Programmazione a Oggetti

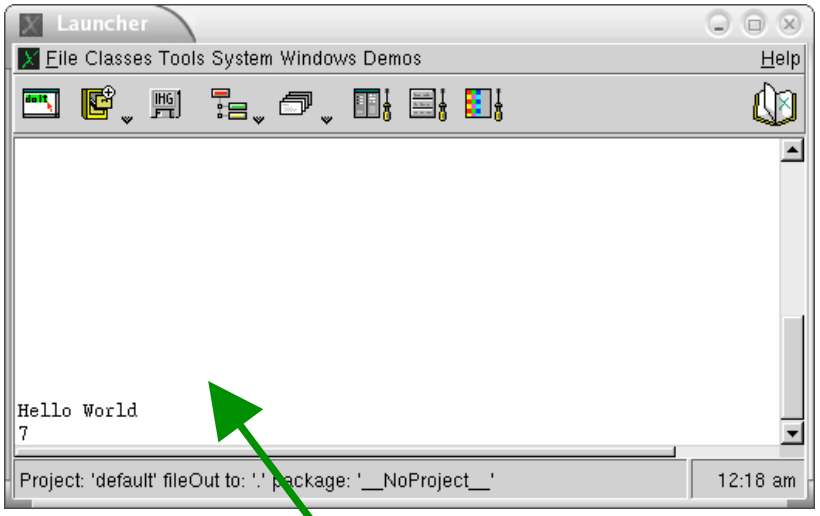
Lezione 6

L'interfaccia grafica
Le Torri di Hanoi

L'interfaccia grafica

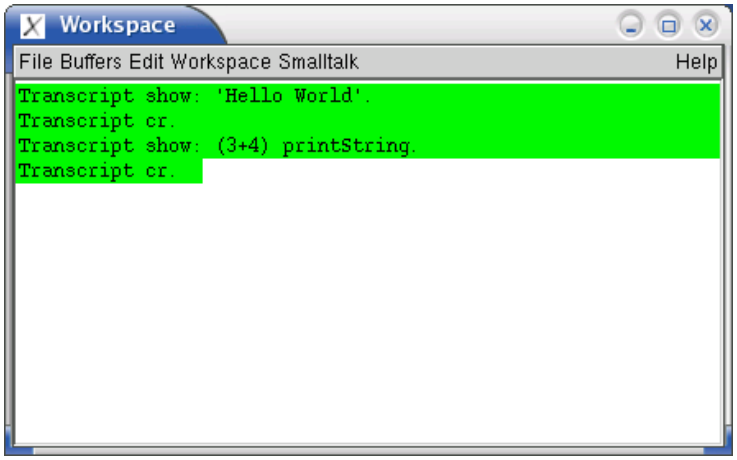
- Launcher: la finestra principale per accedere ai tool del sistema
- Browser: permette di vedere, modificare ed inserire nuove classi e metodi
- Workspace: per interagire con il sistema
- Transcript: per visualizzare la risposta dei programmi

L'interfaccia grafica

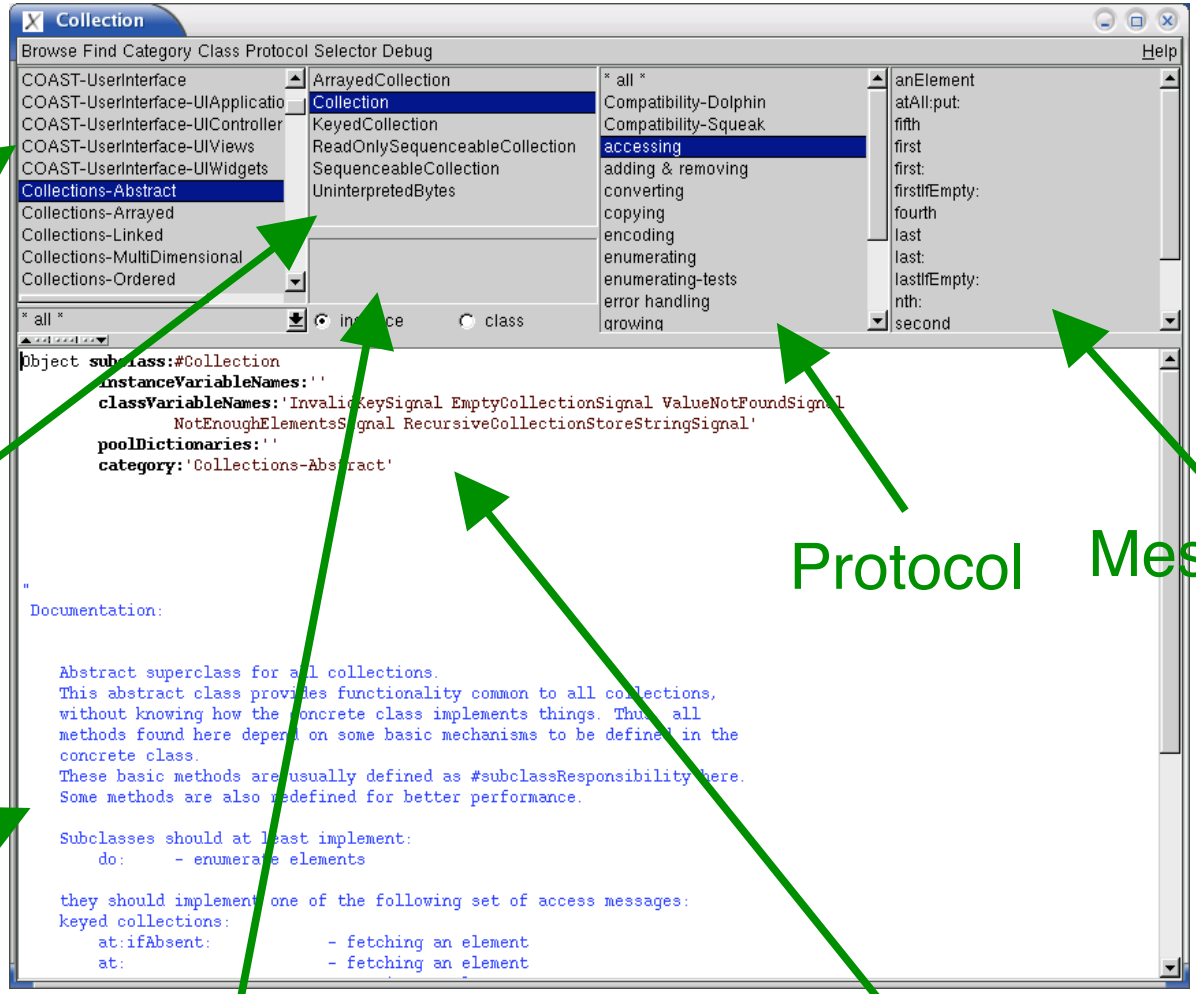


Per eseguire il blocco nel workspace: selezionare il testo e inviare "do it".

Transcript



System browser



Category

Class

Commento

Variabili

Definizione

Protocol

Message

Nuove classi

- Ex novo: inserendo tutti i dati nella finestra di testo del browser

```
Object subclass:#MyCollection
      instanceVariableNames: ''
      classVariableNames: ''
      poolDictionaries: ''
      category: 'Collections-
  Abstract'
```

Nuove classi 2

- Per modifica di classi esistenti.
  La finestra testo del browser pu_ essere editata. Per salvare le modifiche _ sufficiente inviare il messaggio "accept" .

- Il commento alla classe non viene inserito durante la definizione della classe. Va inserito a parte.
  Selezionare "comment" dal menu opportuno.

Metodi

- Metodo di creazione e modifica simile a quello delle classi.
- Si possono automaticamente create i metodi di accesso e modifica. Protocollo: “accessing”.

- Protocolli molto utili per classificare i metodi in base alle loro funzioni.

Interfaccia Testo

- E' il formato con cui vengono esportati e importati file di testo anche nei sistemi con un interfaccia grafica.
- Creazione di una classe:

```
Object subclass: #Account
     instanceVariableNames: 'balance'
     classVariableNames: ''
     poolDictionaries: ''
     category: nil !
```

Interfaccia Testo 2

- Commenti: inseriti e visualizzati per mezzo di oppotuni messaggi alla classe:

```
account comment:
     'I represent a place to deposit
and withdraw money' !


(Account comment) print !
```

Interfaccia Testo 3

- Metodi di classe

```
!Account class methodsFor: 'instance creation'!
      new
            | r |
            r := super new.
            r init.
            ^r
      ! !
```

- Metodi di istanza

```
!Account methodsFor: 'instance initialization'!
        init
            balance := 0
        ! !
```

Interfaccia Testo 4

- Schema generale per la definizione dei metodi

```
! <class expression>
     methodsFor: <category name> !

     <method definition 1> !
     <method definition 2> !
     ...
     <method definition n> ! !
```

Torri di Hanoi

- Creo dei metodi in un nuovo protocollo (games) nella classe Object:

- Osservate che in questo problema non sono necessarie variabili di istanza.

Torri di Hanoi: movetower

- Il metodo che esegue la ricorsione:

```
moveTower: height from: fromPin
          to: toPin using: usingPin
"Recursive procedure to move the disk at a
height from one pin to another pin using a
third pin"

(height > 0) ifTrue: [
    self moveTower: (height-1) from: fromPin
                to: usingPin using: toPin.
    self moveDisk: fromPin to: toPin.
    self moveTower: (height-1) from:usingPin
                to: toPin using: fromPin]
```

Torri di Hanoi: movedisk

- Il metodo che visualizza le mosse:

```
moveDisk: fromPin to: toPin
"Move a disk from a pin to another
 pin. Print the result in the
 Transcript window"

Transcript cr.
Transcript show: (fromPin printString,
                  '->', toPin printString).
```

Torri di Hanoi: esecuzione

- Nel workspace:

```
(Hanoi new) moveTower: 3
     from: 1 to: 3 using: 2.
```

- Output nel Transcript:

```
1->3
1->2
3->2
1->3
2->1
2->3
1->3
```

Torri di Hanoi: esecuzione

- E se scrivessimo:

```
(Hanoi new) moveTower: 3
  from: 'North' to: 'South' using: 'East'.
```

- No problem!

```
North->South
North->East
South->East
North->South
East->North
East->South
North->South
```

Guidando l'input...

- Possiamo scrivere una procedura che chiede l'input all'operatore:

```
hanoi
"Tower of Hanoi program: ask the number of
  disk"
|height aString|
aString:=fillnTheBlank request:
  'Please type the number of disk'.
height:= aString asNumber.
self moveTower:height from:1 to:3 using:2.
"(Object new) hanoi"
```

Verso una versione grafica…

- Scriviamo ora un programma che effettivamente muove dei dischi su pile:

```
Object subclass:#TowerOfHanoi
  instanceVariableNames: 'stacks'
      classVariableNames: ''
      poolDictionaries: ''
      category: 'games'
```

Overriding dei metodi

- Il metodo **hanoi** dovra' ora anche inizializzare gli stack:

```
hanoi
"Tower of Hanoi program: ask the number of
  disk"
|height aString|
heigth:=(fillnTheBlank request:
  'Please type the number of disk') asNumber.
stacks:=(Array new:3).
(1 to 3) do: [:index|stacks at:index
  put:stack new]
(heigth to:1 by:-1) do: [:each|(stacks at:1)
  push: each]
self moveTower:height from:1 to:3 using:2.
"(TowerOfHanoi new) hanoi"
```

Overriding dei metodi

- Il metodo **moveDisk** dovra' ora anche modificare gli stack. Riusciamo a riusare il vecchio metodo:

```
moveDisk: fromPin to: toPin
"move a disk form a pin to another"
|disk|
 disk:= (stacks at: fromPin) pop.
 (stacks at: toPin) push: disk.
 super moveDisk: fromPin to: toPin.
 Transcript show: (` `, disk
  printString).
```

E se volessi uan versione grafica?

- Potrei definire una sottoclasse **`GraphicTowerofHanoi`**
- Sufficiente Ridefinire la **`moveDisk`**
- Osservare che e' necessario dove andare a disegnare e cancellare i dischi (se gli stack hanno un metodo size, no problem!)
- Osservare come sia piu' facile riusare il codice quando i metodi fanno operazioni elementari