CS 535 Object-Oriented Programming & Design
Fall Semester, 2001
Doc 10 More Collections
Contents

Arrays

Similar to arrays in other languages

Once created can not grow

Some Array Operations

```
a := #( 1 3 5 7 6 4 2 ).
b := Array with: 2 with: 9.

secondElement := a at: 2.
firstElement := a first.
a
      at: 1
      put: 12.

jointList := a , b.          "Comma concatinates collections"
sublist := a
      copyFrom: 2
      to: 4.
copyList := a copyUpTo: 6.
copyWithout := jointList copyWithout:  2.

fiveIndex := a indexOf: 5.
location := a
      indexOfSubCollection: #( 6 4)
      startingAt: 2
      ifAbsent: [-1].
a
      replaceAll: 3
      with: 12.
numberOfElements := a size.
even := jointList select:  [:each | each even].
```

OrderedCollections

A growable array
- When add elements, OrderedCollection object grows if needed

Like Java's Vector or ArrayList
Use more than arrays

Some OrderedCollection Operations

```
a := #( 1 3 5 7 6 4 2 ) asOrderedCollection.
b := OrderedCollection new.
b
        add: 2;
        add: 5.

secondElement := a at: 2.
firstElement := a first.
a
        at: 1
        put: 12.

jointList := a , b.
sublist := a
        copyFrom: 2
        to: 4.
copyList := a copyUpTo: 6.
copyWithout := jointList copyWithout:  2.

fiveIndex := a indexOf: 5.
location := a
        indexOfSubCollection: #( 6 4)
        startingAt: 2
        ifAbsent: [-1].
a
        replaceAll: 3
        with: 12.
numberOfElements :=  a size.
evenElements := jointList select:  [:each | each even].

a remove: 5
```

Dictionary

A hash table, like Java's Hashtable or HashMap

In arrays and ordered collections indexes are integers

In hash tables indexes can be any object

```
| phoneNumbers |

phoneNumbers := Dictionary new.
phoneNumbers
      at: 'whitney'
      put: '594-3535'.
phoneNumbers
      at: 'beck'
      put: '594-6807'.
phoneNumbers
      at: 'donald'
      put: '594-7248'.

phoneNumbers at: 'donald'            "Returns '594-7248' "

phoneNumbers
      at: 'sam'
      ifAbsent: ['Not found'].

phoneNumbers do:
      [:value |
      Transcript
            show: value;
            cr].
```

"Continued from previous slide"

```
keyphoneNumbers keysDo:
    [:key |
    Transcript
        show: key;
        cr].

phoneNumbers keysAndValuesDo:
    [:key :value |
    Transcript
        show: key;
        tab;
        show: value;
        cr].
```

Strings & Symbols

A String is an array of characters

'The cat in the hat'

Characters can be any Unicode character

Symbols are strings that are represented uniquely

Examples of symbols

#ASymbol
#'CanUseSingleQuotes'
#cat

There is only one copy of a symbol with a given sequence of characters in the image

'cat' = 'cat' "true"
'cat' == 'cat' "false"

#cat = #cat "true"
#cat == #cat "true"

Identity Problem

```
| phoneNumbers |
phoneNumbers := Dictionary new.
phoneNumbers
      at: 'whitney'
      put: '594-3535'.
phoneNumbers
      at: 'beck'
      put: '594-6807'.
phoneNumbers
      at: 'donald'
      put: '594-7248'.

phoneNumbers                  "Returns 'Not Found' "
      keyAtValue: '594-3535'
      ifAbsent: ['Not found']
```

keyAtValue:ifAbsent: compares values using ==

Using Symbols

```
| phoneNumbers |
phoneNumbers := Dictionary new.
phoneNumbers
      at: 'whitney'
      put: '594-3535' asSymbol.
phoneNumbers
      at: 'beck'
      put: '594-6807' asSymbol.
phoneNumbers
      at: 'donald'
      put: '594-7248' asSymbol.


"This works"
phoneNumbers keyAtValue: '594-3535' asSymbol
```

Blocks and Returns

When a block evaluates a return (^) it exits the method the block was defined in

Example

```
Smalltalk.CS535 defineClass: #BlockExample
	superclass: #{Core.Object}
	etc

doExample
	Transcript
		show: 'Start doExample';
		cr.
	self start.
	Transcript
		show: 'End doExample';
		cr.!

evaluate: aBlock
	Transcript
		show: 'Start evaluate';
		cr.
	aBlock value.
	Transcript
		show: 'End evaluate';
		cr.!

start
	Transcript
		show: 'Start start';
		cr.
	self evaluate: [^nil].
	Transcript
		show: 'End start';
		cr.! !
```

Running the Example

Evaluate:

```
BlockExample new doExample
```

The output in the Transcript is:

```
Start doExample
Start start
Start evaluate
End doExample
```