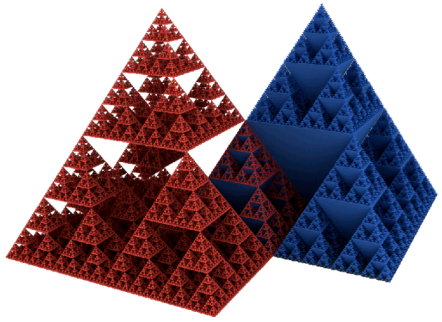
COMPSCI 101
Principles of Programming

Lecture 13

Arrays (chapter 15)



Review

Looping is a common program activity. Loops normally can be decomposed into four parts:
continuation test,
body,
updating the continuation test components,
initialisation.

We have seen how to use both *while* and *for* loops.

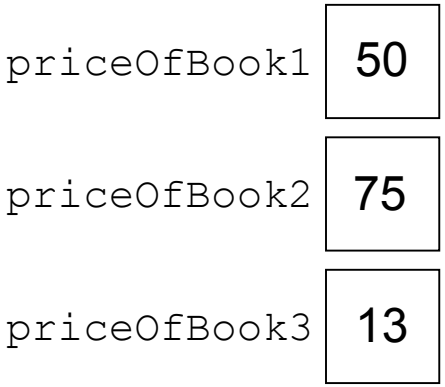
We have also seen how to use *break* statements to "exit" loop statements and to use *continue* statements to "skip" the rest of the current iteration of the loop and "continue" on to the next iteration.

Arrays

Why use arrays?

Let's say we wanted to store the prices of several books. We might declare three variables of type **`int`**:

```
int priceOfBook1;
int priceOfBook2;
int priceOfBook3;
```



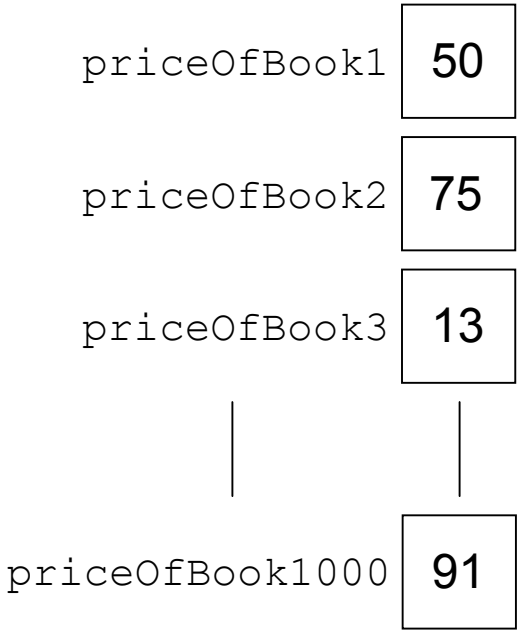
How would we calculate the total cost of the books?

```
int totalPrice = priceOfBook1 + priceOfBook2 +
                                    priceOfBook3;
```

Why arrays?

What if we now have 1000 books?

```
int priceOfBook1;
int priceOfBook2;
int priceOfBook3;
....
int priceOfBook1000;
```



Now calculating the total cost of the books is very cumbersome:

```
int totalPrice = priceOfBook1 + priceOfBook2 +
                 priceOfBook3 + priceOfBook4 +
                 ... + ... + ... + ...
                 + priceOfBook1000;
```

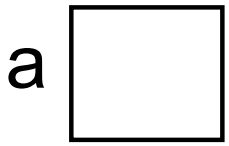
Analogy

single variable

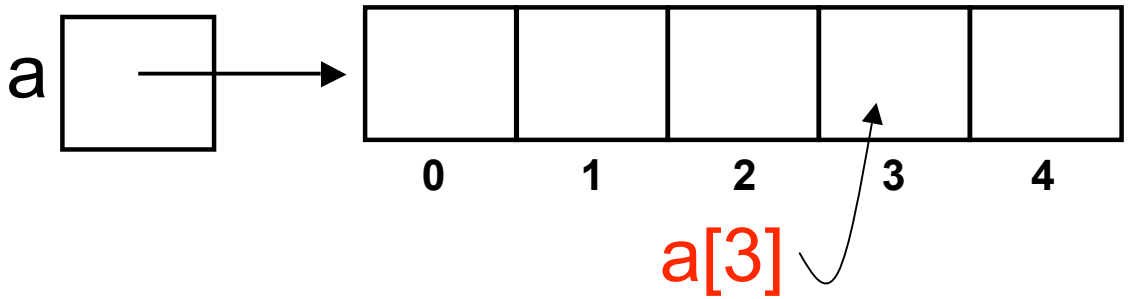
8172 Green St

array

3 / 156 Green St



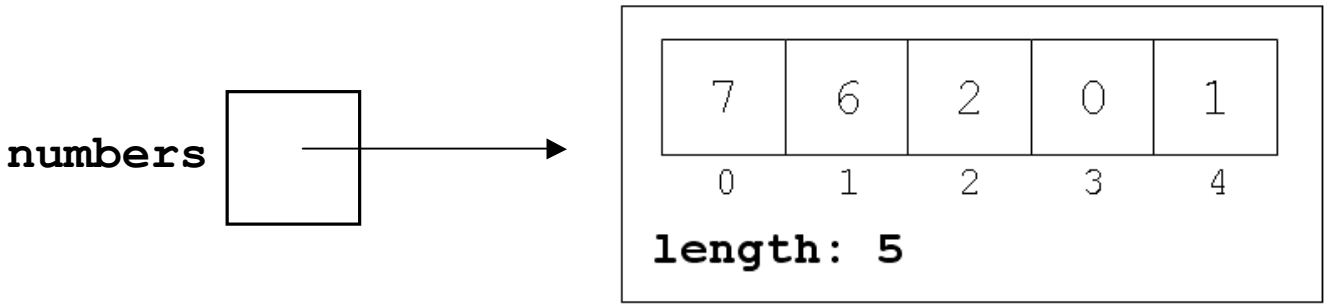
Referred to as: a



One dimensional arrays

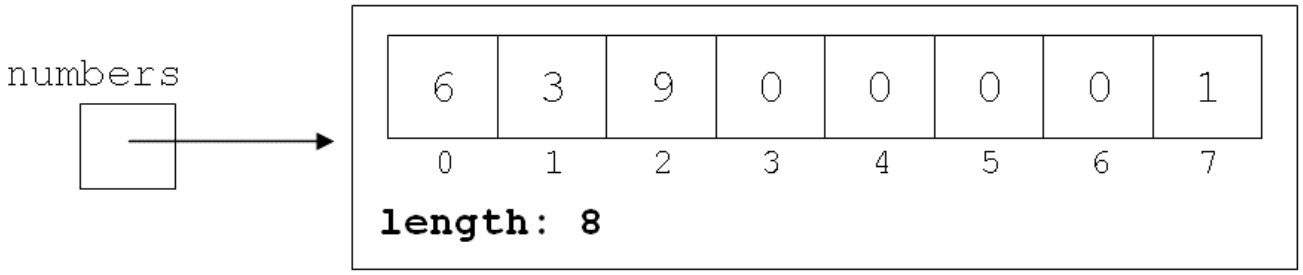
An array:

- holds a sequence of elements of the ***same type***
- each ***element*** in the array is associated with an ***index*** value
- index values start at 0
- every array has a ***length*** field which stores the number of elements in the array
- ***arrays are objects***

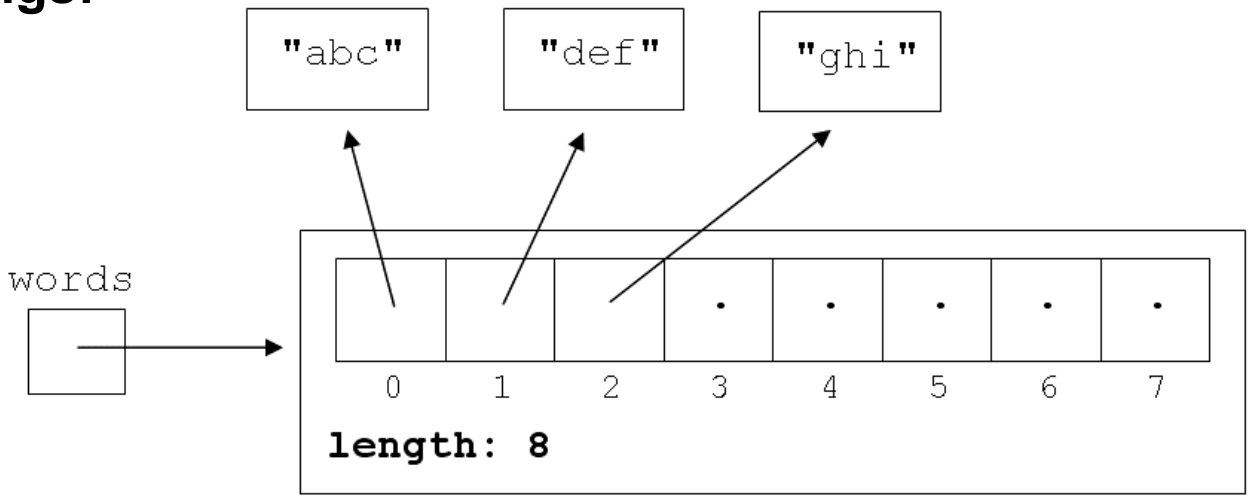


Arrays of different types

An array of integers:



An array of Strings:



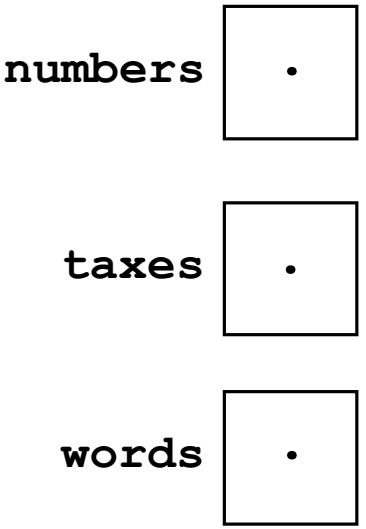
Declaring an array variable

Formal syntax:

```
<type>[] <identifier>;
```

For example:

```
int[] numbers;
double[] taxes;
String[] words;
```



These declarations do **not** set aside any space for the elements of the arrays

After the declarations, these variables store the value `null` (which means they don't point to anything)

Creating an array

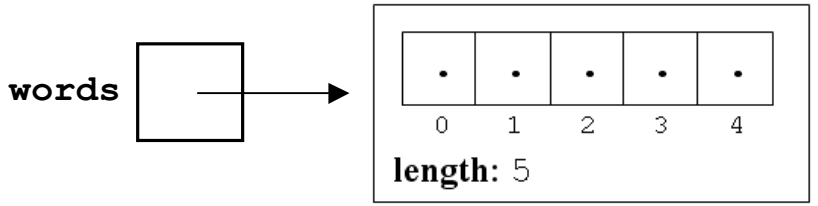
Formal syntax:

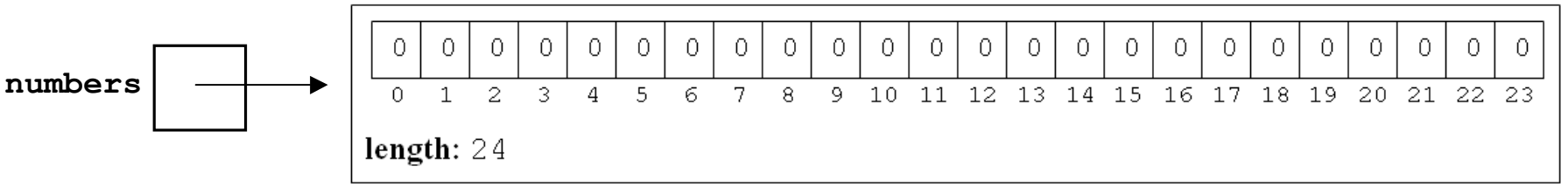
```
<identifier> = new <type>[<number of elements>];
```

For example:

```
numbers = new int[24];
taxes = new double[912];
words = new String[5];
```

choose carefully – we cannot ***change the size*** of an array once it has been created





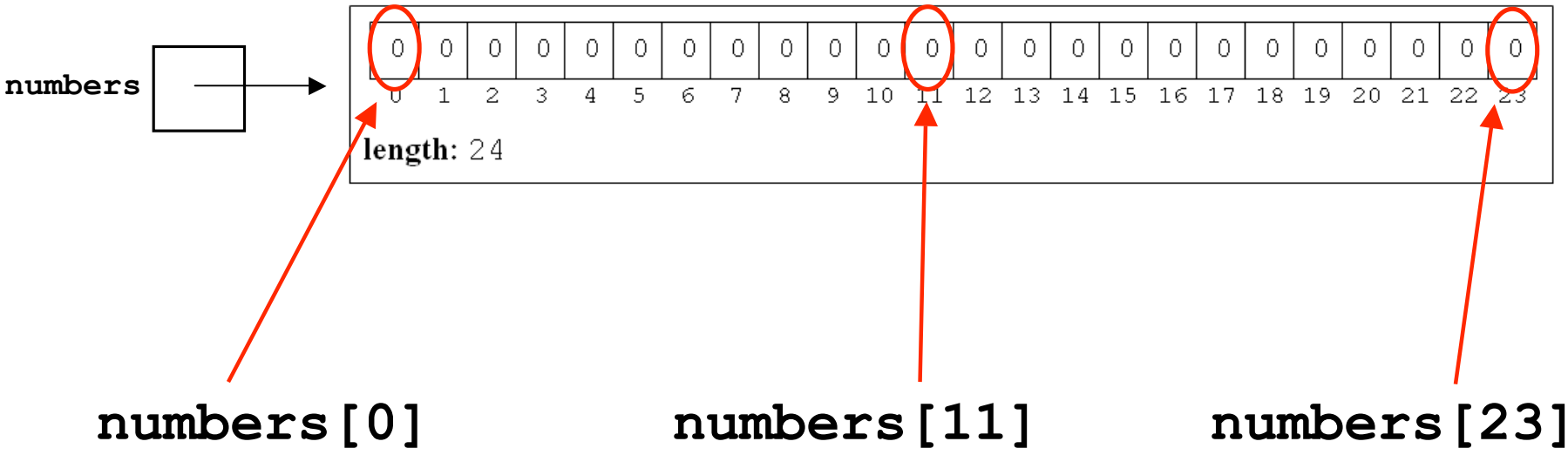
Initial values for elements

When we create an array, all the elements are initialised automatically to standard default values:

Type	*Default value*
Whole-valued numeric types (`byte`, `short`, `int`, `long`)	`0`
Real-valued numeric types (`float`, `double`)	`0.0`
`boolean`	`false`
`char`	`\u0000`
All object types	`null`

Using an array

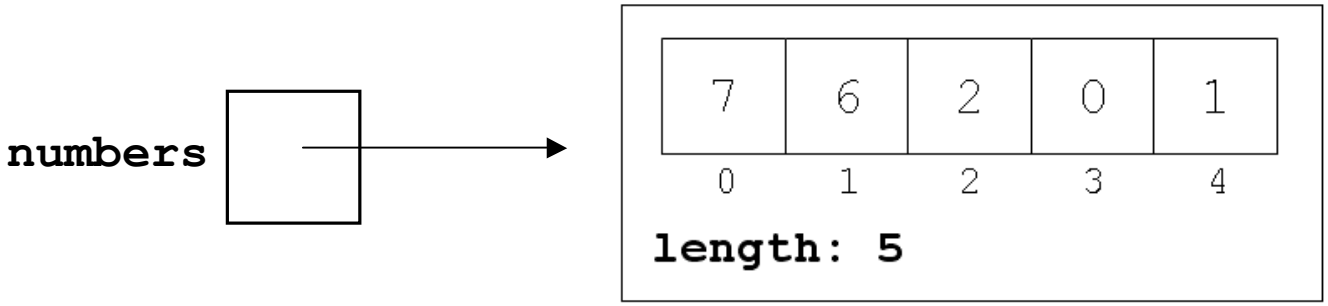
The most important feature of an array is the ability to refer to a specific element by using its index number:



`numbers[24]` **is a run-time error**

Arrays and loops

We can access every element in an array systematically using a loop:




```
numbers[i]
```

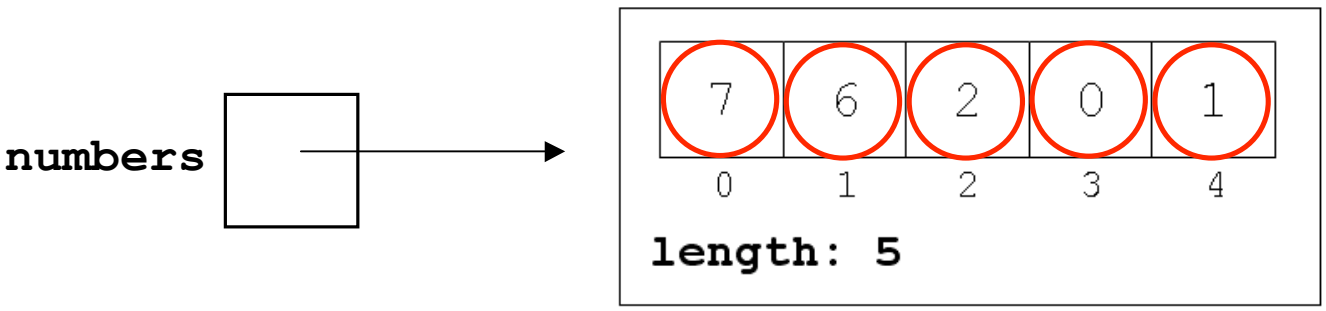
`i` is a loop variable which counts from 0 up to the last index of the array

Arrays and loops

This is commonly done with a `for` loop like:

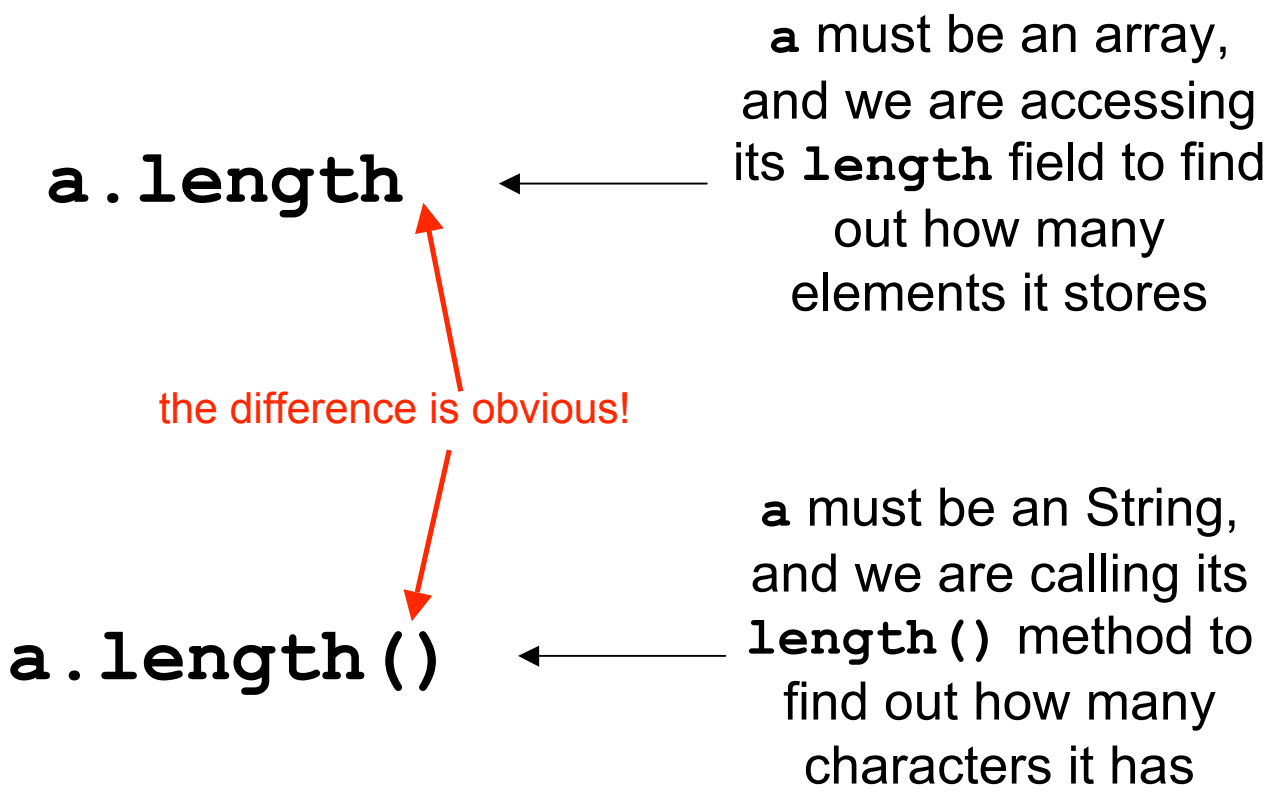
this is how we access the `length` field of an array

```
for (int i = 0; i < numbers.length; i++) {
      System.out.println(numbers[i]);
}
```



The length field

Don't confuse accessing the `length` field of an array with calling the `length()` instance method on a String:



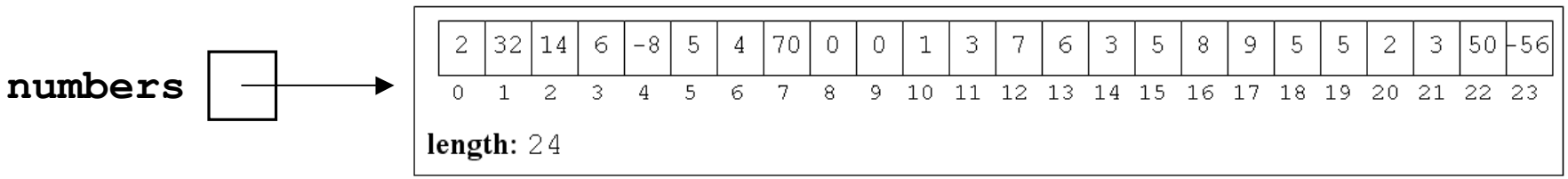
Initialising an array

We can assign each element a value in turn:

```
numbers[0] = 2;
numbes[1] = 32;
.....
numbers[23] = -56;
```

Or, we can construct the array with an *array initialiser* statement:

```
int[] numbers = {2, 32, ....., -56};
```



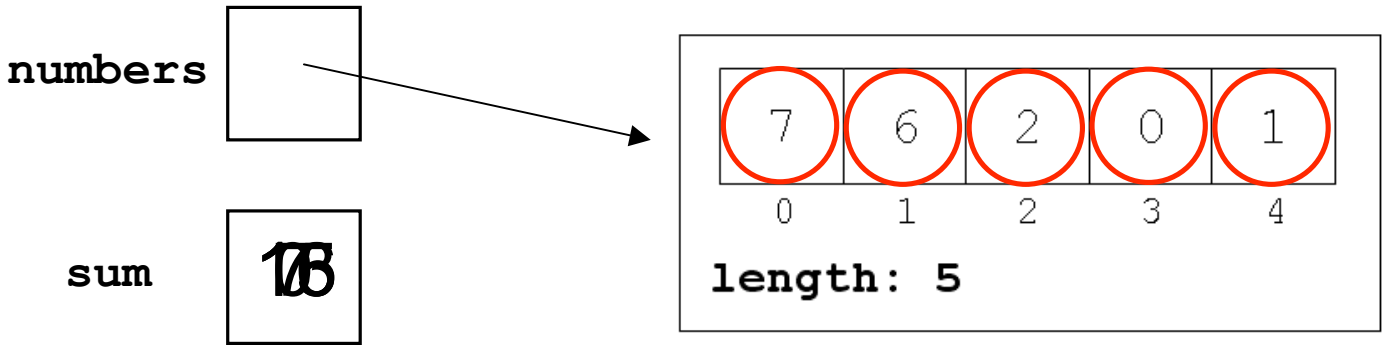
Processing arrays

Processing an array usually means performing some operation on every element of the array.

Examples:

- summing the elements

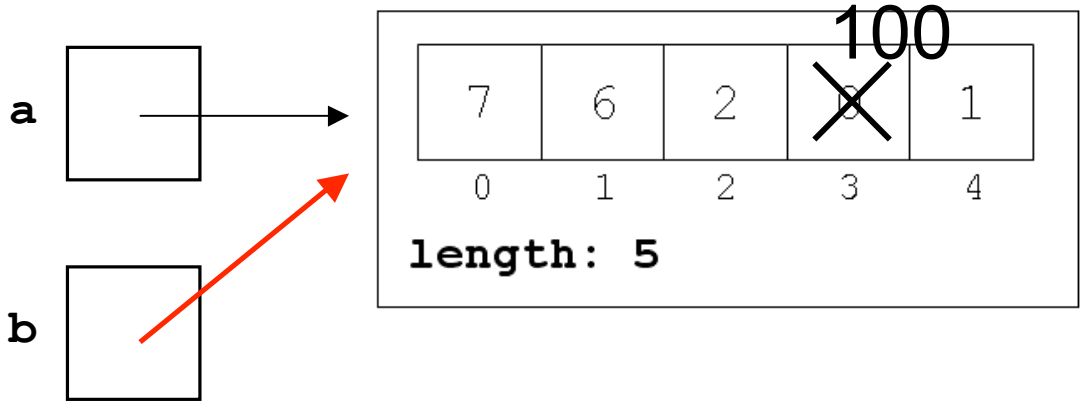
```
int sum = 0;
for (int i = 0; i < nums.length; i++) {
    sum += nums[i];
}
```



“Copying” array references

Consider the following code:

```
int[] a = {7, 6, 2, 0, 1};
int[] b;

b = a;

b[3] = 100;
```



Aliasing *(not in book)*

If we say that "John Smith" is an alias for Walter Jones, we're saying that both the name "John Smith" and the name "Walter Jones" refer to the same person!

If we say that the object variable "x" is an alias for the object variable "y" then both names refer to the same object.

For example, in

```
int[] a = {7, 6, 2, 0, 1};
int[] b;
b = a;
```

After the assignment operation, both "b" and "a" refer to the same array object and are therefore aliases.

Aliasing *(not in book)*

Aliasing can lead to some counter-intuitive behaviour if you don't understand that this is going on. For example, consider the following:

```
int [] a = {1,3,5,7};
int [] b = a;
System.out.println(a[2]);
```

What is printed out?

5

```
b[2] = 7;
System.out.println(a[2]);
```

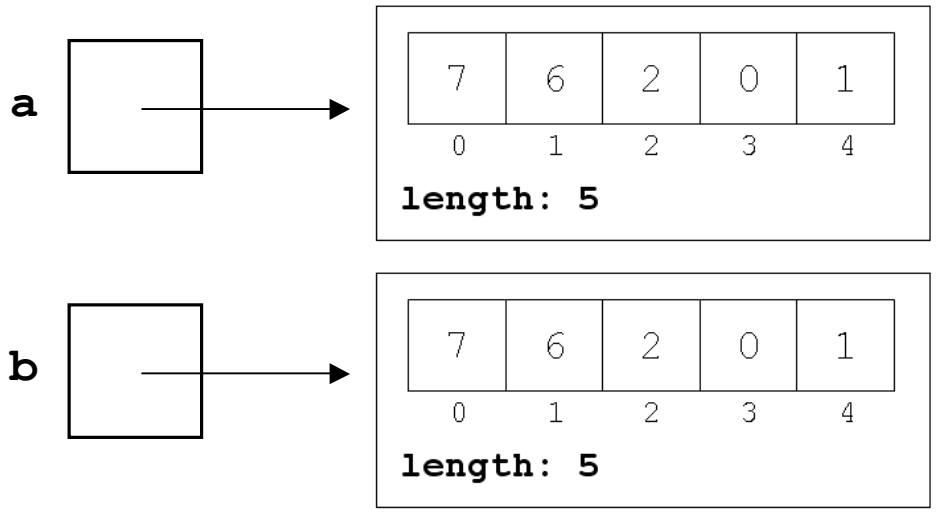
What is printed out? 7

Aliasing can make your program harder to understand!! Use it with care!!

Copying array values

To actually *duplicate* an array, we actually have to create a new array object and copy the values across from the old array to the new one, we can use a loop to copy the values across.

```
int[] a = {7, 6, 2, 0, 1};
int[] b;
b = new int[a.length];
for (int i = 0; i < b.length; i++) {
    b[i] = a[i];
}
```



Arrays as parameters

There is nothing unusual about using arrays as parameters, or as return types from a method:

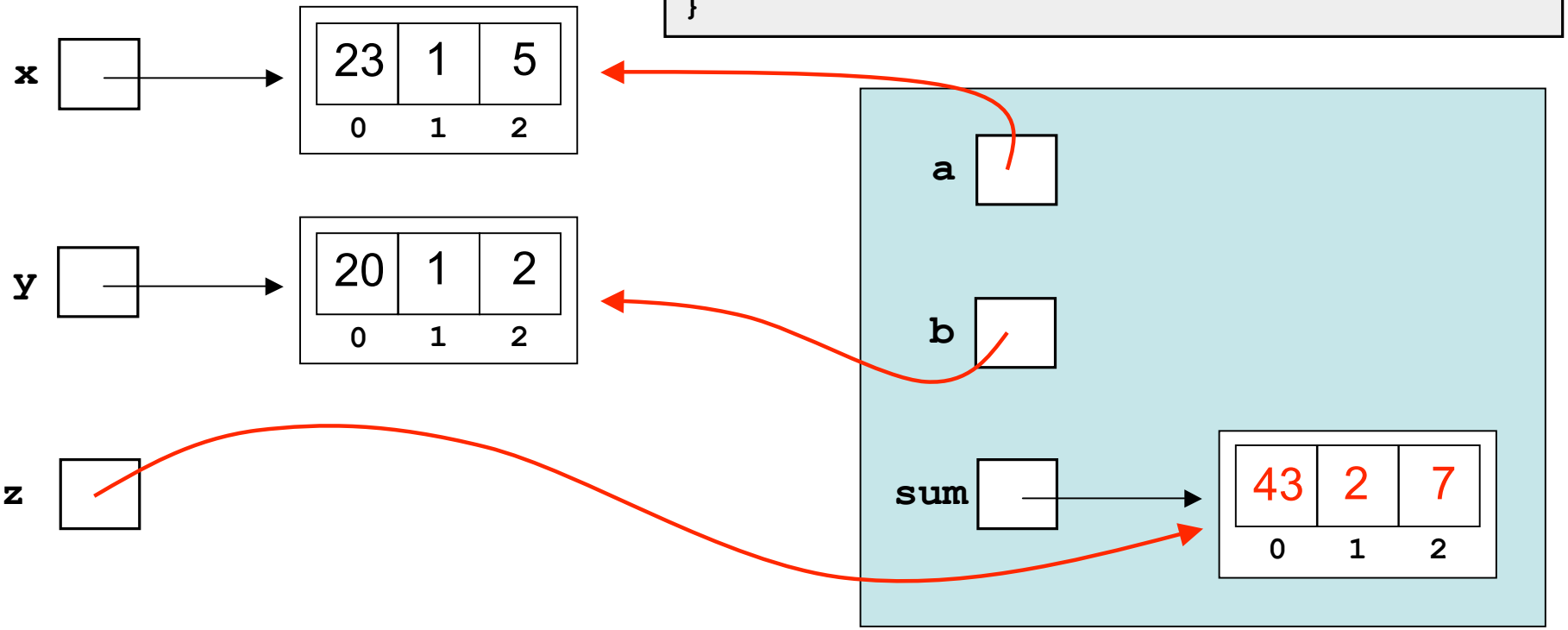
Consider the `addArrays()` method given below:

```
public int[] addArrays(int[] a, int[] b) {

      int[] sum = new int[a.length];

      for (int i = 0; i < a.length; i++) {
            sum[i] = a[i] + b[i];
      }

      return sum;
}
```

Arrays as parameters

```
int[] x = {23, 1, 5};
int[] y = {20, 1, 2};

int[] z = addArrays(x, y);
```

```
public int[] addArrays(int[] a, int[] b) {

        int[] sum = new int[a.length];

        for (int i = 0; i < a.length; i++) {
                sum[i] = a[i] + b[i];
        }

        return sum;
}
```

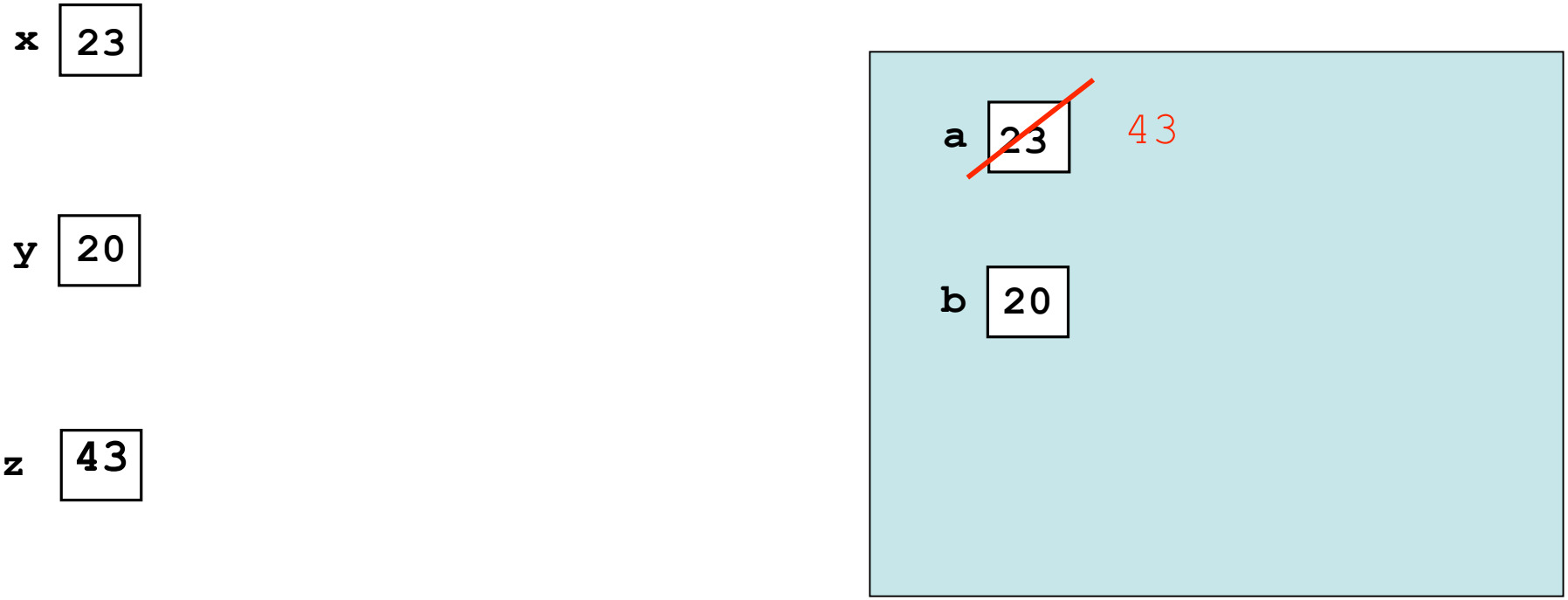


Understanding Parameters - Review

```
int x = 23;
int y = 20;

int z = add(x, y);
System.println(x);
```

```
public int add(int a, int b) {
        int sum = a + b;

        a = sum;
        return sum;
}
```

23



Understanding Parameters

```
public int[] addAs(int[] a, int[] b){

  int[] sum = new int[a.length];

  for (int i = 0; i < a.length; i++){
          sum[i] = a[i] + b[i];
  }
  a[0] = sum[0];
  return sum;
}
```

```
public int[] addAs(int[] a, int[] b){

  int[] sum = new int[a.length];

  for (int i = 0; i < a.length; i++){
          sum[i] = a[i] + b[i];
  }
  a = sum;
  return sum;
}
```

```
int[] x = {23, 1, 5};
int[] y = {20, 1, 2};

int[] z = addAs(x, y);
System.println(x[0]);
```

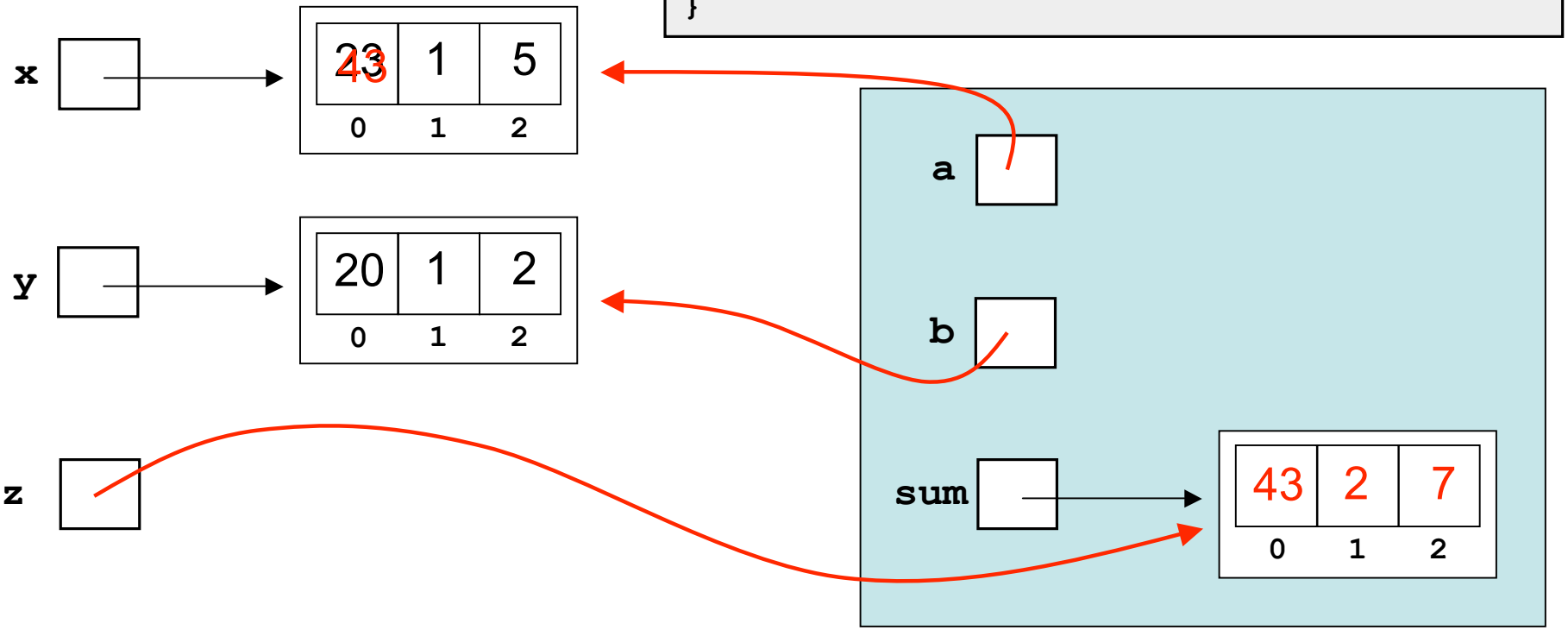
What difference (if any) will using the left version (instead of the right version) of “addAs” make when “x[0]” is printed by the above code?

Understanding Parameters

```
int[] x = {23, 1, 5};
int[] y = {20, 1, 2};

int[] z = addArrays(x, y);
System.println(x[0]);
```

43

```
public int[] addArrays(int[] a, int[] b) {

        int[] sum = new int[a.length];

        for (int i = 0; i < a.length; i++) {
                sum[i] = a[i] + b[i];
        }
        a[0] = sum[0];
        return sum;
}
```

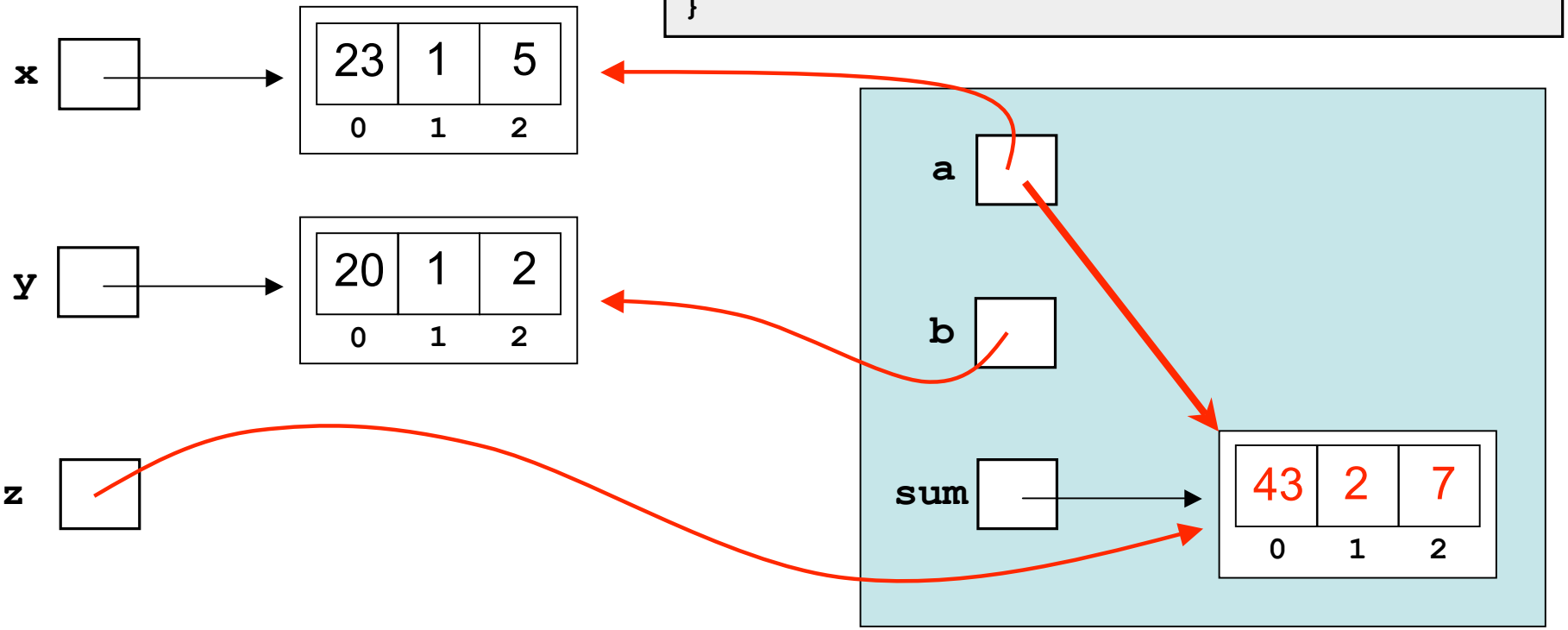


Understanding Parameters

```
int[] x = {23, 1, 5};
int[] y = {20, 1, 2};

int[] z = addArrays(x, y);
System.println(x[0]);
```

23

```
public int[] addArrays(int[] a, int[] b) {

        int[] sum = new int[a.length];

        for (int i = 0; i < a.length; i++) {
                sum[i] = a[i] + b[i];
        }
        a = sum;
        return sum;
}
```



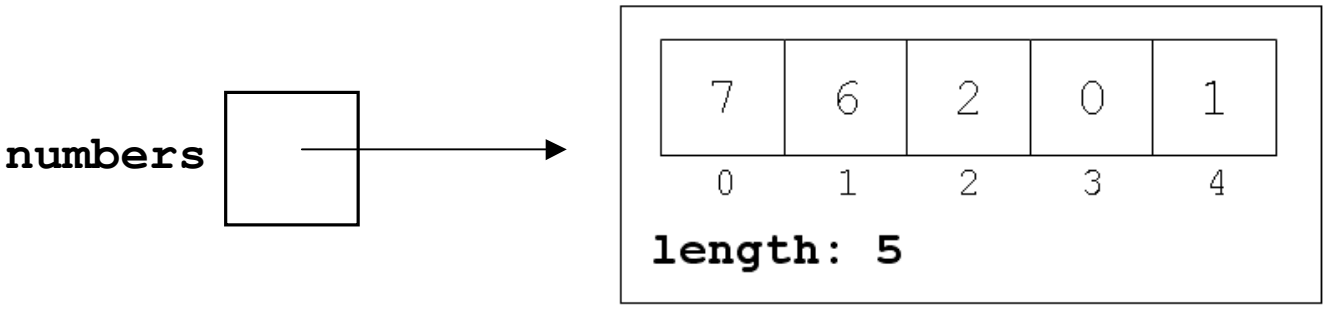
Summary

We have introduced the use of arrays for both primitive types and for objects.

An array:

- holds a sequence of variables of the ***same type***
- each ***element*** in the array is associated with an ***index*** value
- index values start at 0
- every array has a ***length*** field which stores the number of elements in the array
- ***arrays are objects***

int [ ] numbers = {7, 6, 2, 0, 1};



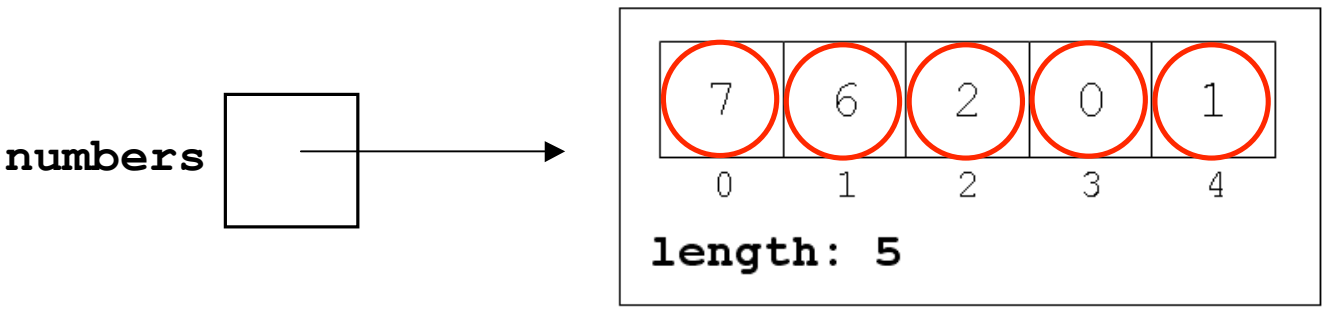
Summary

We can access every element in an array systematically using a loop:

This is commonly done with a `for` loop like:

this is how we access the `length` field of an array

```
for (int i = 0; i < numbers.length; i++) {
      System.out.println(numbers[i]);
}
```



Summary

We discussed how aliasing works with object variables, and how it can lead to confusing code.

We also showed how arrays can be passed as parameters or as return values.

Lastly we examined how object parameters are a form of aliasing and saw how an understanding of aliasing allows us to predict the effect of "updating" object parameters.