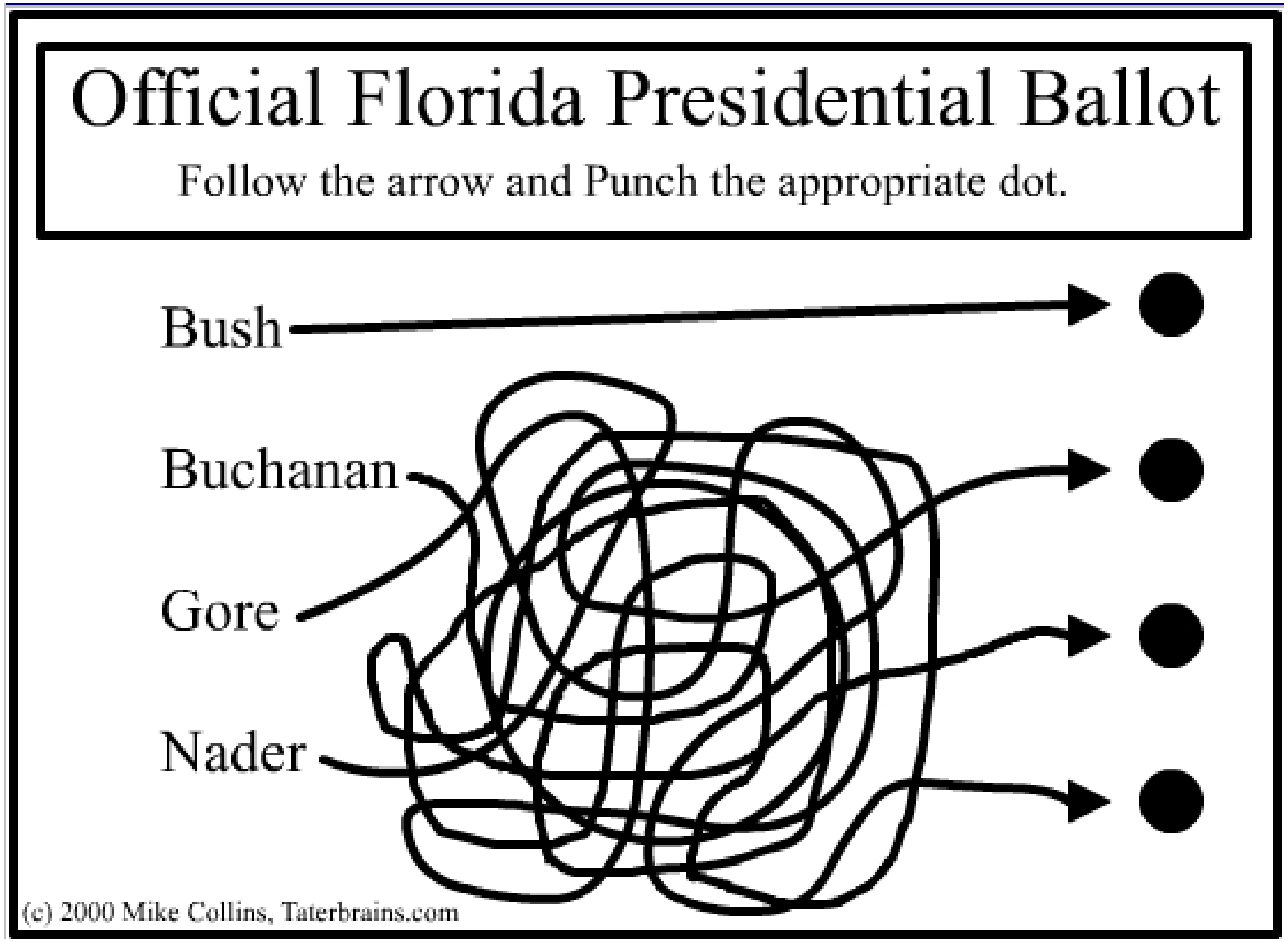
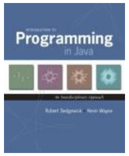
Linked Lists


Official Florida Presidential Ballot
Follow the arrow and Punch the appropriate dot.
Bush
Buchanan
Gore
Nader

(c) 2000 Mike Collins, Taterbrains.com



Section 4.3

Sequential vs. Linked Allocation

Sequential allocation: Put items one after another.

- Java: array of objects.

Linked allocation: Include in each object a link to the next one.

- Java: link is reference to next item.

Key distinctions:

get ith item

- Array: random access, fixed size.
- Linked list: sequential access, variable size.

get next item

addr	value
B0	"Alice"
B1	"Bob"
B2	"Carol"
B3	-
B4	-
B5	-
B6	-
B7	-
B8	-
B9	-
BA	-
BB	-

array
(B0)

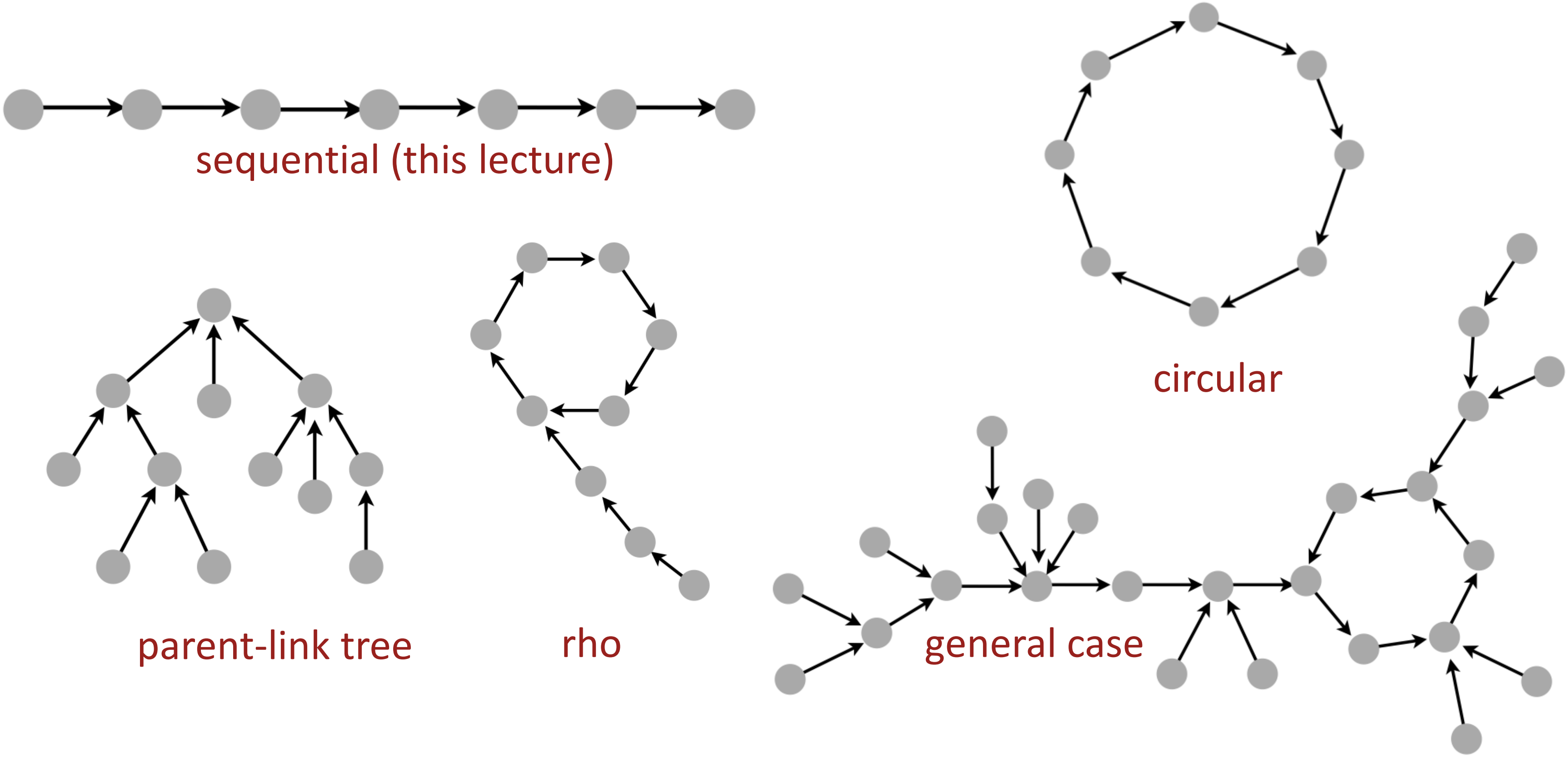
addr	value
C0	"Carol"
C1	null
C2	-
C3	-
C4	"Alice"
C5	CA
C6	-
C7	-
C8	-
C9	-
CA	"Bob"
CB	C0

linked list
(C4)

Singly-Linked Data Structures

From the point of view of a particular object:

all of these structures look the same! ●→



sequential (this lecture)

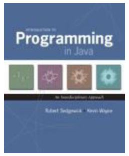
circular

parent-link tree

rho

general case

Multiply-linked data structures: Many more possibilities.



Section 4.3

Linked Lists

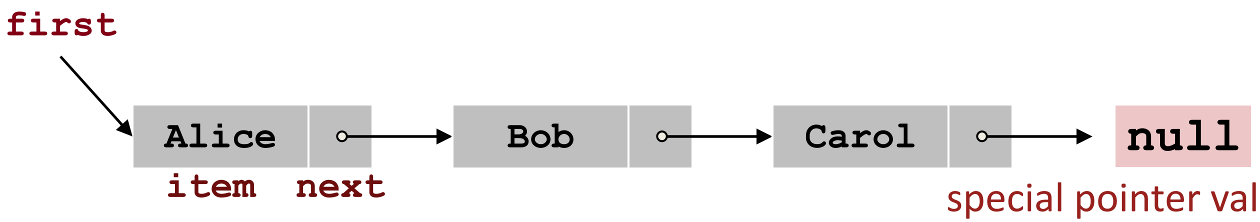
Linked list:

- A recursive data structure.
- An item plus a pointer to another linked list (or empty list).
  - Unwind recursion: linked list is a sequence of items.

Node data type:

- A reference to a **String**.
- A reference to another **Node**.

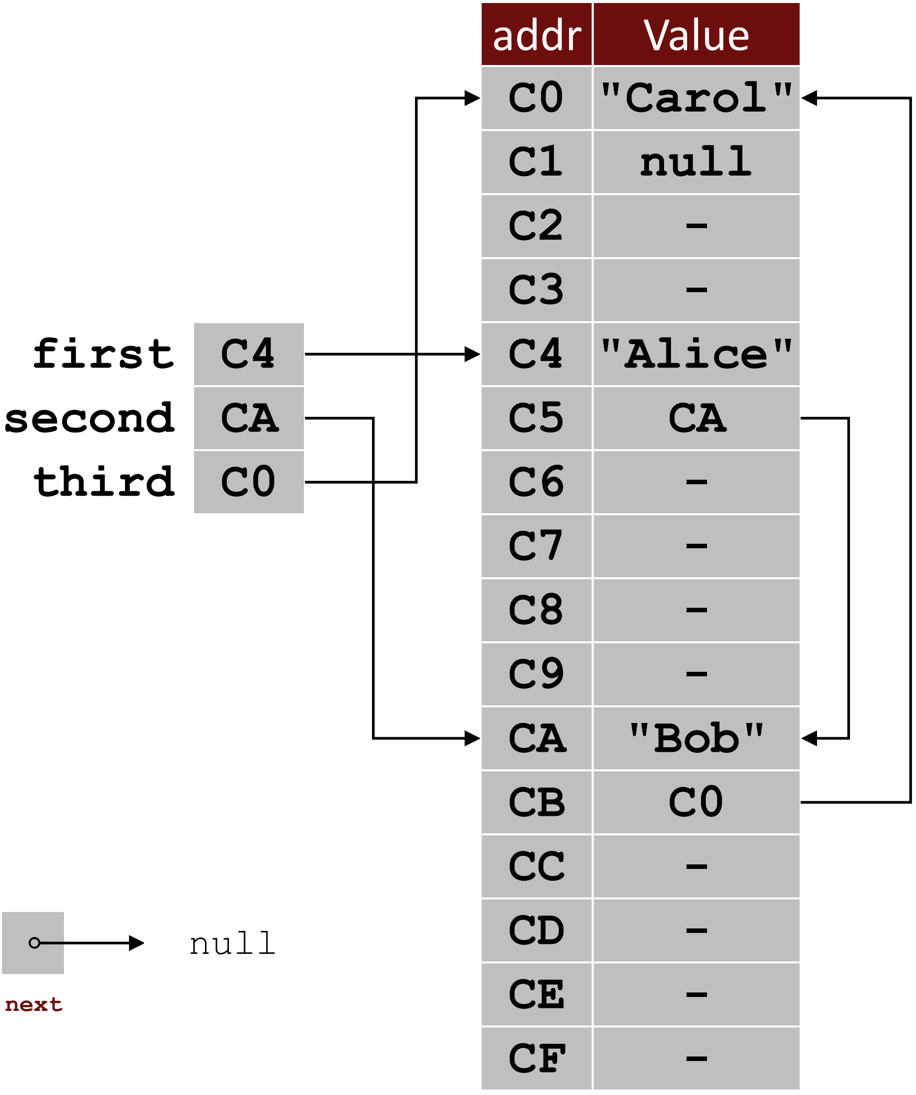
```
public class Node {
   public String item;
   public Node next;
}
```



special pointer value **null** terminates list

Building a Linked List

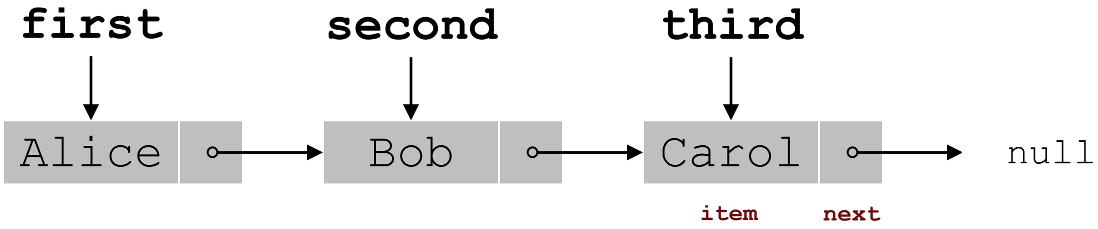
```
Node third   = new Node();
third.item   = "Carol";
third.next   = null;

Node second = new Node();
second.item = "Bob";
second.next = third;

Node first   = new Node();
first.item   = "Alice";
first.next   = second;
```



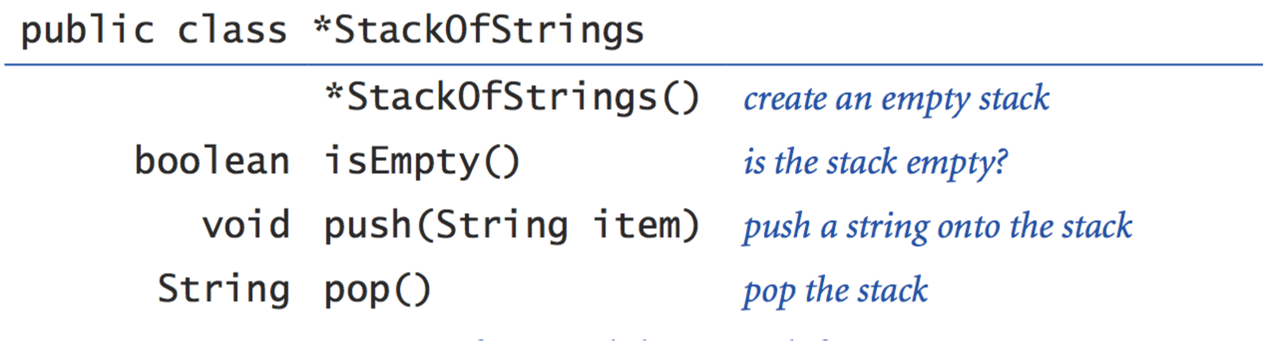
first	C4
second	CA
third	C0

addr	Value
C0	"Carol"
C1	null
C2	-
C3	-
C4	"Alice"
C5	CA
C6	-
C7	-
C8	-
C9	-
CA	"Bob"
CB	C0
CC	-
CD	-
CE	-
CF	-

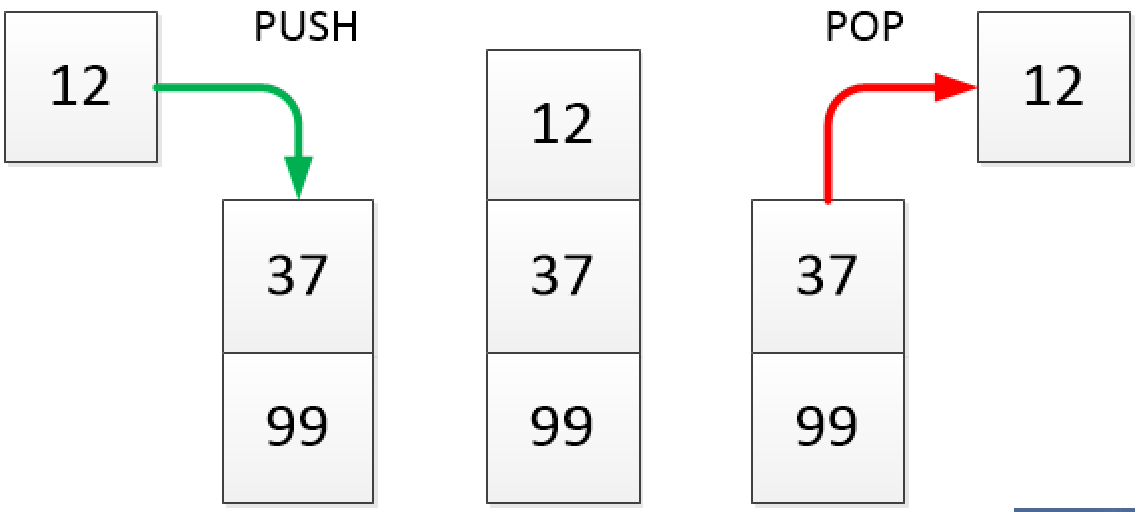
main memory

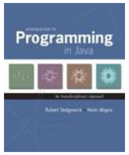


Stack API



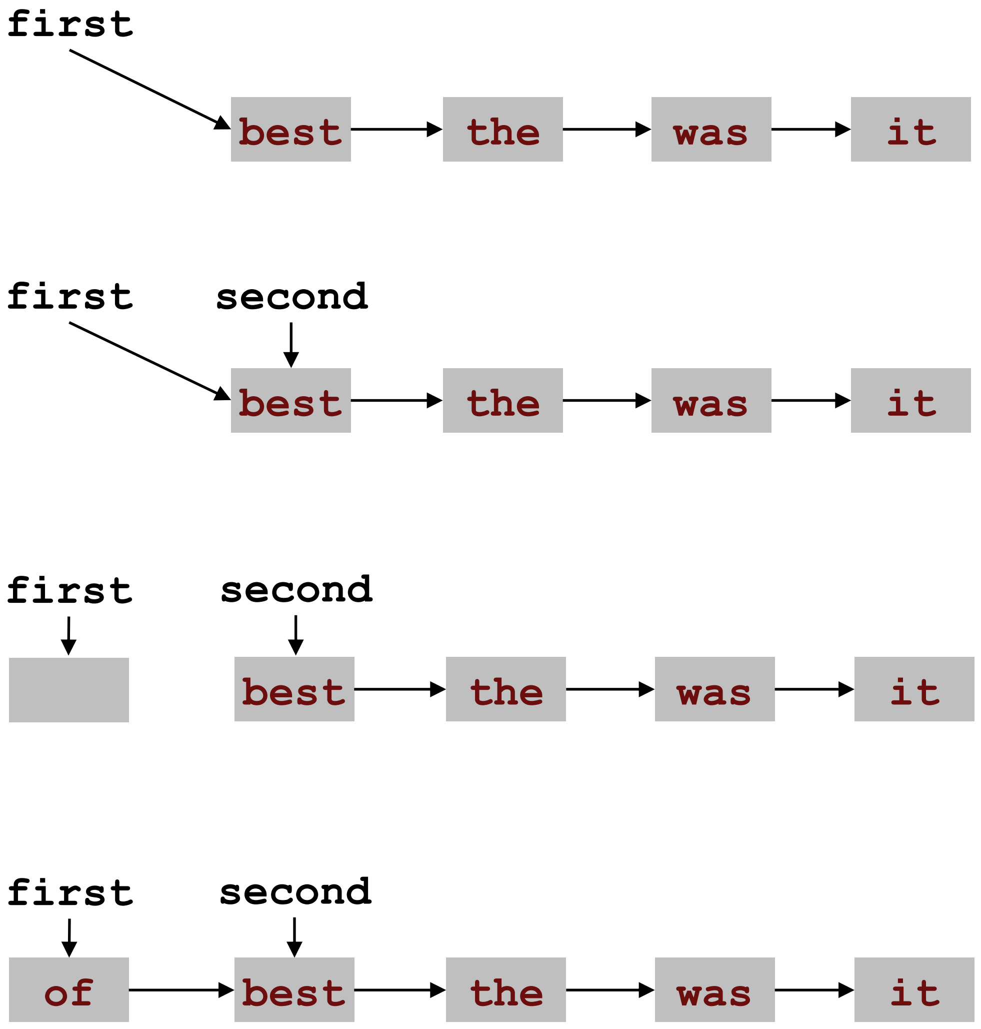
```
public class *StackOfStrings
_____________________________________________________________
                 *StackOfStrings()    create an empty stack
         boolean  isEmpty()           is the stack empty?
            void  push(String item)   push a string onto the stack
          String  pop()               pop the stack
```





Section 4.3

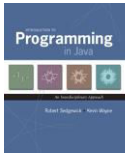
Stack Push: Linked List Implementation



```
Node second = first;
```

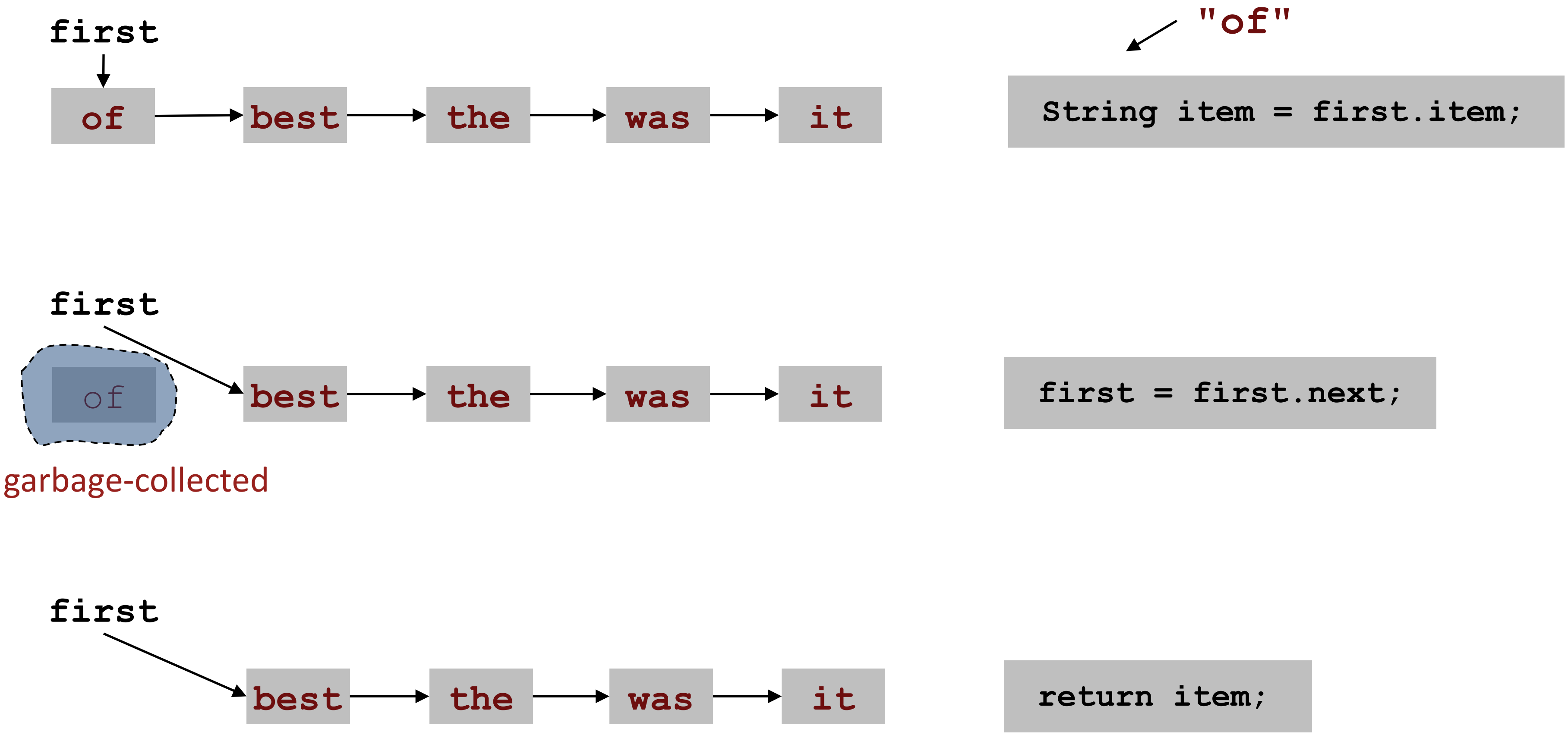
```
first = new Node();
```

```
first.item = "of";
first.next = second;
```



Section 4.3

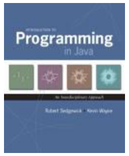
Stack Pop: Linked List Implementation



```
String item = first.item;
```

```
first = first.next;
```

```
return item;
```



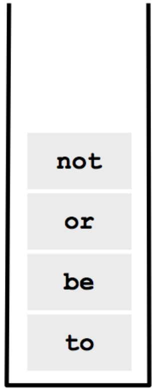
Section 4.3

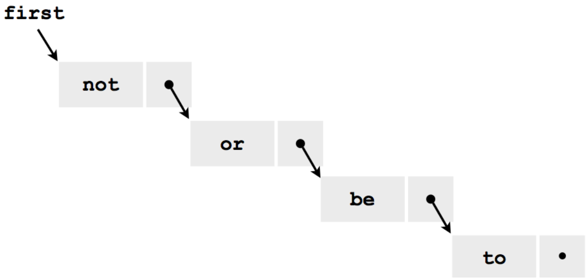
Stack: Linked List Implementation

```
public class LinkedStackOfStrings {
   private Node first = null;

   private class Node {
      private String item;
      private Node next;
   }

   public boolean isEmpty() { return first == null; }

   public void push(String item) {
      Node second = first;
      first = new Node();
      first.item = item;
      first.next = second;
   }

   public String pop() {
      String item = first.item;
      first = first.next;
      return item;
   }
}
```

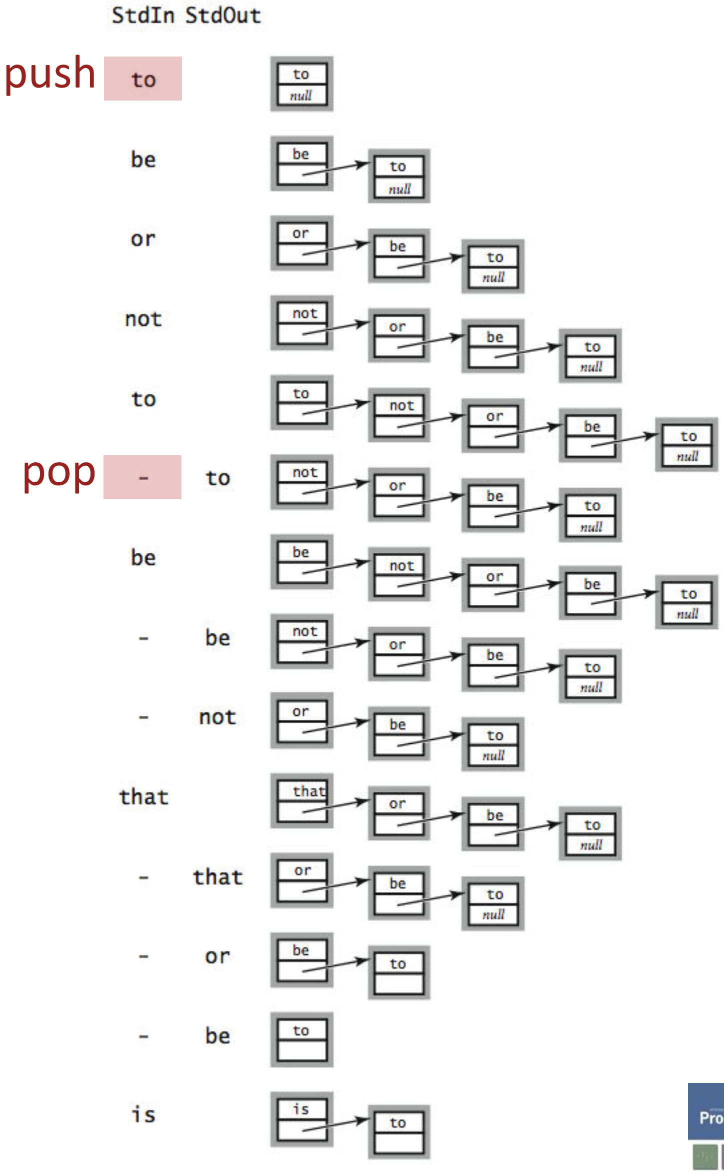
"inner class"

stack and linked list contents after 4th push operation

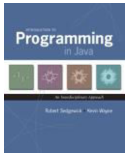




Linked List Stack: Test Client Trace



	StdIn	StdOut	
push	to		to null
	be		be → to null
	or		or → be → to null
	not		not → or → be → to null
	to		to → not → or → be → to null
pop	-	to	not → or → be → to null
	be		be → not → or → be → to null
	-	be	not → or → be → to null
	-	not	or → be → to null
	that		that → or → be → to null
	-	that	or → be → to null
	-	or	be → to
	-	be	to
	is		is → to



Section 4.3

Stack Data Structures: Tradeoffs

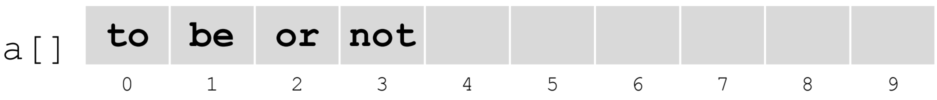
Two data structures to implement **`Stack`** data type.

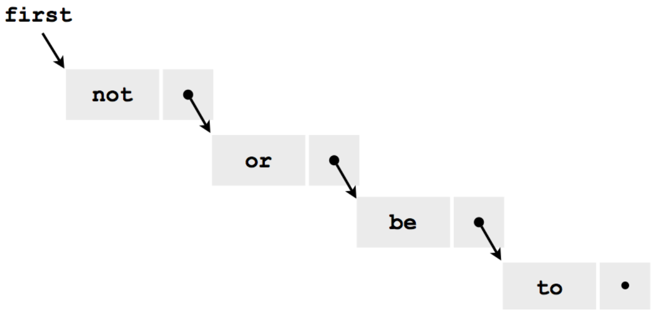
Array:

- Every push/pop operation take constant time.
- But… must fix maximum capacity of stack ahead of time.

Linked list:

- Every push/pop operation takes constant time.
- Memory is proportional to number of items on stack.
- But… uses extra space and time to deal with references.

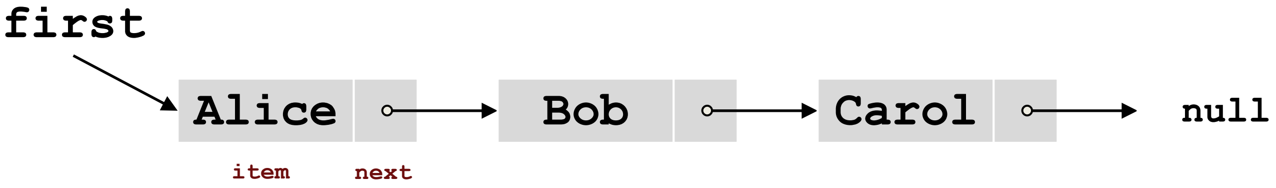


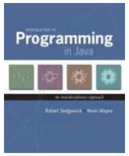


List Processing Challenge 1

What does the following code fragment do?

```
for (Node x = first; x != null; x = x.next) {
   System.out.println(x.item);
}
```



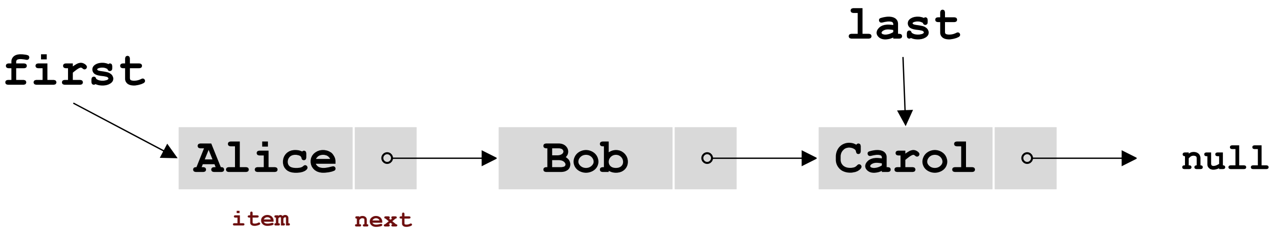


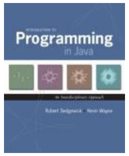
Section 4.3

List Processing Challenge 2

What does the following code fragment do?

```
Node last = new Node();
last.item = args[0];
last.next = null;
Node first = last;
for (int i = 1; i < args.length; i++) {
    last.next = new Node();
    last = last.next;
    last.item = args[i];
    last.next = null;
}
```





Section 4.3