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.

**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

**Using ArrayIntList**
- **construction**
  ```java
  int[] numbers = new int[5];
  ArrayIntList list = new ArrayIntList();
  ```
- storing a value, retrieving a value:
  ```java
  numbers[0] = 42;
  int val = numbers[0];
  list.add(42);
  int val = 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 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;

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

Type Parameters (Generics)

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

- `add(value)`: appends value at end of list
- `add(index, value)`: inserts given value at given index, shifting subsequent values right
- `clear()`: removes all elements of the list
- `indexOf(value)`: returns first index where given value is found in list (-1 if not found)
- `get(index)`: returns the value at given index
- `remove(index)`: removes/returns value at given index, shifting subsequent values left
- `set(index, value)`: replaces value at given index with given value
- `size()`: returns the number of elements in list
- `toString()`: returns a string representation of the list such as "[3, 42, -7, 15]"

ArrayList methods 2

- `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.

ArraysList 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 -- Example

- Every java primitive has a class dedicated to it.
  ```java
  Example:
  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);
  double sum = 0.0;
  for (int i = 0; i < grades.size(); i++) { //AutoUnboxing from Double to double
    sum += grades.get(i);
  }
  ```

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?
  - How to do this correctly?

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, e.g., 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.