CS 535 Object-Oriented Programming & Design
Fall Semester, 2013
Doc 13 Assignment 3 Comments
Oct 15 2013

Avg

```
	| avg |
(self Sum isKindOf: Integer) | (self Sum isKindOf: Float)
ifTrue:
[
avg := ((self Sum)/(self size)) asFloat.

^(avg)
]

ifFalse:
[
^(self Sum).
].
```

Grading Policy For Rest of Term - Formatting

Each method with such poor formatting
loses 1 point per method

Avg

```
| avg |
(self Sum isKindOf: Integer) | (self Sum isKindOf: Float)
ifTrue:
[
avg := ((self Sum)/(self size)) asFloat.

^(avg)
]

ifFalse:
[
^(self Sum).
].
```

control-o control-o

control-o control-o

control-o control-o

control-o control-

control-o control-

control-o contro

control-o contro

control-o contr

control-o con

control-o co

control-o co

control-o c

Grading Policy For Rest of Term - Names

Each name that does not following Smalltalk naming structure

Loses 1 point

Up to 10 points

Avg

```
| avg |
(self Sum isKindOf: Integer) | (self Sum isKindOf: Float)
ifTrue:
[
avg := ((self Sum)/(self size)) asFloat.

^(avg)
]

ifFalse:
[
^(self Sum).
```

Grading Policy For Rest of Term - Data Class

Data class (Struct class)

Data and only accessor method

Lose 3 or more points per class

Grading Policy Rest of Term - Helper Methods

Helper Methods

1 point per helper method

Other Ways to lose points

Using Global variables (Shared Variables)

Improper use of inheritance

- Node subclass of BST
- Matrix subclass of BST

Providing access to instance variables that should be private

- Example Accessor method for root in BST

Assigning values to variables then assigning again

```
valuesBetween: a and: b
| x |
      x := Array new: 20.
x := self select: [:each | (each > a) & (each < b)].
^(x)
```

Other Ways to lose points

Making local variable or arguments into instance variables

```
Smalltalk.Core defineClass: #Matrix
	superclass: #{Core.Array}
	indexedType: #objects
	private: false
	instanceVariableNames: 'noOfRows noOfCols cellValue aMatrix bMatrix matrixSum
'
	classInstanceVariableNames: ''
	imports: ''
	category: ''
```

Some Solution

```
Collection>>average

    self isEmpty ifTrue: [^0].
    ^self sum / self size
```

What happens when collection contains non-numbers?

```
Collection>>sum

    self isEmpty ifTrue: [^0].
    ^self fold: [:a :b | a + b]
```

```
testVariance

	self assert:( #( 17 15 23 7 9 13) variance - 33.2) < 0.000001
```

Note that all test method names start with "test"

First Method for BinarySearchTree

testAdd

```
| tree |
tree := BinarySearchTree new.
self assert: tree isEmpty.
#(1 2 3) do: [:each |
      self deny: (tree includes: each)].
tree add: 2.
self assert: tree size = 1.
self assert: (tree includes: 2).
#(1  3) do: [:each |
      self deny: (tree includes: each)].
tree add: 3.
self assert: tree size = 2.
#(2 3) do: [:each |
      self assert: (tree includes: each)].
```

Classes

BinarySearchTree

root

Node

key
left
right

NullNode

Object is parent class of all three

BinarySearchTree methods

```
clear

        root := nil.
```

```
isEmpty

        ^self size = 0
```

```
includes: aMagnitude

        root ifNil: [^false].
        ^root includes: aMagnitude
```

includes:

```
Node>>includes: aKey

	key = aKey ifTrue: [^true].
	aKey < key ifTrue: [^left includes: aKey].
	aKey > key ifTrue: [^right includes: aKey].
```

left and right are either
- Node
- NullNode

```
NullNode>>includes: aKey

	^false
```

Creating a Node

```
Node class>>key: aMagnitude

        ^super new setKey: aMagnitude
```

```
Node>>setKey: aMagnitude

    key := aMagnitude.
    left := NullNode instance.
    right := NullNode instance.
```

NullNode

```
Smalltalk.Core defineClass: #NullNode
	superclass: #{Core.Object}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: 'instance '
	imports: ''
	category: ''
```

```
NullNode>>instance

	instance ifNil: [instance := NullNode new].
	^instance
```

add:

```
BinarySearchTree>>add: aMagnitude

	self isEmpty
		ifTrue:
			[root := Node key: aMagnitude.
			^aMagnitude].
	^root add: aMagnitude
```

```
Node>>add: aMagnitude

	key = aMagnitude ifTrue: [^aMagnitude].
	aMagnitude < key ifTrue: [^self addLeft: aMagnitude].
	^self addRight: aMagnitude
```

Private methods in Node

```
Node>>addRight: aMagnitude

	right
		ifNil:
			[right := Node key: aMagnitude.
			^aMagnitude].
	^right add: aMagnitude
```

```
Node>>addLeft: aMagnitude

	left
		ifNil:
			[left := Node key: aMagnitude.
			^aMagnitude].
	^left add: aMagnitude
```

do:

```
BinarySearchTree>>do: aBlock

    self isEmpty ifTrue:[^self].
    root do: aBlock.
```

```
Node>>do: aBlock

    left do: aBlock.
    aBlock value: key.
    right do: aBlock.
```

```
NullNode>>do: aBlock
    "Null so do nothing"
```

do:

```
Node>>do: aBlock

	left do: aBlock.
	aBlock value: key.
	right do: aBlock.
```

```
Node>>do: aBlock

	left ifNotNil: [left do: aBlock].
	aBlock value: key.
	right ifNotNil: [ right do: aBlock].
```

Don't need NullNode here

Use ifNotNil:

size

```
BinarySearchTree>>size

    | size |
    root ifNil: [^0].
    size := 0.
    root do: [:each | size := size + 1].
    ^size.
```

Could have kept count in tree each time added element

printOn:

```
BinarySearchTree>>printOn: aStream

        aStream nextPutAll: 'BST('.
        root
                ifNotNil:
                        [root do:
                                [:each |
                                aStream
                                        nextPutAll: each printString;
                                        space]].
    aStream nextPutAll: ')'
```

BinarySearchTree & Collection

Follow the conventions of your language/libraries when possible

In Smalltalk all collection class inherit from Collection (or subclass)

Same is true in Java

So make BinarySearchTree subclass of Collection

Have to implement

add:

do:

remove:ifAbsent:

Get all other collection methods

Then can do

```
BinarySearchTree>>printOn: aStream

        aStream nextPutAll: 'BST('.
        root
                ifNotNil:
                        [root do: [:each | aStream nextPutAll: each printString]
                                separatedBy: [aStream space]].
        aStream nextPutAll: ')'
```

Search Trees, Keys and Values

Normally Node in Search tree contains

- left
- right
- key
- value

Tree ordered by keys

Some Issues

```
valuesBetween: a and: b

	| valueArray |

		valueArray := self select: [:each | each > a].
		valueArray := valueArray select: [:each | each < b].
		valueArray isEmpty ifTrue: [^' Nothing Found!!'].
		^valueArray
```

```
initializeRows: rows columns: columns
	"Initialize a Matrix object with the given number of rows and columns"

	super initialize.
	" *** Edit the following to properly initialize instance variables ***"
	numberOfRows := rows.
	numberOfColumns := columns.
	cells := Array new: rows * columns.
	" *** And replace this comment with additional initialization code *** "
	^self
```

```
squares

    | s |
    s := OrderedCollection new.
    1 to: self size
        do:
            [:i |
            ((self at: i) isKindOf: Integer) | ((self at: i) isKindOf: Float)
                ifFalse: [^'Non Numeric Values are not allowed']].
    s := self collect: [:i | i squared].
    ^s
```

MeanNumbers

```
| sum avg |
sum := 0.
self do:
        [:each |
        ((each isInteger) or: (each isKindOf: Float))
                ifTrue: [sum := each + sum]
                ifFalse: [^'pls enter valid input']].
avg := sum / self size asFloat.
^avg
```

where are you printing?

```
printSampleTree:aBlock node:traverseNode

        traverseNode = nil
                ifFalse:  [self printSampleTree:aBlock node: traverseNode left.
                           self printSampleTree:aBlock node: traverseNode right.
                    ^aBlock value:traverseNode data. ].
```

```
add: nodeValue
"|tree|"
"tree:= BInarySearchTree new."
root isNil
        ifTrue: [root:= nodeValue]
        ifFalse: [nodeValue < root
                        ifTrue: [root left: nodeValue]
                        ifFalse: [root right: nodeValue].
                        ].
```

```
"self:= Array new."
"1 to: (BinarySearchTree size) do: [:each |self at: each put: (Array new:)]."
        "ifFalse:[root to: (BinarySearchTree size) do: [:each | self       ]]."
^self
```

```
squares
	"I return a collection that contains the squares of the values in the receiver
collection"

	^self collect: [:collectionElement |
		(collectionElement isKindOf: Number) ifFalse: [
			^'Collection contains non-numeric values. Hence squares calculation is
not possible.'.
		].
		collectionElement * collectionElement.
	].
```

```
do: aBlock
        | left middle right |
        root isNil ifTrue: [
                ^''
        ]ifFalse: [
                left := self inOrderTreeTraversal: root left.
                middle := root value printString.
                aBlock value: 'I am in a block!'. "No"
                right := self inOrderTreeTraversal: root right.
                ^left, ' ', middle, ' ', right
        ].
```

```
average
	| sum total |
	total:=0.
	sum:=0.
	 total:= self size.
	total=0
	ifTrue: [ ^ 0 ]
	ifFalse: [ self do: [ :i | sum := sum +i ] ].
	 ^ ( sum / total ) asFloat
```

```
average
	| sum total |
	self isEmpty ifTrue: [^0].
	sum := 0.
	self do: [:i | sum := sum + i].
	^ (sum / self size) asFloat
```

Tree or Node?

```
Smalltalk.Core defineClass: #BinarySearchTree
        superclass: #{Core.Object}
        indexedType: #none
        private: false
        instanceVariableNames: 'count key value rightChild leftChild treeRoot '
        classInstanceVariableNames: ''
        imports: ''
        category: ''
```

Duh Comments

```
* aMatrixObject
|objectArraySize objectRows objectColumns sumMatrix temp|
"Returns the size of the object Matrix"
objectArraySize:= aMatrixObject getSize.
"Returns the number of rows in object Matrix"
objectRows:= aMatrixObject getTotalRows.
"Returns the number of columns in object Matrix"
objectColumns:= aMatrixObject getTotalColumns.
(totalColumns = objectRows)
ifTrue:[
        sumMatrix:= Matrix new.
        sumMatrix rows: totalRows columns: objectColumns.
        1 to: totalRows do:[:i|
        1 to: totalColumns do:[:m|
                temp:= (self row: i column: m) +     (aMatrixObject row:i column: m).
                sumMatrix row: i column:m put:temp.    ].
                ].
        ^sumMatrix.
        ]
```

```
do: aBlock startingFrom: aNode
root isNil
 ifFalse: [aBlock value: aNode. self do: aBlock startingFrom: aNode leftChild. self do: aBlock startingFrom: aNode rightChild]
```

```
average
|i sum|
i := self size.
i > 0
ifTrue:
[
sum := 0.
self do: [ :each | sum := sum + (each)].
^((sum/i) asFloat)
].
```

Sum or Average?

```
sumAverage
        |sum|
        sum := 0.0.
        self do: [: aNumber | sum := sum + aNumber ].
        ^( ((sum )/ (self size) ) )
```

```
withall: aCollection


 | b  temp |
b:=BinarySearchTree new.
1 to: aCollection size do:[:i |
temp:= (aCollection at: i).
b add:temp.
].
^b.
```