Arrays

Chapter 8

Objectives

You will be able to

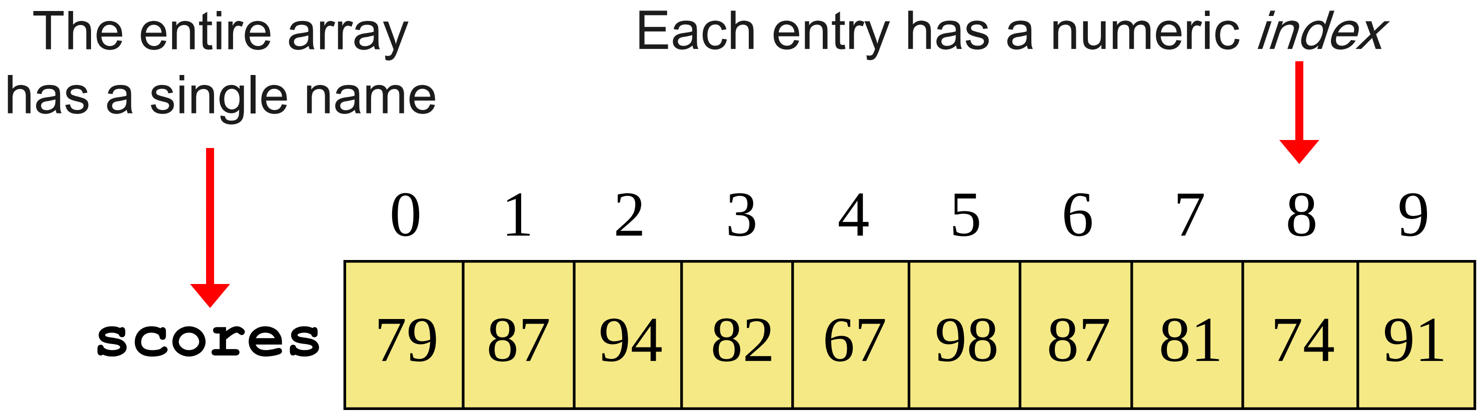
- Use arrays in your Java programs to hold a large number of data items of the same type.
- Initialize an array in the declaration.
- Write a loop to process the items of an array.

Chapter 8: Arrays

- Arrays are objects that help us organize large amounts of information.
- Today we will focuses on:
  - Array declaration and use
  - Bounds checking and capacity
  - Arrays that store object references
  - Pitfall: Scanner class, nextLine method (Not in the textbook)

Arrays

An *array* is an ordered list of values


The entire array has a single name
Each entry has a numeric *index*
0 1 2 3 4 5 6 7 8 9
scores
79 87 94 82 67 98 87 81 74 91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Declaring Arrays

- The **`scores`** array could be declared as follows:

```
int[] scores = new int[10];
```

- The type of the variable **`scores`** is **`int[]`**
  - Array of integers
- The reference variable **`scores`** is set to a new array object that can hold 10 integers.

Declaring and Using Arrays

- Some other examples of array declarations:

```
double[] prices = new double[500];


boolean[] flags;

flags = new boolean[20];


char[] codes = new char[1750];
```

Array Elements

- A particular entry in an array is referenced using the array name followed by the index in brackets.
- For example, the expression

```
scores[2]
```

  refers to the 3rd value in the array scores.

Array Entries

```
    scores[2]
```

- The expression represents a place to store a single integer and can be used wherever an integer variable can be used.

Array Elements

- The values held in an array are called *array elements.*
- An array stores multiple values of the same type
  - The *element type*
- The element type can be a primitive type or an object reference.
- Therefore, we can create an array of integers, an array of characters, an array of **`String`** objects, etc.

Array Elements

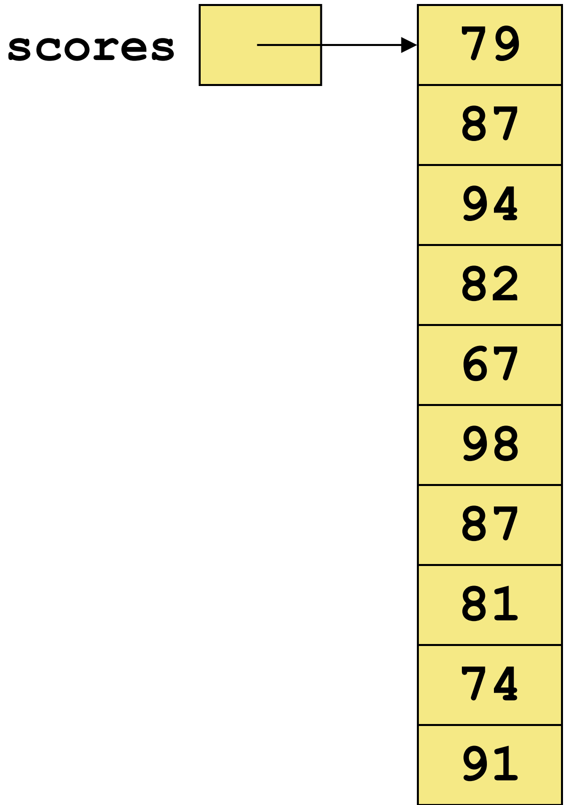
- An array element can be assigned a value, printed, or used in a calculation:

```
scores[2] = 89;

scores[0] = scores[0] + 2;

System.out.println ("Top = " + scores[5]);
```

Anything that we could do with an integer variable.

Arrays

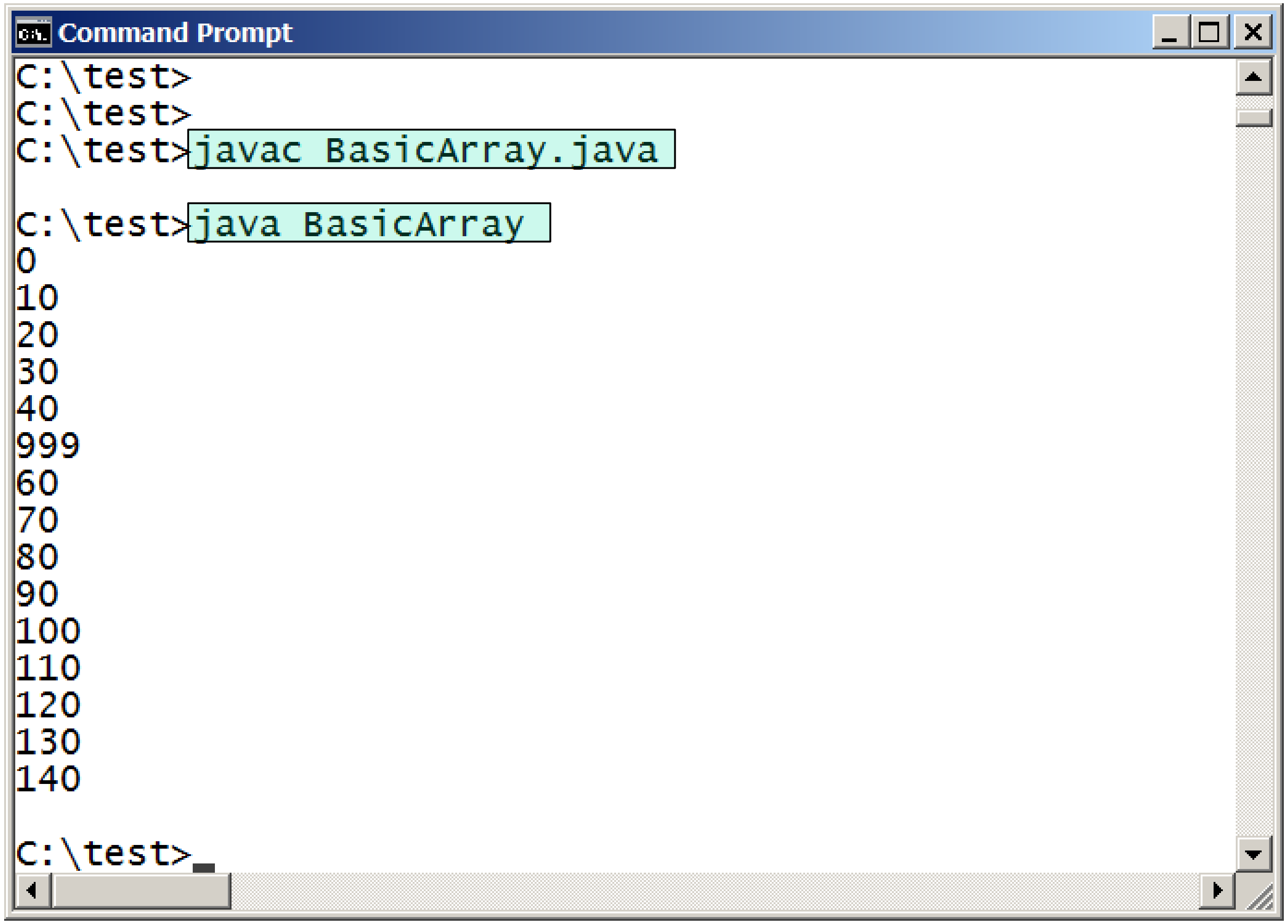
- In Java, the array itself is an object that must be instantiated.
- Another way to picture the **`scores`** array:



BasicArray.java

```java
public class BasicArray
{
    //----------------------------------------------------
    //  Creates an array, fills it with various integer
    //  values,modifies one value, then prints them out.
    //----------------------------------------------------
    public static void main (String[] args)
    {
        final int LIMIT = 15, MULTIPLE = 10;
        int[] list = new int[LIMIT];

        //  Initialize the array values
        for (int index = 0; index < LIMIT; index++)
        {
            list[index] = index * MULTIPLE;
        }

        list[5] = 999;  // change one array value

        for (int index = 0; index < LIMIT; index++)
        {
             System.out.println(list[index]);
        }
    }
}
```

BasicArray Running



```
C:\test>
C:\test>
C:\test>javac BasicArray.java

C:\test>java BasicArray
0
10
20
30
40
999
60
70
80
90
100
110
120
130
140

C:\test>_
```

Exercise

Write a program to declare an array named **`nums`** of ten elements of type int and initialize the elements (starting with the first) to the values 2, 4, 6, 8,..., 18, 20 respectively.

End of Section

Array Index and Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element.
  - The index value must be in range 0 to N-1
- The Java interpreter throws an **`ArrayIndexOutOfBoundsException`** if an array index is out of bounds
- This is called automatic *bounds checking.*

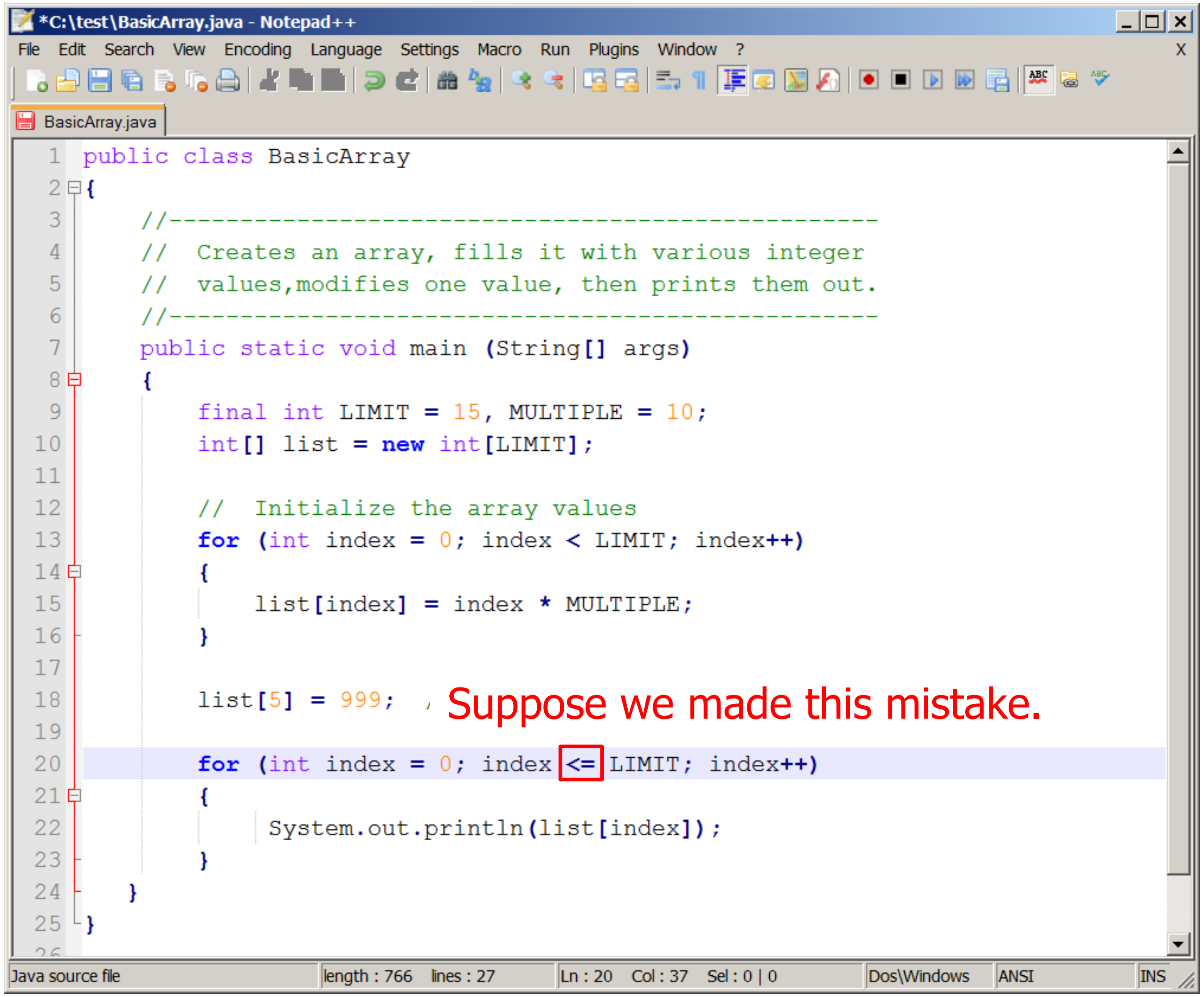
Bounds Checking

- If the array **`codes`** can hold 100 values, it can be indexed using only the numbers 0 to 99.
- The following reference will cause an exception to be thrown:

problem

```
for (int index=0; index <= 100; index++)
{
    codes[index] = index*50 + epsilon;
}
```

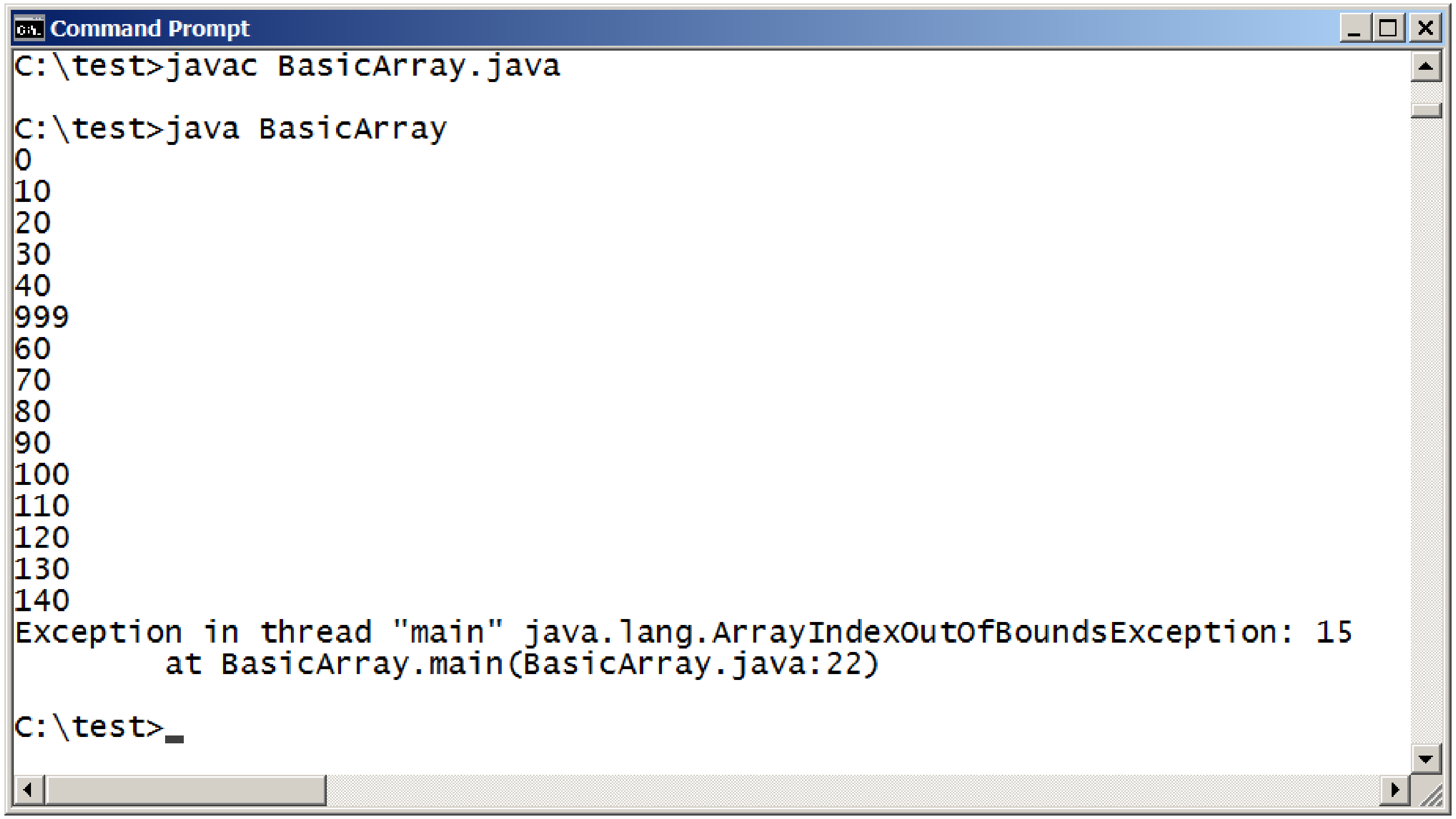
BasicArray Example


*C:\test\BasicArray.java - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
BasicArray.java

```java
public class BasicArray
{
    //-----------------------------------------------------------------
    //  Creates an array, fills it with various integer
    //  values,modifies one value, then prints them out.
    //-----------------------------------------------------------------
    public static void main (String[] args)
    {
        final int LIMIT = 15, MULTIPLE = 10;
        int[] list = new int[LIMIT];

        //  Initialize the array values
        for (int index = 0; index < LIMIT; index++)
        {
            list[index] = index * MULTIPLE;
        }

        list[5] = 999;

        for (int index = 0; index <= LIMIT; index++)
        {
            System.out.println(list[index]);
        }
    }
}
```

Suppose we made this mistake.

Here is what would happen



```
C:\test>javac BasicArray.java

C:\test>java BasicArray
0
10
20
30
40
999
60
70
80
90
100
110
120
130
140
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 15
        at BasicArray.main(BasicArray.java:22)

C:\test>_
```

Bounds Checking

- Each array object has a public constant called **`length`** that stores the size of the array.
- It is referenced using the array name:

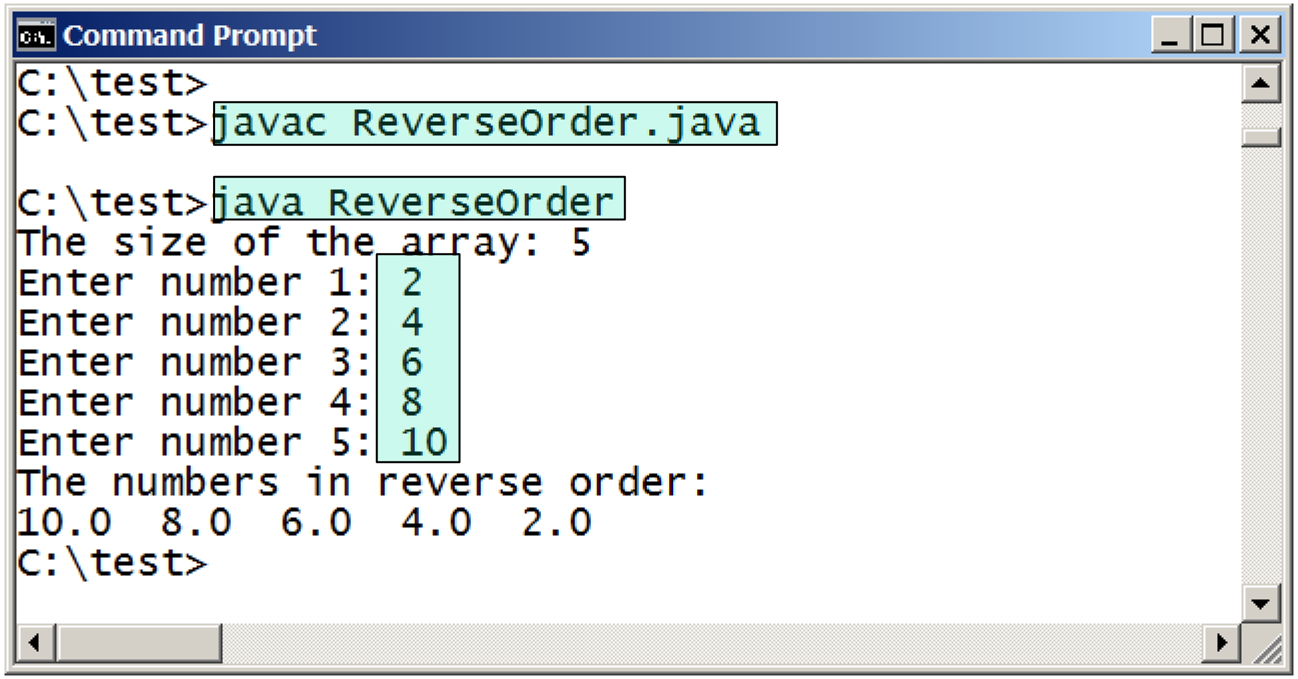
```
scores.length
```

- Note that **`length`** holds the number of elements, not the largest index.
- No parentheses
  - Unlike the String method length()

ReverseOrder.java

```
import java.util.Scanner;
public class ReverseOrder
{
    public static void main (String[] args)
    {
        Scanner scan = new Scanner (System.in);

        double[] numbers = new double[5];

        System.out.println ("The size of the array: " +
                             numbers.length);

        for (int index = 0; index < numbers.length; index++)
        {
            System.out.print ("Enter number " + (index+1) + ": ");
            numbers[index] = scan.nextDouble();
        }

        System.out.println ("The numbers in reverse order:");

        for (int index = numbers.length-1; index >= 0; index--)
        {
             System.out.print (numbers[index] + "  ");
        }
    }
}
```

ReverseOrder Running



```
C:\test>
C:\test>javac ReverseOrder.java

C:\test>java ReverseOrder
The size of the array: 5
Enter number 1: 2
Enter number 2: 4
Enter number 3: 6
Enter number 4: 8
Enter number 5: 10
The numbers in reverse order:
10.0  8.0  6.0  4.0  2.0
C:\test>
```

Exercise

- Assume that an array of integers named *numbers* has been declared and initialized.
- Write a single statement that assigns a new value to the first element of the array.
- The new value should be equal to twice the value stored in the last element of the array.

End of Section

Initializer Lists

- An *initializer list* can be used to instantiate and fill an array in one step.
- The values are delimited by braces and separated by commas.
- Examples:

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

Initializer Lists

- Note that when an initializer list is used:
  - The **`new`** operator is not used.
  - No size value is specified.
- The size of the array is determined by the number of items in the initializer list.
- An initializer list can be used only in the array declaration.
  - Not in assignment statements.

Arrays of Objects

- The elements of an array can be object references.
- The following declaration reserves space to store 5 references to **`String`** objects

```
String[] words = new String[5];
```

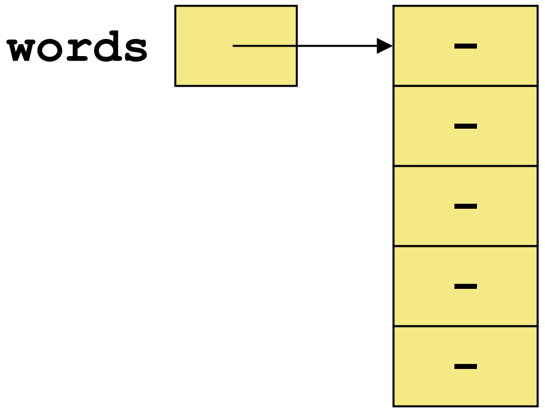
Arrays of Objects

```
String[] words = new String[5];
```

- This does NOT create any `String` objects.
- Initially an array of objects holds `null` references.
- Each object stored in an array must be instantiated separately.
  - It is a *reference* to the object, not the object itself, that is stored in the array.

Arrays of Objects

- The **`words`** array when initially declared:

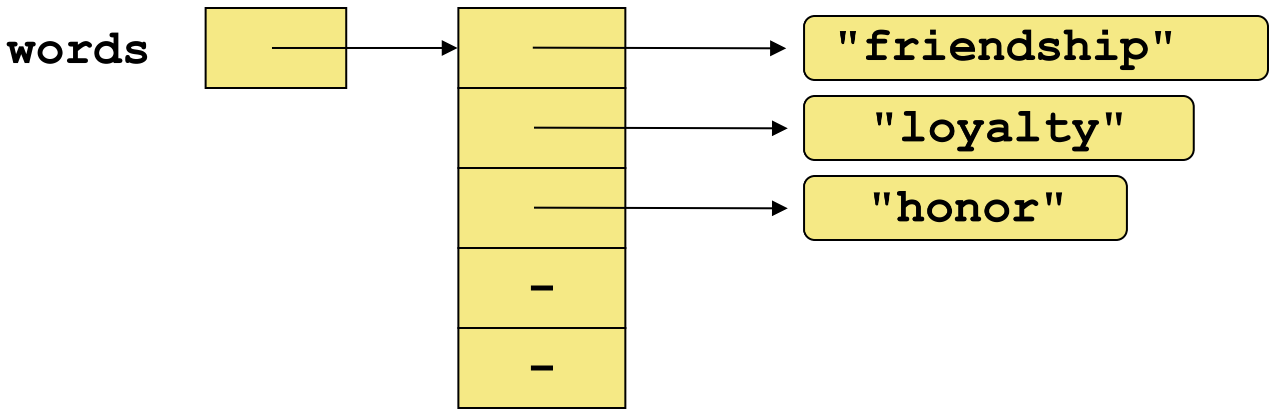


  - At this point, the following reference would throw a **`NullPointerException`**:

```
System.out.println (words[0]);
```

Arrays of Objects

After some `String` objects are created and stored in the array:



Arrays of Objects

- Keep in mind that **`String`** objects can be created using literals.
- The following declaration creates an array object called **`verbs`** and fills it with four **`String`** objects created using string literals.

```
String[] verbs = {"play", "work", "eat", "sleep"};
```

End of Section

Pitfall: Scanner Class, nextLine method

Scanner, nextLine method:

- Reads the remainder of a line of text and returns as a string.
- Starting whenever the last keyboard reading left off
- The line terminator '\n' is read and discarded, not included in the string returned.

```
int number = scan.nextInt();
String str1 = scan.nextLine();
String str2 = scan.nextLine();
```

Pitfall: Scanner Class, nextLine method

- Pitall: calling nextLine after next, nextInt, nextDouble method
  - next, nextInt, or nextDouble reads the input but does not read the new line ('\n') character.
  - The new line character is left in the keyboard input buffer.
  - So the nextLine invocation after reads the rest of the line, which is nothing other than '\n' and returns the empty string.

Pitfall: Scanner Class, nextLine method

Fix the problem:

- Include an extra call to nextLine() to clear the New Line character after calling next, nextInt, or nextDouble.

```
int number = scanner.nextInt();
scanner.nextLine();
String str1 = scanner.nextLine();
String str2 = scanner.nextLine();
```

Readings and Assignments

- Reading: Chapter 8.1-8.3

- Self-Assessment Exercises:
  - Self-Review Questions Section
    - SR 8.3, 8.10, 8.11, 8.15
  - After Chapter Exercises
    - EX 8.1, 8.4, 8.5 a, b, 8.7, 8.8

- Project 8

End of Presentation