4.3 Stacks, Queues, and Linked Lists

CIS 11 We Have .CLASS

Penn Engineering

Introduction to Programming in Java

Section 4.3

Data Types and Data Structures

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

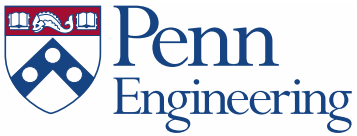
- Some are built into the Java language: `int`, `double[]`, `String`, ...
- Most are not: `Complex`, `Picture`, `Stack`, `Queue`, `ST`, `Graph`, ...

this lecture

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, ...

this lecture



Section 4.3

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?

Stack: [LIFO = last in first out] ← this lecture

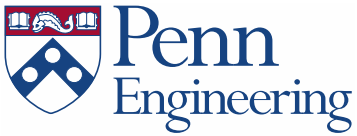
- Remove the item most recently added.
- Ex: Pez, cafeteria trays, Web surfing.

Queue: [FIFO = first in, first out] ← Harp

- Remove the item least recently added.
- Ex: Line for help in TA office hours.

Symbol table:

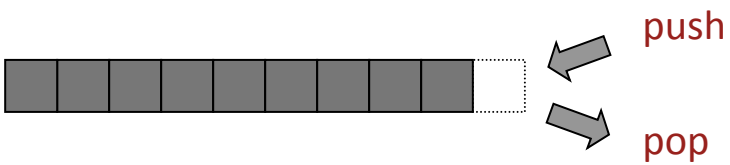
- Remove the item with a given key.
- Ex: Phone book.



Section 4.3

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*



Stack Client Example 1: Reverse

```
public class Reverse {
   public static void main(String[] args) {
      StackOfStrings stack = new StackOfStrings();
      while (!StdIn.isEmpty()) {
         String s = StdIn.readString();
         stack.push(s);
      }
      while (!stack.isEmpty()) {
         String s = stack.pop();
         StdOut.println(s);
      }
   }
}
```

```
% more tiny.txt
it was the best of times

% java Reverse < tiny.txt
times of best the was it
```

stack contents when standard input is empty

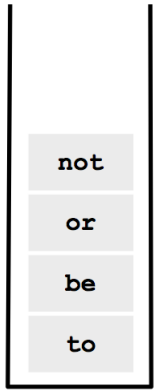
Stack: Array Implementation

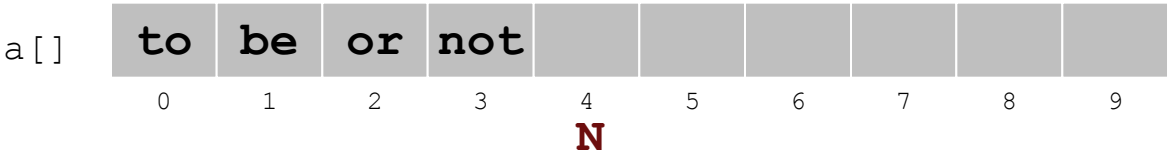
Array implementation of a stack.

how big to make array? [stay tuned]

- Use array `a[]` to store `N` items on stack.
- **`push()`** add new item at `a[N]`.
- **`pop()`** remove item from `a[N-1]`.

stack and array contents after 4th push operation

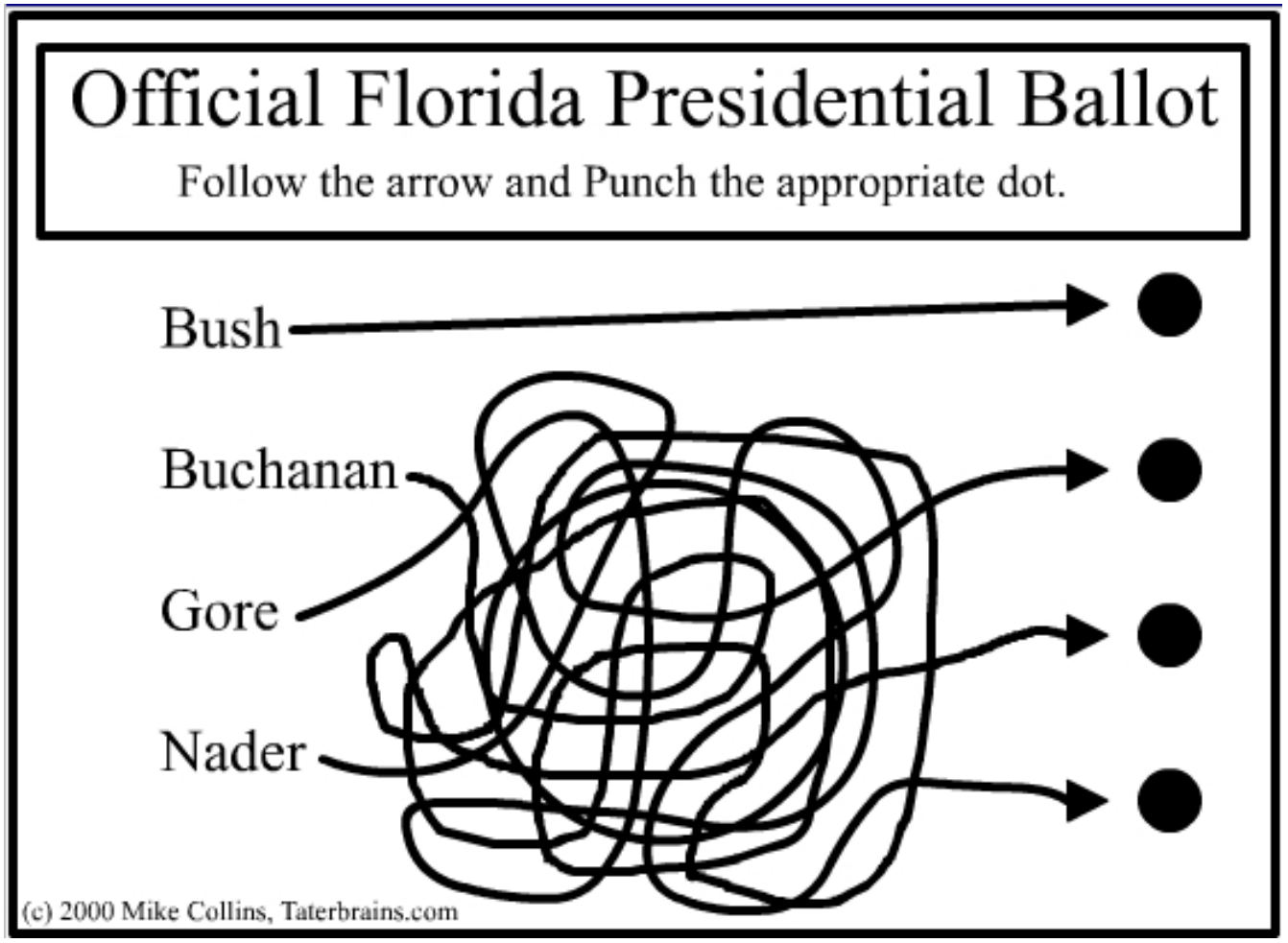





```
public class ArrayStackOfStrings {
   private String[] a;
   private int N = 0;

   public ArrayStackOfStrings(int max) { a = new String[max]; }
   public boolean isEmpty()            { return (N == 0);  }
   public void push(String item)       { a[N] = item; N++; }
   public String pop()                 { N--; return a[N]; }
}
```

temporary solution: make client provide capacity

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.

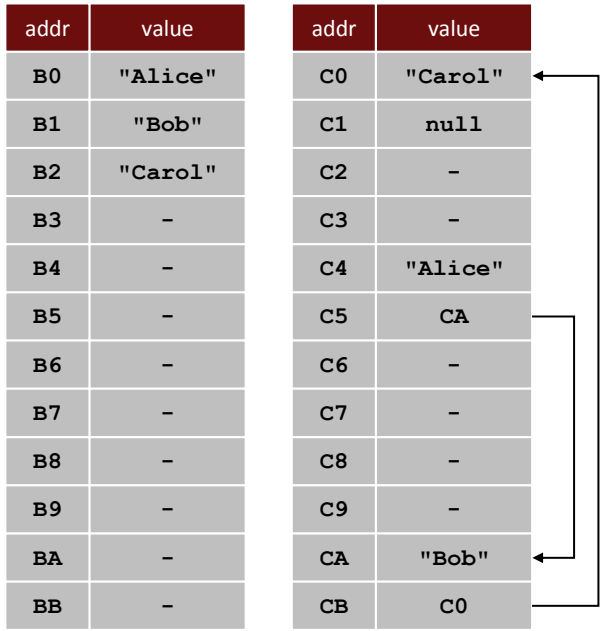
- TOY: link is memory address of next item.
- Java: link is reference to next item.

Key distinctions:

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

get i^{th} item

get next item



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

array (B0)

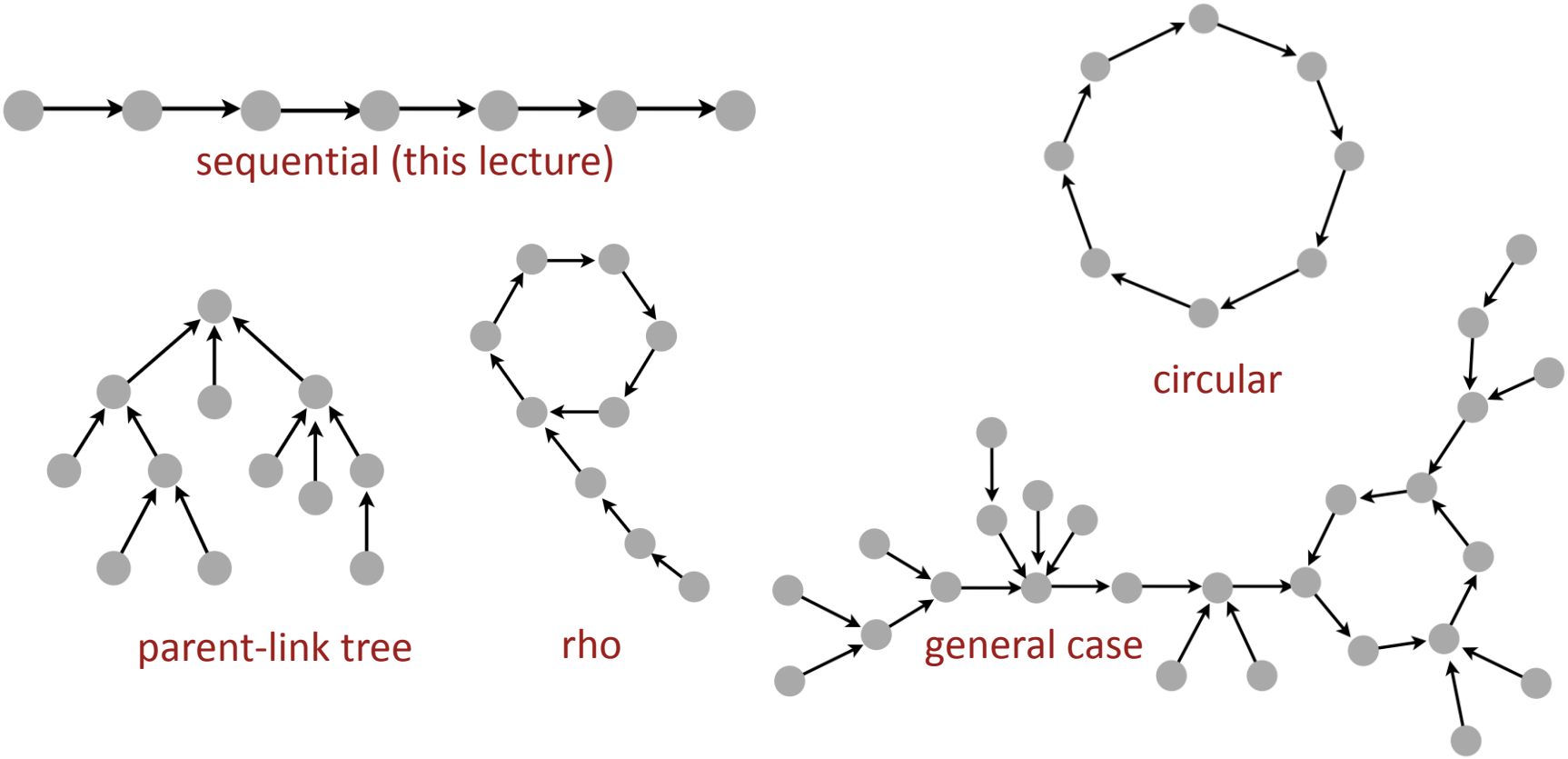
addr	value
C0	"Carol"
C1	null
C2	-
C3	-
C4	"Alice"
C5	CA
C6	-
C7	-
C8	-
C9	-
CA	"Bob"
CB	C0

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

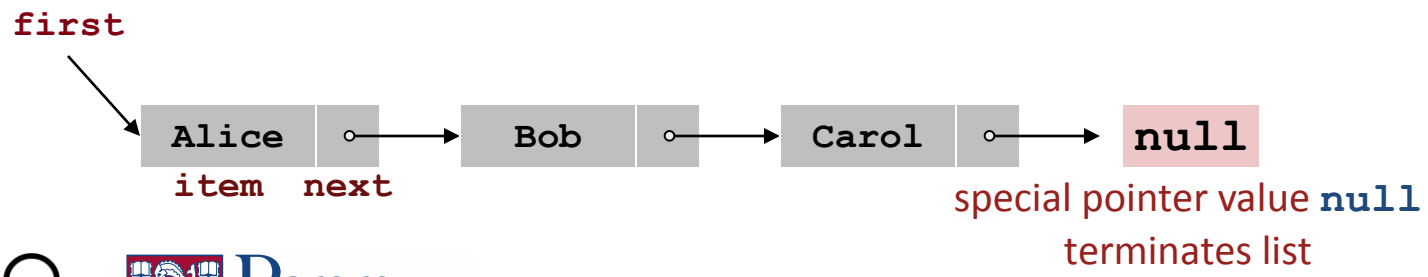
Linked list:

- 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:

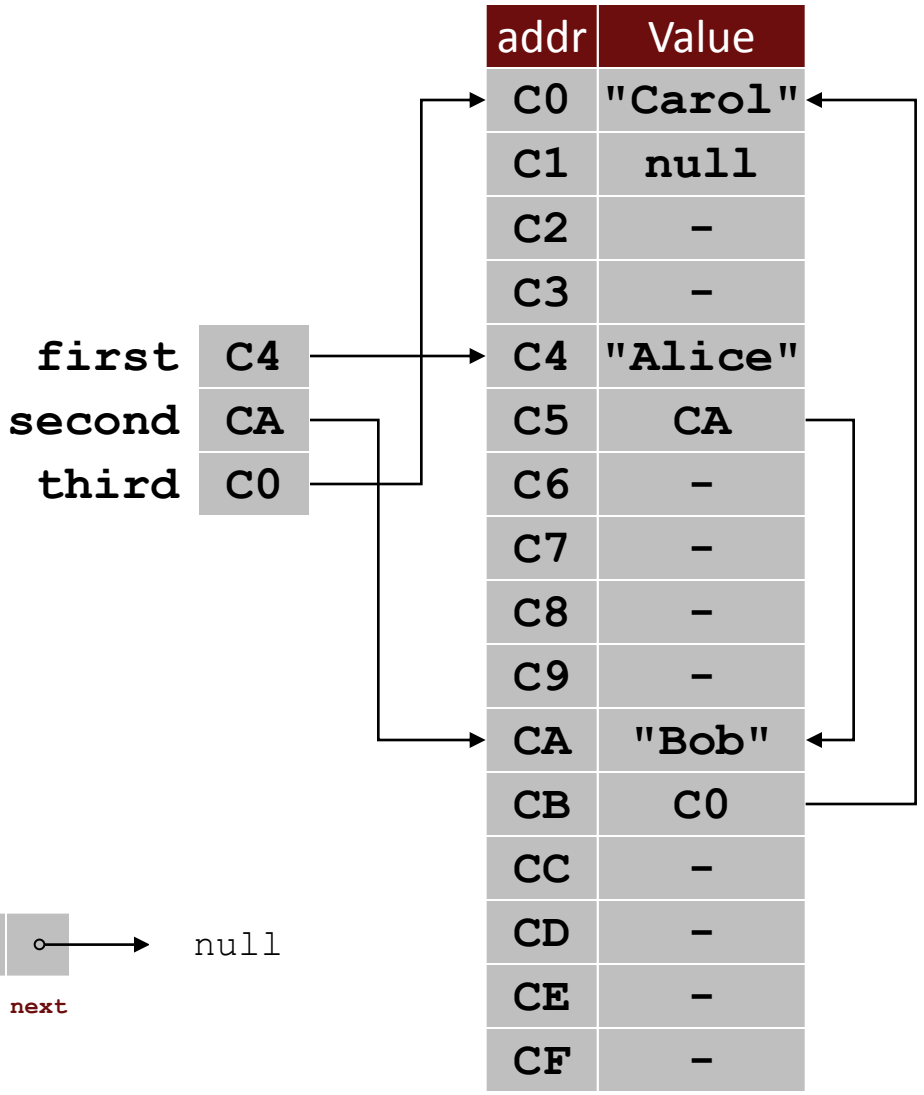
- A reference to a **`String`**.
- A reference to another **`Node`**.

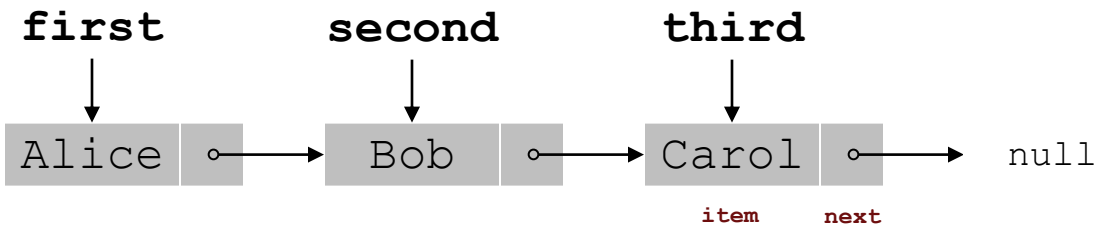
```
public class Node {
  public String item;
  public Node next;
}
```



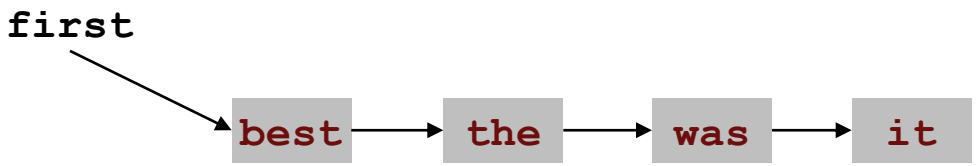
Building a Linked List

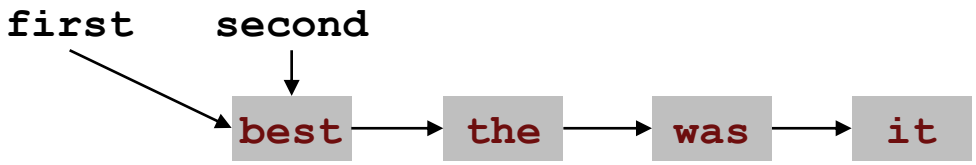
```
Node third  = new Node();
third.item  = "Carol";
third.next  = null;

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

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



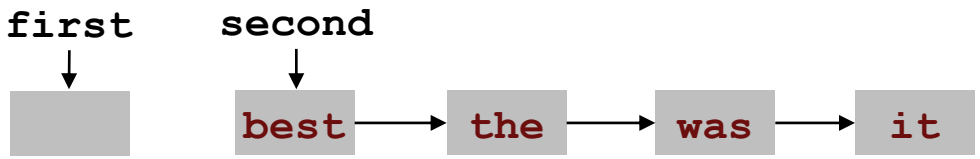


Stack Push: Linked List Implementation

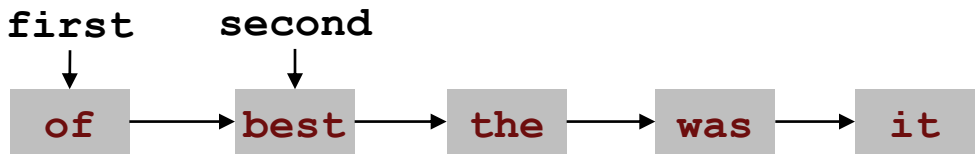





```
Node second = first;
```

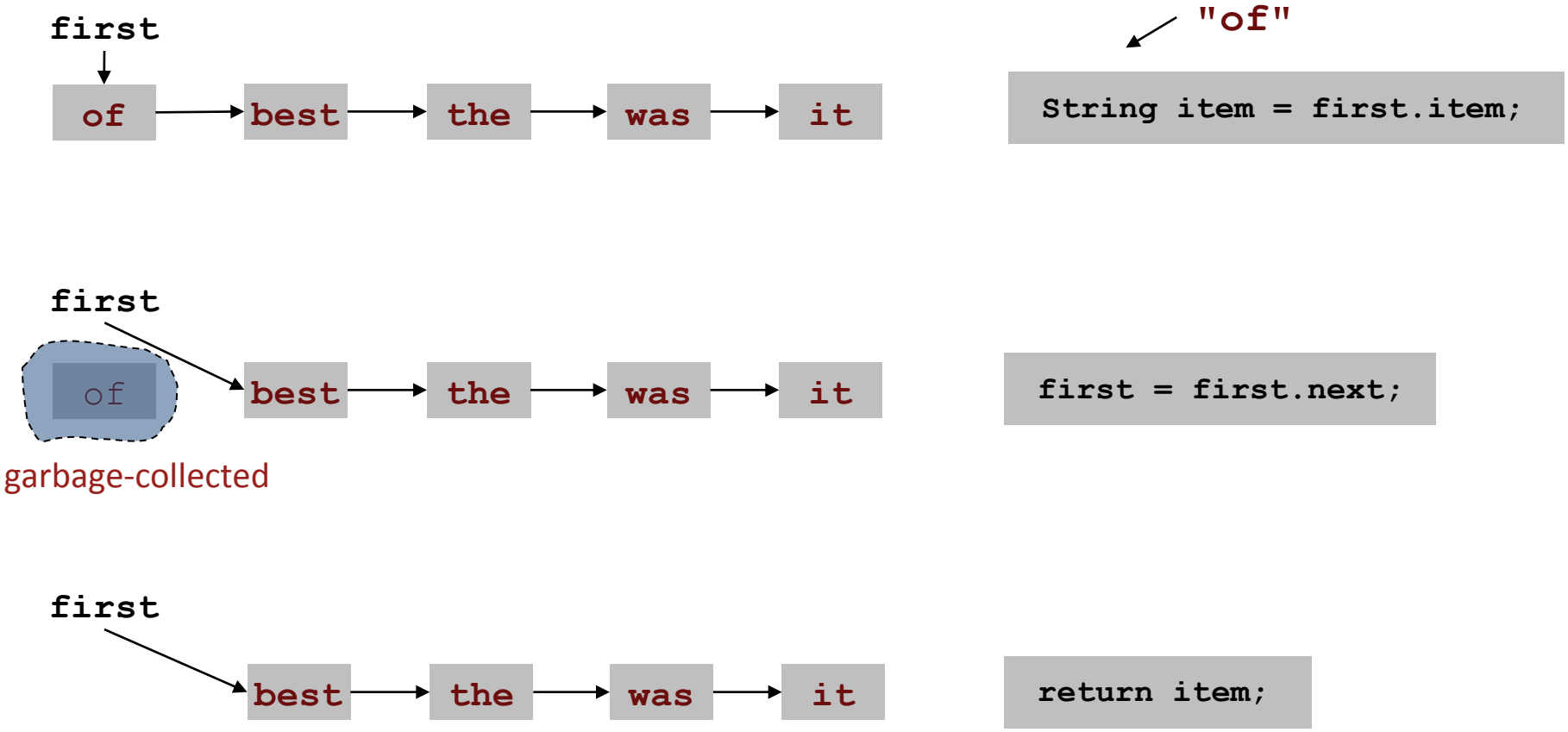



```
first = new Node();
```




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

Stack Pop: Linked List Implementation

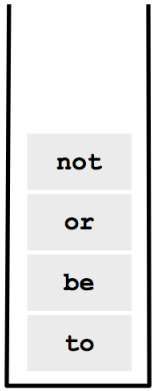


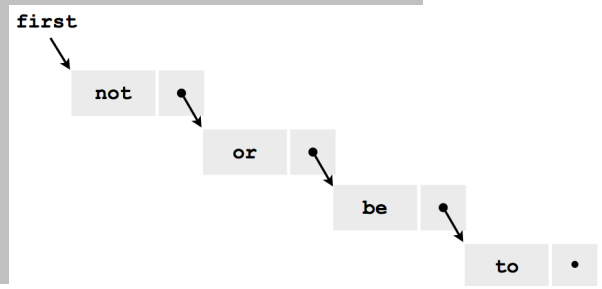
Stack: Linked List Implementation

```
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;
   }
}
```

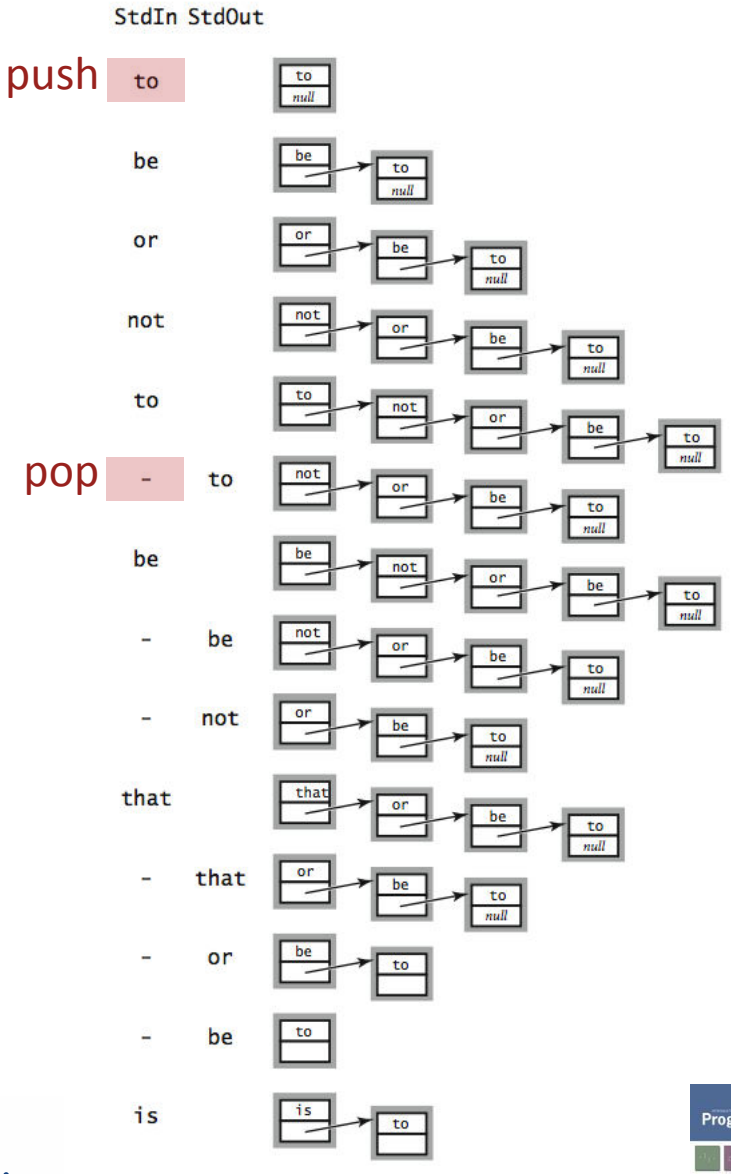
"inner class"

stack and linked list contents after 4th push operation





Linked List Stack: Test Client Trace



Stack Data Structures: Tradeoffs

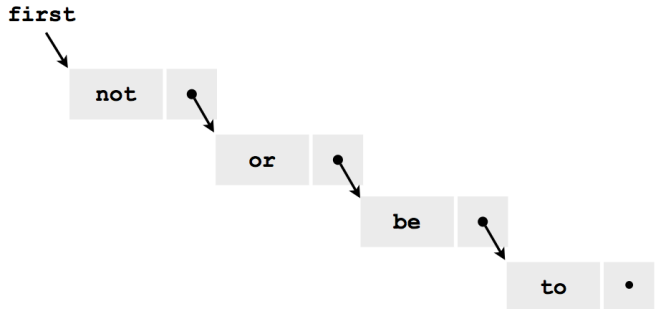
Two data structures to implement **`Stack`** data type.

Array:

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

Linked list:

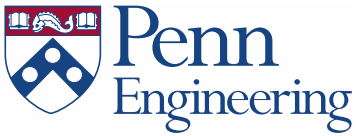
- Every push/pop operation takes constant time.
- Memory is proportional to number of items on stack.
- But… uses extra space and time to deal with references.



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);
}
```

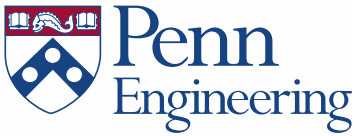


Section 4.3

List Processing Challenge 2

What does the following code fragment do?

```
Node last = new Node();
last.item = 5;
last.next = null;
Node first = last;
for (int i = 1; i < 6; i++) {
   last.next = new Node();
   last = last.next;
   last.item = i;
   last.next = null;
}
```



Section 4.3