G51OOP

7. ArrayLists

Object Oriented Programming

Colin Higgins

Array & Collections Overview

- Arrays
  - Working with arrays
  - Java API support for arrays
- Intro to collection classes
  - Types of collection
- ArrayLists

Java Arrays – The Basics

- Declaring an array

```
int[] myArray;
int[] myArray = new int[SIZE1];
String[] stringArray = new String[SIZE2];
String[] strings = new String[] {“one”, “two”};
```

- Checking an arrays length

```
int arrayLength = myArray.length;
```

- Looping over an array

```
for(int i = 0; i < myArray.length; i++) {
   String s = myArray[i];
}
```

- **Also**

```
   for (int i: myArray) { //etc…
```

Java Arrays – Bounds Checking

- Bounds checking
  - Java does this automatically. Impossible to go beyond the end of an array (unlike C/C++)
  - Automatically generates an `ArrayIndexOutOfBoundsException`

Java Arrays – Copying

- Don’t copy arrays “by hand” by looping over the array
- The System class has an `arrayCopy` method to do this efficiently

```
int array1[] = new int[SIZE_TEN];
int array2[] = new int[SIZE_TEN];
//assume we add items to array1

//copy array1 into array2
System.arrayCopy(array1, 0, array2, 0, 10);
//copy last 5 elements in array1 into first 5 of array2
System.arrayCopy(array1, 5, array2, 0, 5);
```

Java Arrays – Sorting

- Again no need to do this “by hand”.
- The java.util.Arrays class has methods to sort different kinds of arrays

```
int myArray[] = new int[] {5, 4, 3, 2, 1};
java.util.Arrays.sort(myArray);
//myArray now holds 1, 2, 3, 4, 5
```

- Sorting arrays of *objects* involves some extra work, as you need to implement the Comparable interface.

Java Arrays

- Advantages
  - Very efficient, quick to access and add to
  - Type-safe, can only add items that match the declared type of the array
- Disadvantages
  - Fixed size, some overhead in copying/resizing
  - Can't tell how many items in the array, just how large it was declared to be
  - Limited functionality, need more general functionality

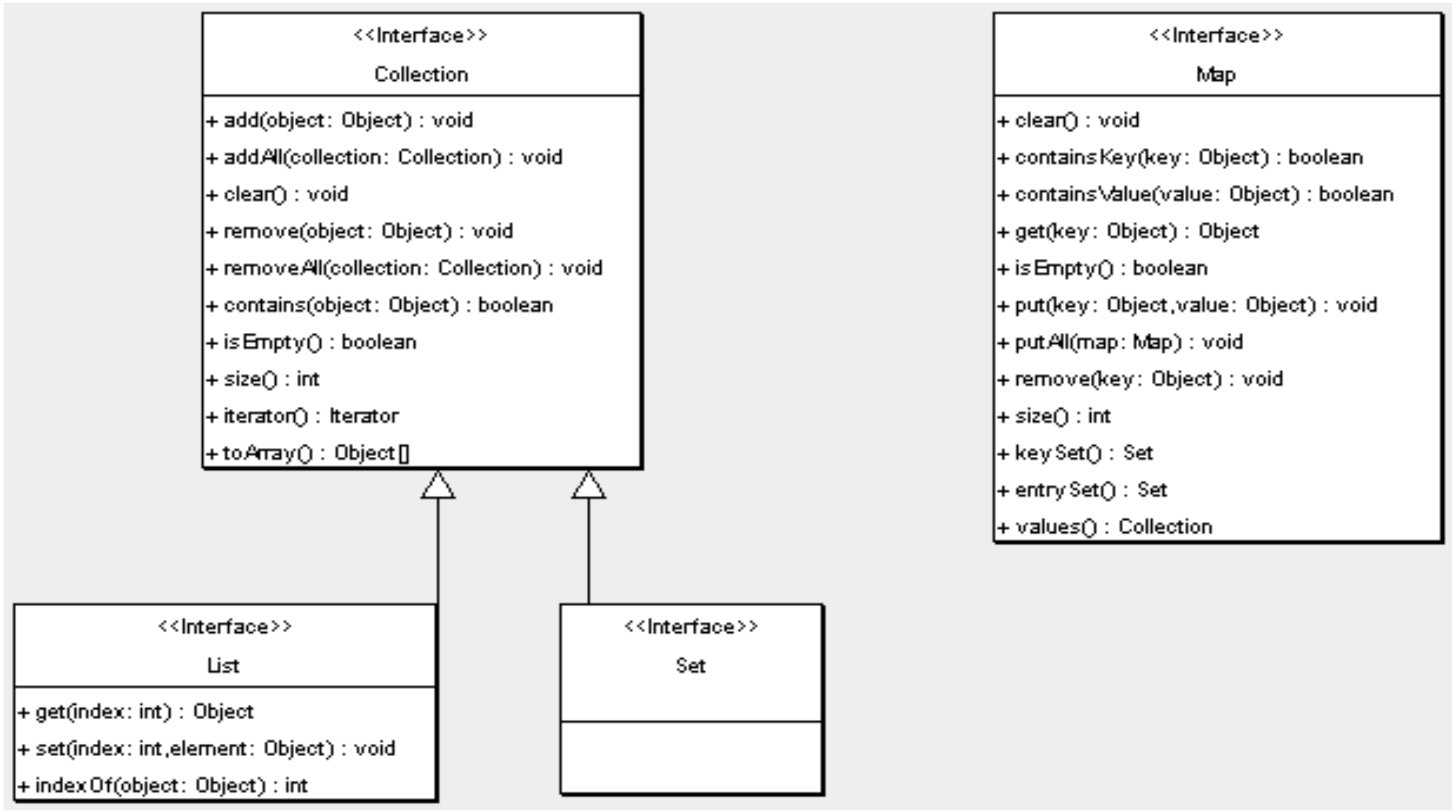
Intro to Java Collections

- What are they?
  - A number of pre-packaged implementations of common 'container' classes, such as LinkedLists, Sets, etc.
  - Part of the `java.util` package.
- Advantages
  - Very flexible, can hold any kind of object
- Disadvantages
  - Not as efficient as arrays (for some uses)
  - Need to make them type-safe using generics (templates)

Java Collections

- Two Types of Containers
- Collections
  - Group of objects, which may be restricted or manipulated in some way
  - E.g. ordered to make a List or LinkedList
  - E.g. a Set, an unordered group which can only contain one of each item
- Maps
  - Associative array, Dictionary, Lookup Table, Hash
  - A group of name-value pairs

Java Collections



<<Interface>>
Collection
+ add(object: Object) : void
+ addAll(collection: Collection) : void
+ clear() : void
+ remove(object: Object) : void
+ removeAll(collection: Collection) : void
+ contains(object: Object) : boolean
+ isEmpty() : boolean
+ size() : int
+ iterator() : Iterator
+ toArray() : Object[]
<<Interface>>
Map
+ clear() : void
+ containsKey(key: Object) : boolean
+ containsValue(value: Object) : boolean
+ get(key: Object) : Object
+ isEmpty() : boolean
+ put(key: Object, value: Object) : void
+ putAll(map: Map) : void
+ remove(key: Object) : void
+ size() : int
+ keySet() : Set
+ entrySet() : Set
+ values() : Collection
<<Interface>>
List
+ get(index: int) : Object
+ set(index: int, element: Object) : void
+ indexOf(object: Object) : int
<<Interface>>
Set

Java Collections

- Several implementations associated with each of the basic interfaces
- Each has its own advantages/disadvantages
- Maps
  - HashMap, SortedMap
- Lists
  - ArrayList, LinkedList
- Sets
  - HashSet, SortedSet

Generics

- Beginning with version 5.0, Java allows class and method definitions that include parameters for types
- Such definitions are called *generics*
  - Generic programming with a type parameter enables code to be written that applies to any class
- Eg

```
ArrayList<String> aList = new ArrayList<String>();
```

The **ArrayList** Class

- **ArrayList** is a class in the standard Java libraries
  - Unlike arrays, which have a fixed length once they have been created, an **ArrayList** is an object that can grow and shrink while your program is running
- In general, an **ArrayList** serves the same purpose as an array, except that an **ArrayList** can change length while the program is running

The **ArrayList** Class

- The class **ArrayList** is implemented using an array as a private instance variable
  - When this hidden array is full, a new larger hidden array is created and the data is transferred to this new array

Using the **ArrayList** Class

- In order to make use of the **ArrayList** class, it must first be imported from the package **java.util**
- An **ArrayList** is created and named in the same way as object of any class, except that you specify the base type as follows:

```
ArrayList<BaseType> aList =  new ArrayList<BaseType>();
```

Using the **ArrayList** Class

- An initial capacity can be specified when creating an **ArrayList** as well
  - The following code creates an **ArrayList** that stores objects of the base type **String** with an initial capacity of 20 items

    ```
    ArrayList<String> list = new ArrayList<String>(20);
    ```
  - Specifying an initial capacity does not limit the size to which an **ArrayList** can eventually grow
- Note that the base type of an ArrayList is specified as a *type parameter*

Using the **ArrayList** Class

- The **add** method is used to set an element for the first time in an **ArrayList**

  ```
  list.add("something");
  ```

  – The method name **add** is overloaded

  – There is also a two argument version that allows an item to be added at any currently used index position or at the first unused position

Using the **ArrayList** Class

- The **size** method is used to find out how many indices already have elements in the **ArrayList**

```
int howMany = list.size();
```

- The **set** method is used to replace any existing element, and the **get** method is used to access the value of any existing element

```
list.set(index, "something else");
String thing = list.get(index);
```

ArrayList v Array 1

Why use an array instead of an **ArrayList**?

1. An **ArrayList** is less efficient than an array (for simple operations)
2. **ArrayList** does not have the convenient square bracket notation
3. The base type of an **ArrayList** must be a class type (or other reference type). It cannot be a primitive type. (Although wrappers, auto boxing, and auto unboxing make this a non-issue with Java 5)

ArrayList v Array 2

Why use an **ArrayList** instead of an array?

1. Arrays can't grow. Their size is fixed at compile time.
   - **ArrayList** grows and shrinks as needed while your program is running

2. You need to keep track of the actual number of elements in your array (recall partially filled arrays).
   - **ArrayList** will do that for you.

3. Arrays have no methods (just **length** instance variable)
   - **ArrayList** has powerful methods for manipulating the objects within it

ArrayList v Array 3

- construction

  String[] names = new String[5];

  ArrayList<String> list = new ArrayList<String>();

- storing a value

  names[0] = "Jessica";

  list.add("Jessica");

- retrieving a value

  String s = names[0];

  String s = list.get(0);

ArrayList v Array 4

- doing something to each value that starts with "B"

```
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
}
```

`for (int i = 0; i < list.size(); i++) {`
`    if (list.get(i).startsWith("B")) { ... }`
`}`

- seeing whether the value "Colin" is found

```
for (int i = 0; i < names.length; i++) {
    if (names[i].equals("Colin")) { ... }
}
```

`if (list.contains("Colin")) { ... }`

ArrayList as a parameter

- Removes all plural words (ending in ‘s’) from the given list.

```
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- You can also return a list:

```
public ArrayList<Type> methodName(params)
```