CSCI 1103: Array-based Data Structures

Chris Kauffman

Last Updated:
Mon Nov 13 11:15:14 CST 2017
Logistics

<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture</th>
<th>Outside</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon 11/13</td>
<td>Expandable Arrays</td>
<td>Lab 10 on Stacks</td>
</tr>
<tr>
<td>Wed 11/15</td>
<td>Stacks/Queues</td>
<td>P4 Due</td>
</tr>
<tr>
<td>Fri 11/17</td>
<td>Queues</td>
<td></td>
</tr>
<tr>
<td>Mon 11/20</td>
<td>Review</td>
<td>Lab 10 Due, Review</td>
</tr>
<tr>
<td>Wed 11/22</td>
<td>Exam 2</td>
<td></td>
</tr>
</tbody>
</table>

Reading from Eck

- Ch 5 on Objects/Classes
- Ch 8.3.3 on Throwing exceptions
- Ch 7.4 on ArrayList
- Ch 9.3 on Stacks and Queues

Lab10: Stack Data Structure

- Define a new class for Stacks of Strings
- Two types

Project 4

- Due Wednesday
- Questions?
Exceptions for Errors

- Java’s mechanism for indicating errors is to **throw exceptions**
- There are a wide variety of exception kinds available
- Can also create your own: they are a class
- For simple situations, `RuntimeException` suffices
- Construct one with a `String` error message indicating problem
  ```java
  RuntimeException e = new RuntimeException("Ya done messed up.");
  ```
- Raise the exception with the keyword `throw`
  ```java
  throw e;
  ```
- Frequently do this in one-liners
  ```java
  throw new RuntimeException("Ya done messed up.");
  ```
Exceptions share return semantics

- Uncaught throw statements immediately exit a method, similar to return
- Control flows up and out, usually crashes program

```java
// Divide num by denom and return the quotient.
// Raise a RuntimeException if denom is 0.
public static int divide(int num, int denom) {
    if (denom == 0) {
        throw new RuntimeException("Divide by 0"); // error: immediately
    }

    int quotient = num / denom;
    return quotient; // immediately return
}
```
Additional Info on Exceptions

- We will work with RuntimeExceptions as they are simple sufficient
- Exceptions are a complex topic, include
  - try/catch blocks to recover form exceptions
  - method signatures with throws
  - inheritance of exception types
- We will revisit some of these topics later when discussing File Input/Output as many methods in I/O involve exception handling
Basic Data Structures

▶ Information frequently comes/goes in patterns
▶ To make life easier for programmers and utilize the machine more efficiently, data structures provide a way to organize data for easy use
▶ The purpose of a creating data structure is to make programming another task easier
▶ We will discuss some simple data structures
  ▶ Expandable Arrays (today)
  ▶ Stacks built on arrays (lab 10)
  ▶ Queues built on arrays (later in week)
▶ Textbook discusses some alternatives
  ▶ Linked lists
  ▶ Stacks built from linked nodes
  ▶ Queues built for linked nodes
▶ You will likely study these in later CS courses
Expandable Arrays

Standard Array

► Recall Java’s standard arrays

1. Length is fixed at creation
2. Initially filled with zeroey elements (0 or null or similar)
3. Random access based on index number using square braces: arr[i]
4. Cannot grow

► Inability to grow is a drag as one frequently wants to add without knowing limit

► The goal of an expandable array or ArrayList is to making adding possible

Expandable array

► Independent class created by us

1. Length is NOT fixed
2. Initially empty: size 0
3. Random access based on index number using methods: a.get(i) and a.set(i,x)

► No magic: a field of the expandable array will be a standard array

► When standard array fills up, make a bigger one, copy over elements
First pass: FixedList doesn’t grow

Create/Initial Add

Welcome to DrJava.
> FixedList f = new FixedList(3);
> f.toString()
[]
> f.size()
0
> f.get(2)
java.lang.RuntimeException:
out of bounds
at FixedList.get(FixedList.java:22)
> f.add("A")
> f.size()
1
> f.toString()
[A]
> f.get(0)
A
> f.get(1)
java.lang.RuntimeException:
out of bounds
at FixedList.get(FixedList.java:22)

Further Adds/Set

> f.add("B")
> f.toString()
[A, B]
> f.get(1)
B
> f.size()
2
> f.add("C")
> f.toString()
[A, B, C]
> f.get(2)
C
> f.set(1,"X")
> f.toString()
[A, X, C]
> f.add("D")
java.lang.RuntimeException:
list array is full
at FixedList.add(FixedList.java:40)
Exercise: Accessor/Mutators Methods

Define `size()`

```java
class FixedList{
    // number of elements
    private int size;
    // that have been added
    // contents of the array
    private String[] data;

    // Create the array backing the fixed list
    public FixedList(int maxSize){
        this.size = 0;
        this.data = new String[maxSize];
    }

    // Return how many elements are in the list
    public int size(){
        // YOUR CODE HERE
    }
}
```

Define `set()`

```java
// Return element i of the list. Check that the index is in bounds (greater than or equal to 0 and less than the list size)
public String get(int i){
    if(i < 0 || i >= this.size){
        // out of bounds
        String msg = "out of bounds";
        throw new RuntimeException(msg);
    }
    return this.data[i];
}

// Change element i of the list. Check that the index is in bounds (greater than or equal to 0 and less than the list size)
public void set(int i, String x){
    // YOUR CODE HERE
}
```
Exercise: add() Method

Define add(x) method that allows

> f.toString()

[]
> f.add("A")
> f.add("B")
> f.toString()

[A, B]
> f.size()

2

public class FixedList{
    // number of elements that have been added
    private int size;
    // contents of the array
    private String[] data;

    // Add the given string to the list at the end. If there is not
    // sufficient space for the addition, throw an exception
    public void add(String x){
        // YOUR CODE HERE to:
        // Check for space in array, throw exception if none
        // Put x in array
        // Increment size
    }
}
ExpandableList: changes

A modification to `add(x)` allows as many additions as memory supports.

- **Draw pictures** to demonstrate how `add(x)` works
- **How much** does the array size increase during expansion?

```java
// A class wrapper for a list of Strings. This version grows the underlying array when needed.
public class ExpandableList{
    private int size; // number of elements that have been added
    private String[] data; // contents of the array

    // Add the given string to the list at the end. If there is not sufficient space for the addition, expand the underlying array to accommodate it.
    public void add(String x){
        if(this.size >= this.data.length){ // check for space
            String newData[] = new String[this.data.length*2]; // new larger array
            for(int i=0; i<this.data.length; i++){ // copy old elements
                newData[i] = this.data[i];
            }
            this.data = newData; // point at new array
        }
        this.data[this.size] = x; // add on element
        this.size++; // increase size
    }
}
```