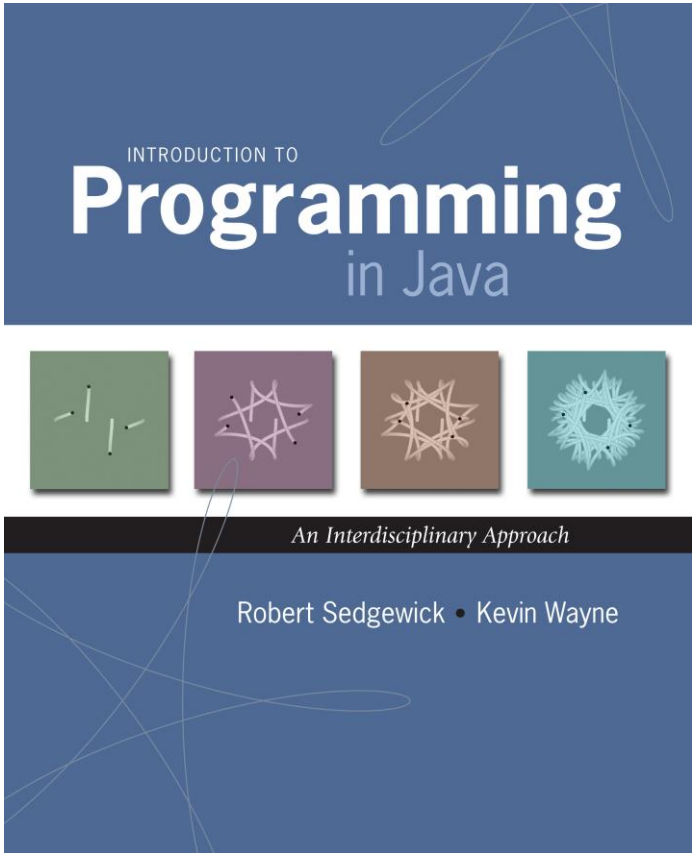
# 4.3 Stacks, Queues, and Linked Lists



# Data Types and Data Structures

Data types. Set of values and operations on those values.

- Some are built into the Java language: `int`, `double[]`, `String`, ...
- Most are not: `Complex`, `Picture`, `Stack`, `Queue`, `ST`, `Graph`, ...

Data structures.

- Represent data or relationships among data.
- Some are built into Java language: arrays.
- Most are not: linked list, circular list, tree, sparse array, graph, ...

# Collections

## Fundamental data types.

- Set of operations (add, remove, test if empty) on generic data.
- Intent is clear when we insert.
- Which item do we remove?

http://www.gatreasures.com/index.php?main_page=page&id=7

## Stack. [LIFO = last in first out]

- Remove the item most recently added.
- Ex: Pez, cafeteria trays, Web surfing.

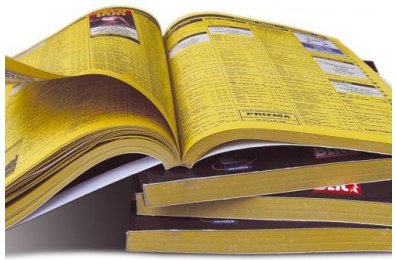
http://www.zdnet.com/apple-fans-queue-for-ipads-uk-launch-3040089083/

## Queue. [FIFO = first in, first out]

- Remove the item least recently added.
- Ex: Guitar Hero (RingBuffer)

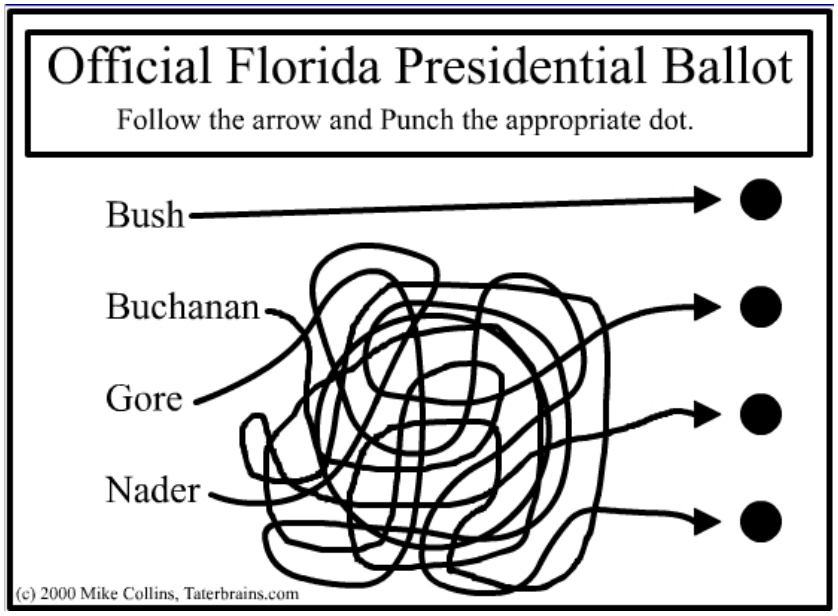
## Symbol table.

- Remove the item with a given key.
- Ex: Phone book.



http://www.otreva.com/web-design-scranton-phonebook-vs-internet-marketing/

# Linked Lists



# Sequential vs. Linked Allocation

**Sequential allocation.** Put items one after another.

- TOY: consecutive memory cells.
- Java: array of objects.

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

- TOY: link is memory address of next item.
- Java: link is reference to next item.

**Key distinctions.**

- Array: random access, fixed size. (get $i^{th}$ item)
- 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 | - |

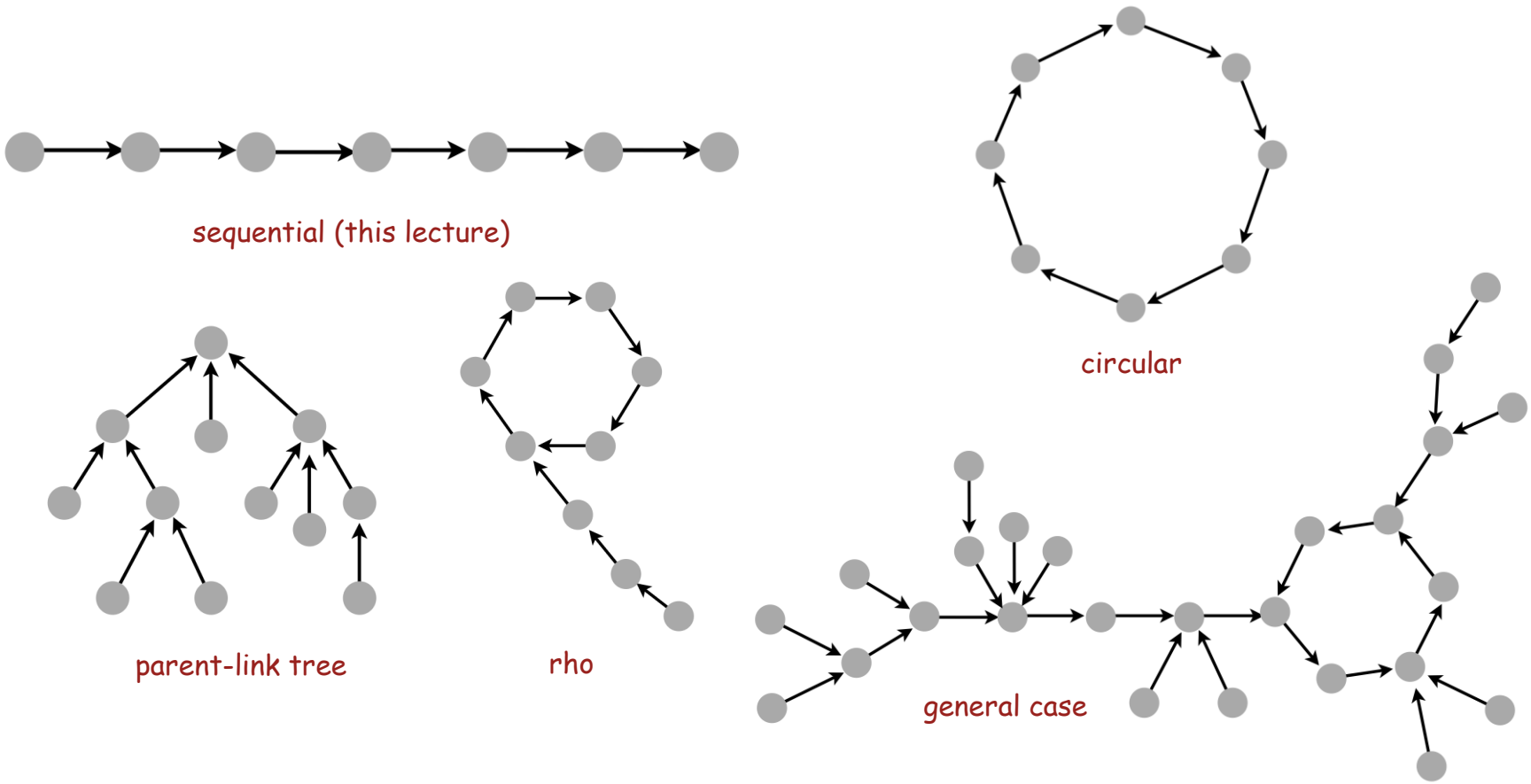
array (B0)

| 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.




Multiply-linked data structures. Many more possibilities.

# Linked Lists

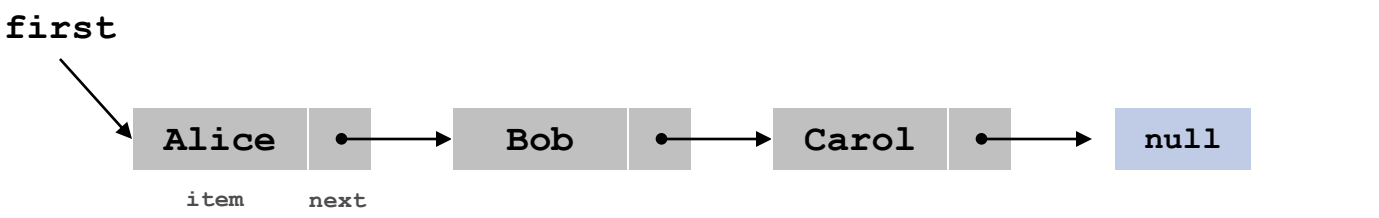
## Linked list.

- 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`.

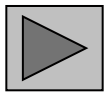
```
public class Node {
  public String item;
  public Node next;
}
```

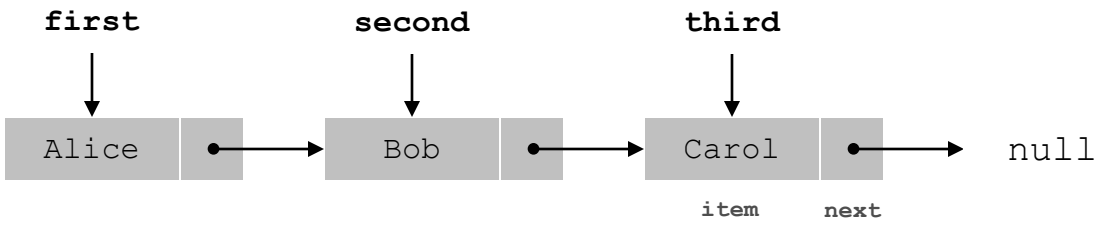


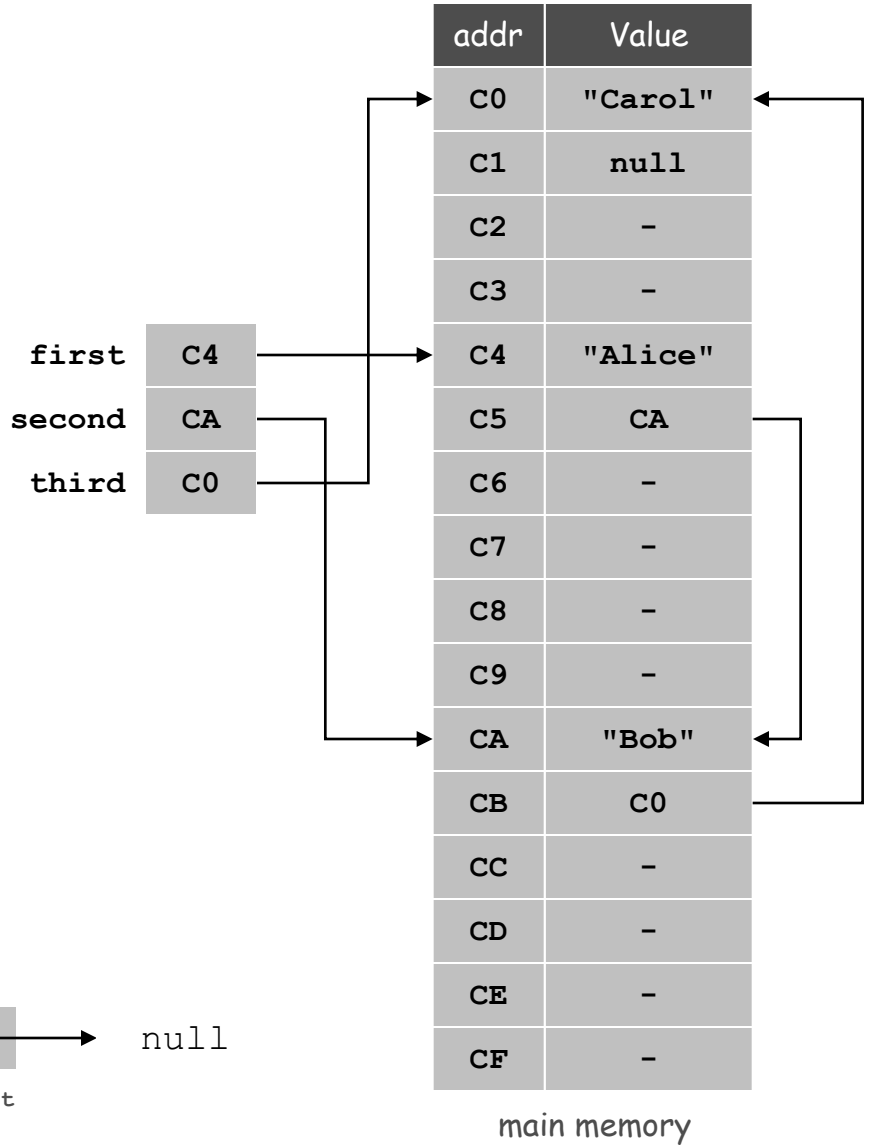

special pointer value `null` terminates list

# Building a Linked List

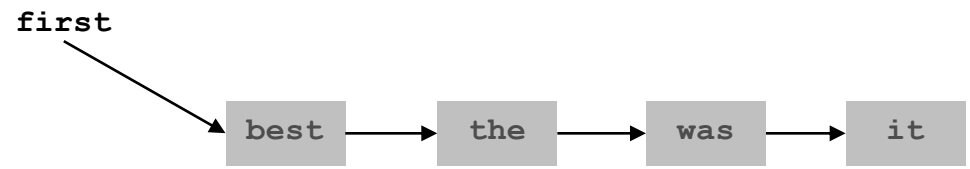
```
Node third  = new Node();
third.item  = "Carol";
third.next  = null;

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

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

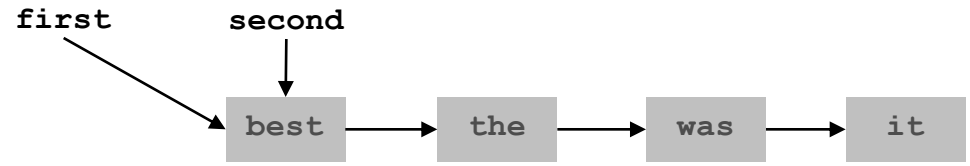




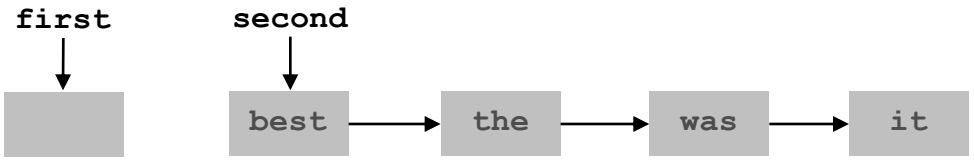



# Stack Push (Insert at Front): Linked List Implementation

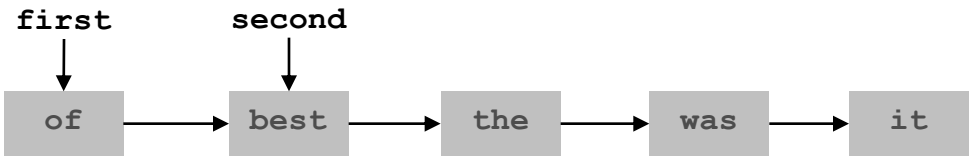



```
Node second = first;
```

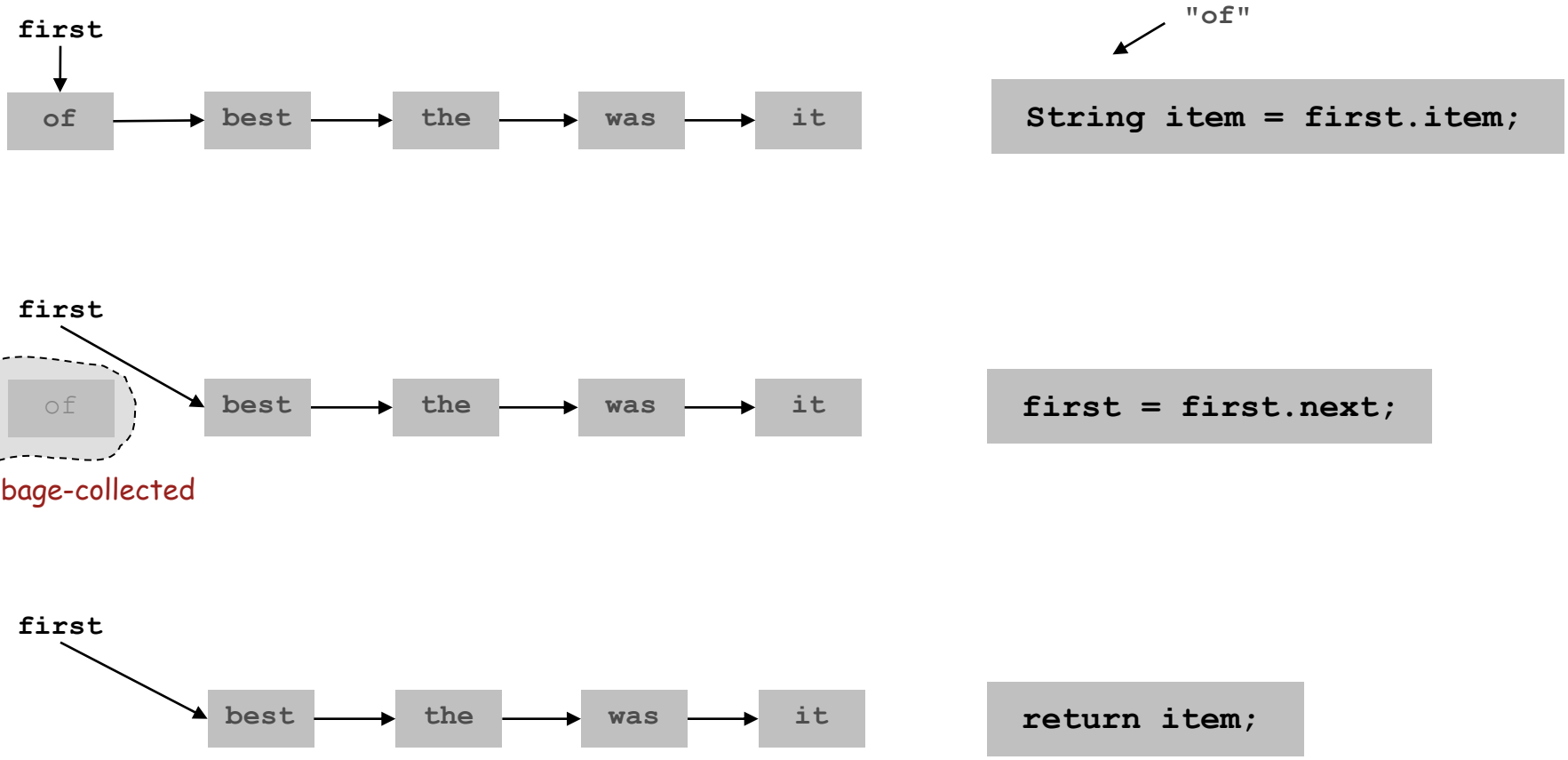



```
first = new Node();
```




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

# Stack Pop (Remove from Front): Linked List Implementation

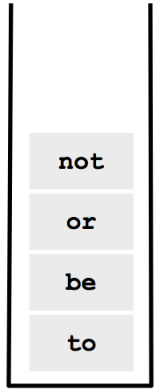


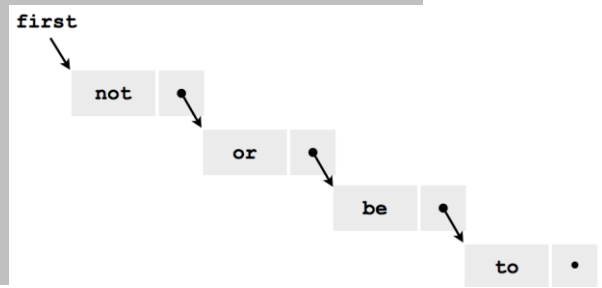
# Stack: Linked List Implementation

```
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



# Stack Data Structures: Tradeoffs

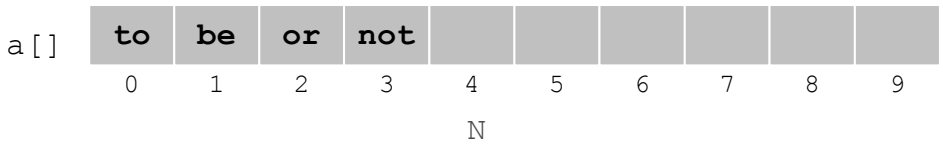
Two data structures to implement `stack` data type.

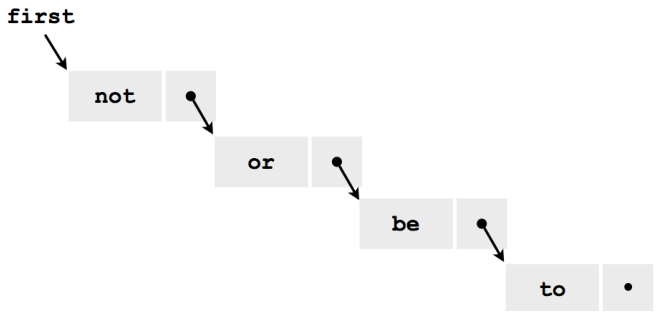
Array.

- 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.

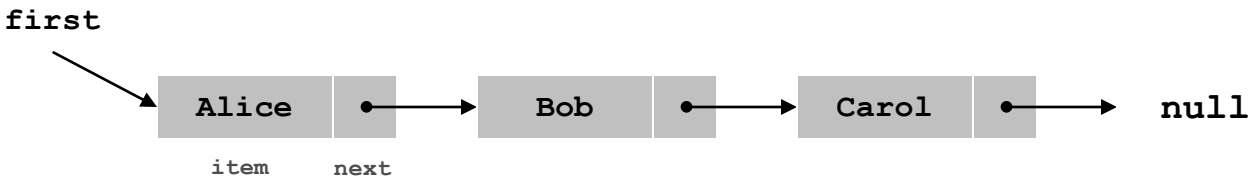


# List Processing Challenge 1

Q. What does the following code fragment do?

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



# List Processing Challenge 2

Q. What does the following code fragment do?

```
Node last = new Node();
last.item = StdIn.readString();
last.next = null;
Node first = last;
while (!StdIn.isEmpty()) {
   last.next = new Node();
   last = last.next;
   last.item = StdIn.readString();
   last.next = null;
}
```