Linked Lists
with some bells and whistles

Nested classes
Exceptions
Home-brew iterator
$ javadoc *.java
Loading source file WList.java...
Loading source file WordCount.java...
Constructing Javadoc information...
Standard Doclet version 1.6.0_26
...
An **iterable** linked list of (word, count) pairs

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <strong>addCount</strong>(java.lang.String word)</td>
</tr>
<tr>
<td>Increment the count associated with a word.</td>
</tr>
<tr>
<td>int <strong>countOf</strong>(java.lang.String word)</td>
</tr>
<tr>
<td>Get the count associated with a word.</td>
</tr>
<tr>
<td>int <strong>headCount</strong>()</td>
</tr>
<tr>
<td>Get count at head of list.</td>
</tr>
<tr>
<td>java.lang.String <strong>headWord</strong>()</td>
</tr>
<tr>
<td>Get word at head of list.</td>
</tr>
<tr>
<td>void <strong>insert</strong>(java.lang.String word)</td>
</tr>
<tr>
<td>Insert a word in the list, with an initial count of 1.</td>
</tr>
<tr>
<td>WList.WListIterator <strong>iterator</strong>()</td>
</tr>
<tr>
<td>Allow users to iterate through the list</td>
</tr>
<tr>
<td>void <strong>remove</strong>()</td>
</tr>
<tr>
<td>Remove word from head of list.</td>
</tr>
</tbody>
</table>
public class WList implements Iterable {

    Typically we would use the Java collections library, which has a good linked list implementation.

    To explore how it works, we’ll build one ourselves.
public class WList implements Iterable {

    private WCell head;

    // Nested class - we typically do this in Java when it will be used
    // only from this class.

    public class WCell {
        public String word;    // Public so I can use it in the iterator
        public int count;      // Public so I can use it in the iterator
        private WCell next;
    }
}
public class WList implements Iterable {
  
  private WCell head;

  public class WCell {
    public String word;
    public int count;
    private WCell next;
  }
}
/**
 * Insert a word in the list, with an initial count of 1.
 * @param word String to insert in the list (at the front).
 */
 public void insert(String word) {
     WCell cell = new WCell();
     cell.word = word;
     cell.count = 1;
     cell.next = head;
     head = cell;
 }
public void insert(String word) {
    WCell cell = new WCell();
    cell.word = word;
    cell.count = 1;
    cell.next = head;
    head = cell;
}
/**
 * Remove word from head of list.
 */
public void remove() {
    assert (head != null);  // Something is there to remove
    head = head.next;
}
/** Increment the count associated with a word.  
* @param word Increase the count associated with this word.  
* If this word is not in the list, insert it.  
*/

public void addCount(String word) {
    WCell cur = head;
    while (cur != null) {
        if (cur.word.equals(word)) {
            cur.count++;
            return;
        }
        cur = cur.next;
    }
    // Reached end of list, but didn't find the word
    this.insert(word);
}
And so on ...

Methods to insert, delete, find count of word, etc.

Now we want to make it easy to write a “for” loop to iterate through the list (e.g., to print it) from another class. We want ...

public class WList implements Iterable
Iterable and Iterator

Interface Iterable
requires a method Iterator iterator()

Interface Iterator
requires methods hasNext(), next()
also remove(), but it doesn’t have to work!
/**
 * Our own iterator.
 */

public class WListIterator implements java.util.Iterator {
    private WCell cur;       // Here is our finger

    public WListIterator(WCell head) { ... }

    public boolean hasNext() { ... }

    public WCell next() { ... }

    public void remove() { ... }
}
In another Java class ...

WList.WListIterator counts = wordCounts.iterator();
while (counts.hasNext()) {
    WList.WCell cell = counts.next();
    System.out.println(cell.word + "\t" + cell.count);
}
public class WListIterator implements java.util.Iterator {
  private WCell cur; // Here is our finger

  /**
   * Construct an iterator for a list.
   */
  public WListIterator(WCell head) {
    cur = head;
  }

  /**
   * Is there at least one more element in this list?
   * @return true iff there is at least one more element.
   */
  public boolean hasNext() {
    return (cur != null);
  }
}
/**
 * Return current cell and move forward.
 */

public WCell next() {
    if (cur == null) {
        throw new NoSuchElementException(
                "Off the end of the WList!");
    }
    WCell prev = cur;
    cur = cur.next;
    return prev;
}
/**
 * Return current cell and move forward.
 */

public WCell next() {
    if (cur == null) {
        throw new NoSuchElementException(
            "Off the end of the WList!");
    }
    WCell prev = cur;
    cur = cur.next;
    return prev;
}
Why Exceptions?

We could use “assert”
We could print a message
   and call System.exit(1)
We could return special values (as in C)

What’s wrong with those alternatives?
Why exceptions

Permit recovery
  E.g., on FileNotFoundException, we could prompt for another file

Don’t permit ignoring error
  by accident or through laziness
  A big problem with C error codes

Simplify control flow
try {
    Scanner in = new Scanner( new FileInputStream( args[0] ));
    WList wordCounts = new WList();
    while (in.hasNext()) {
        String word = in.next();
        // System.err.println("Reading word: " + word);
        wordCounts.addCount( word );
    }
    WList.WListIterator counts = wordCounts.iterator();
    while (counts.hasNext()) {
        WList.WCell cell = counts.next();
        System.out.println(cell.word + "\t" + cell.count);
    }
} catch (FileNotFoundException e) {
    System.out.println("No such file");
}
Exceptions

Are subclasses of Exception
Are thrown “up” the call-chain until they are caught
  either by the program, or at the top level
Come in two flavors
  Checked: You might recover, and must either catch or advertise
  Unchecked: Probably a bug or catastrophe
Recap

Linked list class

• Generally use Java’s linked list class instead, but
• You’ll build more complex structures with the same techniques

Java goodies

• Nested classes (typically private)
• Iterators and Iterable interfaces
  » Putting the traversal logic with the data structure class, and letting it look like a sequence to a calling class
• Exceptions
  » A cleaner way to handle exceptional cases