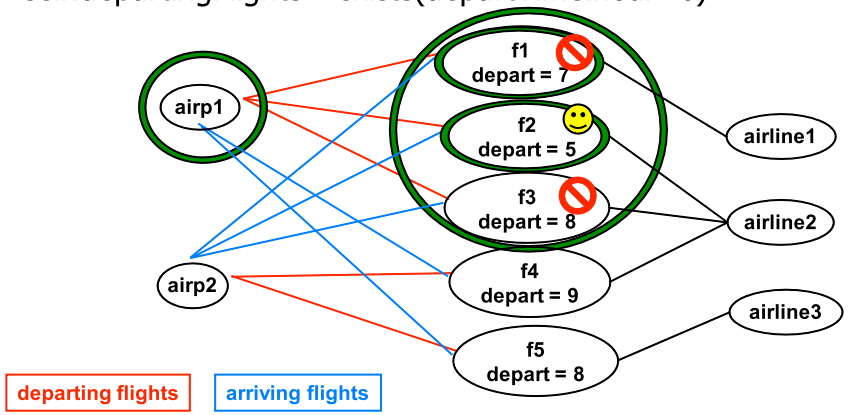


exists

- *Sintassi*: `collection->exists(elems|expr)`
- *Semantica*: corrisponde al quantificatore esistenziale ∃: è vero sse esiste almeno un elemento di `collection` che soddisfa `expr`
- *Note*: `elem` può essere omesso se non necessario
  `collection->exists(expr)`

Esempio (4)

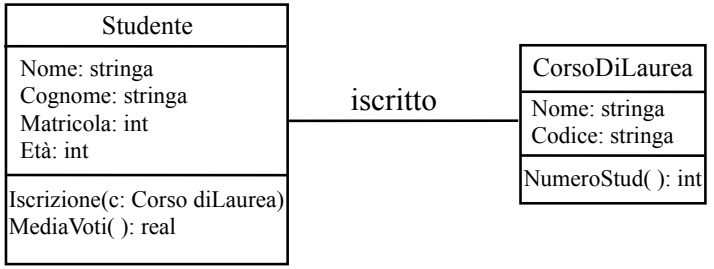
context Airport inv:
self.departingFlights->exists(departTime.hour<6)



Operatori più usati sulle collezioni

- *isEmpty()*: vera se `collection` non ha elementi
- *notEmpty()*: vera se `collection` ha almeno un elemento
- *size()*: numero di elementi nella `collection`
- *count(elem)*: numero di occorrenze di `elem` in `collection`
- *includes(elem)*: vera se `elem` è in `collection`
- *excludes(elem)*: vera se `elem` non è in `collection`
- *includesAll(coll)*: vera se tutti gli elementi di `coll` sono in `collection`
- *excludesAll(coll)*: vera se tutti gli elementi di `coll` non sono in `collection`
- *union(coll):* `collection ∪ coll`
- *intersection(coll):* `collection ∩ coll`
- *any(expr) :* restituisce un qualsiasi elemento di `collection` che soddisfa `expr`

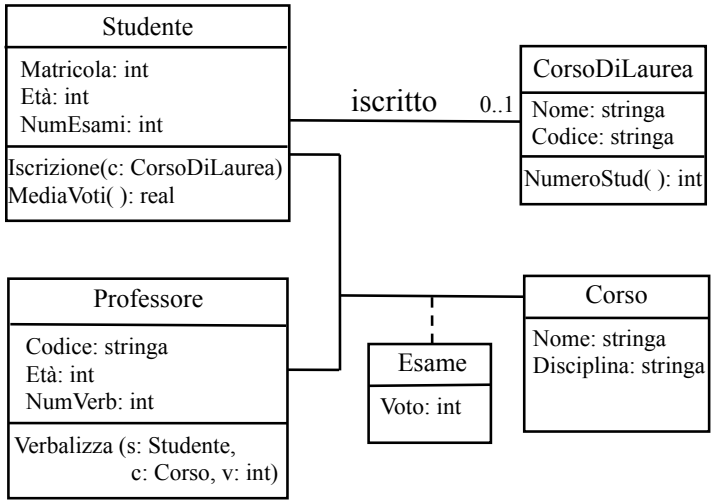
Esempio di specifica di una operazione




```
context  CorsoDiLaurea::
         NumeroStud(): Integer
pre:     --
post:    result = self.iscritto->size()
```

Non c'è precondizione, il risultato è uguale al numero di studenti iscritti al CorsoDiLaurea su cui viene invocata l'operazione

Esempio di diagramma delle classi



Specifica OCL

```
context Professore::
        Verbalizza(s: Studente, c: Corso, v: Integer)
pre:  v >= 18 and v <= 31 and
      not Esame.allInstances() -> exists(ee |
                                          ee.Corso = c and
                                          ee.Studente = s)

post: Esame.allInstances()->
        exists(ee|
           ee.Professore = self and ee.Corso = c and
           ee.Studente = s and ee.voto = v and
           Esame.allInstances()->
            forall(e|Esame@pre.allInstances()->includes(e) or e = ee)
        )
        and
        self.NumVerb = (self.NumVerb@pre) + 1
        and
        s.NumEsami = s.NumEsami@pre + 1
```


`allInstances`

- Operatore **`allInstances`** permette di costruire una collezione di oggetti a partire dalla classe di contesto

```
context Person
… Person.allInstances()->forAll
            (p | p. parents->size() <= 2)
```


Specifica OCL (alternativa)

```
context Professore::
        Verbalizza(s: Studente, c: Corso, v: Integer)
pre:  v >= 18 and v <= 31 and
      not Esame.allInstances() -> exists(ee |
                                          ee.Corso = c and
                                          ee.Studente = s)

post: let NE = Esame.allInstances() -> select(ee |
                                    ee.Professore = self and
                                    ee.Corso = c and
                                    ee.Studente = s and
                                    ee.voto = v)
      in
      NE.notEmpty() and
      Esame.allInstances() = Esame@pre.allInstances() -> union(NE)
      and
      self.NumVerb = (self.NumVerb@pre) + 1
      and
      s.NumEsami = s.NumEsami@pre + 1
```

`let` … `in`

- **`let`** … **`in`** serve per definire nuove espressioni da utilizzare durante la specifica

- Per esempio:

```
let NE = Esame.allInstances() -> select(ee |
                                  ee.Professore = self and
                                  ee.Corso = c and
                                  ee.Studente = s and
                                  ee.voto = v)
in
NE.notEmpty() and …
```

definisce l'insieme NE da usare successivamente nella specifica

Verifica appartenenza a classi in OCL

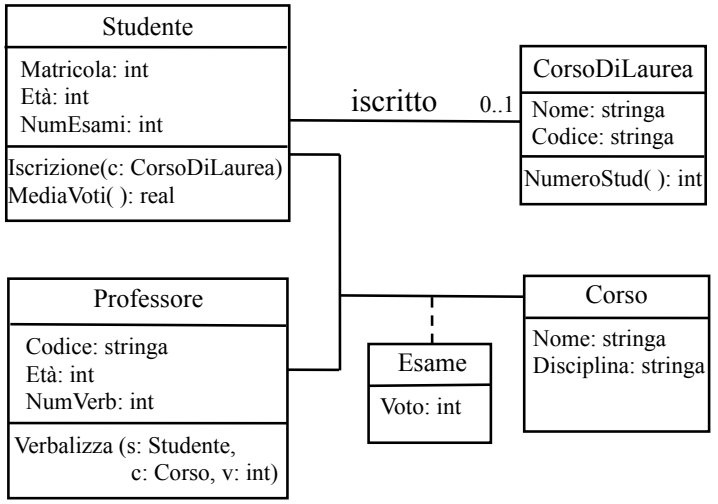
- OCL ha due operatori per scrivere espressioni relative a gerarchie di ereditarietà:
  - **`oclIsTypeOf(<class>)`** restituisce true se l'oggetto di invocazione ha come classe più specifica **`<class>`**
  - **`oclIsKindOf(<class>)`** restituisce true se l'oggetto è istanza della classe **`<class>`**

Specifica operazioni clienti

- E' analoga alla sepecifica di operazioni di classi ma senza la possibilità di fare riferimento a `self`

- Ricordiamo infatti che `self` denota l'oggetto di invocazione che nel caso delle operazioni dei clienti è inesistente.

Esempio di diagramma delle classi



Esempio di specifica di operazioni di un cliente

InizioSpecificaOperazioniCliente Controlli

```
context Controlli::MediaVoti(s: Studente): real
pre:  s.NumEsami > 0
post: result = s^MediaVoti()  --invoca oper. di Studente

context Controlli::NumeroStudenti(c: CorsoDiLaurea): int
pre:  --
post: result = c^NumeroStud()  --invoca oper. di CorsoDiLaurea

context::MediaEsami(c: CorsoDiLaurea): real
pre:  c.allInstances()->nonEmpty()
      -- ha almeno uno studente iscritto
Post: result = c.iscritto.Studente.NumeroEsami -> sum()
        / c.iscritto->size()
      -- è la media del numero di esami sostenuti dagli studenti iscritti c
```

FineSpecifica

Uso di OCL per formalizzare vincoli sul diagramma delle classi UML

- Inoltre si possono esprimere vincoli sul diagramma delle classi come "**invarianti**", ovvero espressioni che devono essere vere per tutte le possibili istanziazioni del diagramma delle classi
- Da usare nei commenti del diagramma delle classi al posto del linguaggio naturale

Esempi di vincolo OCL sul diagramma delle classi UML

```
context Esame
inv 18 <= voto and voto <= 31
```

I voti degli esami sono compresi tra 18 e 31 (31 per la lode)

```
context Esame
inv Esame.allInstances()->forall(e1 |
      Esame.allInstances()->forall(e2 |
       e1.Studente = e2.Studente and e1.Corso = e2.Corso
                  implies e1=e2
      )
    )
```

Non possono esistere due esami distinti che riguardano lo stesso studente e lo stesso corso

Operatori OCL

- `--` (inline) o `/* */` (più righe) indicano un commento
- Operatori OCL da usare nella specifica:

Operatori sul tipo generico

Expression	Result Type
object = (object2 : OclAny)	Boolean
object <> (object2 : OclAny)	Boolean
object.oclIsUndefined()	Boolean
object.oclIsKindOf(type : OclType)	Boolean
object.oclIsTypeOf(type : OclType)	Boolean
object.oclIsNew()	Boolean
object.oclInState()	Boolean
object.oclAsType(type : OclType)	type
object.oclInState(str: StateName)	Boolean
type::allInstances()	Set(type)

`oclIsNew()`: *L'oggetto di invocazione è un nuovo oggetto è stato creato durante l'esecuzione dell'operazione*

Operation	Notation	Result Type
or	a or b	Boolean
and	a and b	Boolean
exclusive or	a xor b	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implies	a implies b	Boolean

Operatori predefiniti sui Boolean

Operation	Notation	Result Type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiplication	a * b	Integer or Real
division	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs()	Integer or Real
maximum of a and b	a.max(b)	Integer or Real
minimum of a and b	a.min(b)	Integer or Real
round	a.round()	Integer
floor	a.floor()	Integer

Operatori predefiniti sui Real ed Integer

Operation	Expression	Result Type
concatenation	string.concat(string)	String
size	string.size()	Integer
to lower case	string.toLower()	String
to upper case	string.toUpper()	String
substring	string.substring(int,int)	String
equals	string1 = string2	Boolean
not equals	string1 <> string2	Boolean

Operatori predefiniti su String

Operation	Description
count(object)	The number of occurrences of the object in the collection
excludes(object)	True if the object is not an element of the collection
excludesAll (collection)	True if all elements of the parameter collection are not present in the current collection
includes(object)	True if the object is an element of the collection
includesAll (collection)	True if all elements of the parameter collection are present in the current collection
isEmpty()	True if the collection contains no elements
notEmpty()	True if the collection contains one or more elements
size()	The number of elements in the collection
sum()	The addition of all elements in the collection. The elements must be of a type supporting addition (such as *Real* or *Integer*).

Operatori predefiniti sulle collezioni

Operation	Set	OrderedSet	Bag	Sequence
=	X	X	X	X
<>	X	X	X	X
-	X	X	-	-
append(object)		X	-	X
asBag()	X	X	X	X
asOrderedSet()	X	X	X	X
asSequence()	X	X	X	X
asSet()	X	X	X	X
at(index)	-	X	-	X
excluding(object)	X	X	X	X
first()	-	X	-	X
flatten()	X	X	X	X
including(object)	X	X	X	X
indexOf(object)	-	X	-	X
insertAt(index, object)	-	X	-	X
intersection(coll)	X	-	X	-
last()	-	X	-	X
prepend(object)	-	X	-	X
subOrderedSet(lower, upper)	-	X	-	-
subSequence(lower, upper)	-	-	-	X
symmetricDifference(coll)	X	-	-	-
union(coll)	X	X	X	X

Dettaglio sull'operatore `flatten()`

```
Set { Set { 1, 2 }, Set { 2, 3 }, Set { 4, 5, 6 } }
Set { 1, 2, 3, 4, 5, 6 }
```


Operation	Description
any(expr)	Returns a random element of the source collection for which the expression *expr* is true
collect(expr)	Returns the collection of objects that result from evaluating *expr* for each element in the source collection
collectNested (expr)	Returns a collection of collections that result from evaluating *expr* for each element in the source collection
exists(expr)	Returns true if there is at least one element in the source collection for which *expr* is true
forAll(expr)	Returns true if *expr* is true for all elements in the source collection
isUnique(expr)	Returns true if *expr* has a unique value for all elements in the source collection
iterate(...)	Iterates over all elements in the source collection
one(expr)	Returns true if there is exactly one element in the source collection for which *expr* is true
reject(expr)	Returns a subcollection of the source collection containing all elements for which *expr* is false
select(expr)	Returns a subcollection of the source collection containing all elements for which *expr* is false
sortedBy(expr)	Returns a collection containing all elements of the source collection ordered by *expr*

ESEMPI

Specifica in linguaggio naturale

InizioSpecificaClasse Studente

Iscrizione(c: CorsoDiLaurea)
pre : this non è iscritto ad alcun CorsoDiLaurea
post : this è iscritto al CorsoDiLaurea c,
MediaVoti() : real
pre : this.NumEsami > 0
post : result è la media dei voti degli esami sostenuti da this
FineSpecifica

Specifica mediante la logica

InizioSpecificaOperazioniClasse Studente

Iscrizione(c: CorsoDiLaurea)
pre : ¬(∃ c2 | <this,c2> ∈ iscritto)
post : iscritto = pre(iscritto) ∪ {<this,c>}

MediaVoti() : real
pre : this.NumEsami > 0
post : definiamo Voti come l'insieme
{ v ∈ int | ∃ p | p ∈ Professore ∧ c ∈ Corso ∧
<this,p,c> ∈ Esame ∧ (this,p,c).voto = v}

$$result = \frac{\sum_{v \in Voti} v}{this.NumEsami}$$

FineSpecifica

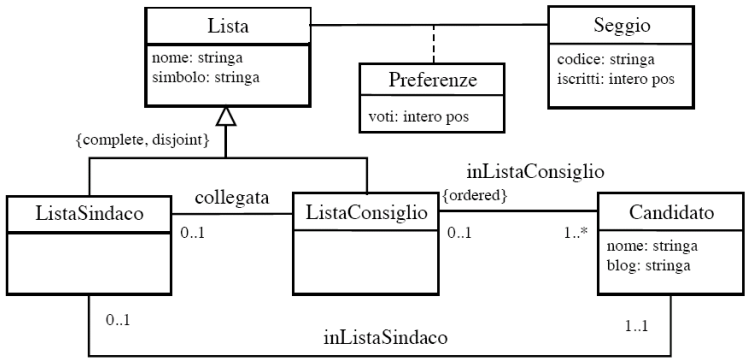
Specifica OCL

```
context Studente::Iscrizione(cl: CorsoDiLaurea)
pre:   not iscritto.allInstances() -> exists(is | is.Studente = self)
post:  iscritto.allInstances() ->
       exists(is|is.Studente = self and is.CorsoDiLaurea = cl and
          iscritto.allInstances()->
             forAll(i|iscritto@pre.allInstances()->includes(i) or i = is)
       )


context Studente:: MediaVoti() : Real
pre:   self.NumEsami <> 0
post:  result = (self.Esame.voto -> sum()) / self.NumEsami
```

Specifica OCL (alternativa)

```
context Studente::Iscrizione(cl: CorsoDiLaurea)
pre:   not iscritto.allInstances() -> exists(is | is.Studente = self)
post:  let NI = iscritto.allInstances() -> select(is |
                                  is.Studente = self and
                                  is.CorsoDiLaurea = cl)
       in
       NI.notEmpty() and
       iscritto.allInstances() = iscritto@pre.allInstances() -> union(NI)


context Studente:: MediaVoti() : Real
pre:    self.NumEsami <> 0
post:   result = (self.Esame.voto -> sum()) / self.NumEsami
```


Altro Esempio (1)



Altro Esempio (2)

- Siamo interessati ad effettuare diversi controlli sui seggi e le liste, in particolare:
  - data una lista l, restituire l'insieme dei candidati a sindaco contenuti in l, cioè se l è una lista per il sindaco restituire l'insieme costituito dal solo candidato presente in essa, se l è una lista per il consiglio restituire l'insieme formato dai candidati presenti in essa che sono anche candidati in una lista per il sindaco;
  - dato un seggio s, restituire la percentuale di votanti, cioè il rapporto tra voti ottenuti dalle varie liste nel seggio s e numero di iscritti ad s

Altro Esempio (3)

InizioSpecificaOperazioniCliente Controlli

CandidatiSindaco : Insieme(Candidato)

pre: *true*

post: Definiamo l'insieme C tale che:

se $l \in ListaSindaco$

$$C \doteq \{c \in Candidato \mid (c,l) \in inListaSindaco\}$$

altrimenti, se $l \in ListaConsiglio$:

$$C \doteq \{c \in Candidato \mid (c,l) \in inListaConsiglio \wedge \exists l' \in ListaSindaco.(c,l') \in ListaSindaco\}$$

$result = C$

PercentualeVotanti : Real

pre: $s.iscritti \neq 0$

post: Definiamo l'insieme

$$C \doteq \{(l,v) \mid (s,l) \in preferenze \wedge preferenze.voti(s,l) = v\}$$

$result = (\sum_{(l,v)\in C} v \setminus s.iscritti) * 100$

Fin

Specifica OCL

```
InizioSpecificaOperazioniCliente Controlli
context Controlli::candidatiSindaco(l:Lista):Set<Candidato>
pre :   --
post :  (l.oclIsTypeOf(ListaSindaco) implies
               result = l.inListaSindaco.Candidato
        )
        and
        (l.oclIsTypeOf(ListaConsiglio) implies
               result = l.inListaConsiglio.Candidato ->
                     select (c | c.inListaSindaco -> notEmpty())
        )


context Controlli::percentualeVotanti(s:Seggio):Real
pre :   s.iscritti <> 0
post :  result = ((s.Preferenze.voti -> sum()) / s.iscritti)*100
FineSpecifica
```