Lecture 9: Array Lists and Assertions

CS 62
Spring 2019
William Devanny & Alexandra Papoutsaki
public static boolean greaterThanOne(int x) {
    if (x > 1) {
        return true;
    } else {
        return false;
    }
}
public static boolean greaterThanOne(int x) {
    return x > 1;
}

• The if condition is a boolean value!

• When possible simply return the boolean.
Adding to ArrayList

• Suppose there are $n$ elements in ArrayList and you want to add one more. What is the cost of this operation?

• If enough space (size<capacity):
  • Add to end is $O(1)$
  • Add to beginning is $O(n)$

• If not space:
  • What is the cost of ensureCapacity?
  • $O(n)$ because $n$ elements in array
Amortized Time Analysis

As the ArrayList increases in size, the doubling happens half as often but costs twice as much.

Insertion Cost

Copying Cost

Total Cost
Amortized Time Analysis

We will use the aggregate method – the simplest method for amortized time analysis

Others: accounting (banker’s) and potential (physicist’s).

Think of it as an average. For a total of $n$ operations: $O(\text{total cost}) / \text{(number of operations)}$ (in this case additions)

$O(\text{total cost of operations}) = \sum \text{cost of insertions} + \sum \text{cost of copying}$

Total cost of insertions: $n$ of them, each $O(1)$ cost, therefore $nO(1) = O(n)$

Total cost of copying during the $n$ insertions: $1 + 2 + 2^2 + \cdots + 2^{\log n} \leq 2n$ which is $O(n)$

$O(\text{total cost}) = O(n) + O(n) = O(n)$

Amortized time = $O(n)/n = O(1)$ but “lumpy”
EnsureCapacity

• What if we only increase the capacity by 1 element each time?
  • Adding $n$ elements one at a time to end
    • Total cost of $n$ insertions: $1 + 2 + 3 + \cdots + (n - 1) = n(n - 1)/2$
    • Total cost of $O(n^2)$
  • Average cost of each is $O(n)$
ArrayList Operations

- **Best case:**
  - $O(1)$: `size()`, `isEmpty()`, `get(int i)`, `set(int i, E elet)`, `remove()`, `add()`

- **Worst case:**
  - $O(1)$: `size()`, `isEmpty()`, `get(int i)`, `set(int i, E elet)`
  - $O(n)$: `remove`, `add()`

- `add()` runs in amortized constant time: adding $n$ elements requires $O(n)$ time.
Won’t use `Assert` class from Bailey.

Command to check assertions in standard Java

Two forms:

```
assert boolExp
assert boolExp : message
```

Article on when to use `assert`:
[https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html](https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html)
Assertions help...

- Defensive programming
- Little cost to executing assertions ... and can turn off checking
- Extremely useful in debugging in tracking down what is going wrong - can be better than inserting println’s.
- Also useful in checking cases that should not occur
  - e.g., defaults in switch, other control paths not taken.
- Do NOT use argument checking in public methods
  - Should throw an exception instead
- Do NOT use to perform action that is critical for the program
  - Instead perform the action before the assertion and then assert that the action succeeded
Turning on assert

Turn on assertions when run program, by adding `-ea` as virtual machine argument in arguments tab in Eclipse when set up runtime configuration.

If you leave it off, then it ignores assert statements.

If on and the assertion is false, then it will raise an `AssertionError` exception and will print associated message

They should not be caught as represents a program error