

Collections Get Supercharged

John Thickstun
Sirius Software Inc.



What is a collection? A reminder.

- It's a kind of object
- Like a \$list, but better!
 - It's OO'ified: you can use methods on it and everything else you know about objects.
- Contains a set of values, each of the same type
 - Values can be anything, including objects
 - Collections are objects too, so you can even have a collection of collections.
- They come in several flavors
 - Arraylists, NamedArraylists, and more



Maximum Power, Minimum Pain

- Two new collections methods:
 - Maximum and Minimum
- Find the 'biggest' or 'smallest' items in a collection
 - Convenient—one line of UL in many cases
 - Fast—No need for UL loops or temporary variable
- Customizable
 - Decide for yourself what 'big' and 'small' mean
 - (We'll get to this later)



Examples of Maximum/Minimum

```
begin
    %primes                is arrayList of float
    %primes = new
    %primes:add(7)
    %primes:add(31)
    %primes:add(127)
    %primes:add(3)
end
```

```
begin
    %names                is arrayList of longstring
    %names = new
    %names:add('Gary')
    %names:add('John')
    %names:add('Dave')
end
```

```
%primes:maximum = 3
%primes:minimum = 4
%names:maximum = 2
%names:minimum = 3
```



Maximum and Minimum Again

- Maximum and Minimum are customizable
 - Use method objects to decide what 'big' and 'small' mean
 - Get exactly what you want out of a collection
 - Even if your collection is made up of objects
- How can you do it?
 - Define a method for items in your collection:
 - (arrayList of myObject):myMethod is <numeric or string>
 - Maximum/Minimum of your objects are determined by the size of the returned value.



Custom Ordering for Ininsics

```
begin

local function (float):divisors is float
'''
end function

%numbers          is arrayList of float

%numbers = new
%numbers:add(5)
%numbers:add(22)
%numbers:add(12)
%numbers:add(77)
%numbers:add(4)

printtext {~} = {%numbers:maximum(divisors)}
printtext {~} = {%numbers:minimum(divisors)}

end
```

```
%numbers:maximum(divisors) = 12
%numbers:minimum(divisors) = 5
```



Ordering a class - Part I

```
class sale
  public
    variable state is longstring
    variable items is NamedArrayList of float

    constructor new
      function total is float
    end public

  constructor new
    %this:items = new
    %this:state = 'MO'
  end constructor

  function total is float
    %i is float
    %sum is float

    %sum = 0
    for %i from 1 to %this:items:count
      %sum = %sum + %this:items:itemByNumber(%i)
    end for

    if state = 'NY'
      %sum = %sum + .10 * %sum
    end if

    return %sum
  end function
end class
```



Ordering a class - Part II

```
%mySale      is object sale  
%saleList    is arrayList of object sale
```

```
%mySale = new  
%saleList = new
```

```
%mySale:items('shirt') = 30.00  
%mySale:items('pants') = 50.00  
%mySale:items('hat') = 20.00  
%saleList:add(%mySale)
```

```
%mySale = new  
%mySale:items('headphones') = 30.00  
%mySale:items('microphone') = 15.00  
%saleList:add(%mySale)
```

```
%mySale = new  
%mySale:items('shoes') = 60.00  
%saleList:add(%mySale)
```

```
printtext {~} = {%saleList:maximum(total)}  
printtext {~} = {%saleList:minimum(total)}
```

```
%orderList:maximum(total) = '1'  
%orderList:minimum(total) = '2'
```



The Sort and SortNew methods

- SortNew can sort Arraylist, NamedArraylist, FloatNamedArraylist, and UnicodeNamedArraylist collections and produces a new Arraylist collection
- Sort can only sort an Arraylist collection and sorts the Arraylist “in place”
- Sort and SortNew are very fast
 - Much of the logic is done in assembly
 - Even when a UL method indicates the order, only N method calls are needed
 - Most languages use “comparison” methods, requiring $N \cdot \log(N)$ method calls
- Sort and SortNew can be customized



Some data to sort

```
class City
  public
    variable name          is longstring
    variable state        is longstring
    variable country      is longstring
    variable population    is float

    function toString is longstring
    ...
  end public
  ...
  function toString is longstring
    return %this:name
  end function
end class

%cities is arrayList of object City
%cities = new

%cities:add(new(name='St. Louis',
                State='MO',
                Country='USA',
                population=354,361))
```



Simple Sorting

St. Louis	MO	USA	354,361
New York City	NY	USA	8,363,710
Los Angeles	CA	USA	3,833,995
London		England	7,556,900
Sydney	NSW	Australia	4,504,469

```
%cities:sortNew(ascending(size))
```

St. Louis	MO	USA	354,361
Los Angeles	CA	USA	3,833,995
Sydney	NSW	Australia	4,504,469
London		England	7,556,900
New York City	NY	USA	8,363,710

```
%cities:sortNew(descending(country))
```

St. Louis	MO	USA	354,361
Los Angeles	CA	USA	3,833,995
New York City	NY	USA	8,363,710
London		England	7,556,900
Sydney	NSW	Australia	4,504,469



SortOrders

- SortOrders define an order on collections
 - Same idea as minimum/maximum, except on steroids
- SortOrders are flexible
 - You can define any ordering you want: use any expression that assigns a string or numeric value to the items in your collection
 - In particular, you can use a local function to determine an ordering
- Three kinds of SortOrder
 - Ascending and Descending
 - List - these let you “nest” orders. A list SortOrder will first sort the items into categories per the first SortOrder, and then it will sort those categories per the second SortOrder, etc.



More Specific Sorts

```
%cities:sortNew(list(ascending(country), descending(size)))
```

Sydney	NSW	Australia	4,504,469
London	—	England	7,556,900
New York City	NY	USA	8,363,710
Los Angeles	CA	USA	3,833,995
St. Louis	MO	USA	354,361

```
local function (City):nameLength is float  
    return %this:name:length  
end function
```

```
%cities:sortNew(ascending(nameLength))
```

Sydney	NSW	Australia	4,504,469
London	—	England	7,556,900
St. Louis	MO	USA	354,361
Los Angeles	CA	USA	3,833,995
New York City	NY	USA	8,363,710



Finding an item

- Search through the collection looking for an item
- Four ways to find an item:
 - FindNextItem, FindNextItemNumber
 - FindPreviousItem, FindPreviousItemNumber
- Use a SelectionCriterion to describe an item



A simple example

```
b
%haystack is arrayList of unicode
%needle is float

%haystack = new
%haystack:add('hay')
%haystack:add('needle!')
%haystack:add('hay')
%haystack:add('hay')
%haystack:add('hay')
%haystack:add('needle!')

%needle = %haystack:findNextItemNumber(eq(this,'needle!'))
printtext {~} = {%needle}
%needle = %haystack:findNextItemNumber(eq(this,'needle!'),
                                       start=%needle)

printtext {~} = {%needle}
%needle = %haystack:findPreviousItemNumber(eq(this,'needle!'),
                                           start=%needle)

printtext {~} = {%needle}
end
```

%needle = 2

%needle = 6

%needle = 2



Sirius Software, Inc.

SelectionCriterion

- Use a SelectionCriterion to explain what kind of item you're looking for
- Eleven types of criteria:
 - Eq, Ne, Le, Lt, Ge, Gt
 - And, Or, Not
 - True, False
- Use Eq/Ne/etc. for particular comparisons
- The boolean operators allow you to chain together conditions
- True/False can be used as “starting points” for composing more complicated criteria—like null SortOrders



Find me more: SubsetNew

- Like Find, except that it finds every item that matches the SelectionCriterion
- The matching items are returned in a new collection.
 - Hence the 'New' in SubsetNew
- Also known as a 'filter', or a 'selection', you can use this to filter elements out of a list or select particular items from the list



Using SubsetNew

```
%cities:subsetNew(eq(country, 'USA')):print
```

```
printtext *****
```

```
%cities:subsetNew(ne(country, 'USA')):print
```

```
1: St. Louis  
2: New York  
3: Los Angeles  
*****  
1: London  
2: Sydney
```



A caveat

```
%sc          is object SelectionCriterion for longstring
```

```
%sc = or(eq(this,'alice'),  
         eq(this,'bob'),  
         eq(this,'charlie'),  
         ...  
         eq(this,'zack'))
```

- Try something like this instead:

```
%names is arrayList of longstring
```

```
%names = new  
%names:add('alice')  
%names:add('bob')  
...  
%names:add('zack')
```

```
local function (string):name_check is float expose  
    return %names:findNextItemNumber(eq(this,%this))  
end function
```

```
%sc = ne(name_check,0)
```



Conclusions

- Collections provide a large new set of tools for organizing and viewing data
- The common pattern of passing a method variable (or combination thereof) to an intrinsic method make these tools quite consistent
- Have your cake and eat it to: speed and flexibility
 - Intrinsic assembly methods make for fast searching and sorting of collections
 - Method variables make these methods highly tunable to particular problems

