CS 535 Object-Oriented Programming & Design
Fall Semester, 2008
Assignment 3 Comments
Oct 7 2008
Problem 1 - Average of Odd Numbers

Array>>avrOddNmbrs
    | count total sum |
sum := 0.
count := 0.
total := self
    inject: 0
    into:
        [ :sum2 :element |
            element odd
                ifTrue: [count := count + 1.
                    sum := sum2 + element].
            sum := sum2 + element].
        sum].
^((total/count) asFloat

Look at the names used. Do they follow Smalltalk naming conventions? Do the names help understand what is going on? What is the difference between sum, sum2 and total? Do the names tell you or do you have to read the code to find out?
Solution 2

Array>>averageOfOdds
| odds size |

odds := self select: [:each | each odd].
sum := odds fold: [:a :b | a +b].
^sum/(odds size)

This is a lot shorter due to separating finding the odd number, computing the sum and the number of odd numbers. Size is an existing function.
Solution 3

Collection>>sum
  self isEmpty ifTrue: [^0].
  ^self fold: [:a :b | a + b]

Collection>>average
  self isEmpty ifTrue: [^nil].
  ^self sum / self size

Array>>averageOfOdds
  | odds |
  odds := self select: [:each | each odd].
  ^odds average

Not clear that the policy should be for sum and average on an empty collection – return 0, nil or throw an exception. By breaking solution 2 into three methods we get some useful code. Not very likely that we will be able to use averageOfOdds in another project, but sum and average are common operations.
Useless Operations

| sum a c avg |
sum := 0.
a := Array new.
a := self select: [:each | each odd].
etc
Names

<table>
<thead>
<tr>
<th>sum</th>
<th>a</th>
<th>c</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>a collection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total numOdds average</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>words dollarWords</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TestString</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>originArr arrWords arrVal val</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
a := OrderedCollection new.
b := Dictionary new.
((self collect: [:each | each tokensBasedOn: Character space] do: [:each | b at: each put: (each inject: 0 into: [:partialSum :number |
(number asInteger > 64 and: [number asInteger < 91])
ifTrue: [partialSum + (number asInteger - $A asInteger + 1)]
ifFalse:
[
(number asInteger > 96 and: [number asInteger < 123])
ifTrue: [partialSum + (number asInteger -$a asInteger + 1)]
ifFalse: [partialSum + 0]
]
]))}
| a b |
a := OrderedCollection new.
b := Dictionary new.
((self collect: [:each | each]) tokensBasedOn: Character sapce)
do:
  [:each |
    b
    at: each
    put:
      (each
        inject: 0
        into:
          [:partialSum :number |
            (number asInteger > 64 and: [number asInteger < 91])
              ifTrue: [partialSum + (number asInteger - $A asInteger + 1)]]
            ifFalse:
              [(number asInteger > 96 and: [number asInteger < 123])
                ifTrue: [partialSum + (number asInteger - $a asInteger + 1)]]
              ifFalse: [partialSum + 0]]))))
Problem 3 - Words

Character>>isWordSeparator
  self isSeparator ifTrue: [^true].
  (',.;?!' includes: self) ifTrue: [^true].
  ^false

String>>words
  ^self runsFailing: [:each | each isWordSeparator]
Problem 4 - Dollar Words

Character>>dollarValue

    self isAlphabetic ifFalse: [^0].
    ^self asLowercase asInteger - $a asInteger + 1

String>>dollarValue

    ^self
    inject: 0
    into: [:sum :each | sum + each dollarValue]
Problem 4 - Dollar Words

String>>isDollarWord

^self dollarValue = 100

String>>dollarWords

^self words select: [:each | each isDollarWord]