Arrays

James Brucker

Array

An array is a sequence of elements of the same type; a single variable (x) refers to the whole series.

```
float[] x = new float[10]; // array of 10 values
```

Refer to each element using an **index**, starting at 0.

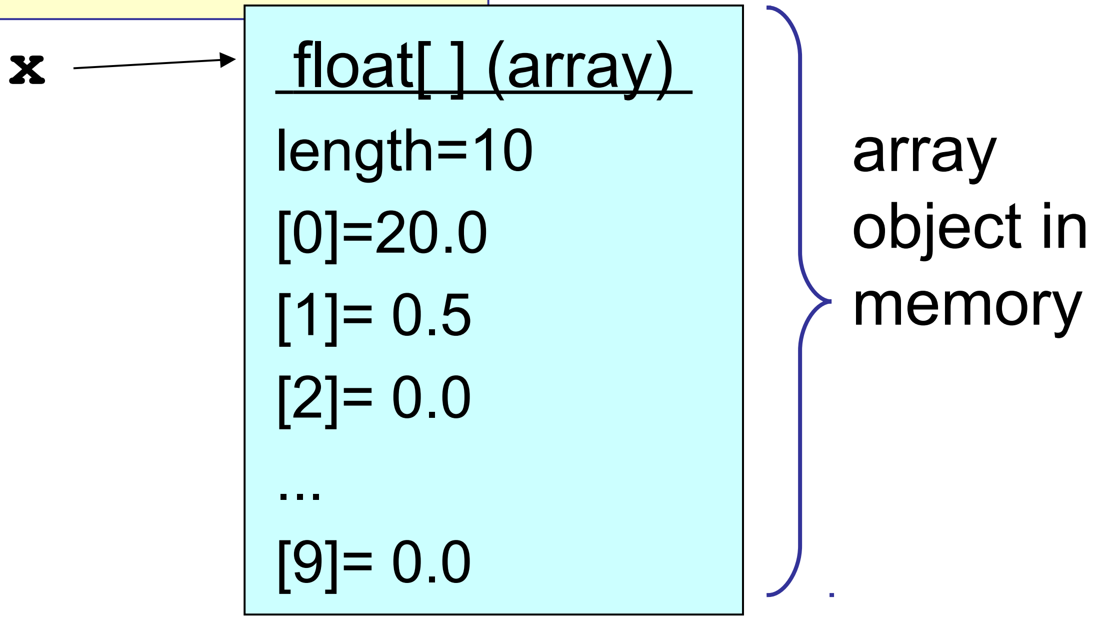
```
x[0] = 20;           // first element of x
x[1] = 0.5F;         // 2nd element of x
x[2] = -3;
x[9] = 9000;         // last element has index 9
                     // (not 10)
```

Structure of an array

The first element has index 0.

An array has a fixed length (size cannot be changed).

```
float[] x = new float[10];
x[0] = 20;
x[1] = 0.5F;
```



Array knows its own size!

Every array has an *attribute* named `length`

```
    double[] x = new double[20];
    x.length    // returns 20
```

`x.length is 20.`

The *first* element is `x[0]`,

the ***last*** element is `x[x.length -1]`.

Don't forget -1 !

In Java, an array is an ***object***.

`length` is a **property** (attribute) of the array object.

Why Use Arrays?

- Make it easy to process lots of data using loops.
- Perform operations on vectors and matrices.

Examples are given in later slides.

3 Steps to create an array

There are 3 steps to define & initialize an array.

Memorize them! A common programming error is to omit one of these steps.

1. Define array variable (reference)	`double[ ] x;`	`String[ ] colors;`
2. Create the array & specify its size.	`x = new double[10];`	`colors = new String[3];`

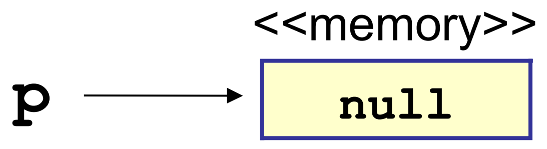
3. Assign values to array elements.	`x[0] = 10.0;`
	`x[1] = 0.5;`
	`. . .`
		`colors[0] = "red";`
		`colors[1] = "blue";`
		`colors[2] = "green";`

1. Define array reference

Declare p as type "array of int".

OK to omit space after "int" and between [ ].

```
int [] p;
```



This creates an array *reference* p,
but does not create an array.

p does not refer to anything yet!
Just like:

```
        String s;
```

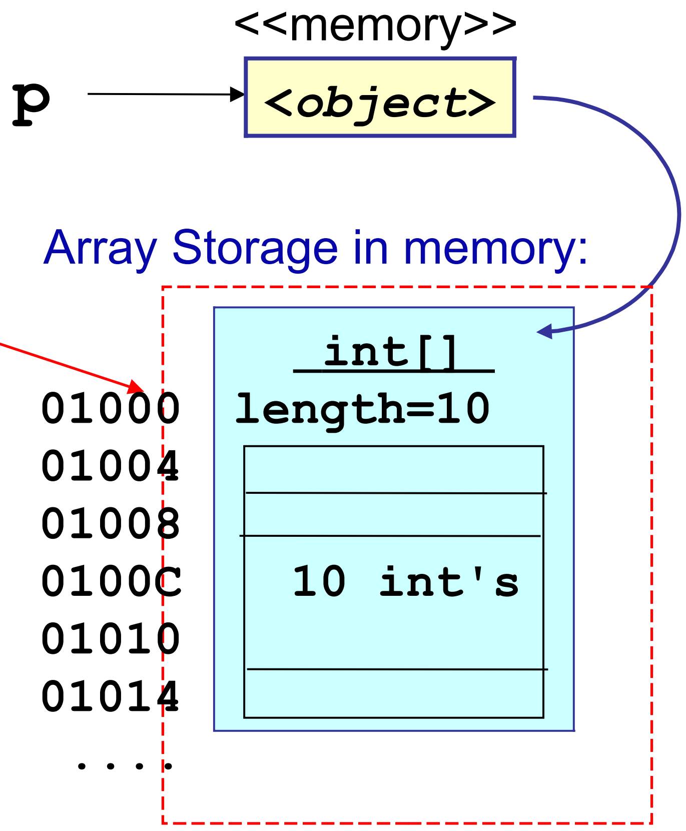
defines a String *reference* but does not create a string.

2. Create the Array object

Create the array using "new".

```
array = new DataType[ size ]
```

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

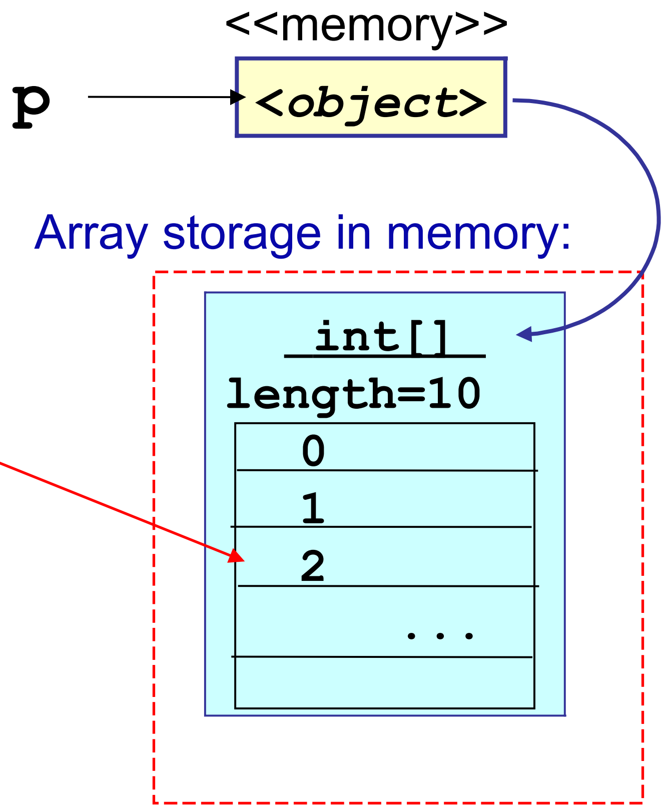


"new" creates a new object.
Here, it creates an *array* containing 10 "int" values.
It sets p to *refer* to this object.

3. Initialize elements of the array

When you create the array, Java does not initialize the array elements. You must do this.

```
for(int k=0; k < 10; k++)
    p[k] = k;
```



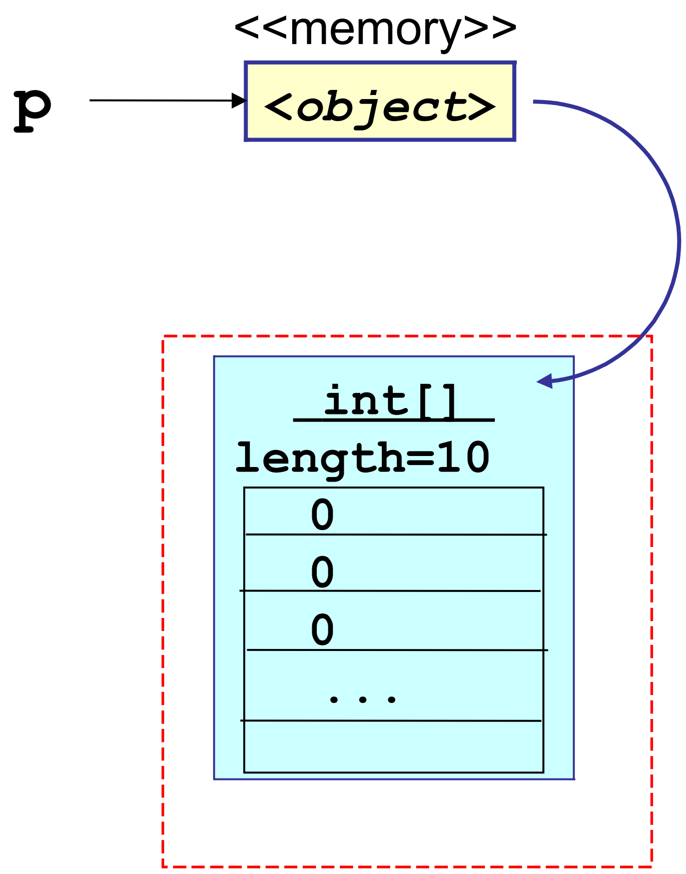
You can initialize array elements any way you like.

Some examples in later slides.

Short-cut to create an Array

You can combine steps (1) and (2) into one statement:

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



This statement does two things:

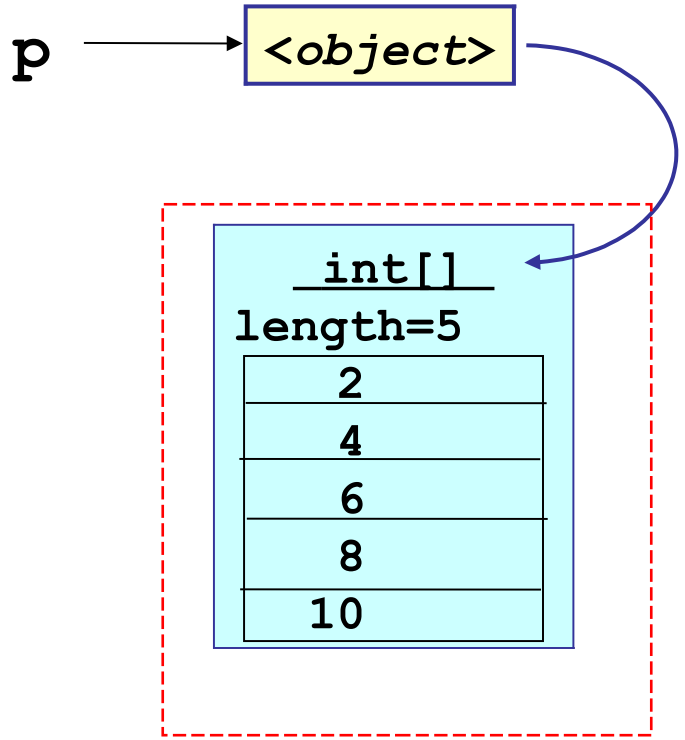
1) define p as an array reference

2) create an array with 10 elements and assign it to p

Another short-cut

If you have fixed values to put in the array, you can combine steps 1 - 3 into one statement:

```
int[] p = { 2, 4, 6, 8, 10};
```



This statement does 3 things:

1) define p as an array reference
2) create array with 5 int's
3) stores values 2, 4, ... 10 in the array

Summary: steps to create array

1. Define an array reference:

```
double [] x;
```

2. Create the array (allocate storage for elements) :

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

3. Assign values to the array elements:

```
for(int k=0; k<x.length; k++) x[k] = 2*k;
```

Short-cut: define array reference and create object

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

Meaning of `[]` in "`String[] x`"

The **`[ ]`** means *"array of ..."* or *"... array"*.

- **`int[]`** means "*int array*" or "*array of int*".
- **`Foo[]`** means "*Foo array*" or "*array of Foo*".

`int[ ] x;`	x is <u>type</u> "int array"
`main(String[] args)`	args is <u>type</u> "String array"
`char[] c = {'c','a','t'}`	c is <u>type</u> "char array"
`double[] getScores()`	getStores returns "array of double"

Inspect an array using BlueJ

Demo in class.

Use BlueJ to see inside an array

(called *inspection*)

Example: an Array to hold data

Suppose we have some numbers we want to store in an array, and compute the average.

The input data looks like this:

```
10        (number of values to read)
83.4      (first data value)
72.5      (second data value)
 .
 .
 .
92.0      (last data value)
```

What to do

1. Read the first line (size of the data): int size = 10
2. Create array to hold the values
3. Read all the values

```
10
83.4
72.5
  .
  .
  .
92.0
```

Code (1) - read into an array

```
Scanner console = new Scanner(System.in);
// read size of data and create the array
int size = console.nextInt();
double[] data = new double[size];

// read all the data or until array is full
int count = 0;
while( console.hasNextDouble() &&
       count < data.length )
{
    data[count] = console.nextDouble();
    count++;    // same as: count = count + 1
}
```

Code (2) - compute average

```
// Compute the average
double sum = 0.0;
for(int k=0; k<count; k++) sum = sum + data[k];

double average = sum/count;

System.out.printf("The average is %f\n",average);
```

Notice: using an array we can *easily* process all the data in a loop. Just 1 line (or 2 lines) of code!

We can also use a "for-each" loop that is even simpler:

```
for(double x: data) sum = sum + x;
```

Array as parameter

Use the same syntax as declaring an array variable.

```
/** Print the array elements. */
public void printArray( String[] array ) {
    for(int k=0; k< array.length; k++)
        System.out.printf("[%d] = %s\n",
                          k, array[k] );
```

```
/** Return maximum element in array. */
public double max( double[] array ) {
    double max = array[0];
    for(int k=1; k<array.length; k++) {
        if (array[k] > max) max = array[k];
    return max;
}
```

main has String array param

The main method accepts array of Strings.

```
/** args = command line arguments */
public static void main( String[] args ) {
    for(int k=0; k < args.length; k++)
        System.out.printf("args[%d] = %s\n",
                          k, args[k] );
```

The parameters to `main` are strings given on command line when running the class in the JVM.

For example:

```
cmd>  java MyClass hi there
```

```
args[0] = "hi"
```

```
args[1] = "there"
```

Method can return an array

A method can return an array:

```
/** Create an array and fill it with "1" */
static double[] makeOnes(int size) {
    double x = new double[size];
    // use Arrays.fill() is better
    for(int k=0; k<size; k++) x[k]=1;
    return x;
}
```

Avoid this Common Mistake!

What does "`b = a`" do?

What will be printed?

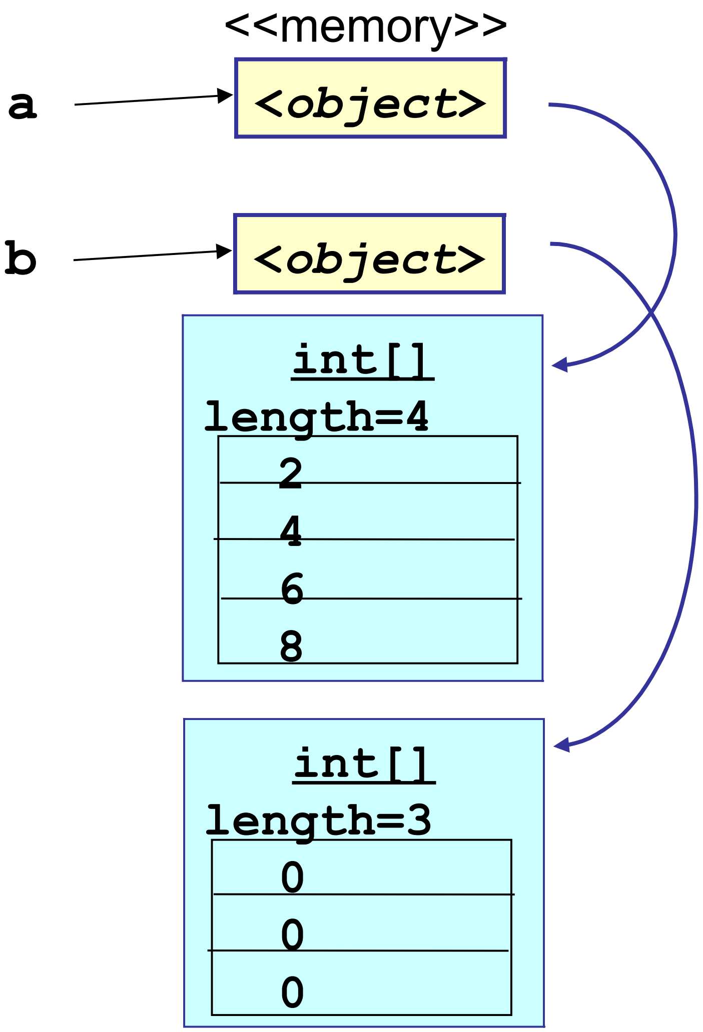
```
int [] a = { 2, 4, 6, 8 };
int [] b = { 0, 0, 0 };
b = a;                     // What does this do?
b[2] = 999;
System.out.println( a[2] );
System.out.println("b.length=" + b.length );
```

An Array Variable is a *Reference*

What does "`b = a`" do?

What will be printed?

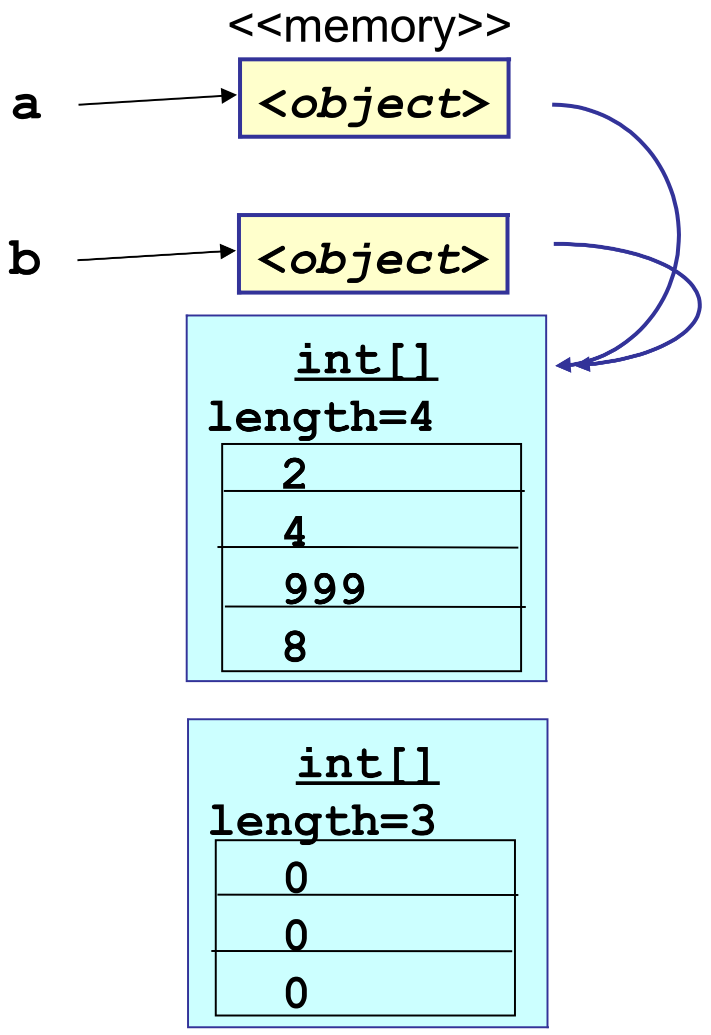
```
int [] a = { 2, 4, 6, 8 };
int [] b = { 0, 0, 0 };
b = a;       // Does what?
b[2] = 999;
System.out.println(a[2]);
System.out.println(
  "b.length=" + b.length );
```



"b = a" copies the *reference, not the array*

`b = a;`
makes **b** refer to same array as **a**.

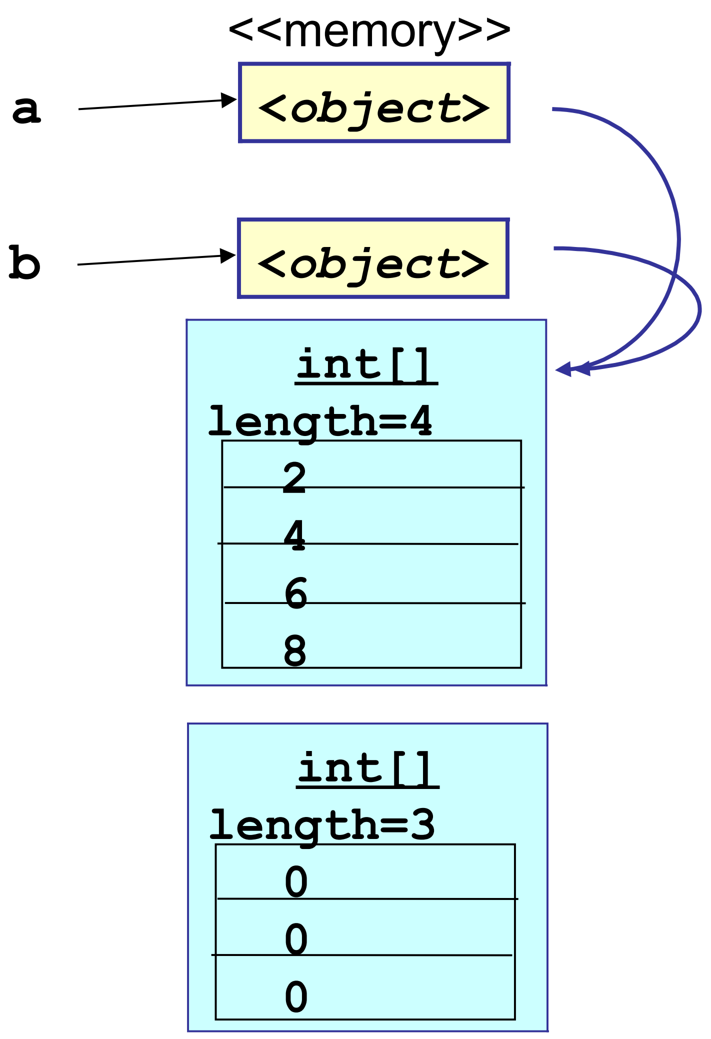
```
b = a;
b[2] = 999;
System.out.println(a[2]);
System.out.println(
  "b.length=" + b.length );
```



The result:

```
b = a;
b[2] = 999;
System.out.println(a[2]);
System.out.println(
 "b.length=" + b.length );
```

```
999
b.length = 4
```



How do you *really* copy an array?

See the next part of this lecture. :-)

Here is one solution:

```
int[] a = { 2, 4, 6, 8 };


// java.util.Arrays.copyOf( ... )
// creates a new array for copy.
int[] b = Arrays.copyOf( a, a.length );
```

See also: `System.arraycopy( ... )`