

# Enhancement Methods and Beyond

**Alex Kodat**  
**Sirius Software Inc.**



# Object-Oriented Methods

- Perform processing on an object
- Usually invoked with `<object>:<method>` syntax
  - Is nice! Read left to right instead of in to out
- Except for shared methods which don't really apply to an object



# A Puzzle

- Suppose I want to apply write a method that applies to a class I can't add a method to
  - The class is a system class
  - The class belongs to another group
- The “classic” solution – write a shared method



# A Shared Method that Operates on a Stringlist

```
class myClass
  public shared
    function stringlistBytes(%sl is object stringlist) is float
  end public shared
  ...
  function stringlistBytes(%sl is object stringlist) is float
    %i is float
    %length is float
    for %i from 1 to %sl:count
      %length = %length + %sl:itemLength(%i)
    end for
    return %length
  end function
end class

...
%list is object stringlist

...
printText {~} = {(myClass):stringListBytes(%list)}
```



# The Problem with using Shared Methods to Add Methods to Off-limits Classes

- The invocations read unnaturally
  - Inside to outside, like \$functions, instead of left to right
  - Parentheses always required to bracket the object to which the method applies
    - Many object method invocations have no parameters
- The method names must be unique within a class, even if they apply to different classes
  - We can't have two Bytes methods inside a class, one that applies to Stringlists and, another to XmlDocs
    - Hence the funny name, StringlistBytes in our example



# Extending the Stringlist Class

```
class myStringlist extend stringlist inherit
  public
    function bytes is float
end public shared

...
function bytes is float
  %i is float
  %length is float
  for %i from 1 to %this:count
    %length = %length + %this:itemLength(%i)
  end for
  return %length
end function
end class

...
%list is object myStringlist

...
printText {~} = {%list:bytes}
```



# Now That Looks Nicer

- O-O gurus would be pleased
  - Go ahead, inhale deeply from the inheritance bong
- But...
- How does this work if %list was an input parameter?
  - Change all callers to use the MyStringlist class?
- What if YourStringlistClass also adds methods?
  - Multiple inheritance?
  - But now everyone needs to change declarations to use OurStringlist class to pick up a few methods?
- So this approach only really workable if a single site-wide class extends another class and everyone uses that class rather than the base class



# A New Solution to the Puzzle

## Enhancement Methods

- A new type of shared method
  - Shared because doesn't apply to instances of the containing class
  - Though does apply to instances of other classes
- Support added in Sirius Mods 7.2
  - Now available in an Online near you
- VB.Net also recently added support for these
  - Although they call them “extension methods” for maximal confusion
- The term “enhancement” does not appear in User Language
  - Used for documentation and message, only



# Our Stringlist Method as an Enhancement Method

```
class myClass
  public shared
    function (stringlist):bytes is float
  end public shared
  ...
  function (stringlist):bytes is float
    %i is float
    %length is float
    for %i from 1 to %this:count
      %length = %length + %this:itemLength(%i)
    end for
    return %length
  end function
end class

...
%list is object stringlist

...
printText {~} = {%list:(+myClass)bytes}
```



# So What Have We Bought?

- Method name can be shorter
  - Because class of object to which it applies distinguishes like-named methods
- Method invocation more natural
  - Reads left to right
  - Nice for stringing methods as in  
**`%list:(+myClass)bytes:right(10, pad='0')`**
  - No parentheses required to hold method object
- So method declarations and invocations shorter and easier to read



# Intrinsic Enhancement Methods

- Operate on basic 204 string and numeric datatypes
- But using natural, left to right O-O syntax
- Methods must be declared as operating on Float or String class
  - But conversion done automatically from string to float or vice versa at invocation
  - String and Float method names must be unique within a class
  - For consistency, continues yucky 204 tradition of converting non-numeric strings to zero
    - When in Rome...



# Example Intrinsic Enhancement Method

```
class myClass
  public shared
    function (string):everyOther is longstring
  end public shared
  ...
  function (string):everyOther is longstring
    %result is longstring
    %position is float
    for %position from 1 to %this:length by 2
      %result = %result with %this:substring(%position, 1)
    end for
    return %result
  end function
end class

...
%data is longstring

...
auditText {~} = {%data:(+myClass)everyOther}
auditText {~} = {'Sgehcrraeyty':(+myClass)everyOther}
```



# Enhancement Method Miscellany

- Can be applied to **any** class
  - Collections
  - File classes
  - Enumerations
  - System or User Language classes
  - Well, except for exception classes of containing class
- Containing class indicated by (+<*className*>) before the method name
  - + there to distinguish from other class name references in the same context
    - Narrowing assignment
    - Base class method reference



# Collection Enhancement

- Collections can be enhanced
- A class can extend a collection of objects of its own class
- In such a case, the enhancement method name does not need to be qualified with the name of the enhancing class
  - In other words, for any method on a collection of a class, the compiler first looks for enhancement methods in the class of the collection components



# Example Collection Enhancement Method

```
class customer
  public shared
    function (arraylist of object customer):important is -
      collection arraylist of object customer
    ...
  end public shared
  ...
  function (arraylist of object customer):important is -
    collection arraylist of object customer
    %result is collection arraylist of object customer
    %i is float
    %result = new
    for %i from 1 to %this:count
      ...
    end for
    return %result
  end function
end class

...
%custlist          is collection arraylist of object customer
%importantCustlist is collection arraylist of object customer
...
%importantCustlist = %custlist:important
```



# Collection Enhancement Methods

- Gives a class “control” over collections of that class
- Which makes sense since collections of a class are as much a part of that class as objects of the class
- Different methods required for different collections (ArrayLists, NamedArrayLists, FloatNamedArrayLists)
  - Which makes sense as behavior of different collections is different
  - But same name can be used for methods for different collections
    - And some code shared?

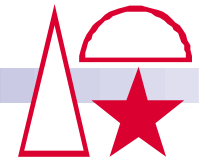


# Summary

- Enhancement methods make it possible to add methods to a class one doesn't “own”
  - Without the hassles and limitations of extending a class
- Intrinsic enhancement methods especially useful
- Coming soon... local enhancement methods
  - Enhancement methods only available within a context (method)
  - Since many methods only have localized use



# Let's Play Around with Enhancement Methods a Bit



Sirius Software, Inc.