Lecture 6: ArrayList Implementation

CS 62
Fall 2018
Alexandra Papoutsaki & William Devanny
Programming Assignment

• Weak AI/Natural Language Processing:
  • Generate text by building frequency lists based on pairs of words. `ArrayList` of `Associations` of `String` (words) and `Integer` (count of that word)

• Harder assignment, start early!
The picture so far...

• When you wanted to store a collection of data you would use arrays

• The problem: fixed length (**final** instance variable **length**)
  • Once you have created them, they cannot grow or shrink

• Useful when we know in advance the number of elements to hold, but how often is this the case?

• Don’t play nicely with Generics
Data Structures

• Collections of:
  • Data
  • Their relationships
  • The operations that can be applied to them

• In OOP a collection is an object of elements

• Some collections are ordered, some are not

• Some allow duplicate elements, some do not

• Typically collections provide operations that allow us to add, remove, search for an element, and ask for their size (# elements)
Collections in Java

List ADT

• A collection storing elements in an ordered fashion
• Can access elements by a 0-based index
• It has a known size which is the number of elements it holds
• Elements can be added at the front, rear, or intermediately

![List ADT Diagram]

- size = 8
Array list

• Automatically resizable list

• By default, add a new element to end of list

• In Java, import java.util.ArrayList;
  • public class ArrayList<E> extends AbstractList<E>
    implements List<E>

• Remember wrapper classes
ArrayList\<E\> major methods

- `ArrayList()`: constructs an empty list with initial capacity of 10.
- `ArrayList(int N)`: constructs an empty list with initial capacity of `N`.
- `boolean add(E e)`: appends element at end of list
- `boolean add(int index, E e)`: appends element at index
- `void clear()`: removes all elements from list
- `E get(int index)`: returns element at index
- `E remove(int index)`: removes & returns element at index
- `boolean remove(Object o)`: removes & returns first occurrence of element, if it exists
- `int size()`: returns number of elements
ArrayList\<E>\\n\\n• Standard Java libraries have lots of extra methods not in our implementation\\n• Many involve working on other collections  
  • irrelevant for us at this point:\n  • `addAll`, `contains`, `containsAll`, `listIterator`, `removeAll`, `replaceAll`, `retainAll`, `sort`, `spliterator`, `sublist`, `toArray`
ArrayList\<E\> and Generics

• When instantiating an array list, we need to specify the type of elements E that it will hold

• `ArrayList<Type> al = new ArrayList<Type>();`
ArrayList\<E\> vs arrays

- String[] faculty = new String[2];
  - ArrayList<String> faculty = new ArrayList<String>();

- faculty[0] = “Melanie Wu”;
  - faculty.add(“Melanie Wu”);

- String name = faculty[0];
  - String name = faculty.get(0);

- for(int i=0; i<faculty.length; i++)
  System.out.println(faculty[i]);
  - for(int i=0; i<faculty.size(); i++)
    System.out.println(faculty.get(i));
  - for(String name: faculty)
    System.out.println(name); //for-each loop
Tamassia & Goodrich \texttt{ArrayIndexList\langle E\rangle}

- Interface is \texttt{IndexList\langle E\rangle}

\texttt{ArrayIndexList\langle E\rangle}
  - Similar to \texttt{ArrayList}
  - Instance variables:
    - \texttt{elts}: array instance variable
    - \texttt{eltsFilled}: number of slots filled.

- Creating new array list is weird
  - Can’t construct array of variable type!
  - Create array of \texttt{Object}, but coerce to believe array of \texttt{E}
public interface IndexList<E> {

    public int size();

    public boolean isEmpty();

    public void add(int i, E e) throws IndexOutOfBoundsException;

    public E get(int i) throws IndexOutOfBoundsException;

    public E remove(int i) throws IndexOutOfBoundsException;

    public E set(int i, E e) throws IndexOutOfBoundsException;
}

public class ArrayIndexList<E> implements IndexList<E> {

    private E[] elts; //array storing the elements
    private int capacity = 16; // initial length of array elts
    private int eltsFilled = 0; // number of elements stored

    @SuppressWarnings("unchecked")
    public ArrayIndexList() {
        elts = (E[]) new Object[capacity];
        // the compiler may warn, but this is ok
    }

    private void checkIndex(int r, int n) throws IndexOutOfBoundsException {
        if (r < 0 || r >= n) throw new IndexOutOfBoundsException("Illegal index: " + r);
    }

    //fill the rest
}
/**
 * @return the number of elements in the indexed list.
 */

public int size() {
    return eltsFilled;
}

/**
 * @return whether the indexed list is empty.
 */

public boolean isEmpty() {
    return size() == 0;
}

/**
 * @return the element stored at the given index.
 */

public E get(int r) throws IndexOutOfBoundsException {
    checkIndex(r, size());
    return elts[r];
}
/**
* @param r the index to be updated
* @param newElt the element to go in slot r
* @return the element originally in slot r
* post: The element stored now at index r is now newElt
*/

public E set(int r, E newElt) {
    checkIndex(r, size());
    E temp = elts[r];
    elts[r] = newElt;
    return temp;
}
/**
 * post: If elts is full, then copy elements of elts to new array with twice the
 * capacity. There is now at least one empty slot in the array representation
 */

@SuppressWarnings("unchecked")
private void ensureCapacity() {
    if(eltsFilled == capacity){
        capacity *= 2;
        E[] newElts = (E[]) new Object[capacity];
        for (int i=0; i<EltsFilled; i++){
            newElts[i]=elts[i];
        }
        elts = newElts;
    }
}
}
/**
 * @param r the index to be updated
 * @param newElt the element to go in slot r
 * @return the element originally in slot r
 * post: The element stored now at index r is now newElt
 */

public void add(int r, E newElt) {
    checkIndex(r, size()+1);
    ensureCapacity();
    for(int i = eltsFilled-1; i>=r; i--)
        elts[i+1] = elts[i]; //shift elements to the right
    elts[r]=elt;
    eltsFilled++;
}

/**
 * @param elt element to be added to the rear of indexed list.
 * post: The element stored now at rear is newElt
 */

public void add(E elt) {
    add(size(),elt);
}
/**
 * @param r the index of the element to be removed
 * @return the element at index r
 * post: Removes the element at index r, shifts all elements on its right one to left
 */

public E remove(int r) {
    checkIndex(r, size());
    E temp = elts[r];
    for(int i = r; i < eltsFilled - 1; i++)
        elts[i] = elts[i+1] //shift elements to the right one to the left
    eltsFilled--;
    return temp;
}