CS 310: ArrayList Implementation

Chris Kauffman

Week 2-2
Logistics

At Home

- Read Weiss Ch 5: Big-O
- Read Weiss Ch 15: ArrayList implementation

Reminder to DrJava Users

- Consider Using GMU Edition of DrJava
- Download here: https://cs.gmu.edu/~kauffman/drjava/

Goals

- Build an array list
- Analyze its complexity
Java has a nice library of containers, Collections framework

- Interfaces that provide `get()`, `set()`, `add()`
- All have parameterized types: `ArrayList<E>`, `TreeSet<E>`

At present, most interested in `ArrayList`

- Like arrays but lacking nice `[ ]` syntax
- Use `get()` and `set()` instead
- Can `add()` elements at end (high index)

Demonstrate `ArrayList` in DrJava
Basic Premise of the Expandable Array

Use an underlying array

```java
public class MyArrayList<T>{
    // This almost works
    T data[];
    ...}

▶ data is a standard fixed size array
▶ get()/set() are array ops
```

Adding and Expanding

- Add elements into `data`
- If/when `data` runs out of space
  1. Allocate a new larger array `data2`
  2. Copy elements from `data` to `data2`
  3. Add new element(s) to `data2`
  4. Set `data` to `data2`
  5. Original array gets garbage collected

Questions

- What’s the notion of `size` now?
- How much should the array grow on expansion?
- Is there wasted space? How much?
Create MyArrayList

```java
public class MyArrayList<T>{
    T data[]; int size; // Holds elements, virtual size
    public MyArrayList(); // Initialize fields
    public int size(); // Virtual size of AL
    public void add(T x); // Add an element to the end
    public T get(int i); // Retrieve element i
    public void set(int i, T x); // Replace element i with x
    public void insert(int i, T x); // Insert x at position i, shift
    // elements if necessary
    public void remove(int i); // Remove element at position i, 
    // shift elements to remove gap
}
```

`add(x)`
If/when data runs out of space

1. Allocate a new larger array `data2`
2. Copy from `data` to `data2`
3. Add new element(s) to `data2`
4. Set `data` to `data2`
5. GC gets the old array

Respect My `size()`
get()/set()/insert()/remove() must respect size() which is always smaller than or equal to data.length;
check for out of bounds access
Examine Results

- Code up versions together quickly
- Simple version: MyArrayList.java in code distrib
- Also included java.util.ArrayList from Java 1.7 source
- May also want to look at Weiss’s version in textbook-source/weiss/util/ArrayList.java

Complexity

What are the complexities for methods like
- set(i, x) and get(i)
- insert(i, x) and remove(i, x)
- add(x): this is the big one
Limits of Types

Unfortunately, java type system has some limits.

new T[10] Not Allowed

public class MyArrayList<T> {
    private T [] data;
    public MyArrayList(){
        this.data=new T[10]; // Grrrr
    }
    public T get(int i){
        this.rangeCheck(i);
        return this.data[i];
    }
}

Instead: Object[] + Caste

public class MyArrayList<T> {
    private T data[];
    public MyArrayList(){
        this.data=(T[]) new Object[10];
    }
    public T get(int i){
        this.rangeCheck(i);
        return this.data[i];
    }
}
Unsafe Operations in MyArrayList

lila % javac MyArrayList.java
Note: MyArrayList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

lila % javac -Xlint:unchecked MyArrayList.java
MyArrayList.java:77: warning: [unchecked] unchecked cast
found    : java.lang.Object[]
required: T[]
    this.data = (T[]) new Object[10];
^
1 warning
Unsafe Operations

Suppress Warnings
Offending code is

```java
private T [] data;

public MyArrayList(){
    this.data=(T[]) new Object[10];
}
```

▶ It is unsafe, but so is fire.
▶ Tell the compiler to shut up

```java
// I know what I'm doing
@SuppressWarnings("unchecked")
public MyArrayList(){
    this.data=(T[]) new Object[10];
}
```

Alternative:
This version uses casting in get()

```java
public class MyArrayList<T> {
    Object [] data;
    public MyArrayList(){
        this.data =new Object[10];
    }
    @SuppressWarnings("unchecked")
    public T get(int i){
        this.rangeCheck(i);
        return (T) this.data[i];
    }
}
```

Also needed anywhere else type T stuff is returned so less preferred: more @SuppressWarnings
HW and Unsafe Operations

- Proper use of generics creates good compile-time type checking
- **Rarely** is casting required; ArrayList implementation is one such exception
- **HW1** is NOT such a case
  - Should not need to caste anything
  - Should not need to use `@SuppressWarnings`
  - Doing either may result in penalties
Warmup: Finish methods for MyArrayList

```java
public class MyArrayList<T>{
    T data[]; int size; // Holds elements, virtual size
    public MyArrayList(); // Initialize fields
    public int size(); // Virtual size of AL
    public T get(int i); // Retrieve element i
    public void add(T x); // Add an element to the end

    // FINISH THESE
    public void set(int i, T x); // Replace element i with x
    public void insert(int i, T x); // Insert x at position i, shift elements if necessary
    public void remove(int i); // Remove element at position i, shift elements to remove gap
}
```

- Three methods of MyArrayList remain - finish them
- Note common patterns that should be factored into helpers (e.g. expansion, bounds checking)
- Note: al.insert(i,x) is called al.add(i,x) in java.util.ArrayList
Exercise: **ArrayList Complexities**

- ArrayList of with $N$ elements
- Time/Space Complexities of methods
- Worst-case or Average/Amoritzied

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method</th>
<th>Worst Runtime</th>
<th>Average Runtime</th>
<th>Worst Space</th>
<th>Average Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size()</td>
<td>al.size()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get(i)</td>
<td>al.get(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set(i,x)</td>
<td>al.set(i,x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add(x)</td>
<td>al.add(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert(i,x)</td>
<td>al.add(i,x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove(i)</td>
<td>al.remove(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What is the space complexity of an ArrayList with $N$ elements?
- Is that a tight bound?
Expanding with Magic Numbers

- Size increase when expansion is required is interesting
- Can’t be constant: increase size by 1, or 2, or 10 will not give good complexity
- Standard Java ArrayList increases to $3/2 \times \text{oldSize} + 1$
- Chosen based on engineering experience rather than theory, can use bit shifts to compute it fast
- Default ArrayList size is 10
- **Magic Numbers**: $3/2$ and 10, magic because there is no good reason for them
Average/Amortized Complexity

- Worst case complexity for `arrayList.add(x)` is $O(N)$ when expansion is required
- But expansion happens rarely if size increase by 150% during expansion
- Over many add operations, the average `add(x)` takes $O(1)$ time complexity
- **Amortized Analysis**: sort of like average case (definition is close enough for this class)