CSCI 136
Data Structures &
Advanced Programming

Lecture 7

Fall 2017

Instructors: Bill & Bill

Last Time

- Associations
- Code Samples
  - WordFreq, Dictionary (Associations, Vectors)
- Generic Data Types
- Lab 2 Design and Strategies
- Vector Implementation

Today: Linked Lists

- Vector Implementation continued
- Condition Checking
  - Pre- and post-conditions, Assertions
- List: A general-purpose structure
- Implementing Lists with linked structures
  - Singly and Doubly Linked Lists

Basic Vector<E> Methods

```
public class Vector<E> {
public Vector()            // Make a small Vector
public Vector(int initCap) // Make Vector of given capacity
public void add(E elt)     // Add elt to (high) end of Vector
public void add(int i, E elt)     // Add elt at position i
public E remove(E elt)     // Remove (and return) elt
public E remove(int i)     // Remove (and return) elt at pos i
public int capacity()      // Return capacity
public int size()          // Return current size
public boolean isEmpty()   // Is size == 0?
public boolean contains(E elt) // Is elt in Vector?
public E get(int i)        // Return elt at position i
public E set(int i, E elt) // Change value at position i
public int indexOf(E elt)  // Return earliest position of elt
}
```

Class Vector : Basic Methods

- Much work done by few methods:
  - indexOf(E elt, int i) // find first occurrance of elt at/after pos. I
    - Used by indexOf(E elt)
    - remove methods use indexOf(E elt)
  - firstElement(), lastElement() use get(int i)
- Method names/functions in spirit of Java classes
  - indexOf has same behavior as for Strings
- Methods are straightforward except when array is full
- How do we add to a full Vector?
  - We make a new, larger array and copy values to it

Extending the Array

- How should we extend the array?
- Possible extension methods:
  - Grow by fixed amount when capacity is reached
  - Double array when capacity is reached
- How could we compare the two techniques?
  - Run speed tests?
    - Hardware/system dependent
  - Count operations!
  - We'll do this soon

ensureCapacity

- How to implement `ensureCapacity(int minCapacity)`?

```
// post: the capacity of this vector is at least minCapacity
public void ensureCapacity(int minCapacity) {
   if (elementData.length < minCapacity) {
      int newLength = elementData.length; // initial guess
      if (capacityIncrement == 0) {
      // increment of 0 suggests doubling (default)
         if (newLength == 0) newLength = 1;
             while (newLength < minCapacity) {
               newLength *= 2;
             }
       } else {
      // increment != 0 suggests incremental increase
         while (newLength < minCapacity) {
             newLength += capacityIncrement;
         }
      }
```

ensureCapacity

```
  // assertion: newLength > elementData.length.
     Object newElementData[] = new Object[newLength];
     int i;

  // copy old data to array
     for (i = 0; i < elementCount; i++) {
        newElementData[i] = elementData[i];
     }

     elementData = newElementData;
        // garbage collector will pick up old elementData
  }
  // assertion: capacity is at least minCapacity
}
```

Pre and Post Conditions

- Recall `charAt(int index)` in Java String class
- What are the pre-conditions for charAt?
  - 0 <= index < length()
- What are the post-conditions?
  - Method returns char at position index in string
- We put pre and post conditions in comments above most methods

```
/* pre: 0 ≤ index < length
 * post: returns char at position index
 */
 public char charAt(int index) { … }
```

Pre and Post Conditions

- Pre and post conditions “form a contract”
- Post-condition is guaranteed if method is called when pre-condition is true
- Examples:
  - `s.charAt(s.length() - 1)`: index < length, so valid
  - `s.charAt(s.length() + 1)`: index > length, not valid
- These conditions document requirements that user of method should satisfy
- But, as comments, they are not enforced

Other Examples

- Other places pre and post conditions are useful

```
// Pre: other is of type Card
// Post: Returns true if suits and ranks match
public boolean equals(Object other) {
    if ( other instanceof Card ) {
        Card oc = (Card) other;
        return this.getRank() == oc.getRank() &&
            this.getSuit() == oc.getSuit();
    }
    else return false;
}
```

Assert Class

- Pre- and post-condition comments are useful as a programmer, but it would be *really* helpful to know as soon as a pre-condition is violated (and return an error)

- The Assert class (in structure5 package) allows us to programmatically check for pre- and post-conditions

Assert Class

The Assert class contains the methods:

```
public static void pre(boolean test, String message);
public static void post(boolean test, String message);
public static void condition(boolean test, String message);
public static void fail(String message);
```

If the boolean test is NOT satisfied, an exception is raised, the message is printed and the program halts

Assert Example

- Let's look in CardsWithBaileyAssert
- This time, we'll use assertions to check for pre-conditions
  - Have to import structure5.Assert (in bailey.jar)
- Use `instanceof` to check `Object other` in `equals()` method
  - This allows Java to print **useful** error messages when something is wrong

General Rules about Assert

1. State pre/post conditions in comments
2. Check conditions in code using “Assert”
3. Use Fail in unexpected cases (such as the default block of a switch statement)

- Any questions?
- From this point on:
  - You should use pre- and post-conditions
  - You are (strongly) encouraged to use assertions

The Java assert keyword

- An alternative to Duane's Assert class
- Added in Java 1.4
- Two variants
  - assert boolean_expression
    - Throws an AssertionError if the expression is false
  - assert boolean_expression : other_expression
    - In addition, prints value of other_expression
- See CardsWithJavaAssert.java

Assertions Help Debug

- No need to slow down "production" code
  - Assertions are disabled at runtime by default
  - Use `—enableassertions` or `—ea` to turn on assertions

```
javac —ea AbstractCard.java
```

Pros and Cons of Vectors

Pros

- Good general purpose list
- Dynamically Resizeable
- Fast access to elements
  - vec.get(387425) finds item 387425 in the same number of operations regardless of vec's size

Cons

- Slow updates to front of list (why?)
- Hard to predict time for add (depends on internal array size)
- Potentially wasted space

Today we look at another way to store data: Linked Lists

But First : List Interface

```
interface List {
   size()
   isEmpty()
   contains(e)
   get(i)
   set(i, e)
   add(i, e)
   remove(i)
   addFirst(e)
   getLast()
   .
   .
   .
}
```

- Flexible interface
- Can be used to describe many different types of lists
- It' s an interface…therefore it provides no implementation
- Vector implements List
- Other implementations are possible

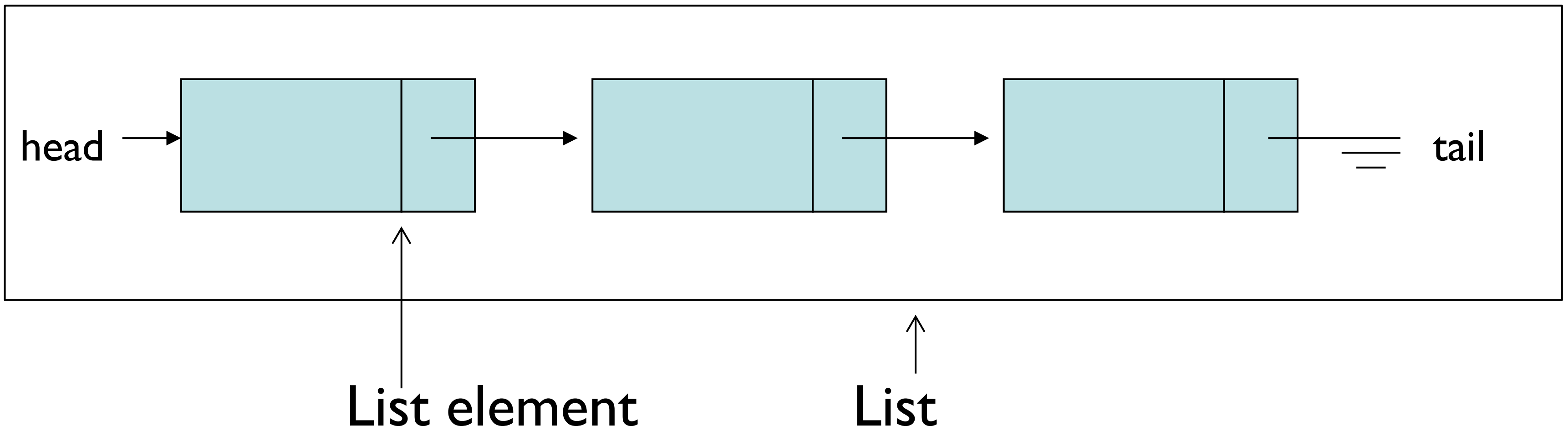
List Implementations

- General concept for storing/organizing data
- Vectors implement the List interface
- We now explore other List implementations
    - SinglyLinkedList
    - CircularlyLinkedList
    - DoublyLinkedList

Linked List Basics

- There are two key aspects of Lists
    - Elements of the list
    - The list itself

- Visualizing lists



head

tail

List element

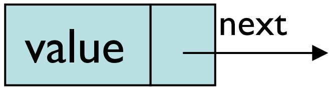
List

Linked List Basics

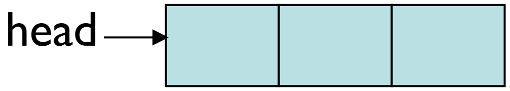
- List nodes are recursive data structures
- Each “node” has:
  - A data value
  - A “next” value that identifies the next element in the list
  - Can also have “previous” that identifies the previous element (“doubly-linked” lists)
- What methods does Node class need?

SinglyLinkedLists

- How would we implement SinglyLinkedListNode?
  - SinglyLinkedListNode = SLLN in my notes
  - SLLN = Node in the book (in Ch 9)



  - How about SinglyLinkedList?
  - SinglyLinkedList = SLL in my notes



- What would addFirst(E d) look like?
- getFirst()?
- addLast(E d)? (more interesting)
- getLast()?