String, Array and Sort

동서대학교 인터넷공학부
2005년 2학기
김태용

String

- Examples of declaring Strings:
  - String s1 = "This is a String";
  - String s2 = new String("This is also a String");
- Strings may be appended using the + operator:
  - String s1 = "This is part1 ";
  - String s2 = "and part2";
  - String s3 = s1 + s2; // s3 equals "This is part1 and part2"

String

- Strings are immutable objects, which means that they cannot be changed.
- On the second line of the following example, the old String "old" is discarded and a new String "new" is actually recreated.
  - String s = "old";
  - s = "new"; // s is now equal to "new"

String

- One must be careful when comparing Strings. Strings, like all classes, are descendants of the Object class.
- Using the equality operator (==) compares the references (pointers) of the two objects, not their contents.
- It is advisable to use the *equals* method to compare two Strings.
  - String s1 = new String("TestString");
  - String s2 = new String("TestString");

String

- if (s1 == s2)
- System.out.println("s1 == s2");
- // This will NOT print because s1 and s2
- // reference two different objects.
- if (s2.equals(s1))
- System.out.println("s2.equals(s1)");
- // This will print because s1 and s2
- // contain identical string values.

- String objects cannot be directly accessed as character arrays. If you need more control over your string, use the *StringBuffer* class.

String

- To obtain a substring of a string, use the *substring* method of the String object.
- The first parameter of the *substring* method is the 0-based position of the first character of the desired substring.
- The second parameter of the *substring* method is 1 greater than the 0-based position of the last character of the desired substring:
  - String s1 = "abcdefg";
  - String s2 = s1.substring(3,5); // s2 is equal to "de"

String

- To search for a substring within a given string, use the indexOf method of the String object.
- The *indexOf* method returns the 0-based position within this string of the first occurrence of the specified substring. If the specified substring is not found in the string, -1 is returned.
  - String s = "abcdefg";
  - int x = s.indexOf("de"); // x is equal to 3

String

- The length of a String may be obtained by using the *length* method.
  - String s = "abcdefg";
  - int x = s.length(); // x is equal to 7

Array - Declaring and Defining

- The following two statements are equivalent for declaring integer array variables:
  - int[] myIntArray1; // Declare an integer array variable
  - int myIntArray2[]; // Declare another integer array variable
- Defining an array will allocate memory to store the specified number of elements of the array type as shown below:
  - myIntArray = new int[10]; // Define an array of 10 integers
- Arrays can be declared and defined in one statement:
  - int myIntArray[] = new int[10]; // Declare and define an array of 10 integers
- Arrays can be initialized when they are declared:
  - int myIntArray[] = { 150, 350, 700, 120 } // An array of 4 integers
  - char myCharArray[] = { 'd', 'u', 'd', 'e' } // An array of 4 characters
  - double myDblArray[] = { 1.0, 2.5, 4.3, 2.8 } // An array of 4 doubles

Array - Using Arrays

- Arrays in Java are indexed beginning at 0.
- The following example shows how to store and access values in an array:
  - static final int MY_ARRAY_LEN = 10; // Declare a constant of 10
  - int myIntArray[] = new int[MY_ARRAY_LEN]; // Declare and define an array
  - for (int i = 0; i < MY_ARRAY_LEN; i++)
  - myIntArray[i] = i * i; // Populate the array
  - int sum;
  - for (int j = 0; j < MY_ARRAY_LEN; j++)
  - sum += myIntArray[j]; // Sum the elements of the array
- To get the size of the array, use the length method of the array:
  - int average = sum / myIntArray.length;

Array - Multidimensional Arrays

- Arrays of arrays can be declared in Java
  - int [][] my2DimArray = [100][5]; // 2-D array
  - int [][][] my3DimArray = [100][5][3]; // 3-D array
- The following example populates a 2 dimensional array:
  - for (int i = 0; i < 100; i++)
  - for (int j = 0; j < 5; j++)
  - my2DimArray[i][j] = i * j

Conversions

- To convert from an **int** to a **String** you may simply append the int value to the String as shown below:
  - String s = "" + 3; // s is now equal to "3"
- A **double** can be converted to a **String** the same way:
  - String s = "" + 64.321; // s is now equal to "64.321"
- To convert from a **String** to a **double** it is necessary to use the *Double* object:
  - String s = "64.321";
  - Double D1 = new Double (s);
  - double d1 = D1.doubleValue(); // d1 is now equal to 64.321
- To convert from a **String** to an **int** you may use the *parseInt* method of the Integer object:
  - String s = "23";
  - int x = Integer.parseInt(s); // x is now equal to 23

Bubble Sort

- Algorithm to sort a list of items X0, X1, X2, … , Xn into ascending order
  - For i = 0 to n
  - For j = n down to i+1
  -
  - If Xj-1 > Xj Then
  - Exchange Xj-1 and Xj
  - Next
  - Next

Bubble Sort - Implementation 1

```
int nums[] = { 99, -10, 1000123, 18, -987, 5623, 463, -9, 287, 49 };
int i, j;
int n = nums.length;
for ( i = 0; i < n; i++ )
{
    for (int j = n-1; j > i; j--)
    {
        if ( nums[j-1] > nums[j] )
        {
            temp = nums[j-1];
            nums[j-1] = nums[j];
            nums[j] = temp;
        }
    }
}
```

Bubble Sort - Implementation 2

(From Java2: A Beginner's Guide)

```
int nums[] = { 99, -10, 1000123, 18, -987, 5623, 463, -9, 287, 49 };
int a,b,t;
int size;
size = 10;  // number of elements to sort
// This is the Bubble sort
for ( a=1; a < size; a++ )
{
    for ( b = size – 1; b >= a; b- - )
    {
        if ( nums[b-1] > nums[b] )  // if out of order
        {
            // exchange elements
            t = nums[b-1];
            nums[b-1] = nums[b];
            nums[b] = t;
        }
    }
}
```