Programmazione a Oggetti
Lezione 5

Inside Collection Classes

Contenuto della lezione

- Implementazione di alcuni metodi visti sulle collection classes
- Obiettivi:
  - far vedere come viene usata l'astrazione fornita da Smalltalk
  - Le collection classes sono implementate in Smalltalk!

collect

- Restituisce una nuova collezione con gli oggetti modificati dall' esecuzione di un blocco:

```
collect: aBlock
 |newCollection|
newCollection := self species new.
self do: [:each | newCollection
  add:(aBlock value:each)]
^newCollection
```

species

- Restituisce la specie cui appartiene un' istanza;
- Non si usa direttamente il metodo **`class`** perche' potrebbe essere utile ridefinire il comportamento di **`species`**:

```
species
        ^self class
```

inject

- Ripete il calcolo di una operazione su tutti gli elementi di una collezione partendo da un valore base (utile per calcolare sommatorie, produttorie etc.)

```
inject: t1 into: t2
 |t3|
 t3 := t2.
 self do: [:t4 | t3:= t1 value: t3
                       value: t4].
  ^t3
```

addAll

- Implementata in **Collection**
- Usa **add**. Rimane parametrica rispetto a future implementazione di **add**:

```
addAll: t1
  t1 do: [:t2 | self add: t2].
  ^t1

add: t1
  self subclassResponsibility.
  ^self
```

Add (in Set)

- Evita di generare i duplicati (influenza tutti i metodi che costruiscono insiemi e usano add):

```
add: t1
  | t2 |
  t1 == nil ifTrue: [^t1].
  t2 := self findElementOrNil: t1.
  (self basicAt: t2)== nil
     ifTrue: [self atNewIndex: t2 put: t1].
  ^t1
```

findElementOrNil (in Set - private)

- Cerca un oggetto in un Set
- Restituisce l'indice di dove si trova o l'indice dove dovrebbe essere inserito
  - Calcola il valore hash dell'elemento (e lo scala rispetto alla dimensione dell'insieme)
  - Se lo slot e' libero o e' occupato dall'oggetto ha finito
  - Altrimenti devo verificare che l'oggetto sia quello che cerco (ci potrebbero essere collisioni nella funzione hash)
  - Si scorrono tutti gli indici fino alla fine o fin quando si trova l'oggetto cercato
  - Se si arriva alla fine senza trovarlo, si alloca nuova memoria per far crescere la collezione

findElementOrNil

```
findElementorNil: anObject
| index length probe pass|
length := self basicSize.
pass:=1.
index:=self initialIndexFor: (anObject hash)
    boundedBy: lenght.
[(probe:=self basicAt: index)==nil
or:[probe=anObject]]
    whileFalse [(index:=index+1)>length
          ifTrue: [index:=1.
                   pass:=pass+1.
                   pass > 2
                      ifTrue: [^self grow
findElementorNil: anObject]]].
^index
```

addAll

- Osservate che addAll non duplica gli elementi della collezione che viene aggiunta
- Quindi dopo l'istruzione:

```
t1 addAll: t2
```

- Modifiche agli oggetti della collezione t2 (ancora accessibili via t2) influenzano la collezione t1.

Set: osservazioni

- Implementati (per efficienza) con strutture a indici, ma non accessibili con indici
- Importanza della funzione hash (da ridefinirsi con =. E' opportuno che elementi uguali abbiano valori hash uguali)

```
hash
    ^xcoord hash

= aPoint
    ^self xcoord = aPoint xcoord
```

Remove (Array, eredita da Collection)

```
remove: t1 ifAbsent: t2
  self subclassResponsibility.
  ^self
```

Il metodo non e' implementato

Remove (Ordered Collection)

```
remove: t1 ifAbsent: t2

  | t3 t4 |
  t3 := firstIndex.
  t4 := lastIndex.
  [t3 <= t4]
     whileTrue:
      [t1 = (self basicAt: t3)
         ifTrue:
           [self removeIndex: t3.
                      ^t1].
           t3 := t3 + 1].
  ^t2 value
```

RemoveIndex (OrderedCollection - private)

```
removeIndex: t1
   | t2 |
   t2 := t1.
   [t2 < lastIndex]
      whileTrue:
       [self basicAt: t2 put:
            (self basicAt: t2 + 1).
           t2 := t2 + 1].
   self basicAt: lastIndex put: nil.
   lastIndex := lastIndex - 1.
   ^self
```

Size e Capacity

- size e' il numero di elementi presenti nella collezione
- capacity e' il numero massimo di elementi memorizzabili nella collezione
- Capacity puo' essere modificata dinamicamente
- Conviene pero' prevedere la massima dimensione di una collezione

```
tmp:=OrderedCollection new:15
```

Conversioni: asArray (SequenceableCollection)

```
asArray
 | t1 t2 t3 |
 t1 := Array new: self size.
 t2 := 1.
 t3 := self size.
 [t2 <= t3]
    whileTrue:
    [t1 at: t2 put: (self at: t2).
        t2 := t2 + 1].
 ^t1
```