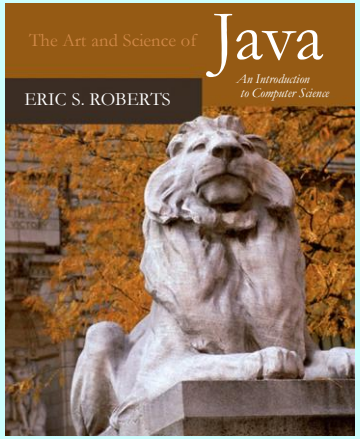
CHAPTER 11

Arrays and ArrayLists

Little boxes, on a hillside, little boxes made of ticky-tacky
Little boxes, little boxes, little boxes, all the same
There's a green one and a pink one and a blue one and a yellow one
And they're all made out of ticky-tacky and they all look just the same
—Malvina Reynolds, "Little Boxes," 1962



CS102 @ Özyeğin University

Slides are adapted from the originals available at
http://www-cs-faculty.stanford.edu/~eroberts/books/ArtAndScienceOfJava/

The `ArrayList` Class

- Although arrays are conceptually important as a data structure, they are not used as much in Java as they are in most other languages. The reason is that the **`java.util`** package includes a class called **`ArrayList`** that provides the standard array behavior along with other useful operations.

The **`ArrayList`** Class

- Although arrays are conceptually important as a data structure, they are not used as much in Java as they are in most other languages. The reason is that the **`java.util`** package includes a class called **`ArrayList`** that provides the standard array behavior along with other useful operations.
- The main differences between Java arrays and **`ArrayList`**s stem from the fact that **`ArrayList`** is a Java class rather than a special form in the language. As a result, all operations on **`ArrayList`**s are indicated using method calls. For example, the most obvious differences include:
  - You create a new **`ArrayList`** by calling the **`ArrayList`** constructor.
  - You get the number of elements by calling the **`size`** method rather than by selecting a **`length`** field.
  - You use the **`get`** and **`set`** methods to select individual elements.

The **`ArrayList`** Class

- Although arrays are conceptually important as a data structure, they are not used as much in Java as they are in most other languages. The reason is that the **`java.util`** package includes a class called **`ArrayList`** that provides the standard array behavior along with other useful operations.
- The main differences between Java arrays and **`ArrayList`**s stem from the fact that **`ArrayList`** is a Java class rather than a special form in the language. As a result, all operations on **`ArrayList`**s are indicated using method calls. For example, the most obvious differences include:
  - You create a new **`ArrayList`** by calling the **`ArrayList`** constructor.
  - You get the number of elements by calling the **`size`** method rather than by selecting a **`length`** field.
  - You use the **`get`** and **`set`** methods to select individual elements.
- The next slides summarize the most important methods in the **`ArrayList`** class. The notation **`<T>`** indicates the base type.

The `ArrayList` Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

Methods in the `ArrayList` Class

`boolean add(<T> element)` Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.
`void add(int index, <T> element)` Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.
`<T> remove(int index)` Removes the element at the specified position and returns that value.
`boolean remove(<T> element)` Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.
`void clear()` Removes all elements from the **`ArrayList`**.
`int size()` Returns the number of elements in the **`ArrayList`**.
`<T> get(int index)` Returns the object at the specified index.
`<T> set(int index, <T> value)` Sets the element at the specified index to the new value and returns the old value.
`int indexOf(<T> value)` Returns the index of the first occurrence of the specified value, or −1 if it does not appear.
`boolean contains(<T> value)` Returns **`true`** if the **`ArrayList`** contains the specified value.
`boolean isEmpty()` Returns **`true`** if the **`ArrayList`** contains no elements.

The `ArrayList` Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

The `ArrayList` Class

```
int[] array1 = new int[5];


for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

0	0	0	0	0

The `ArrayList` Class

```
int[] array1 = new int[5];


for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

0	1	0	0	0

The **`ArrayList`** Class

```
int[] array1 = new int[5];


for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

0	1	2	0	0

The `ArrayList` Class

```
int[] array1 = new int[5];


for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

0	1	2	3	0

The `ArrayList` Class

```
int[] array1 = new int[5];


for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

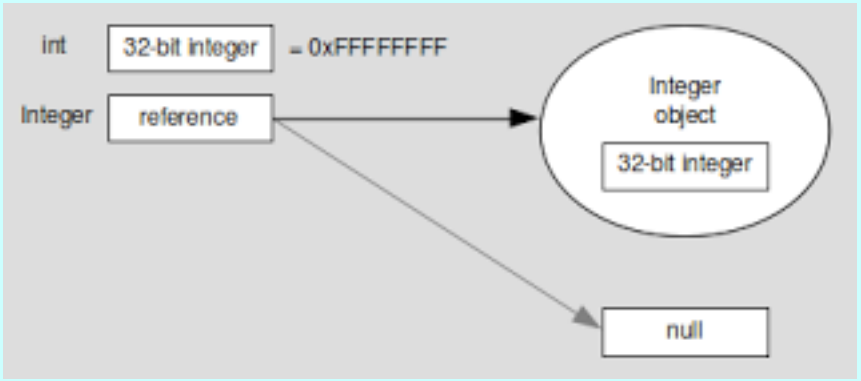
0	1	2	3	4

The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

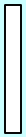
Java Collection classes do not accept primitive types. Instead they use wrapper classes.

`int vs. Integer`



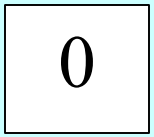
The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```



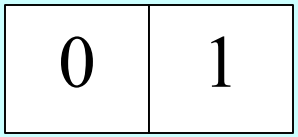
The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```



The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```



The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

0	1	2	3

The `ArrayList` Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

0	1	2	3	4

The `ArrayList` Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();
for(int i = 0; i < 5; i++) {
    array1[i] = i;
    println(array1.length);
}
println("----------")
for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
    println(arrayList1.size());
}
```

The **`ArrayList`** Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();
for(int i = 0; i < 5; i++) {
    array1[i] = i;
    println(array1.length);
}
println("----------")
for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
    println(arrayList1.size());
}
```

```
5
5
5
5
5
---------
1
2
3
4
5
```

`add` *method extends the size by one by adding a new element to the end of the list.*

The `ArrayList` Class

```
array[2] = 42;
println(array[2]);

arrayList.set(2, 42);
println(arrayList.get(2));
```

The `ArrayList` Class

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.add(44);
arrayList.set(2, 45);
println(arrayList.get(2));
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
println(arrayList.get(2));
arrayList.add(44);
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.set(2, 45);
arrayList.add(44);
println(arrayList.get(2));
```

?

The `ArrayList` Class

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.add(44);
arrayList.set(2, 45);
println(arrayList.get(2)); // prints 45
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
println(arrayList.get(2)); // ERROR!!!
arrayList.add(44);
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.set(2, 45); // ERROR!!!
arrayList.add(44);
println(arrayList.get(2));
```

?

The `ArrayList` Class

```
ArrayList<Integer> intList = new ArrayList<Integer>();
ArrayList<Double> doubleList = new ArrayList<Double>();
ArrayList<String> stringList = new ArrayList<String>();

intList.add(42);
doubleList.add(42.0);
stringList.add("42");
```

Reversing an `ArrayList` (Java 5.0)

```
import java.util.*;

/**
 * This program reads in a list of integers and then displays that list in
 * reverse order. This version uses an ArrayList<Integer> to hold the values.
 */
public class ReverseArrayList {

   public static void main(String[] args) {
      println("This program reverses the elements in an ArrayList.");
      println("Use " + SENTINEL + " to signal the end of the list.");
      ArrayList<Integer> list = readArrayList();
      reverseArrayList(list);
      printArrayList(list);
   }

/* Reads the data into the list */
   public static ArrayList<Integer> readArrayList() {
      ArrayList<Integer> list = new ArrayList<Integer>();
      while (true) {
         int value = readInt(" ? ");
         if (value == SENTINEL) break;
         list.add(value);
      }
      return list;
   }
```

Reversing an `ArrayList` (Java 5.0)

```
/* Prints the data from the list, one element per line */
   public static void printArrayList(ArrayList<Integer> list) {
      for (int i = 0; i < list.size(); i++) {
         int value = list.get(i);
         println(value);
      }
   }

/* Reverses the data in an ArrayList */
   public static void reverseArrayList(ArrayList<Integer> list) {
      for (int i = 0; i < list.size() / 2; i++) {
         swapElements(list, i, list.size() - i - 1);
      }
   }

/* Exchanges two elements in an ArrayList */
   public static void swapElements(ArrayList<Integer> list, int p1, int p2) {
      int temp = list.get(p1);
      list.set(p1, list.get(p2));
      list.set(p2, temp);
   }

/* Private constants */
   private static final int SENTINEL = 0;

}
```

Generic Types in Java 5.0

- The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language. In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array. Class definitions that include a type parameter are called **generic types**.

Generic Types in Java 5.0

- The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language. In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array. Class definitions that include a type parameter are called **generic types**.
- When you declare or create an **ArrayList**, it is a good idea to specify the element type in angle brackets. For example, to declare and initialize an **ArrayList** called **names** that contains elements of type **String**, you would write

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

Generic Types in Java 5.0

- The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language. In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array. Class definitions that include a type parameter are called **generic types**.
- When you declare or create an **ArrayList**, it is a good idea to specify the element type in angle brackets. For example, to declare and initialize an **ArrayList** called **names** that contains elements of type **String**, you would write

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

- The advantage of specifying the element type is that Java now knows what type of value the **ArrayList** contains. When you call **set**, Java can ensure that the value matches the element type. When you call **get**, Java knows what type of value to expect, eliminating the need for a type cast.

```
import java.util.ArrayList;

public class Experiments {
        public static void main(String[] args) {
                ArrayList<Integer> list = new ArrayList<Integer>();
                list.add(42);
                list.add(43);
                printArrayList(list);
                list.add(0, 44);
                printArrayList(list);
                list.add(1, 45);
                printArrayList(list);
                list.remove(1);
                printArrayList(list);
                list.remove(new Integer(42));
                printArrayList(list);
//              list.remove(43); // ERROR!!!
//              printArrayList(list);

                ArrayList<String> list2 = new ArrayList<String>();
                list2.add("a");
                list2.add("b");
                list2.add("c");
                list2.add("d");
                printStringArrayList(list2);
                list2.remove(1);
                printStringArrayList(list2);
                list2.remove("c");
                printStringArrayList(list2);
        }
```

```
        public static void printStringArrayList(ArrayList<String> list) {
                for(int i = 0; i < list.size(); i++){
                        System.out.print(" " + list.get(i));
                }
                System.out.println();
        }

        public static void printArrayList(ArrayList<Integer> list) {
                for(int i = 0; i < list.size(); i++){
                        System.out.print(" " + list.get(i));
                }
                System.out.println();
        }
}
```

```
import java.util.ArrayList;

public class Experiments {
        public static void main(String[] args) {
                ArrayList<Integer> list = new ArrayList<Integer>();
                list.add(42);
                list.add(43);
                printArrayList(list);
                list.add(0, 44);
                printArrayList(list);
                list.add(1, 45);
                printArrayList(list);
                list.remove(1);
                printArrayList(list);
                list.remove(new Integer(42));
                printArrayList(list);
//              list.remove(43); // ERROR!!!
//              printArrayList(list);

                ArrayList<String> list2 = new ArrayList<String>();
                list2.add("a");
                list2.add("b");
                list2.add("c");
                list2.add("d");
                printStringArrayList(list2);
                list2.remove(1);
                printStringArrayList(list2);
                list2.remove("c");
                printStringArrayList(list2);
        }
```

```
42 43
44 42 43
44 45 42 43
44 42 43
44 43
a b c d
a c d
a d
```