ArrayLists

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

ArrayIntList

- Let's consider the methods of a class called ArrayIntList that represents 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
ArrayIntList

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

- storing a given value: retrieving a value
  ```java
  numbers[0] = 42;
  int val = numbers[0];
  list.add(42);
  int val = list.get(0);
  ```

- searching for a given value
  ```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 how to use the class
Java Collections and ArrayLists

- Java includes a large set of powerful classes that provide functionality for storing and accessing collections of objects.
- The most basic, `ArrayList`, 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addAll(list)</code></td>
<td>adds all elements from the given list at the end of this list</td>
</tr>
<tr>
<td><code>addAll(index, list)</code></td>
<td>inserts the list at the given index of this list</td>
</tr>
<tr>
<td><code>contains(value)</code></td>
<td>returns true if given value is found somewhere in this list</td>
</tr>
<tr>
<td><code>containsAll(list)</code></td>
<td>returns true if this list contains every element from given list</td>
</tr>
<tr>
<td><code>equals(list)</code></td>
<td>returns true if given other list contains the same elements</td>
</tr>
<tr>
<td><code>remove(value)</code></td>
<td>finds and removes the given value from this list</td>
</tr>
<tr>
<td><code>removeAll(list)</code></td>
<td>removes any elements found in the given list from this list</td>
</tr>
<tr>
<td><code>retainAll(list)</code></td>
<td>removes any elements <em>not</em> found in given list from this list</td>
</tr>
<tr>
<td><code>subList(from, to)</code></td>
<td>returns the sub-portion of the list between indexes <em>from</em> (inclusive) and <em>to</em> (exclusive)</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>returns an array of the elements in this list</td>
</tr>
</tbody>
</table>
Learning about classes

- The Java API specification website contains detailed documentation of every Java class and its methods.

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 and above

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<>();
```

---

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?

<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:
    ```java
    // 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.
  ```java
  ArrayList<Integer> list = new ArrayList<Integer>();
  ```

Wrapper classes: Example

- Every java primitive has a class dedicated to it.

Example:

```java
int x = 3;
Integer y = new Integer(5);

int z = x + y;
int z = x + y.intValue();  // convert wrapper to primitive

// can also construct an Integer from a string:
y = new Integer(“5”);
```
**ArrayLists of wrapper type objects**

<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>
<tr>
<td>float</td>
<td>Float</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);
  ```

---

**ArrayLists of wrapper type objects**

- **Autoboxing:**

  ```java
  ArrayList<Double> grades = new ArrayList<Double>();
  // Autoboxing: create Double from double 3.2
  grades.add(3.2);
  grades.add(2.7);
  double sum = 0.0;
  for (int i = 0; i < grades.size(); i++) {
    //AutoUNboxing from Double to double
    sum += grades.get(i);
  }
  ...
  ```
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 ArrayList.

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 common behavior amongst classes. Example: the List interface is implemented by several Collections classes (LinkedList, ArrayList, Vector, Stack).