initialize
"Initialize a newly created instance. This method must answer the receiver."

super initialize.
" *** Edit the following to properly initialize instance variables ***"
data := nil.
next := nil.
prev := nil.
" *** And replace this comment with additional initialization code *** "
^self
increase
   " an instance method that increases the count of the counter"

   count := count + 1.
   SharedVar := SharedVar + 1.
   ^self

   count
   ^countVar
increase

" *** increase self counter *** "
counter := counter +1.
" *** Increase class master counter *** "
self class increaseMasterCount
addFirst: anObject
  "adds the argument to the front of the list"

| obj |
obj := Node new.
head isNil
  ifTrue:
    [head := obj.
     head data: anObject]
  ifFalse:
    [head prev: obj.
     obj next: head.
     obj prev: nil.
     head := obj.
     head data: anObject]
incMain

MainCount := MainCount + 1

count

^myCount
getNodeValueAt: anInteger
|evaluatingNode|
evaluatingNode := _firstNode.
anInteger - 1 timesRepeat: [
    evaluatingNode := evaluatingNode nextNode.
].
^evaluatingNode value.
at: anInteger

| temp check|

anInteger < 0 | (anInteger > size)
   ifTrue: [self error: 'Integer out of bounds].
size < 1
   ifTrue: [self error: 'Empty list].
temp := Node new.
temp := head.

check := 0.
[check ~= anInteger]
   whileTrue:
      [temp := temp next.
         check := check + 1].
^temp value
newNode: anObject
    |temp |
    temp := super new.
temp data: anObject.
^temp
increase
  count:= count + 1.
  MasterCount:= MasterCount+1.
  Transcript print: count.
  Transcript cr.

masterCount
  Transcript show: 'Master'.
  Transcript print: MasterCount.
  Transcript cr.
See the Difference?

add: anObject
  "comment stating purpose of message"

  | newNode |

  newNode := Node new.
nodenode link: anObject.
firstNode isNil
ifTrue:[self lastNode: newNode]
ifFalse:[newNode nextNode: firstNode.
  self firstNode previousNode: newNode].
self firstNode: newNode.
^self firstNode link

addFirst: anObject
  "comment stating purpose of message"

  | newNode |

  newNode := Node new.
nodenode link: anObject.
firstNode isNil
ifTrue:[self lastNode: newNode]
ifFalse:[newNode nextNode: firstNode.
  self firstNode previousNode: newNode].
self firstNode: newNode.
^self firstNode link
at: anInteger

| currentNode |
anInteger < 1 | anInteger > self size
  ifTrue: [^self error: 'The index you entered is out of bounds.'].
currentNode := self head.
anInteger - 1 timesRepeat: [currentNode := currentNode next].
^currentNode value value
DoubleLinkedList>>head
  "info hiding -2"
^head
# Node Class

<table>
<thead>
<tr>
<th>Instance Variables</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>previous</td>
<td>previous</td>
</tr>
<tr>
<td>data</td>
<td>data</td>
</tr>
</tbody>
</table>
addFirst: anObject

|aNode|
aNode:=Node value:anObject.
head = nil
    ifTrue: [tail := aNode]
    ifFalse: [head previous:aNode].
aNode next: head.
head := aNode.
size := size + 1.
^aNode
addFirst: anObject

|aNode|
aNode := Node value: anObject.
aNode append: head.
head := aNode.

self isEmpty
   ifTrue: [tail := aNode].
size := size + 1.
^aNode