Linked Lists

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

From the point of view of a particular object:
al of these structures look the same!

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.

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

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

first

Alice → Bob → Carol → null

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 API

```java
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
}
```

Diagram:
- `12` is pushed onto the stack.
- `37` is pushed afterward.
- `99` is pushed after `37`.
- `POP` removes `12` from the stack.
- `POP` removes `37` from the stack.
- `POP` removes `99` from the stack.

Section 4.3
Stack Push: Linked List Implementation

```java
Node second = first;
first.item = "of";
first.next = second;
first = new Node();
```
Stack Pop: Linked List Implementation

```java
String item = first.item;
first = first.next;
return item;
```

Section 4.3
Stack: Linked List Implementation

```java
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

What does the following code fragment do?

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

```plaintext
Alice Bob Carol null
```

Section 4.3
What does the following code fragment do?

```java
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;
}

first

Alice → Bob → Carol → null

last
```