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.
  - Remember studentGr? We had to create a completely new grades array to add a grade to it.
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
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.

  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>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>
</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>
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
    }
}
```
There are two main approaches to learning about classes in Java:

1. **Java API Specification**: This is a huge web page containing documentation about every Java class and its methods. You can find the link to the API Specifications on the course web site.

2. **Online Resources and Tutorials**: There are numerous online resources and tutorials available that provide detailed explanations and examples for learning about Java classes. These resources are often updated regularly to reflect changes in the Java language.
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?
- How to do this correctly?
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");
```
Wrapper classes

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);
```

<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>
Wrapper classes - continued

- 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);
}
...
```
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 ArrayList.