

Lecture - Java Arrays

Outline

Introduction to arrays

- Creating and accessing arrays
- Use for loops with arrays

Arrays and references

- Programming with arrays
- Sorting

ArrayList

Multidimensional arrays

Introduction to Arrays

An array is a data structure used to process a collection of data that is all of the same type. An array behaves like a numbered list of variables with a uniform naming mechanism

- It has a part that does not change: the name of the array
- It has a part that can change: an integer in square brackets

For example, given five scores:

```
score[0], score[1], score[2], score[3], score[4]
```

Declaring and creating an array

An array that behaves like this collection of variables, all of type double, can be created using one statement:

```
double[] score = new double[5];
```

variable
hold five double values.

Or using two statements:

```
double[] score;  
score = new double[5];
```

length of array is five.

The first statement declares the variable score to be of the array type double[].

The second statement creates an array with five numbered variables of type double and makes the variable score a name for the array.

Naming

The individual variables that together make up the array are called *indexed variables*

- They can also be called *subscripted variables* or *elements* of the array

The number in square brackets is called an *index* or *subscript*

In Java, indices must be numbered starting with 0.

Advanced: This is because of how the array is laid out in memory. A reference to the array refers to the first element in the array. The *i*th element is stored at <array reference> + *i* * (size of element). The element stored at <array reference> is number *i*=0.

Sizes

The number of indexed variables in an array is called the *length* or *size* of the array. When an array is created, the length of the array is given in square brackets after the array type. The indexed variables are then numbered starting with 0, and ending with the integer that is **one less than** the length of the array

```
double[] score = new double[5];  
score[0] = score[1] = score[2] = score[3] = score[4] = 0;
```

length of array five here
access the array is start from 0.

Indexing

So far, we haven't done anything that couldn't be done with variables `score0`, `score1` etc.

The value of arrays is that any integer expression (such as a variable) can be placed in the square brackets to say which variable we are referring to. That lets us loop over variables.

```
class ArrayDemo {
    public static void main (String[] arg) {
        double[] score = new double[5];

        for (int i = 0; i < score.length; i++) {
            score[i] = 10 + i;
        }

        for (int i = 0; i < score.length; i++)
            System.out.print(score[score.length - (i+1)] + " ");
    }
}
```

be careful of the boundary
of the array.

Size chosen at run time

Even the size of the array can be chosen at run-time. A variable or integer expression can replace the [5] in the above code. Modify the code above to generate a random number between 0 and 10, using `Math.random` (Hint: modify the template `int rand = (int)(Math.random() * range) + min;`), and use that as the size of the array.

Base types

You can create an array of any type, including class types. However, all elements in the array must have the same type. This type is called the *base type* of the array.

Advanced: If you want to have an array of different types of objects, you can use polymorphism, which will be covered much later in this course.

An array is declared and created in almost the same way that objects are declared and created:

```
BaseType[] ArrayName = new BaseType[size];
```

The `size` may be given as an expression that evaluates to a nonnegative integer, for example, an `int` variable.

```
char[] line = new char[80];
double[] reading = new double[count];
Person[] specimen = new Person[2 * line.length];
```

```
import java.util.Scanner;

// Array of scores
public class ArrayOfScores {
    /**
     * Reads in 5 scores and shows how much each
     * score differs from the highest score.
     */
}
```

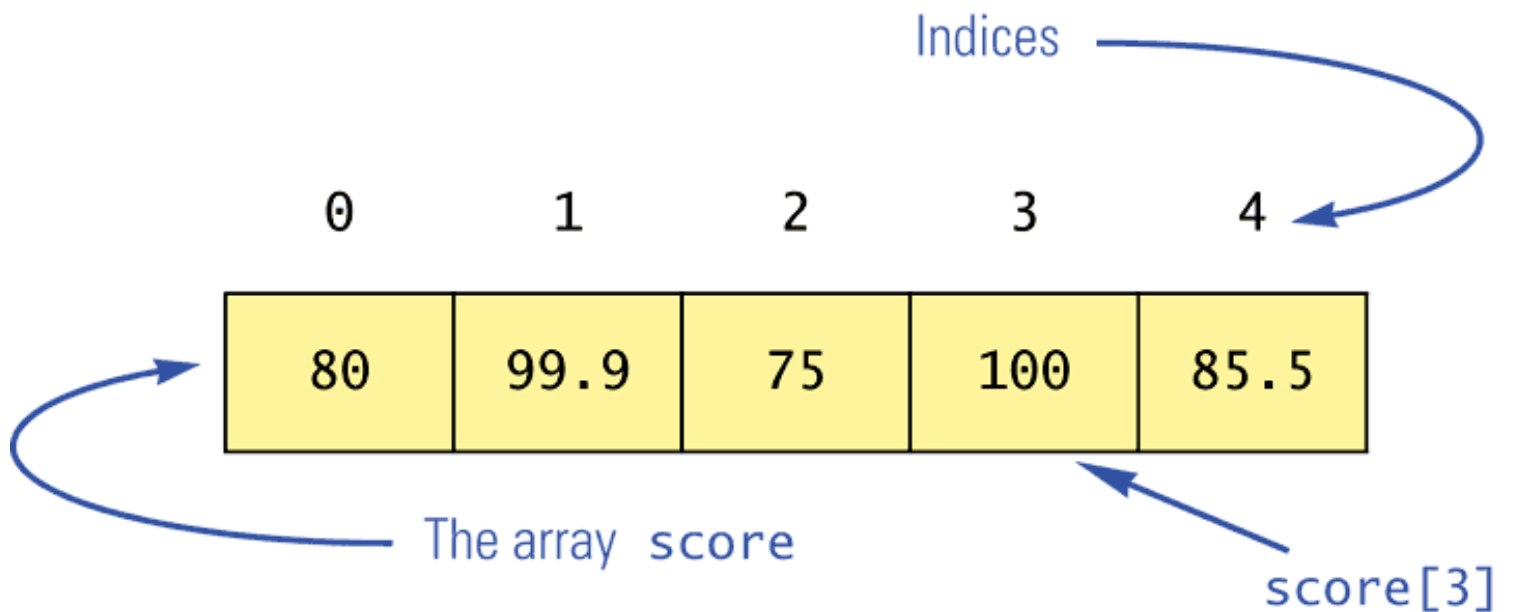
```

*/
public static void main (String[] args) {
    Scanner keyboard = new Scanner(System.in);
    double[] score = new double [5];
    int index;
    double max;

    System.out.println("Enter 5 scores:");
    score[0] = keyboard.nextDouble();
    max = score[0];
    for (index = 1; index < score.length; index++) {
        score[index] = keyboard.nextDouble();
        if (score[index] > max)
            max = score[index];
        // max s the largest of the values score[0]...score[index]
    }

    System.out.println("The highest score is " + max);
    System.out.println("The scores are:");
    for (index = 0; index < 5; index++)
        System.out.println(score[index] +
            " differs from max by " + (max - score[index]));
    }
}

```



Referring to arrays and array elements

Each array element can be used just like any other single variable by referring to it using an indexed expression

```
score[0]
```

The array itself (i.e., the entire collection of indexed variables) can be referred to using the array name without any square brackets

```
score
```

call by reference.

You would use this form if you wanted to pass the array as a parameter to a method, for example.

An array index can be computed when a program is run.

- It may be a single variable: `score[index]`
- It may be an expression that evaluates to an integer in the right range: `score[next+1]`

Three ways square brackets are used with an array name

Square brackets are used in three totally different ways with arrays.

- The most common way: Square brackets are used to name an element of an array:

```
max = score[0];
```

- Square brackets can be used with an integer value to create a new array

```
score = new double [5];
```

- Square brackets can be used to create a type name. In this context, the brackets are only used to mean "this is an array", because brackets are associated with arrays due to the first two uses.

```
double [] score;
```

The length instance variable, and out-of-bounds errors

An array is considered to be an object

Since other objects can have instance variables, so can arrays

Every array has exactly one instance variable named `length`

When an array is created, the instance variable `length` is automatically set equal to its size.

The value of `length` cannot be changed (although a new array of a different size can be assigned to the same variable).

```
double[] score = new double[5];           // score.length now has a value of 5
```

Advanced: *Note that this is an instance variable, not a member method. Some classes, such as `String` have a method named `length()` for a similar role, and others have a method named `size()`. You can remember that arrays are the odd one out (variable not method) because they aren't classes.*

Pitfall: Array index out of bounds

Java array indices always start with **0**, and always end with the integer that is **one less than the size of the array**. The most common programming error made when using arrays is attempting to use a *nonexistent array index*.

When an index expression evaluates to some value other than those allowed by the array declaration, the index is said to be out of bounds. An out of bounds index will cause a program to terminate with a **run-time error message**.

Array indices get out of bounds most commonly at the *first* or *last* iteration of a loop that processes the array: Be sure to test for this! (off-by-one error).

Advanced: *Other languages handle index out of bounds differently. In some languages, an array will automatically grow when you try to assign to an index that is larger than the size of the array, or return a default value (typically 0) when you try to read from one.*

Initializing arrays.

An array can be initialized when it is declared

Values for the indexed variables are enclosed in braces, and separated by commas

The array size is automatically set to the number of values in the braces

```
int[] age = {2, 12, 1};    //no need to new
```

Given age above, `age.length` has a value of 3

Another way of initializing an array is by using a for loop:

```
class ArrayExample {  
    public static void main (String[] args) {  
  
        double[] reading = new double[100];  
        for (int index = 0; index < reading.length; index++)  
            reading[index] = 42.0;  
    }  
}
```

double score = {0, 10, 100}

If the elements of an array are not initialized explicitly, they will automatically be initialized to the default value for their base type.

Arrays are Objects

Arrays are objects, and in most ways behave like class objects.

- An array variable holds a reference to where the actual array is stored in memory

An array can be viewed as a collection of indexed variables.

It can also be viewed as a single item whose value is a collection of values of the base type.

- An array variable names the entire array as a single item: `double[] a`
- A new expression creates an array object and stores the object in memory: `new double[10]`
- An assignment statement places a reference to the memory address of an array object in the array variable: `a = new double[10];`

The previous steps can be combined into one statement:

```
double[] a = new double[10];
```

Note that the new expression that creates an array invokes a constructor that uses a nonstandard syntax. (There are [] but not () .)

Note also that as a result of the assignment statement above, `a` contains a single value: a memory address or reference.

Week 5 stop

Methods that return arrays

In java, a method may also return an array.

The return type is specified in the same way that an array parameter is specified.

```
class ArrayReturnDemo {
    public static double [] incrementArray(double[] a, double increment) {
        double [] temp = new double[a.length];
        for (int i = 0; i < a.length; i++)
            temp[i] = a[i] + increment;
        return temp;
    }

    public static void main (String[] args) {
        // Fill in if you want to call the above method
    }
}
```

Exercise: Add a `main` function to initialize an array to your choice of values (using an initializes like `{..., ..., ...}`), increment by one and then print the values.

Exercise: How could you avoid needing an internal array, `temp`? Algorithms that don't allocate extra space are called "in place" algorithms, and they are very important when dealing with large data sets.

- What side-effects would there be of the in-place algorithm?

The in place algorithm wouldn't strictly need to return the array as a return value. However, it is useful as it allows function chaining, like `squareRoot (incrementArray(a, 1))`.

Pitfall: An array of characters is not a String

An array of characters is conceptually a list of characters, and so is conceptually like a string

However, an array of characters is not an object of the class `String`. (This may be a surprise to C programmers.)

```
char[] a = {'A', 'B', 'C'};  
String s = a; //Illegal!
```

An array of characters can be converted to an object of type `String`, however.

The class `String` has a constructor that has a single parameter of type `char[]`.

```
String s = new String(a);
```

↳ print s → ABC

The object `s` will have the same sequence of characters as the entire array `a` ("ABC"), but is an independent copy.

Another `String` constructor uses a subrange of a character array instead:

```
String s2 = new String(a,0,2);
```

Given `a` as before, the new string object is "AB".

An array of characters does have some things in common with `String` objects. For example, an array of characters can be output using `println`

```
System.out.println(a);
```

Given `a` as before, this would produce the output

ABC

Pitfall: = and == with arrays

Since an array is a reference type, the behavior of arrays with respect to assignment (=), equality testing (==), and parameter passing are the same as that described for classes.

The assignment operator (=) only copies this memory address. It does not copy the values of each element.

- Using the assignment operator will make two array variables be different names for the same array

 `b = a;` same reference

- The memory address in `a` is now the same as the memory address in `b`: They refer to the same array.

A `for` loop is usually used to make two different arrays have the same values in each indexed position:

```
int i;  
for (i = 0; (i < a.length) && (i < b.length); i++)  
    b[i] = a[i];
```

 not same reference

Note that the above code will not make b an exact copy of a, unless a and b have the same length.

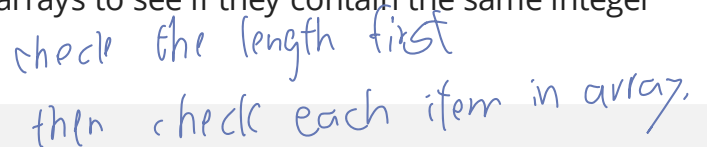
For the same reason, the equality operator (==) only tests two arrays to see if they are stored in the same location in the computer's memory. It does not test two arrays to see if they contain the same values.

```
(a == b)
```

The result of the above boolean expression will be `true` if `a` and `b` share the same memory address (and, therefore, reference the same array), and `false` otherwise

In the same way that an `equals` method can be defined for a class, an `equalsArray` method can be defined for a type of array. This is how two arrays must be tested to see if they contain the same elements. The following method tests two integer arrays to see if they contain the same integer values

```
class Main {  
    public static boolean equalsArray(int[] a, int[] b) {  
        if (a.length != b.length)  
            return false;  
        else {  
            int i = 0;
```

 check the length first
then check each item in array.

```
        while (i < a.length)
        {
            if (a[i] != b[i])
                return false;
            i++;
        }
    }
    return true;
}

public static void main (String[] args) {
    // Fill in if you want to call the above method.
}
}
```

Argument for the method main

You have actually been writing code with arrays since your first java program.

The heading for the `main` method of a program has a parameter for an array of `String`

It is usually called `args` by convention, since it contains the command line arguments.

```
public static void main(String[] args)
```

Note that since `args` is a parameter, it could be replaced by any other non-keyword identifier.

If a Java program is run without giving an argument to `main`, then a default empty array of strings is automatically provided.



Unlike in C/C++ and python, `args[0]` is the first command line argument. The executable name is not passed to the executable.

The shell

When a java program is called from a command line, some arguments are interpreted by the "shell" (the program that interprets commands and calls the correct programs). Different shells interpret special characters in different ways.

In the calculator example, you saw that a Linux/Unix shell (bash) interprets the `*` character, and so you had to put quotes around it ("`*`" instead of `*`). This shows that the Unix shell also interprets the double quote character. In fact, it interprets all of the following characters:

```
! $ ^ * ? ( ) { } ~ ` ' " \ <space> <> | ;
```

The Windows shell (cmd) also interprets several : `" \ / % <space> <> |`

To stop the shell from interpreting these, you can put a `\` before them. To make matters more confusing, these are not always interpreted. In the example below, you can omit the `\` before `!` even though it is interpreted in some contexts.

(These lists are not exhaustive; if you think of any others, please let me know and I will update the list.)

Example

What would the following code do if it is run as below?

```
java SomeProgram Hi \! there
```

In particular, what values would the variables `arg[0]`, `arg[1]` and `arg[2]` have?

Reading Resources

Additional Reading Resources

- WALTER, S. Absolute Java, Global Edition. [Harlow]: Pearson, 2016. (Chapter 6)
- SCHILDT, H. Java: The Complete Reference, 12th Edition: McGraw-Hill, 2022 (Chapter 3)
- Array (accessible on 14-02-2024) Oracle's Java Documentation. Available at:
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>.
- Arrays (accessible on 14-02-2024) Java 7 Documentation. Available at:
<https://docs.oracle.com/javase/specs/jls/se7/html/jls-10.html>.