```
Object subclass: #BinTree
	instanceVariableNames: 'treeObj leftnode rightnode'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'Collections-BinTree'!
!BinTree commentStamp: 'RAB 2/21/2013 11:41' prior: 0!
Tree nodes just store an object and a left/right link to other tree
nodes!


!BinTree methodsFor: 'adding' stamp: 'RAB 2/23/2013 12:07'!
addLeftKid: node
	"Set the left child for this node, must be a BinTree object"

	node class == self class ifFalse: [ ^false ].
	leftnode_node! !

!BinTree methodsFor: 'adding' stamp: 'RAB 2/23/2013 12:07'!
addRightKid: node
	"Set the right child for this node, must be a BinTree object"

	node class == self class ifFalse: [ ^false ].
	rightnode_node! !


!BinTree methodsFor: 'accessing' stamp: 'RAB 2/23/2013 12:09'!
left
	"return node.left"

	^ leftnode! !

!BinTree methodsFor: 'accessing' stamp: 'RAB 2/23/2013 12:09'!
obj
	"return node.object"

	^ treeObj! !

!BinTree methodsFor: 'accessing' stamp: 'RAB 2/23/2013 12:09'!
right
	"return node.right"

	^ rightnode! !


!BinTree methodsFor: 'Initialize' stamp: 'RAB 2/23/2013 12:08'!
initNew: objectItem
	"This is called by the class new:, it sets the node's object
and initialized left and right to nil"

	treeObj_objectItem.
```

```
        leftnode_nil.
        rightnode_nil.
        ^self
        ! !


!BinTree methodsFor: 'print' stamp: 'RAB 2/23/2013 12:11'!
print
        "comment stating purpose of message"


        ^treeObj printString! !

!BinTree methodsFor: 'print' stamp: 'RAB 3/4/2013 18:23'!
visit
        "routine called from recursive traversals"


        Transcript show: (self obj); show: '  -  '. ! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

BinTree class
        instanceVariableNames: ''!

!BinTree class methodsFor: 'class initialization' stamp: 'RAB
2/23/2013 12:13'!
new: objectItem
        "this creates an instance of the BinTree object (node) and
initializes it with the object passed in"

        ^(self new)
        initNew: objectItem
! !



Object subclass: #InorderTraversal
        instanceVariableNames: 'st root'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Collections-BinTree'!

!InorderTraversal methodsFor: 'initialize-release' stamp: 'RAB
2/23/2013 13:05'!
init: theTree
        "called by new: to set the root"


        root := theTree.
```

```
        st := Stack new.
        ^self! !


!InorderTraversal methodsFor: 'Iterators' stamp: 'RAB 3/4/2013 21:20'!
first
        "Reset the traversal and return the first node"

        "Clear Stack"
        self clearStack.

        "Initialize stack from root"
        self doStack: root.

        "return the next node, in this case it is node number 1"
        ^self next

! !

!InorderTraversal methodsFor: 'Iterators' stamp: 'RAB 2/23/2013
14:10'!
next
        "return the next node in the traversal"

        | node |
        st isEmpty ifTrue: [node := nil.] ifFalse: [node := st pop.
self doStack: (node right).].
        ^ node

        ! !


!InorderTraversal methodsFor: 'tracing' stamp: 'RAB 2/27/2013 17:21'!
getStack
        "return the stack used internally for tracing purposes"

        ^st! !


!InorderTraversal methodsFor: 'private' stamp: 'RAB 2/23/2013 13:23'!
clearStack
        "comment stating purpose of message"

        st isEmpty ifFalse: [st pop. self clearStack]! !

!InorderTraversal methodsFor: 'private' stamp: 'RAB 2/23/2013 13:28'!
doStack: node
        "comment stating purpose of message"

        node == nil ifFalse: [st push: node. self doStack: (node
```

```
left)]! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

InorderTraversal class
        instanceVariableNames: ''!

!InorderTraversal class methodsFor: 'class initialization' stamp: 'RAB
3/4/2013 21:19'!
new: theTree
        "instantiate a new InorderTraversal speficify the root node"

        ^(self new)
        init: theTree
        ! !


!InorderTraversal class methodsFor: 'traverse' stamp: 'RAB 3/4/2013
17:02'!
recurse: node
        "Show the entire tree using InorderTraversal"


        node == nil ifFalse: [
                self recurse: (node left).
                node visit.
                self recurse: (node right).].

        ! !



Object subclass: #PostorderTraversal
        instanceVariableNames: 'root st'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Collections-BinTree'!

!PostorderTraversal methodsFor: 'private' stamp: 'RAB 2/23/2013
15:32'!
clearStack
        "comment stating purpose of message"

        st isEmpty ifFalse: [st pop. self clearStack]! !

!PostorderTraversal methodsFor: 'private' stamp: 'RAB 2/23/2013
16:45'!
doStack: node
        "comment stating purpose of message"
```

```
        node == nil ifFalse: [st push: node. self doStack: (node
left). (node left) == nil ifTrue: [self doStack: (node right)].]! !


!PostorderTraversal methodsFor: 'iterators' stamp: 'RAB 2/23/2013
16:10'!
first
        "Initialize and return first node of traversal"

        "Clear Stack"
        self clearStack.

        "Initialize stack from root"
        self doStack: root.

        "return the next node, in this case it is node number 1"
        ^self next

! !

!PostorderTraversal methodsFor: 'iterators' stamp: 'RAB 2/23/2013
17:01'!
next
        "return the next node in the traversal"

        | node |
        st isEmpty ifTrue: [node := nil.] ifFalse: [node := st pop. st
isEmpty ifFalse: [node == ((st top) right) ifFalse: [self doStack:
((st top) right)].].].
        ^node

        ! !


!PostorderTraversal methodsFor: 'initialize-release' stamp: 'RAB
2/23/2013 15:27'!
init: theTree
        "called by new: to set the root"

        root := theTree.
        st := Stack new.
        ^self! !


!PostorderTraversal methodsFor: 'tracing' stamp: 'RAB 2/27/2013
17:21'!
getStack
        "return the stack used internally for tracing purposes"

        ^st! !
```

```
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

PostorderTraversal class
	instanceVariableNames: ''!

!PostorderTraversal class methodsFor: 'class initialization' stamp:
'RAB 3/4/2013 21:20'!
new: theTree
	"instantiate a new PostorderTraversal speficify the root node"

	^(self new)
	init: theTree
	! !


!PostorderTraversal class methodsFor: 'traverse' stamp: 'RAB 3/4/2013
17:02'!
recurse: node
	"Show the entire tree using PostorderTraversal"


	node == nil ifFalse: [
		self recurse: (node left).
		self recurse: (node right).
		node visit
		].

! !



Object subclass: #PreorderTraversal
	instanceVariableNames: 'root st'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'Collections-BinTree'!

!PreorderTraversal methodsFor: 'private' stamp: 'RAB 2/23/2013 14:26'!
clearStack
	"comment stating purpose of message"

	st isEmpty ifFalse: [st pop. self clearStack]! !


!PreorderTraversal methodsFor: 'iterators' stamp: 'RAB 2/23/2013
15:10'!
first
	"Initialize and return first node of traversal"
```

```
        "Clear Stack"
        self clearStack.

        "Initialize stack from root"
        st push: root.

        "return the next node, in this case it is node number 1, the
root"
        ^self next

! !

!PreorderTraversal methodsFor: 'iterators' stamp: 'RAB 2/23/2013
15:17'!
next
        "return the next node in the traversal"

        | node |
        st isEmpty ifTrue: [node := nil.] ifFalse: [node := st pop.
                (node right) == nil ifFalse: [st push: (node right)].
                (node left) == nil ifFalse: [st push: (node left)].].

        ^ node

        ! !


!PreorderTraversal methodsFor: 'tracing' stamp: 'RAB 2/27/2013 17:21'!
getStack
        "return the stack used internally for tracing purposes"

        ^st! !


!PreorderTraversal methodsFor: 'initialize-release' stamp: 'RAB
2/23/2013 14:58'!
init: theTree
        "called by new: to set the root"

        root := theTree.
        st := Stack new.
        ^self! !

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

PreorderTraversal class
        instanceVariableNames: ''!

!PreorderTraversal class methodsFor: 'traverse' stamp: 'RAB 3/4/2013
17:03'!
```

```
recurse: node
        "Show the entire tree using PreorderTraversal"


        node == nil ifFalse: [
                node visit.
                self recurse: (node left).
                self recurse: (node right).
                ].

! !


!PreorderTraversal class methodsFor: 'class initialization' stamp:
'RAB 3/4/2013 21:20'!
new: theTree
        "instantiate a new PreorderTraversal speficify the root node"

        ^(self new)
        init: theTree
        ! !



Object subclass: #TraversalTest
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Collections-BinTree'!

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

TraversalTest class
        instanceVariableNames: ''!

!TraversalTest class methodsFor: 'test method' stamp: 'RAB 3/4/2013
18:57'!
largeTest
        "Function for testing the BinTree and traversal classes.
        A severnteen node binary tree is created and traversed."

| nodeA nodeB nodeC nodeD nodeE nodeF nodeG nodeH nodeI nodeJ nodeK
nodeL nodeM nodeN nodeO nodeP nodeQ traverse nextNode travStack
tempStack tempEntry |

Transcript clear.
Transcript show: 'Test case: a binary tree with seventeen nodes.'; cr.
Transcript show: 'See documentation for tree layout.'; cr; cr.
nodeA := BinTree new: 'A'.
nodeB := BinTree new: 'B'.
```

```
nodeC := BinTree new: 'C'.
nodeD := BinTree new: 'D'.
nodeE := BinTree new: 'E'.
nodeF := BinTree new: 'F'.
nodeG := BinTree new: 'G'.
nodeH := BinTree new: 'H'.
nodeI := BinTree new: 'I'.
nodeJ := BinTree new: 'J'.
nodeK := BinTree new: 'K'.
nodeL := BinTree new: 'L'.
nodeM := BinTree new: 'M'.
nodeN := BinTree new: 'N'.
nodeO := BinTree new: 'O'.
nodeP := BinTree new: 'P'.
nodeQ := BinTree new: 'Q'.

nodeA addLeftKid: nodeB.
nodeA addRightKid: nodeC.
nodeB addLeftKid: nodeD.
nodeB addRightKid: nodeE.
nodeC addLeftKid: nodeF.
nodeD addLeftKid: nodeG.
nodeD addRightKid: nodeH.
nodeE addRightKid: nodeI.
nodeI addLeftKid: nodeM.
nodeG addLeftKid: nodeJ.
nodeG addRightKid: nodeK.
nodeH addRightKid: nodeL.
nodeL addRightKid: nodeP.
nodeK addLeftKid: nodeN.
nodeK addRightKid: nodeO.
nodeN addLeftKid: nodeQ.

tempStack := Stack new.

Transcript show: 'In-order:'; cr.
traverse := InorderTraversal new: nodeA.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
                        ].
                [tempStack isEmpty]
                        whileFalse: [
```

```
                                tempEntry := tempStack pop.
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Pre-order.'; cr.
traverse := PreorderTraversal new: nodeA.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
                        ].
                [tempStack isEmpty]
                        whileFalse: [
                                tempEntry := tempStack pop.
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Post-order.'; cr.
traverse := PostorderTraversal new: nodeA.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
                        ].
                [tempStack isEmpty]
                        whileFalse: [
                                tempEntry := tempStack pop.
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
```

```
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Recursive InOrder Traversal.'; cr.
InorderTraversal recurse: nodeA.
Transcript cr.

Transcript cr; cr; show: 'Recursive PreOrder Traversal.'; cr.
PreorderTraversal recurse: nodeA.
Transcript cr.

Transcript cr; cr; show: 'Recursive PostOrder Traversal.'; cr.
PostorderTraversal recurse: nodeA.
Transcript cr; cr.
! !

!TraversalTest class methodsFor: 'test method' stamp: 'RAB 3/4/2013
18:52'!
smallTest
        "Function for testing the BinTree and traversal classes.
        A three node binary tree is created and traversed."

| node1 node2 node3 traverse nextNode travStack tempStack tempEntry |

Transcript clear.
Transcript show: 'Test case: a binary tree with three nodes.'; cr.
Transcript show: 'A is the root node, C is left of A, B is right of
C.'; cr; cr.

node1 := BinTree new: 'A'.
node2 := BinTree new: 'C'.
node3 := BinTree new: 'B'.
node1 addLeftKid: node2.
node2 addRightKid: node3.

tempStack := Stack new.

Transcript show: 'In-order:'; cr.
traverse := InorderTraversal new: node1.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
```

```
                        ].
                [tempStack isEmpty]
                        whileFalse: [
                                tempEntry := tempStack pop.
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Pre-order.'; cr.
traverse := PreorderTraversal new: node1.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
                        ].
                [tempStack isEmpty]
                        whileFalse: [
                                tempEntry := tempStack pop.
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Post-order.'; cr.
traverse := PostorderTraversal new: node1.
nextNode := traverse first.
[nextNode notNil]
        whileTrue: [
                Transcript show: 'Current node: '.
                Transcript show: nextNode obj; cr.
                Transcript show: 'Current stack: '.
                travStack := traverse getStack.
                [travStack isEmpty]
                        whileFalse: [
                                tempStack push: travStack pop.
                        ].
                [tempStack isEmpty]
                        whileFalse: [
                                tempEntry := tempStack pop.
```

```
                                Transcript show: tempEntry print.
                                travStack push: tempEntry.
                        ].
                Transcript cr.
                nextNode := traverse next
        ].

Transcript cr; cr; show: 'Recursive InOrder Traversal.'; cr.
InorderTraversal recurse: node1.

Transcript cr; cr; show: 'Recursive PreOrder Traversal.'; cr.
PreorderTraversal recurse: node1.

Transcript cr; cr; show: 'Recursive PostOrder Traversal.'; cr.
PostorderTraversal recurse: node1.
! !
```