CS 535 Object-Oriented Programming & Design
Fall Semester, 2011
Doc 14 Assignment 3 Comments 2
Oct 13 2011

Typical Node Class

```
Smalltalk.Core defineClass: #Node
        superclass: #{Core.Object}
        instanceVariableNames: 'data next '
```

```
Node>>data

        ^data
```

```
Node>>next

        ^next
```

```
Node>>data: anObject

        data := anObject
```

```
Node>>next: anObject

        next := anObject
```

Typical Stack operations

```
Stack>>do: aBlock
    | current |
    current := topOfStack.
    [current isNil] whileFalse: [
        aBlock value: current data.
        current := current next.]
```

```
Stack>>push: anObject
    | newTop |
    newTop := Node new.
    newTop data: anObject.
    newTop next: topOfStack.
    topOfStack := newTop.
    size := size++.
```

Note how stack extracts/sets Node data

Node is 1/2 class

Just data

No operations

Stack has to do all the work

Heuristics

Heuristic 2.8
Keep related data and behavior in one place

Heuristic 3.3
Beware of classes that have many accesor methods in their public interface.
Having many implies that related data and behavior are not being kept in one place.

First Node operation

Constructor method that accepts data and next

Creates Node object that is usable

Why should users always repeat these lines

```
newElement := Node new.
newElement data: anObject.
newElement next: topOfStack
```

Node methods

```
Node class>>data: anObject next: aNodeOrNil

        ^super new setData: anObject next: aNodeOrNil
```

```
Node>>setData: anObject next: aNodeOrNil

        data := anObject.
        next := aNodeOrNil
```

Now Stack>>push:

```
Stack>>push: anObject

    topOfStack := Stack data: anObject next: topOfStack.
    size := size +1.
```

Another Operation do:

```
Node>>do: aBlock
    aBlock value: data.
    next ifNotNil:  [next do: aBlock].
```

```
Stack>>do: aBlock
    self isEmpty ifTrue: [^nil].
    topOfStack do: aBlock
```

Stack methods

```
Node>>push: anObject
    topOfStack := Stack data: anObject next: topOfStack.
    size := size +1.
```

```
Node>>pop
    | topData |
    self isEmpty ifTrue: [self error: 'stack empty, no elements to pop'].
    size := size -1.
    topData := topOfStack data.
    topOfStack := topOfStack next.
    ^topData
```

```
Node>>clear
    topOfStack := nil.
    size = 0.
```

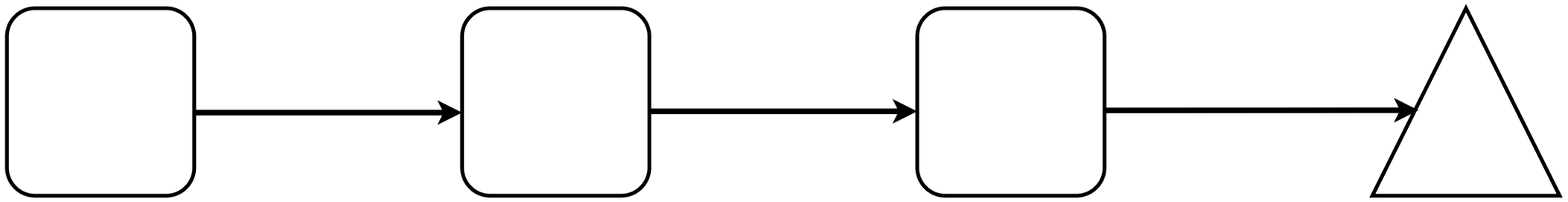
Stack methods

```
Node>>size
    ^size
```

```
Node>>do: aBlock
    self isEmpty ifTrue: [^nil].
    topOfStack do: aBlock
```

```
Node>>printOn: aStream
    aStream nextPut: $(.
    self isEmpty ifTrue: [topOfStack printOn: aStream].
    aStream nextPut: $).
```

Replacing if Statements



Use special node to represent end of list

TailNode

Empty node at end of list

```
TailNode>>isEndOfList
    ^true
```

```
Node>>isEndOList
    ^false
```

```
TailNode>>do: aBlock
```

```
Node>>do: aBlock
    aBlock value: data.
    next do: aBlock
```

New Stack methods

```
Stack>>initialize
    self clear.
```

```
Stack>>do: aBlock
    topOfStack do: aBlock
```

```
Stack>>isEmpty
    ^topOfStack isEndOfList
```

```
Stack>>clear
    topOfStack := nil.
    size = 0.
```