List Implementations that Use Arrays

Chapter 13
Pre-requisites: Chapter 12 & Chapter 2
public interface ListInterface <T> {
    public void add(T newEntry);
    public void add(int newPosition, T newEntry);
    public T remove(int givenPosition);
    public void clear();
    public boolean replace(int givenPosition, T newEntry);
    public T getEntry(int givenPosition);
    public boolean contains(T anEntry);
    public int getLength();
    public boolean isEmpty();
    public T[] toArray();
}

// Q? Compare with java.util.List interface using java-doc, e.g.,
• Positions inside a list start with a 0
• Method get(int index) instead of getEntry(int givenPosition), size() instead of getLength()
• Many more methods, e.g., equals, indexOf, iterator(), listIterator(),
    • Remove(Object O), removeAll(Object O), retainAll(Collection c), subList(int fromIndex, toIndex)
Old Java version uses Object class instead of generic type
The Java Implementation

Figure 13-3 UML notation for the class `AList`
Chapter 13 Objectives

• Implement ADT list by using
  ▪ either array that you can resize
  ▪ or instance of Vector
  ▪ Or linked nodes (chapter 14)

• Discuss advantages, disadvantages of
  ▪ implementations presented
List ADT Implementation: Alternatives

1. Use an **array**
   - When all space used, move data to larger array
2. Use Java class **Vector**
   - Like an array that can expand automatically
3. Chain of **linked nodes**
   - Insertion/deletion anywhere is harder
Contents

• Using an Array to Implement the ADT List
  ▪ An Analogy
  ▪ The Java Implementation
  ▪ Efficiency of Array based Implementation

• What is it challenging?
  ▪ Lists have variable size, but, arrays have fixed size
    • Approach: Resize Array if it is full
  ▪ Which part of array should be used for small lists?
    • Approach: Keep empty slots at higher indices
    • Implication: Add(T) puts new entry at highest empty index
    • Implication: Shift elements if adding/deleting from lower indices
Listing 13-1 — Resizing

public class AList <T> implements ListInterface <T> {
    private T[] list;  private int numberOfEntries;
    private static final int DEFAULT_INITIAL_CAPACITY = 25;
    public AList() { this(DEFAULT_INITIAL_CAPACITY); }
    public AList(int initialCapacity) {
        numberOfEntries = 0;
        @SuppressWarning("unchecked") // Annotation for following statement
        T [] tempList = (T[]) new Object[initialCapacity]; list = tempList;
        // list = new T[initialCapacity]; is not legal in Java as explained on pp. 31
    }
    public void add(T newEntry) {
        ensureCapacity(); list[numberOfEntries] = newEntry; numberOfEntries++;
    }
    private void ensureCapacity() {
        if (numberOfEntries == list.length) list = Arrays.copyOf(list, 2 * list.length) ;
        // resizing explained on pp. 50-55 (Fig. 2-8, 2-9, 2-10)
        // java.util.ArrayList.ensureCapacity() may be slightly different!
    }
    // Note: Annotation type SuppressWarnings is described on pp. 31 (Chapter 2) and at
    // docs.oracle.com/javase/1.5.0/docs/api/java/lang/SuppressWarnings.html

Copyright © 2012 by Pearson Education, Inc. All rights reserved
Listing 13-1

// continued from last slide
public void add(int newPosition, T newEntry) {} // slides 11-12,
public T remove(int givenPosition) {} // slides 13-14
public void clear() {} // Question 10, pp. 330
public boolean replace(int givenPosition, T newEntry) {} // pp. 330-331
public T getEntry(int givenPosition) {} // pp. 331
public boolean contains(T anEntry) {} // pp. 331
public int getLength() { return numberOfEntries; }
public boolean isEmpty() { numberOfEntries == 0; }
public T[] toArray() {
    @SuppressWarnings("unchecked")
    T[] result = (T[]) new Object[numberOfEntries];
    for (int i = 0; i < numberOfEntries; i++) { result[i] = list[i]; }
    return result;
}
}
Shift Analogy: Array vs. Classroom

Figure 13-1 A classroom that contains desks in fixed positions
add(int, T) vs. Seating a new student

Figure 13-2 Seating a new student between two existing students: At least one other student must move.
The Java Implementation

Figure 13-4 Making room to insert Carla as the third entry in an array

- Note Alist code, Listing 13-1

Note: Code listing files must be in same folder as PowerPoint files for links to work
public void add(int newPosition, T newEntry) {
    boolean isSuccessful = true;
    if ((newPosition >= 1) && (newPosition <= numOfEntries + 1)) {
        ensureCapacity();
        makeRoom(newPosition);
        list[newPosition - 1] = newEntry;
        numberOfEntries++;
    } else { isSuccessful = true; }
    return isSuccessful ;
}

private void makeRoom(int newPosition) {
    assert(newPosition >= 1) && (newPosition <= numberOfEntries + 1);
    int newIndex = newPosition - 1;
    int lastIndex = numberOfEntries - 1;
    for (int index = lastIndex; index >= newIndex; index--) {
        list[index + 1] = list[index];
    }
}

//Exercise: Try Questions. 4, 5 and 6 from Textbook page 328.
Figure 13-5 Removing Bob by shifting array entries
public T remove(int givenPosition,)
    {
        T result = null;
        if ((givenPosition >= 1) && (newPosition <= numOfEntries + 1)) {
            assert !isEmpty();
            result = list[givenPosition - 1];
            if (givenPosition < numberOfEntries) removeGap(givePosition);
            numberOfEntries--;
        }
        return result;
    }
private void removeGap(int givenPosition) {
    assert(givenPosition >= 1) && (givenPosition < numberOfEntries);
    int removedIndex = givenPosition - 1;
    int lastIndex = numberOfEntries - 1;
    for (int index = removedIndex; index < lastIndex; index++)
        {  list[index] = list[index + 1] ;  }
}
//Exercise: Try Questions. 7, 8 and 9 from Textbook page 329 - 330.

Copyright ©2 012 by Pearson Education, Inc. All rights reserved
Using a Vector to Implement the ADT List

- View class `VectorList`, **Listing 13-A**

- Note
  - Example of an adaptor class
  - Writing code for the class simple
  - Execution may be slow due to background invocation of `Vector` methods
  - Adding at end of list, retrieving specific entry are fast
  - Adding, removing in middle of list slower
End

Chapter 13