Building Java Programs

Chapter 10
Lecture 10-1: ArrayList

reading: 10.1
Welcome to CSE 143!

I'm Hélène Martin

http://cs.washington.edu/143
CSE 143

- Goal: learn tools for automating complex tasks efficiently
  - Abstraction (client vs. implementation)
  - Data structures
  - Algorithms
- Prerequisite: can automate basic tasks using a programming language (logic, control flow, decomposition)
- For EVERYONE, not just CSE majors
- Learn by doing
- Lots of support (undergraduate TAs, IPL, message board)
Programming

- CS: "efficiently implementing automated abstractions" ¹

- Building things is empowering
  - Small number of fundamentals can solve lots of problems
  - When a program works, it's obvious
  - Welding, chain saws, safety glasses not required

- A LOT of complexity to master: exciting and scary

- Java is our tool in 14x but lessons transfer broadly

¹ http://www.pgbovine.net/what-is-computer-science.htm
Being Successful

- Determination, hard work, focus
- Investing time (~15 hours a week)
  - Starting early
  - Developing problem-solving strategies
  - Developing a consistent style
- Knowing when to ask for help
  - Go to the IPL
  - Talk to me after class, during office hours
- Studying together
  - Homework is individual but studying in groups pays off
Logistics

• Get to know http://cs.washington.edu/143

• 2 sections a week
  • Turn in ONE set of problems each week for credit

• Grading described on syllabus
  • 50% projects, 20% midterm, 30% final (exams open book)

• Weekly programming projects
  • Academic honesty is serious
  • 40 point scale
  • 5 "free late days"; -2 for subsequent days late
Words exercise

- Write code to read a file and display its words in reverse order.

- A solution that uses an array:

```java
String[] allWords = new String[1000];
int wordCount = 0;

Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords[wordCount] = word;
    wordCount++;
}
```

- What's wrong with this?
Recall: Arrays (7.1)

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: 0-based integer to access an element from an array.
  - **length**: Number of elements in the array.

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>12</td>
<td>49</td>
<td>-2</td>
<td>26</td>
<td>5</td>
<td>17</td>
<td>-6</td>
<td>84</td>
<td>72</td>
<td>3</td>
</tr>
</tbody>
</table>

- element 0
- element 4
- element 9

length = 10
Array Limitations

- Fixed-size

- Adding or removing from middle is hard

- Not much built-in functionality (need Arrays class)
List Abstraction

- Like an array that resizes to fit its contents.
- When a list is created, it is initially empty.
  
  ```
  []
  ```

- Use `add` methods to add to different locations in list
  
  ```
  [hello, ABC, goodbye, okay]
  ```

- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
- You can add, remove, get, set, ... any index at any time.
Collections and lists

• **collection**: an object that stores data ("elements")
  
  ```java
  import java.util.*;
  // to use Java's collections
  ```

• **list**: a collection of elements with 0-based **indexes**
  - elements can be added to the front, back, or elsewhere
  - a list has a **size** (number of elements that have been added)
  - in Java, a list can be represented as an **ArrayList** object
Type parameters (generics)

ArrayList<Type> name = new ArrayList<Type>();

- When constructing an ArrayList, you must specify the type of its elements in < >
  - This is called a type parameter; ArrayList is a generic class.
  - Allows the ArrayList class to store lists of different types.
  - Arrays use a similar idea with Type[]

ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
**ArrayList methods (10.1)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(value)</code></td>
<td>appends value at end of list</td>
</tr>
<tr>
<td><code>add(index, value)</code></td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td><code>indexOf(value)</code></td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td><code>get(index)</code></td>
<td>returns the value at given index</td>
</tr>
<tr>
<td><code>remove(index)</code></td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td><code>set(index, value)</code></td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td><code>size()</code></td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td><code>toString()</code></td>
<td>returns a string representation of the list such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
</tbody>
</table>

* (a partial list; see 10.1 for other methods)
ArrayList vs. array

String[] names = new String[5]; // construct
names[0] = "Jessica"; // store
String s = names[0]; // retrieve
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
} // iterate

ArrayList<String> list = new ArrayList<String>();
list.add("Jessica"); // store
String s = list.get(0); // retrieve
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
} // iterate
ArrayList as param/return

public static void name(ArrayList<Type> name) { // param
public static ArrayList<Type> name(params)    // return

• Example:

   // Returns count of plural words in the given list.
   public static int countPlural(ArrayList<String> list) {
       int count = 0;
       for (int i = 0; i < list.size(); i++) {
           String str = list.get(i);
           if (str.endsWith("s")) {
               count++;
           }
       }
       return count;
   }
Words exercise, revisited

- Write a program that reads a file and displays the words of that file as a list.
  - Then display the words in reverse order.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them with all plural words removed.
Exercise solution (partial)

```java
ArrayList<String> allWords = new ArrayList<String>();
Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}

// display in reverse order
for (int i = allWords.size() - 1; i >= 0; i--) {
    System.out.println(allWords.get(i));
}

// remove all plural words
for (int i = 0; i < allWords.size(); i++) {
    String word = allWords.get(i);
    if (word.endsWith("s")) {
        allWords.remove(i);
        i--;
    }
}
```