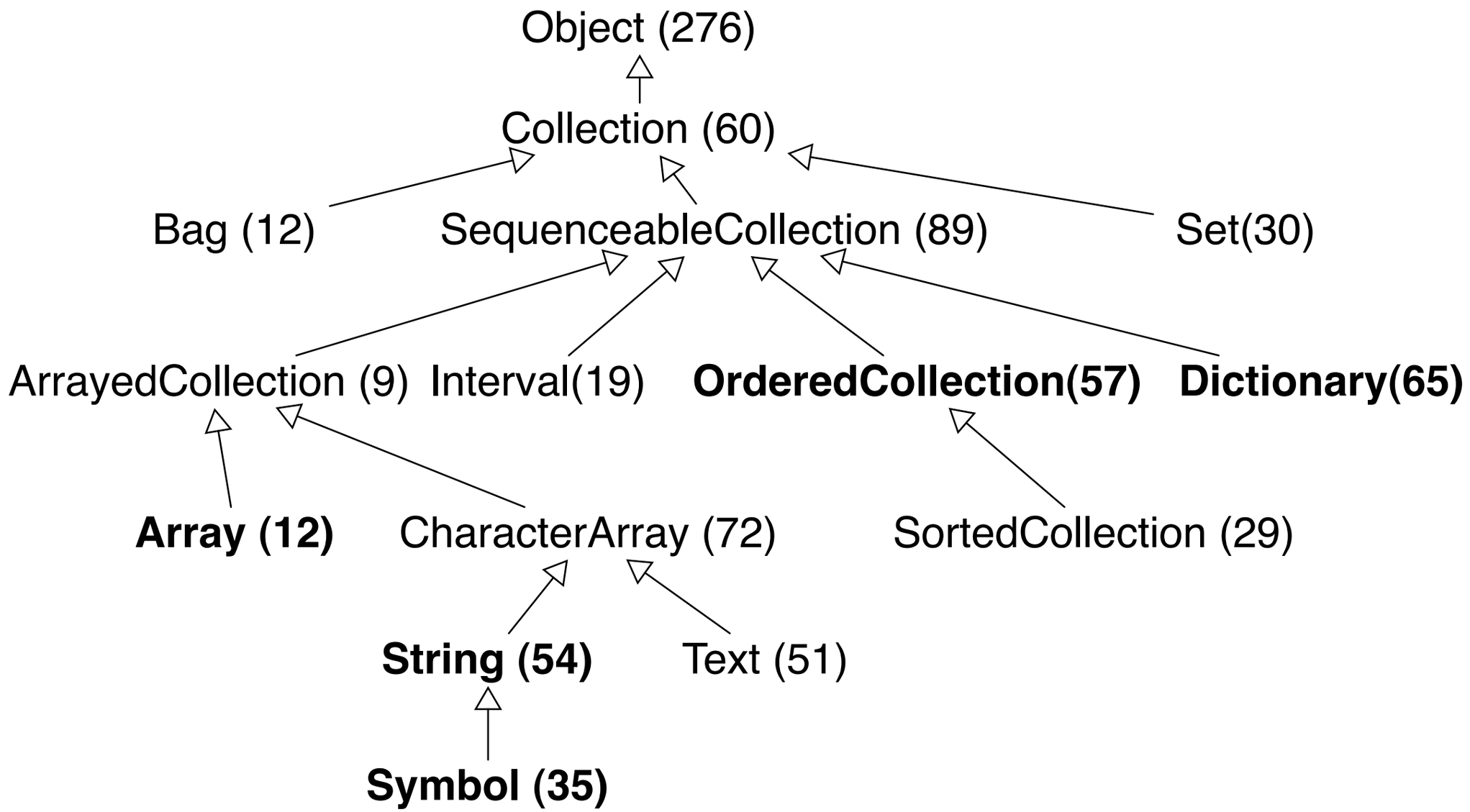
CS 535 Object-Oriented Programming & Design
Fall Semester, 2008
Doc 8 Collections
Sept 23 2008

Reading

Smalltalk by Example, Alex Sharp,
Chapter 11 Collections

Collections



(N) - number of methods defined the class

Bold - commonly used classes

Overview

Array

Fixed size

Elements indexed by integers

Bag

No order or indexing

Repeats allowed

Dictionary

Hash table

Elements indexed by any object

Interval

Finite arithmetic progression

OrderedCollection

Growable array

Set

No order, indexing or repeats

SortedCollection

Sorted growable array

String

Fixed size array of characters

Symbol

String with unique instances

Text

Text that supports fonts, bold etc.

Arrays

Similar to arrays in other languages

Once created can not grow

Creating an Array

```
a := #( 1 3 5 7 6 4 2 ).          “A literal array”
b := Array with: 5 with: 9.       “Create an array with two elements”
c := Array new: 10                “Create array with 10 elements, all nil”
```

Accessing Elements

```
secondElement := a at: 2.      "indexing starts at 1"
firstElement := a first.
lastElement := a last.

a                  "set first element to 12”
   at: 1
   put: 12.
```

Some Array Operations

```
numberOfElements := a size.
locationOfFiveInArray := a indexOf: 5.

jointList := a , b.      "Concatenatation"
sublist := a
   copyFrom: 2
   to: 4.

copyList := a copyUpTo: 6.
copyWithout := jointList copyWithout:  2.

location := a
   indexOfSubCollection: #( 6 4)
   startingAt: 2
   ifAbsent: [-1].
```

More Array Operations

```
a
   replaceAll: 3
   with: 12.

a occurrencesOf: 2.

(a includes: 2) ifTrue: [blah].
(a contains: [:each | each odd] ) ifTrue; [ blah].

(a anySatisfy: [:each | each odd]) ifTrue: [blah].
(a allSatisfy: [:each | each odd]) ifTrue: [blah].

a isEmpty ifTrue; [blah].
```

OrderedCollections

A growable array
When add elements, OrderedCollections grows if needed

Like Java's Vector or ArrayList

Creating an OrderedCollection

```
a := #( 1 3 5 7 6 4 2 ) asOrderedCollection.

b := OrderedCollection new.

c := OrderedCollection with: 5 with: 9.

d := OrderedCollection new: 10.
```

OrderedCollection Methods

```
b       “Add elements grow if needed”
   add: 2;
   add: 5.

secondElement := a at: 2.
firstElement := a first.

a
   at: 1
   put: 12.

jointList := a , b.

copyWithout := jointList copyWithout:  2.
```

```
fiveIndex := a indexOf: 5.

a
   replaceAll: 3
   with: 12.

finumberOfElements :=  a size.
```

```
a remove: 5
```

Size, Capacity & Growing

Size - number of elements in collection

Capacity - number of elements collection can hold without growing

```
| a |
a := OrderedCollection new.
a size.                 “Answers 0”
a capacity              “Answers 5”

6 timesRepeat: [a add: ‘cat’]

a size.                 “ Answers 6”

a capacity.             “ Answers 10”
```

Dictionary

A hash table, like Java's Hashtable or HashMap
In arrays and ordered collections indexes are integers
In dictionaries 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'].
```

hash & =

Both hash and = are used to add/find elements in a dictionary

Hash determine where to start looking

= is used to separate items with the same hash value

If you redefine = in a class make sure that

if a = b then a hash = b hash

Strings & Symbols

A String is an array of characters
Characters can be any Unicode character

Symbols are strings that are represented uniquely

```
#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"
```

Collection Creation Methods

new
new: anInteger
with: anElement
with: with:
with: with: with:
with: with: with: with:
withAll: aCollection

Array new: 5	#(nil nil nil nil nil)
OrderedCollection new: 5	OrderedCollection()
Array with: 2 with: 1	#(2 1)
Bag with: 1 with: 1 with: 2	Bag(1 1 2)
Set with: 1 with: 1 with: 2	Set(1 2)

Converting

asArray
asBag
asSet
asOrderedCollection
asSortedCollection
asSortedCollection: aBlock

cat' asSortedCollection	SortedCollection ($a "16r0061" $c "16r0063" $t "16r0074")
#(3 9 1 4) asSortedCollection	SortedCollection(1 3 4 9)
#(1 2 3 2 1) asBag	Bag(1 1 2 2 3)

Sorting

Expression	Result
#(3 9 1 4) asSortedCollection: [:x :y | x > y]	SortedCollection(9 4 3 1)
#(3 9 1 4) asSortedCollection: [:x :y | x < y]	SortedCollection(1 3 4 9)
#(3 9 1 4) asSortedCollection	SortedCollection(1 3 4 9)
#('dog' 'mat' 'bee') asSortedCollection	SortedCollection('bee' 'dog' 'mat')
#('dog' 'mat' 'bee') asSortedCollection: [:x :y | (x at: 2) < (y at: 2)]	SortedCollection ('mat' 'bee' 'dog')
#('dog' 'mat' 'bee') asSortedCollection: [:x :y | (x at: 2) > (y at: 2)]	SortedCollection ('dog' 'bee' 'mat')

Common Methods

size
capacity
at: indexOrKey
at: indexOrKey put: anElement

isEmpty
isNil
includes: anElement
occurencesOf: anElement

add: anElement
addAll: aCollection

remove: anElement
remove: anElement ifAbsent: aBlock
removeAll: aCollection

Enumerating

```
| sum  |
sum := 0.
#( 3 2 1) do: [:each | sum := sum + each squared].
^sum
```

Verses

```
| data sum  |
data := #( 3 2 1).
sum := 0.
1 to: data size do: [:each | sum := sum + (data at: each) squared].
^sum
```

Common Enumerations

`this is an example'` `  select: [:each |  each isVowel ]`	`'iiaeae'`
#(1 5 2 3 6) reject: [:each | each even]	#(1 5 3)
#(1 2 3 4 5) collect: [:each | each squared].	#(1 4 9 16 25)
`'hi mom' collect:` `  [:each |  each asUppercase ].`	HI MOM'
`#( 1 7 2 3 9 3 50)` `  detect: [:each | each > 8]`	9

Brief Summary

do: aBlock	Evaluate aBlock with receiver's elements
select: aBlock	Return elements that make aBlock true
reject: aBlock	Return elements that make aBlock false
collect: aBlock	Return result of evaluating aBlock on each element
detect: aBlock	Return first element that makes aBlock true
inject: initialValue into: binaryBlock	Accumulate a running value associated with evaluating the argument, binaryBlock, with the current value of the argument, thisValue, and the receiver as block arguments.

inject:into: Example

```
Transcript
   clear;
   show: 'Partial';
   tab;
   show: 'Number';
   cr.
#( 1 2 3 4 5) inject: 0 into:
   [:partialSum :number |
Transcript
   show: partialSum;
   tab;
   show: number;
   cr.
partialSum + number.]
```

Partial	Number
0	1
1	2
3	3
6	4
10	5

Result: 15

More Examples

Product

```
#( 1 2 3 4)
    inject: 1
    into: [:partialProduct :number | partialProduct * number]
```

Vowel Count

```
'hi mom' inject: 0 into:
    [:partial :each |
        each isVowel
            ifTrue:[partial + 1]
            ifFalse:[partial]]
```

keysAndValuesDo

```
'this is an example' keysAndValuesDo:
   [:key :value |
   value isVowel
      ifTrue:
         [Transcript
            show: key;
         tab;
         show: value;
         cr]]
```

Key	Value
3	i
6	i
9	a
12	e
14	a
18	e

Some Fun

```
Transcript
   show: 'Digit';
   tab;
   show: 'Frequency';
   cr.
100 factorial asString asBag sortedElements do:
   [:each |
   Transcript
      show: each key;
      tab;
      show: each value;
      cr]
```

Digit	Frequency
0	30
1	15
2	19
3	10
4	10
5	14
6	19
7	7
8	14
9	20

Useful Enumerations on Sequenceable Collections

with:do:

```
| pairwiseSum |
pairwiseSum := OrderedCollection new.
#(1 3 5 7 9)
      with: #(2 4 6 8 10)
      do: [:first :second | pairwiseSum add: first + second].
^pairwiseSum
```

OrderedCollection (3 7 11 15 19)

do:seperatedBy:

```
#(2 4 6 8)
   do: [:each | Transcript print: each ]
   separatedBy: [Transcript show: ', '].
Transcript
   cr;
   flush
```

Transcript
2, 4, 6, 8

fold:

#(1 2 3) fold: [:a : b | a + b]	`6`
#('A' 'cat' 'in' 'the' 'hat') fold: [:a :b | a , ' ' , b]	`'A cat in the hat'`

Evaluate a block with the 1st and the 2nd element of the receiver, then with the result of the first evaluation and the 3rd element, etc.

piecesCutWhere:do:

A sentence. Another sentence... Yet another sentence.' piecesCutWhere: [:each :next | each = $. and: [next = Character space]] do: [:each | Transcript show: each printString; cr]	A sentence.' ' Another sentence...' ' Yet another sentence.'
#(1 3 7 2 4 5 7 4 1 7 9) piecesCutWhere:[:each :next | each > next] do: [:each | Transcript show: each printString; cr]	#(1 3 7) #(2 4 5 7) #(4) #(1 7 9)

piecesCutWhere: block

- Indicates where to break receiver into pieces
- Does one character look ahead
- Character that cause break is the last element in the piece

runsFailing:do:

```
#( 1 3 7 2 4 5 7 4 1 7 9)
   runsFailing:[:each  | each = 7]
   do:
      [:each |
      Transcript
         show: each printString;
         cr]
```

```
#(1 3)
#(2 4 5)
#(4 1)
#(9)
```

runsFailing: block

Determines where receiver is divided into pieces

Character that cause break is not in any piece

do: block is done on the pieces