ArrayLists

Chapter 12.1 in Savitch

Using arrays to store data

- Arrays: store multiple values of the same type.
- Conveniently refer to items by their index.
- Need to know the size before declaring them:
  ```java
  int[] numbers = new int[100];
  ```
- We often need to store an unknown number of values.
  - Need to either count the values or resize as additional storage space is needed.

Lists

- list: a collection storing an ordered sequence of elements, each accessible by a 0-based index.
  - A list has a size (number of elements that have been added).
  - Elements can be added at any position.

Exercise

- Let's write a class called `ArrayIntList` that implements a list using `int[]`.
  - Behavior:
    - `add(value)`, `add(index, value)`
    - `get(index)`, `set(index, value)`
    - `size()`
    - `remove(index)`
    - `indexOf(value)`
  - The list's size will be the number of elements added to it so far.

Using `ArrayIntList`

- Construction
  ```java
  int[] numbers = new int[5];
  ArrayIntList list = new ArrayIntList();
  ```
- Storing a value, retrieving a value
  ```java
  list.add(42);
  int n = list.get(0);
  ```
- Searching for the value 27
  ```java
  for (int i = 0; i < numbers.length; i++) {
      if (numbers[i] == 27) { ... }
  }
  if (list.indexOf(27) >= 0) { ... }
  ```

Pros/cons of `ArrayIntList`

- Pro (benefits)
  - Simple syntax.
  - Don't have to keep track of array size and capacity.
  - Has powerful methods (`indexOf`, `add`, `remove`, `toString`).

- Con (drawbacks)
  - `ArrayIntList` only works for `ints` (arrays can be any type).
  - Need to learn
Java Collections and ArrayLists

- Java includes a large set of powerful collections classes.
- The most basic, ArrayList, is can store any type of Object.
- All collections are in the `java.util` package.

```java
import java.util.ArrayList;
```

Type Parameters (Generics)

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

- When constructing an ArrayList, you can specify the type of elements it will contain between `< and >.
  - We say that the ArrayList class accepts a type parameter, or that it is a generic class.

```java
ArrayList<String> names = new ArrayList<String>();
names.add("Asa");
names.add("Nathan");
```

ArrayList methods

<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 at given index, shifting subsequent values 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 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>

ArrayList methods 2

```java
addAll(list) adds all elements from the given list at the end of this list.
addAll(index, list) inserts the list at the given index of this list.
contains(value) returns true if given value is found somewhere in this list.
containsAll(list) returns true if this list contains every element from given list.
equals(list) returns true if given other list contains the same elements.
remove(value) finds and removes the given value from this list.
removeAll(list) removes any elements found in the given list from this list.
retainAll(list) removes any elements not found in given list from this list.
subList(from, to) returns the sub-portion of the list between indexes from (inclusive) and to (exclusive).
toArray() returns an array of the elements in this list.
```

Iterating through an array list

- Suppose we want to look for a value in an ArrayList of Strings.
  ```java
  for (int i = 0; i < list.size(); i++) {
    if(value.equals(list.get(i))){
      //do something
    }
  }
  ```

- Alternative:
  ```java
  for (String s : list) {
    if(value.equals(s)){
      //do something
    }
  }
  ```

Note on generics in Java 7

In version 7 of Java, rather than doing:

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

You can save a few keystrokes:

```java
ArrayList<Type> name = new ArrayList<>();
```
Learning about classes

- The Java API Specification is a huge web page containing documentation about every Java class and its methods.
  - The link to the API Specs is on the course web site.

Modifying while looping

- Consider the following flawed pseudocode for removing elements that end with s from a list:

  ```java
  removeEndS(list) {
    for (int i = 0; i < list.size(); i++) {
      get element i;
      if it ends with an 's', remove it.
    }
  }
  ```

  - What does the algorithm do wrong?

ArrayList of primitives?

- The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.
  - The following is illegal:
    ```java
    // illegal -- int cannot be a type parameter
    ArrayList<Integer> list = new ArrayList<Integer>();
    ```
  - But we can still use ArrayList with primitive types by using special classes called wrapper classes in their place.
    ```java
    ArrayList<Integer> list = new ArrayList<Integer>();
    ```

Wrapper classes

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

- A wrapper is an object whose purpose is to hold a primitive value and to provide more functionality.
- Once you construct the list, use it with primitives as normal (autoboxing):
  ```java
  ArrayList<Double> grades = new ArrayList<Double>();
  grades.add(3.2);
  grades.add(2.7);
  ```

Looking ahead: Interfaces

- A Java interface specifies which public methods are available to a user.
- A class implements an interface if it provides all the methods in the interface.
- Interfaces allow for a common behavior amongst classes, eg the Collection interface is implemented by many classes (LinkedList, ArrayList...).
Java Collections

- ArrayList belongs to Java's collections framework.
- Other classes have a very similar interface, so it will be easier to learn how to use those classes once you've learned ArrayLists.