

Primitive Arrays

Produced Dr. Siobhán Drohan
by: Mr. Colm Dunphy
 Mr. Diarmuid O'Connor



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax

Why arrays?

- We look at different pieces of code to explain the concept.
- In each piece of code, we:
 - read in 10 numbers from the keyboard
 - add the numbers
 - print the sum of all the numbers.
- Arrays allow us to reduce the amount of code that's needed to solve problems

Adding 10 numbers

```
import javax.swing.JOptionPane;  
  
int n;  
int sum = 0;  
  
for (int i = 0; i<10; i++) {  
    n = Integer.parseInt  
        (JOptionPane.showInputDialog(  
            "Please enter a number ", "3"));  
    sum += n;  
}  
  
println("The sum of the values you typed in is : " + sum);
```

Reads in 10 numbers
from the keyboard

Adding 10 numbers

```
import javax.swing.JOptionPane;  
  
int n;  
int sum = 0;  
  
for (int i = 0; i<10; i++) {  
    n = Integer.parseInt  
        (JOptionPane.showInputDialog(  
            "Please enter a number ", "3"));  
    sum += n;  
}  
  
println("The sum of the values you typed in is : " + sum);
```

As each number is entered,
it is added to the value
currently stored in **sum**.

Adding 10 numbers

```
import javax.swing.JOptionPane;  
  
int n;  
int sum = 0;  
  
for (int i = 0; i<10; i++) {  
    n = Integer.parseInt  
        (JOptionPane.showInputDialog(  
            "Please enter a number ", "3"));  
    sum += n;  
}  
  
println("The sum of the values you typed in is : " + sum);
```

When the 10 numbers have been read in, the **sum** of the 10 numbers is printed to the console.

Adding 10 numbers

```
import javax.swing.JOptionPane;  
  
int n;  
int sum = 0;  
  
for (int i = 0; i<10; i++) {  
    n = Integer.parseInt  
        (JOptionPane.showInputDialog(  
            "Please enter a number ", "3"));  
    sum += n;  
}  
  
println("The sum of the values you typed in is : " + sum);
```

Notice that,
each time a number is read in,
it overwrites the value stored in **n**.

It doesn't remember
the individual numbers typed in.

Rule – Never lose input data



- Always try to **store** input data for later use
- In real-life systems,
you nearly always need to use it again.
- The previous code has NOT done this.
 - Let's try another way ...

Remembering the Numbers

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;  
int sum = 0;  
  
n0 = Integer.parseInt (JOptionPane.show...  
sum += n0;  
  
n1 = Integer.parseInt (JOptionPane.show...  
sum += n1;  
  
//rest of code for n2 to n8  
  
n9= Integer.parseInt(JOptionPane.show...  
sum += n9;  
  
println("The sum of the values you typed in is : " + sum);
```

This works in the sense that we have retained the input data.

BUT...we no longer use loops.

Imagine the code if we had to read in 1,000 numbers?

We need a new approach...

This is where **data structures** come in!

We will now look at **arrays**.

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax

Arrays (fixed-size collections)

- Arrays are a way to collect associated values.
- Programming languages usually offer a special **fixed-size collection** type: an *array*.
- Java arrays can store
 - objects
 - primitive-type values.
- Arrays use a special syntax.

Primitive types

Primitive type

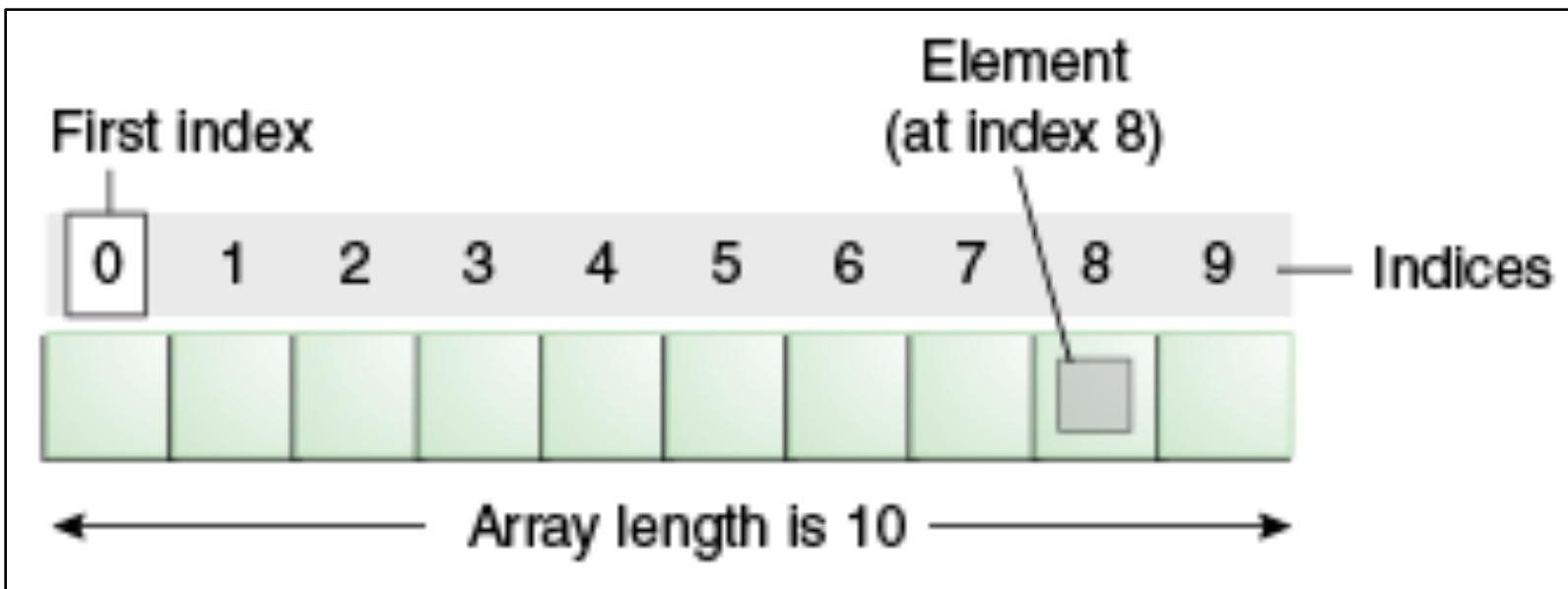
```
int num = 17;
```

Directly stored
in memory...

17

- We are now going to look at a **structure** that can **store many values** of the **same type**.
- Imagine a structure made up of sub-divisions or sections...
- Such a structure is called an **array** and would look like...

Structure of a primitive array



Structure of a primitive array

int[] numbers;

numbers

null

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

numbers

0	0
1	0
2	0
3	0

Structure of a primitive array

```
int[] numbers;
```

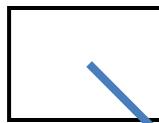
```
numbers = new int[4];
```

We have declared an **array of int**,
with a capacity of four.

Each element is of type **int**.

The array is called **numbers**.

numbers



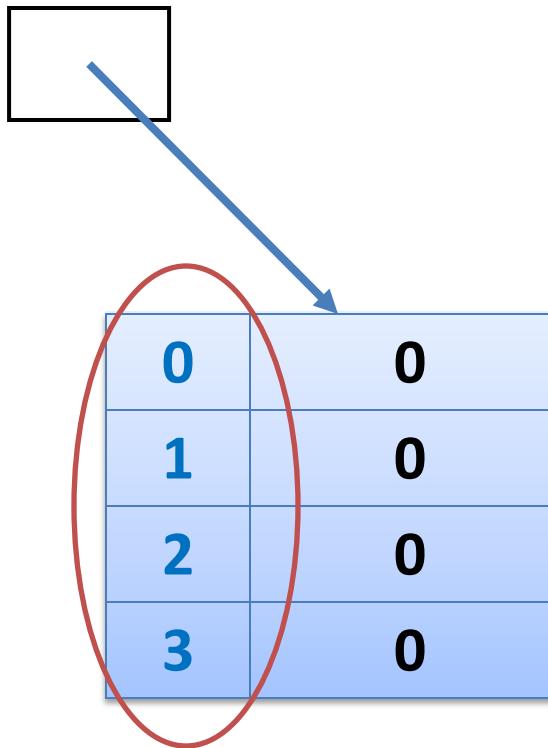
0	0
1	0
2	0
3	0

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

numbers



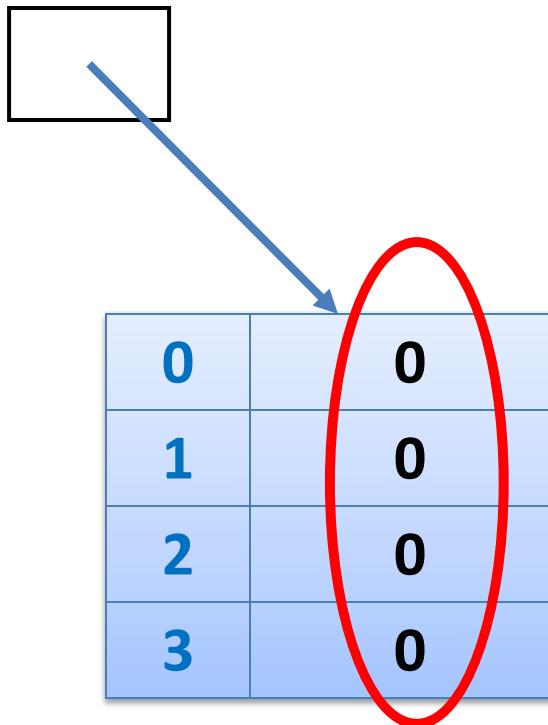
Index of each
element in the array

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

numbers



Default value for each
element of type **int**.

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

```
numbers[2] = 18;
```

We are directly accessing the element at **index 2** and setting it to a **value of 18**.

numbers

The diagram illustrates the structure of a primitive array named "numbers". A blue arrow points from the variable name "numbers" to a 2D grid representing the array's memory. The grid has 4 columns and 4 rows. The columns are indexed 0, 1, 2, and 3 from left to right. The rows are indexed 0, 1, 2, and 3 from top to bottom. The value at index 2 (row 2, column 2) is circled in red and contains the value 18.

0	0
1	0
2	18
3	0

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

```
numbers[2] = 18;
```

```
numbers[0] = 12;
```

We are setting the element at index **0** and to a value of **12**.

numbers

The diagram illustrates the structure of a primitive array named 'numbers'. It shows a variable declaration 'numbers' pointing to a memory location represented by a blue arrow. Below this, a 2D grid represents the array's elements. The grid has 4 columns and 5 rows. The first three columns are labeled with indices 0, 1, and 2, while the last column is labeled 3. The values in the array are: row 0 contains 0 and 12; row 1 contains 1 and 0; row 2 contains 2 and 18; and row 3 contains 3 and 0. The cell containing the value 12 is highlighted with a red oval.

0	12
1	0
2	18
3	0

Structure of a primitive array

```
int[] numbers;
```

```
numbers = new int[4];
```

```
numbers[2] = 18;
```

```
numbers[0] = 12;
```

```
print(numbers[2]);
```

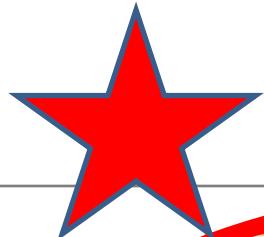
numbers

0	12
1	0
2	18
3	0

Here we are printing the contents of index location 2
i.e. 18 will be printed to the console.

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax

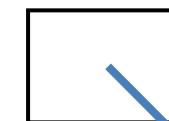


Declaring a primitive array

```
int[] numbers;  
//somecode  
numbers = new int[4];
```

This is how we previously declared our array of four int, called **numbers**.

numbers



0	0
1	0
2	0
3	0

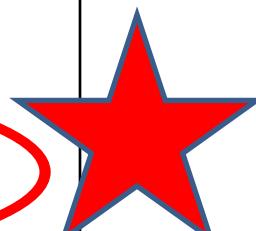
Declaring a primitive array

```
int[] numbers;  
  
//somecode  
  
numbers = new int[4];
```

We can also
(combine both statements)
and declare it like
this...

int[] numbers = new int[4];

numbers



0	0
1	0
2	0
3	0

Let's returning to our method
that reads in, and sums, 10 numbers
(typed in from the keyboard)...

We'll change it
to use **primitive arrays**...

Version that doesn't save the numbers

```
import javax.swing.JOptionPane;  
  
int n;  
int sum = 0;  
  
for (int i = 0; i<10; i++) {  
    n = Integer.parseInt  
        (JOptionPane.showInputDialog(  
            "Please enter a number ", "3"));  
    sum += n;  
}  
  
println("The sum of the values you typed in is : " + sum);
```

Notice that,
each time a number is read in,
it overwrites the value stored in **n**.

It doesn't remember
the individual numbers typed in.

Using arrays to remember numbers

```
import javax.swing.JOptionPane;  
  
int numbers[] = new int[10];  
int sum = 0;  
  
//read in the data  
for (int i = 0; i < 10 ; i++) {  
    numbers[i] = Integer.parseInt(JOptionPane.showInputDialog(  
        "Please enter a number ", "3"));  
}  
  
// now we sum the values  
for (int i = 0; i < 10 ; i++) {  
    sum += numbers[i];  
}  
  
println("The sum of the values you typed in is : " + sum);
```

Using an array
to store each value
that was entered.



Using arrays to remember numbers

```
import javax.swing.JOptionPane;  
  
int numbers[] = new int[10];  
int sum = 0;  
  
//read in the data    Loop 1  
for (int i = 0; i < 10 ; i++) {  
    numbers[i] = Integer.parseInt(JOptionPane.showInputDialog(  
        "Please enter a number ", "3"));  
}  
  
// now we sum the values Loop 2  
for (int i = 0; i < 10 ; i++) {  
    sum += numbers[i];  
}  
  
println("The sum of the values you typed in is : " + sum);
```

Q: Can we reduce the code to only have one loop?

Could we move the “sum” code into the first loop?

Using arrays to remember numbers

```
import javax.swing.JOptionPane;  
  
int numbers[] = new int[10];  
int sum = 0;  
  
//read in the data and sum the values Loop 1  
for (int i = 0; i < 10 ; i++) {  
    numbers[i] = Integer.parseInt(JOptionPane.showInputDialog(  
        "Please enter a number ", "3"));  
    sum += numbers[i];  
}  
  
println("The sum of the values you typed in is : " + sum);
```

A: Yes.

Move the “sum” code into the first loop.
-> functionality doesn’t change

What if we wanted the user to decide
how many numbers
they wanted to sum?

```
import javax.swing.*;  
  
int sum = 0;  
  
//Using the numData value  
int numbers[];  
int numData = Integer.parseInt ( JOptionPane.showInputDialog(  
        "How many values do you wish to sum? ", "3"));  
numbers = new int [numData];  
  
//read in the data and sum the values  
for (int i = 0; i < numData ; i++) {  
    numbers[i] = Integer.parseInt(JOptionPane.showInputDialog(  
        "Please enter a number ", "3"));  
    sum += numbers[i];  
}  
  
println("The sum of the values you typed in is : " + sum);
```

1. Delcare **numbers** to be an array of type integer.
2. **numData** takes in the size.
3. Use numData to initialize the array with **new** specifying the size.

What type of data can be stored
in a **primitive array**?

An array can store ANY TYPE of data.

Primitive Types

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

```
byte smallNumbers[] = new byte[4];
```

```
char characters[] = new char[26];
```

Object Types

```
String words = new String[30];
```

```
Spot spots[] = new Spot[20];
```

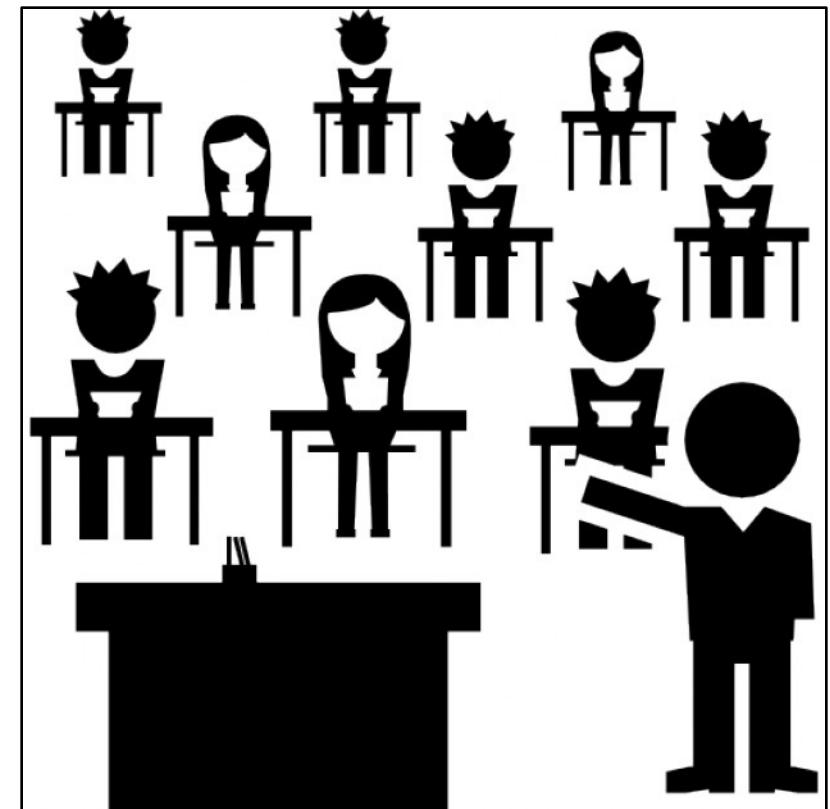
Do we have to use
all the elements in the array?

Do we have to use all elements in the array?

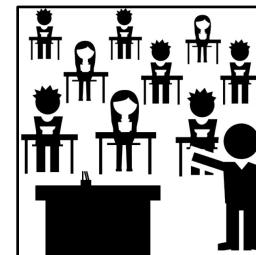
- No.
- **But...**this might cause logic errors,
if we don't take this into consideration
in our coding.
- Consider this scenario...

Scenario – exam results and **average grade**

- We have a class of 15 students.
- They have a test coming up.
- We want to store the results in an array and then find the **average** result.



Average grade



results

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0

We create an **array of int** with a **capacity of 15**

Only 12 students sat the exam.
Their results were recorded in the first 12 elements

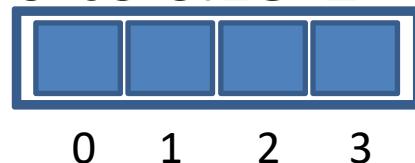
To calculate the average result,
divide by the number of **populated elements**
NOT the array capacity.

Do we have to use all elements in the array?

- If all elements in an array are NOT populated, we need to:
 - have another variable (e.g. int **size**)
 - containing the number of elements in the array **actually used**.
 - ensure size is used when processing the array
 - e.g.
 - for (int i= 0; i < **size**; i++)
- For now though, we assume that all elements of the array are populated and therefore ready to be processed.

Summary - Arrays

- Arrays are structures that can store many values of the same type
- Rule – Never lose input data
 - Arrays enable us to store the data efficiently
 - We can use loops with arrays
- Arrays can store ANY type
- Declaring arrays
 - `int[] arryName;
//somecode
arryName= new int[4];`
 - `int[] arryName= new int[4];`
- Index goes from 0 to size-1



Questions?

