

Fundamentals of Programming

Laboratory 6

JAVA non-primitive data types
Strings and Arrays



CONTAINER OBJECTS

As an introduction to Object Oriented Programming today's tutorial will focus on two popular types of objects: **arrays** and **strings**

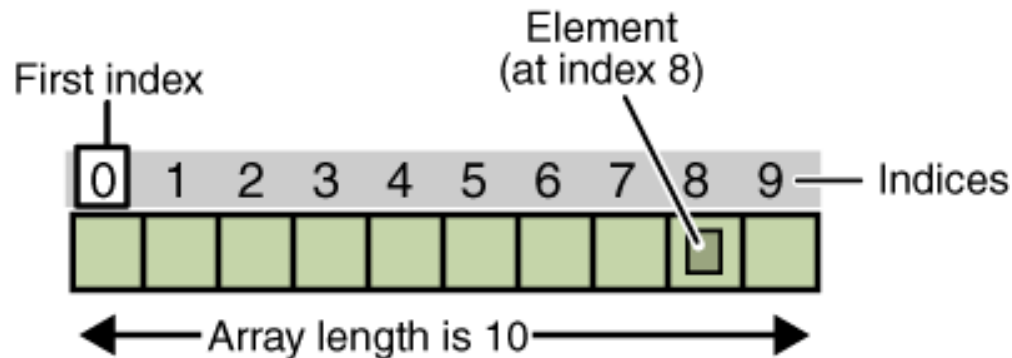
For now think of them as **containers** for a number of values of the **same type** (in case of strings it's **chars** for arrays it can be any **primitive type**) that also have some predefined **methods** that can be performed on the stored data.

ARRAYS

Arrays are simply data arranged into a list (one dimensional array) or a table (multi-dimensional array), that can be accessed by the array object's **identifier** and an **index** (or indexes).

The index of the first element is **0**.

The **length** of an array is established when the array is created.



DECLARATION and ALLOCATION



Arrays in Java are not primitive structures and must be created as **objects**. This means we do not just **declare** it:

```
int[ ] arrayA; //declaration
```

but we also need to **create** or **allocate** the object with the keyword/operator **new**:

```
arrayA = new int[10];  
//allocation
```

After memory allocation, the **length** of the array is **fixed**.

DECLARATION and ALLOCATION



You must both **declare** and **create** an array object:

```
int[] anArray;  
// declares the array
```

```
anArray = new int[10];  
// creates/allocates the array, sets  
size
```

```
int[] anArrayTwo = new int[10];  
// declares and creates the array
```

INITIALIZING ARRAYS

If you initialize the array upon creation, the keyword `new` is unnecessary:

```
char[] anArray2 = { 'a', 'b' } ;
```

```
int[] anArrayOfIntegers = { 1, 2, 3, 8 } ;
```

INDEXES

Values in the array are accessed using their indexes:

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

```
anArray[0] = 100;
```

```
// initializes the zero (first) element in the  
array
```

```
int a = anArray[0];
```

```
// retrieves the value from the first element
```

```
for (int i=0;i<10;i++)
```

```
    anArray[i]=0;    // zeroes all elements of
```

Array exercises (1)

1. Declare an array of integers called Numbers

```
int[] Numbers;
```

2. Create the object for this array, set its size to 5

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

3. Fill the array Numbers with values 1 to 5 using a for loop

```
for (int i=0;i<5;i++)
```

```
    Numbers[i]=i+1;
```



ORAL
EXERCISE

Array exercises (2)

4. Perform operations 1-3 in one statement

```
int[] Numbers = { 1, 2, 3, 4, 5 };
```

5. Switch the first and last value in the array

```
int temp = Numbers[0];  
Numbers[0] = Numbers[4];  
Numbers[4] = temp;
```



Multidimensional arrays

Multidimensional arrays are created just like normal arrays, but with additional brackets.

```
int[][] a2DArray;
```

```
// declares a 2D array
```

```
a2DArray = new int[2][10];
```

```
//creates the 2x10 array
```

```
int[][] anArrayTwo = new int[2][10];
```

```
//both
```

Multidimensional arrays (2)

Each dimension can be allocated separately.

```
int[][] aStrangeArray = new int[2][];  
// creates an array with two rows, but an  
// unknown number of columns
```

```
aStrangeArray[0] = new int[2];  
aStrangeArray[1] = new int[3];  
// creates an array:  
// [ ] [ ]  
// [ ] [ ] [ ]
```

Multidimensional arrays (3)

Two or more indexes are used to access the data in multidimensional arrays.

Nested loops are often used as well:

```
int[][] Numbers = new int[5][5];  
for (int x=0;x<5;x++)  
{  
    for (int y=0;y<5;y++)  
    {  
        Numbers[y][x]=x+y;  
    }  
}
```

Array Task 0:

Copy the below program:

```
class ArrayDemo
{
    public static void main(String[] args)
    {
        int[] a;
        a = new int[3];
        a[0] = 100;
        a[1] = 200;
        a[2] = 300;
        System.out.println("Element at index 0: "+a[0]);
        System.out.println("Element at index 1: "+a[1]);
        System.out.println("Element at index 2: "+a[2]);
    }
}
```



Array Task 1:

Modify the program from Task 0 to allow the user to input all the values to a 10-element array.

Use a loop to ask the user to input the 10 values.

Then ask the user to input a string.

Print all the values from the table separated by that string.

Hint: use the `.length` parameter of arrays:

```
for (int i = 0; i < anArray.length; i++)  
    { ... }
```



Array Task 2:

Prepare and display a 2D array that holds the contents of the multiplication table for numbers 1-10. Use loops to fill the array.



Array Task 3 (optional):

Sort a 10-element array input by the user in descending or ascending order. Count the number of steps taken to sort the array (display the array after each modification).

Use two algorithms (you can write two separate programs):

- a) find the smallest/largest value, move it to index 0, repeat for numbers 1-9, etc.
- b) compare two neighboring values, change places if they are in wrong order, repeat until no changes are made



STRINGS

String - A built-in java class that holds a sequence of characters.

Text variables are objects of the String class, thus to allocate memory for them upon creation either the **new** operator must be used or the string must be **initialized**.

The String class has a large number of **methods** which enable manipulation of string objects.

STRINGS (2)

String objects in java are **read-only** and **immutable**, i.e. their size and content are fixed from the start.

You can assign a new value to a string variable, but it really results in allocating a new object, while the old one still takes up memory (until it is cleaned by the java garbage collector).

STRINGS (3)

Remember! You must both **declare** and **create** a String object, or initialize it right away:

```
String aString = "Hello World!";  
// most common string creation  
String aString2 = new String(aString);  
// creates aString2 with contents of aString  
char[] anArray = {'a', 'b'};  
String aString;  
aString = new String(anArray);  
// creates aString from a char array
```



Commonly used String methods:

```
String aString = "Hello World!";
```

```
int strLength = aString.length();
```

// length() returns the number of characters in the String

```
char letter = aString.charAt(0);
```

//charAt(int i) returns the character at an index i

```
String aString = aString.concat("..and goodbye");
```

```
String bString = " world!";
```

```
String cString = bString.concat(cString);
```

// concatenates(adds) a string to another, returns a new string



Commonly used String methods:

```
String cString = "Hello " + "world!" + aString;  
// more common string adding using the + operator
```

```
String aString = "Hello World!";  
System.out.println(aString.substring(1,4));  
//substring(int beginindex, int endindex) returns a new substring  
from beginindex(inclusive) to endindex(non-inclusive)
```

Each use of text in double quotes (called a string literal) actually creates a string object, so methods can be called on it as well.

```
int strLength = "Hello".length();
```

String exercises:

1. Create a string object named aString containing the word "hello"

```
String aString = "Hello ";
```

2. Add to that string the word " world"

```
String bString = aString+" world";
```

3. Count the number of times the letter "l" appears in the new string (hint: use a for loop and the .charAt() method)

```
int count = 0;
```

```
for (int i=0;i<bString.length();i++)
```

```
if (bString.charAt(i)=='l') count++;
```



Other commonly used String methods:



Accessor methods:

```
length(), charAt(i), getBytes(),  
getChars(istart,iend,gtarget[],itargstart),  
toCharArray(), valueOf(g,iradix), substring(iStart  
[,iEndIndex])
```

Modifier methods:

```
toLowerCase(), toUpperCase(), trim(), concat(g),  
replace(cWhich, cReplacement).
```

Boolean test methods:

```
contentEquals(g), endsWith(g), equals(g),  
equalsIgnoreCase(g), matches(g),  
regionMatches(bIgnoreCase,i1,g2,i3,i4),  
startsWith(g)
```



Inputing Strings

Easiest String input is done using a standard Scanner object and the method `nextLine()`

```
Scanner input  = new Scanner(System.in) ;  
String aString = input.nextLine() ;
```


Escape Sequences

Useful special characters called **escape sequences**, which can be used both as single chars or in strings (in single or double quotes) are used to represent special text:

`\b` Backspace

`\t` Horizontal tab

`\n` New line (line feed)

`\"` Double quotation mark

`\'` Single quotation mark

`\\` Backslash

FORMATING TEXT OUTPUT

So far you have used the `print` and `println` method and `+` operator to output variable values:

```
int price = 1;
```

```
System.out.print("the price is "+a+" dollars" );
```

Another method called `format` involves the use of the `%d` (integers) and `%f` (floats or doubles) operators in text:

```
System.out.format("%d dollars %d cents", a, b);
```



FORMATING TEXT OUTPUT

```
double pi = 3.141593;
```

```
System.out.format("%f", pi);
```

```
//--> "3.141593"
```

```
System.out.format("%.3f", pi);
```

```
//--> "3.142"
```

```
System.out.format("%10.3f", pi);
```

```
//--> "      3.142"
```

FORMATING TEXT OUTPUT

You can use multiple %d or %f operators for many variables, and the %n operator for newlines.

```
int a=1, c=3;
```

```
double b = 2.2;
```

```
System.out.format("a equals %d %n b equals %.1f  
%n c equals %d %n",a,b,c);
```

```
/*
```

```
a equals 1
```

```
b equals 2.2
```

```
c equals 3
```

```
*/
```

String Task 1:

Prepare a program that echoes every line typed in by the user.

Use the **.nextLine ()** method for the Scanner object to read text.

Use an infinite loop.

Break the loop if user types “quit”.



String Task 2:

As task 1, but this time the program splits the text, so that each word is displayed in a new line (basically replace all spaces with new lines).

Use at least two of three ways:

- a) Use **.replace(oldcharsequence, newcharsequence)** to replace " " with "\n"
- b) Using a loop and the **.charAt(index)** method check for spaces
- c) As in b) but print substrings using **.substring(startindex, endindex)** method to copy out each word



String array exercise

Determine the output of the program:

```
int i, j;
String val[][] =
{{"a","b","c"}, {"d","e","f"}, {"g","h","j"}};

for (i = 0; i < val.length; i++)
{
    for (j = 0; j < val[i].length; j++)
    {
        System.out.print("val["+i+"]["+j+"]="+val[i][j]+"\\t");
    }
    System.out.print("\\n");
}
```



HOMEWORK

Generate an array of 100 random integers between 1 and 100.

Print the array. Print the average, min, and max number.

Sort the array from smallest to largest.

Example output:

Unsorted numbers:

88,99,12,13,14,15,5,6,3,99,12...

Average of numbers: 48.5667

Largest number: 99

Smallest number: 3

Sorted numbers: 3,5,6,12,13,14,15...

