Introduction to arrays

Recall the example program that was given at the very beginning of the course:

```
double a[] = {1.55556, 3.2123, 3.11, 1, 0, 2};

double m = 0;
int i = 0;

while (i < a.length)
{
    if (m < a[i])
    {
        m = a[i];
    }
    i = i + 1;
}
System.out.println("Maximum is " + m);
```

In this program, the variable "`a`" refers to a sequence of numbers stored in memory; this is called an *array*.

Arrays are a basic yet very powerful way to group values together in a program. Thanks to arrays, we (as programmers) do not need to give a separate, different name to every value in a sequence. For example instead of:

```
int x1 = 12;
int x2 = 23;
int x3 = 45;
int x4 = 67;
int x5 = 89;
int y = x1 + x2 + x3 + x4 + x5;
```

We can write:

```
int i, y;
int x[] = {12, 23, 45, 67, 89};

y = 0;
for (i = 0; i < x.length; i = i + 1)
   y += x[i];
```

In Java and many other programming languages, arrays are *homogeneous*: *all elements in the array must have the same type*.

To use arrays in Java programs, we must pay special and separate attention to the three following aspects:

- *declaring* an array, which *prepares a name* that can designate an array;

- *defining* an array, which *reserves space in memory* for all its elements;
- *using* an array, when the individual elements are accessed separately.

Using arrays is very simple, so we cover it first.

Array indexing

Syntax:

> <expression> `[` <expression> `]`
>
> (An expression, followed by an opening square bracket "`[`", followed by another expression, followed by a closing square bracket "`]`". The second expression is called the *subscript*.

Semantics:

> The expressions on the left and the right are evaluated. If the result of evaluating the first expression is an array, the value of the subscript is used as index, and the element in the array at that position is accessed. If the first expression does not evaluate to an array, an error is reported.

> **Note**
> The first element in an array has index 0. So if `x` is an array, `x[0]` is its first element.

Examples:

```
// read x's first element and places it in y
y = x[0];

// read x's 11th element and places it in y
y = x[10];

// read the i-th element of x and place it in y
y = x[i];

// change the 2nd element of x to 123
x[1] = 123;

// change the 100th element of x to 42
x[99] = 42;

// change the i-th element of x to 69
x[i] = 69;
```

Declaring and defining arrays in Java

In Java there exists a separation between the *declaration* of an array name, where a name is simply prepared, and the *definition* of an array which reserves space for the array elements in memory.

This separation only exists in a few languages: it exists in C/C++ and Java, but not in ML, Python, R, Matlab and many others.

Moreover, Java also provides a shorthand construct that combines declaration and definition in one to make them shorter. However, for learning purposes it is better to separate them.

Array declaration

Syntax:

> <type> `[ ]` <name> `;`
>
> (A type, followed by opening and closing square brackets without anything in between, followed by a name, followed by a semicolon)

Semantics:

> Prepare the name so that it can be later made to refer to an array.

For example:

```
int[] x;
double[] y;
String[] z;
```

This program fragment prepares the names `x`, `y` and `z` so they can *later* refer to arrays of integer, floating-point and string elements.

Note that the declaration merely prepares the name, so with only a declaration the array is not yet ready for use. For example the following program fragment is INCORRECT:

```
int[] x;

x[10] = 123; // incorrect: x not defined yet
```

Array definition

Syntax:

> `new` <type> `[` <expression> `]`
>
> (The keyword "`new`", followed by a type, followed by an opening square bracket, followed by an expression called the *size*, followed by a closing square bracket)

Semantics:

> First the size expression is evaluated. Then space is reserved in memory for an array able to store that number of elements of the type given on the left. The newly created array is returned as result for the entire construct.

Once an array is created this way, it can be bound to a name prepared by an earlier declaration. For example:

```
// declare the name "x"
int[] x;

// make an array of 100 int's, bind it to "x"
x = new int[100];

// now "x" is attached to an array,
// so we can modify it
x[10] = 123;
```

Combined array declaration and definition

The basic combination of a declaration and assignment, already seen in a previous lecture, applies here:

<type> <name> = <expression> ;

So we can simply combine as follows:

```
int[] x = new int[100];

// is equivalent to:
int[] x;          // prepare a name
x = new int[100]; // reserve space
```

Combined declaration and definition (alternative)

You will encounter example programs or documentation that use another way to expression the combination of an array declaration and definition:

<type> <name> [<expression>];

For example:

```
int x[100];

// nearly equivalent to:
int[] x = new int[100];
```

This construct is used often, mostly because it is shorted to write. However it is subtly different from the construct above, for reasons that will be explained in a later lecture.

For the time being, you should consider only the first form above in your programs.

Size of an array

In Java the *size* of any array can be queried using a special construct, separate from indexing:

Syntax:

<expression> `.length`

Semantics:

Evaluates to the number of elements of the array designated by the expression on the left.

So if `x` designates an array, `x.length` designates its size (number of elements).
For example:

```
int[] x = new int[100];

System.out.println(x.length);
```

This program fragments prints 100.

Examples

1) The following program reads 10 numbers into an array then computes and prints their sum:

```
int[] x = new int[10];
int i, sum;

Scanner in = new Scanner(System.in);
for (i = 0; i < x.length; i = i + 1)
   x[i] = in.nextInt();

sum = 0;
for (i = 0; i < x.length; i = i + 1)
   sum = sum + x[i];
System.out.println(sum);
```

2) **Read an analyze the program Histogram provided in the accompanying source code repository.**

Important concepts

- Construct for array indexing, *subscript* expression;
- Difference between *declaration* and *definition*;
- Construct for array declaration;
- Construct for array definition;
- Combined array declaration and definition using `new`;
- Size of an array with `.length`.

Further reading

- Think Java, chapter 12 (pp. 149-151, up to and including section 12.2), section 12.5 (p. 153), 12.8-12.10 (pp. 155-158)
- Introduction to Programming, section 3.8 up to and including 3.8.4 (pp. 114-120)
- Absolute Java, section 6.1 (pp. 346-355)

Copyright and licensing

Copyright © 2014, Raphael 'kena' Poss. Permission is granted to distribute, reuse and modify this document according to the terms of the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/.