

Chapter 7

The Java Array Object

© Rick Mercer

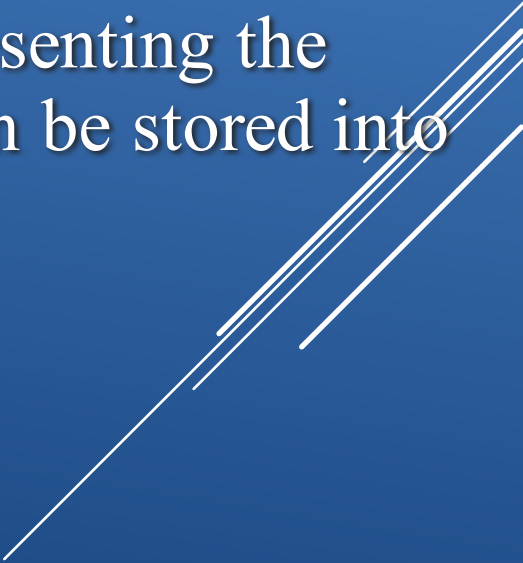
A decorative graphic consisting of several parallel white lines of varying lengths, slanted diagonally from the bottom right towards the top right, located in the lower right corner of the slide.

The Java Array Object

- ◆ Some variables store precisely one value:
 - a double stores one floating-point number
 - an int stores one integer
 - a reference variable stores a "reference" to an object that may store many other variables
- ◆ Java Arrays
 - store a collection of "elements"
 - May be references to objects or may be primitive values
 - programmers access individual objects with subscript notation [] (square brackets)

Array Construction

```
type [] array-reference = new type [capacity] ;
```

- *type* specifies the type of element stored in the array
 - *array-name* is any valid Java identifier that refers to all elements in the array
 - *capacity* is an integer expression representing the maximum number of elements that can be stored into the array
- 

Example Array Declarations

- ◆ An array named x that stores up to 8 numbers

```
double[] x = new double[8];
```

- ◆ This array stores references to 500 strings

```
String[] name = new String[500];
```

- ◆ An array named test to store 2,000 integers

```
int capacity = 1000;
```

```
int[] test = new int[2*capacity];
```

- ◆ An array named customer to stores references to up to 100 unique BankAccount objects

```
BankAccount[] customer = new BankAccount[100];
```

Accessing Individual Array Elements

- ◆ Individual array elements are referenced through subscripts of this form:

array-name [*int-expression*]

- *int-expression* is an integer that should be in the range of $0..capacity-1$

- ◆ Examples:

```
x[0]           // Pronounced x sub 0
name[5]        // Pronounced name sub 5
test[99]       // Pronounced test sub 99
customer[12]   // Pronounced customer sub 12
```

Assignment to individual array elements

```
// All int elements are  
// set to 0 initially  
int[] x = new int[5];
```

```
x[0] = 1;  
x[1] = 2;  
x[2] = 5;  
x[3] = x[0] + x[2];  
x[4] = x[3] - 1;
```

```
for(int j = 0; j < x.length; j++) {  
    // What is the Output?  
    System.out.println(x[j]);  
}
```

Reference	Value
x[0]	1
x[1]	2
x[2]	5
x[3]	6
x[4]	5

Out of Range Subscripts

- ◆ Subscripts can be "out of range"

```
String[] name = new String[1000];  
name[-1] = "Subscript too low";  
name[ 0] = "This should be the first name";  
name[999] = "This is the last good subscript";  
name[1000] = "Subscript too high";
```

- ◆ Two of the above references will cause `ArrayIndexOutOfBoundsException` exceptions and "ungracefully" terminate the program with:

```
java.lang.ArrayIndexOutOfBoundsException: -1
```

Array initializer (a shortcut) and the length variable

- ◆ Arrays can be initialized with an array initializer

```
int[] arrayOfInts = { 55, 33, 77, 22, -99 };
```

```
assertEquals(55, arrayOfInts[0]);  
assertEquals(33, arrayOfInts[1]);  
assertEquals(77, arrayOfInts[2]);  
assertEquals(22, arrayOfInts[3]);  
assertEquals(-99, arrayOfInts[4]);
```

```
// arrayReference.length returns capacity  
assertEquals(5, arrayOfInts.length);
```


Passing Array Arguments

- ◆ Imagine a method that returns true if the first and last values are equal

```
double [] a1 = { 5.0, 4.0, 3.0, 2.0 };
```

```
assertFalse(endsSame(a1));
```

```
double [] a2 = { 5.0, 4.0, 3.0, 2.0, -1.0, 5.0 };
```

```
assertTrue(endsSame(a2));
```

- ◆ The method heading has an array parameter

```
public boolean endsSame(double[] x) {  
}
```

Let's complete this method in a JUnit test

Process all elements with a for loop

- ◆ Imagine a method that returns the average of the array elements

```
double[] x1 = { 5.0, 4.0, 3.0, 2.0 };  
assertEquals(3.5, average(x1), 0.001);  
double[] x2 = { 5.0, 4.0, 3.0, 2.0, -1.0, 5.0 };  
assertEquals(3.0, average(x2), 0.001);
```

- ◆ This method will use an array parameter

```
public double average(double[] x) {  
}
```

Let's complete this method in a JUnit test

Answers

```
public boolean endsSame(double[] x) {  
    return x[0] == x[x.length - 1];  
}
```

```
public double average(double[] x) {  
    int n = x.length;  
    double total = 0;  
    // get the sum of all elements  
    for (int i = 0; i < n; i++) {  
        total += x[i];  
    }  
    // Compute and return the average  
    return total / n;  
}
```