ArrayLists

Savitch Ch. 12.1

Using arrays to store data

```
int[] nums = new int[100];
int size = 0;
```

- We often need to store an unknown number of values.
  - Arrays can be used for this, but we must count the values.
  - Only the values at indexes [0, size - 1] are relevant.
  - Need to resize the array to store additional values

<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>...</th>
<th>98</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>17</td>
<td>93</td>
<td>208</td>
<td>5</td>
<td>-3</td>
<td>100</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>size</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Other possible operations

```
public static void add(int[] list, int size, int value, int index)
public static void remove(int[] list, int size, int index)
public static void find(int[] list, int size, int value)
public static void print(int[] list, int size)
```

- We could implement these operations as methods that accept a list array and its size along with other parameters.
  - But since the behavior and data are so closely related, it makes more sense to put them together into an object.
  - A list object can store an array of elements and a size, and can have methods for manipulating the list of elements.
    - Promotes abstraction (hides details of how the list works)

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 to the back, or elsewhere

Exercise

- Let’s write a class that implements a list using an int[]
  - We’ll call it ArrayIntList
  - 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

```
int[] numbers = new int[5];
ArrayIntList list = new ArrayIntList();
```

- storing a value
  - retrieving a value
    - numbers[0] = 42;
    - int n = numbers[0];
    - list.add(42);
    - int n = list.get(0);

- searching for the value 27
  - 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)
  - syntax is different to learn and use

Implementing add

- Add to end of list is easy; just store element and increase size

```java
public void add(int value) {
  list[size] = value;
  size++;
}
```

- Example:
  ```java
  list.add(42);
  ```

Implementing add (2)

- Adding to the middle or front is hard
  - must shift nearby elements to make room for the new value

```java
public void add(int index, int value) {
  for (int i = size; i > index; i--) {
    list[i] = list[i - 1];
  }
  list[index] = value;
}
```

- Example:
  ```java
  list.add(3, 42);
  ```

Implementing remove

```java
public void remove(int index) {
  for (int i = index; i < size; i++) {
    list[i] = list[i + 1];
  }
  size--;
  list[size] = 0;  // optional (why?)
}
```

- Example:
  ```java
  list.remove(2);
  ```

Preconditions

- What happens if the client tries to access an element that is past the size but within the bounds of the array?
  - Example: list.get(11); on a list of 5 elements, capacity 100
  - We have not addressed this case yet, and currently we just choose to assume that the user will not do such a thing.

- precondition: Something your method assumes is true at the start of its execution.

  // Returns the element at the given index.
  // Precondition: 0 <= index < size
  public void get(int index) {
    return list[index];
  }
Better solution using exceptions

throw new ExceptionType();
throw new ExceptionType("message");

- Causes the program to immediately crash with an exception.
  - Why might this be a good thing?

Common types of exceptions:
- ArithmeticException, ArrayIndexOutOfBoundsException, FileNot FoundException, IllegalStateException, IOException, NullPointerException, RuntimeException, UnsupportedOperationException

```java
public void get(int index) {
    if (index >= size) {
        throw new ArrayIndexOutOfBoundsException();
    }
    return list[index];
}
```

Java Collections Framework

- Java includes a large set of powerful collection classes.
- The most basic, ArrayList, is essentially the same as our ArrayIntList but can store any type of value.
- All collections are in the java.util package.

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

Type Parameters (Generics)

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

- When constructing an ArrayList, you must 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>add(value)</td>
<td>appends value at end of list</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>inserts given value at given index, shifting subsequent values right</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td>indexOf(value)</td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>get(index)</td>
<td>returns the value at given index</td>
</tr>
<tr>
<td>remove(index)</td>
<td>removes/returns value at given index, shifting subsequent values left</td>
</tr>
<tr>
<td>set(index, value)</td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>toString()</td>
<td>returns a string representation of the list such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
<tr>
<td>addAll(list)</td>
<td>adds all elements from the given list to this list at the end of the list, or inserts them at the given index</td>
</tr>
<tr>
<td>addAll(index, list)</td>
<td>inserts given list to this list at the given index</td>
</tr>
<tr>
<td>contains(value)</td>
<td>returns true if given value is found somewhere in this list</td>
</tr>
<tr>
<td>containsAll(list)</td>
<td>returns true if this list contains every element from given list</td>
</tr>
<tr>
<td>equals(list)</td>
<td>returns true if given other list contains the same elements</td>
</tr>
<tr>
<td>remove(value)</td>
<td>finds and removes the given value from this list</td>
</tr>
<tr>
<td>removeAll(list)</td>
<td>removes any elements found in the given list from this list</td>
</tr>
<tr>
<td>retainAll(list)</td>
<td>removes any elements not found in given list from this list</td>
</tr>
<tr>
<td>subList(from, to)</td>
<td>returns the sub-portion of the list between indexes from (exclusive) and to (inclusive)</td>
</tr>
<tr>
<td>toArray()</td>
<td>returns an array of the elements in this list</td>
</tr>
</tbody>
</table>

ArrayList methods 2

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 algorithm to remove plural elements from a list:

```
removePlurals(list) {
    for (int i = 0; i < list.size(); i++) {
        if element i is plural, remove it.
    }
}
```

What does the algorithm do wrong?

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>&quot;she&quot;</td>
<td>&quot;sells&quot;</td>
<td>&quot;seashells&quot;</td>
<td>&quot;by&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;seashore&quot;</td>
</tr>
<tr>
<td>size</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:
  ```
  // illegal -- int cannot be a type parameter
  ArrayList<int> list = new ArrayList<int>();
  ```
- But we can still use `ArrayList` with primitive types by using special classes called `wrapper` classes in their place.
  ```
  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 sole purpose is to hold a primitive value.
- Once you construct the list, use it with primitives as normal:
  ```
  ArrayList<Double> grades = new ArrayList<Double>();
  grades.add(3.2);
  grades.add(2.7);
  ...
  ```