Lists -- How to Build Your Own

Atul Prakash

Reference: Look at ArrayList.java implementation in Java

Source: http://www.docjar.com/html/api/java/util/ArrayList.java.html

Class MyList

- Implement a list using arrays
- Methods:
  - Constructors: default capacity and given capacity
  - add(Object element)
  - Object get(int index)
  - remove element:
    - last
    - first
    - at a given position
  - size(): returns size

Some tests

```
import junit.framework.*;

public class MyListTest extends TestCase {
    public void testCount() {
        MyList l = new MyList();
        assertEquals(0, l.size());
        l.add("abc");
        l.add("def");
        l.add(3);
        assertEquals(3, l.size());
    }

    public void testInserts() {
        MyList l = new MyList();
        l.add("abc");
        l.add("def");
        l.add(3);
        assertEquals(3, l.get(2));
        assertEquals(3, l.size());
        // String s = l.findElement(0); // cast needed.
        // assertEquals("abc", s);
    }
}
```

List State Variables

```
public class MyList { // list of integers
    Object[] data; // list itself. null values at the end
    int capacity;  // maximum capacity of the list
    int num;  // current size of the list
    static final int DEFAULT_CAPACITY = 100;
```

Invariants:

num = # of elements in the list.

0 <= num <= capacity.

data array: first num slots filled, followed by nulls

Constructors

```
public class MyList { // list of integers
    Object[] data; // list itself. null values at the end
    int capacity;  // maximum capacity of the list
    int num;  // current size of the list
    static final int DEFAULT_CAPACITY = 100;

    public MyList() {
        this(DEFAULT_CAPACITY); // call MyList(capacity).
    }

    public MyList(int capacity) {
        this.capacity = capacity;
        data = new Object[capacity]; // null array
        num = 0;
    }
```

Note that array is filled with null values, size is 0. Invariants hold.

adding elements

```
public class MyList { // list of integers
     Object[] data; // list itself. null values at the end
     int capacity;  // maximum capacity of the list
     int num;  // current size of the list
     static final int DEFAULT_CAPACITY = 100;

    public void add(Object a) {
        // add an object a to the end of the list
            data[num] = a;
            num++;
    }
```

Check if num is 0, that this works correctly.

Check also if num == capacity.

Boundary checks

```
public class MyList { // list of integers
     Object[] data; // list itself. null values at the end
     int capacity;  // maximum capacity of the list
     int num;  // current size of the list
     static final int DEFAULT_CAPACITY = 100;

    public void add(Object a) throws CapcityExceeded {
        if (num == capacity) {
            throw new CapacityExceeded("list capacity exceeded");
        }
        data[num] = a;
        num++;
    }
```

Time required: O(1) (constant time)

Retrieving an Element

```
public Object get(int index) {
     // find the element at given index
    if (index < 0 || index >= num) {
       throw RuntimeException("index out of bounds");
    }
     return data[index];
}
```

Deleting Last Element

```
public void deleteLastElement() {
     // delete the last element from the list
     // fill in the code in the class.
    if (num == 0) {
       throw new RuntimeException("list is empty: cannot delete");
    }
    num--;
    data[num] = null;


}
```

Deleting First Element

```
public void deleteFirstElement() {
    // delete first element from the list
    for (int i = 0; i < num-1; i++) {
        data[i] = data[i+1];
    }
    data[num-1] = null;
    num--; // IMPORTANT. Re-establish invariant
}
```

Shift elements to the left. Can be extended to delete an element in the middle

Incorrect shifting

```
public void deleteFirstElement() {
    // delete first element from the list
    for (int i = num-1; i > 0; i--) {
        data[i-1] = data[i];
    }
    num--;
}
```

Some limitations of our lists

- They have a capacity. What if we did not know the capacity?
- Could use additional functionality:
  - updating an element
  - searching for an element
  - inserting an element at a given index

Overcoming Capacity Limitations

- One common strategy:
  - increase the size of the array if it runs out of space
  - But arrays cannot change in size
  - How do you increase the size of the array?

Making a larger array

- Simply create a new, larger array
- Copy data from old array to the new array
- Since the variable data is a reference, it can be made to point to the new array.

Modified add

```
public void add(Object a) {
    if (num == capacity) {
        Object[] datanew = new Object[1+num];
        // copy old data to new data
        for (int i = 0; i < num; i++) {
            datanew[i] = data[i];
        }
        data = datanew; // data now refers to he new array
        capacity = capacity + 1;
    } // done extending the array if necessary.
     // now add the element
    data[num] = a;
    num++;
}
```

Some Inefficiencies

- Every time we exceed the capacity, the array is copied.
- For example, if capacity is 10,000, and we add 3 elements:
  - 10,000*3 elements will be copied (approx).
- Can we make copying less frequent?

Capacity-doubling Strategy

```
public void add(Object a) {
    if (num == capacity) {
        Object[] datanew = new Object[capacity*2];
        capacity = capacity*2;
        // copy old data to new data
        for (int i = 0; i < num; i++) {
            datanew[i] = data[i];
        }
        data = datanew; // data now refers to he new array
    }
    data[num] = a;
    num++;
}
```

More Efficient

- If the capacity is 10,000, adding 3 elements on a full list only causes the array to be copied once.
- Next copy will occur after 10,000 additions.
- Overall, copying becomes less frequent.

Variations of Lists

- Sets (a set of unique elements):
  - Don't allow duplicate objects in the list (modify add to check for duplicates).
- Queues:
  - Only allow deletion from the front
- Stacks:
  - Only allow deletion from the tail