Primitive Arrays

Produced by:
Dr. Siobhán Drohan
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
- Syntax

Why arrays?

- We look at different pieces of code to explain the concept.

Source: Reas & Fry (2014)

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.

Source: Reas & Fry (2014)

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

Source: Reas & Fry (2014)

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**.

Source: Reas & Fry (2014)

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.

Source: Reas & Fry (2014)

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**. This code does not remember the individual numbers typed in.

Source: Reas & Fry (2014)

Rule – Never lose data

- Should always try to store that data for later use (in a more real-life system you would almost always need to use the input data again).

Rule – Never lose data

- Should always try to store that data for later use (in a more real-life system you would almost always need to use the input data again).
- The previous code has not done this.
- We could try another way ...

Remembering the Numbers

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;
int sum = 0;

n0 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n0;

n1 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n1;

//rest of code for n2 to n8

n9= Integer.parseInt(JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n9;

println("The sum of the values you typed in is : " + sum);
```

Remembering the Numbers

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

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;
int sum = 0;

n0 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n0;

n1 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n1;

//rest of code for n2 to n8

n9= Integer.parseInt(JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n9;

println("The sum of the values you typed in is : " + sum);
```

Remembering the Numbers

BUT…we no longer use loops.

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;
int sum = 0;

n0 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n0;

n1 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n1;

//rest of code for n2 to n8

n9= Integer.parseInt(JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n9;

println("The sum of the values you typed in is : " + sum);
```

Remembering the Numbers

BUT…we no longer use loops. Imagine the code if we had to read in 1,000 numbers?

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;
int sum = 0;

n0 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n0;

n1 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n1;

//rest of code for n2 to n8

n9= Integer.parseInt(JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n9;

println("The sum of the values you typed in is : " + sum);
```

Remembering the Numbers

We need a new approach…this is where **data structures** come in! We will now look at **arrays**.

```
int n0,n1, n2, n3, n4, n5, n6, n7, n8, n9;
int sum = 0;

n0 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n0;

n1 = Integer.parseInt (JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n1;

//rest of code for n2 to n8

n9= Integer.parseInt(JOptionPane.showInputDialog("Please enter a number ", "3"));
sum += n9;

println("The sum of the values you typed in is : " + sum);
```

Arrays (fixed-size collections)

- Arrays are a way to collect associated values.

Arrays (fixed-size collections)

- Arrays are a way to collect associated values.
- Programming languages usually offer a special fixed-size collection type: an *array*.

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 or primitive-type values.

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 or primitive-type values.
- Arrays use a special syntax.

Primitive types

Primitive type

```
int num = 17;
```

Primitive types

Primitive type

```
int num = 17;
```

Directly stored in memory…

17

Primitive types

Primitive type

```
int num = 17;
```

Directly stored in memory…

17

num can store **ONE** int value, in this case, 17.

Primitive types

Primitive type

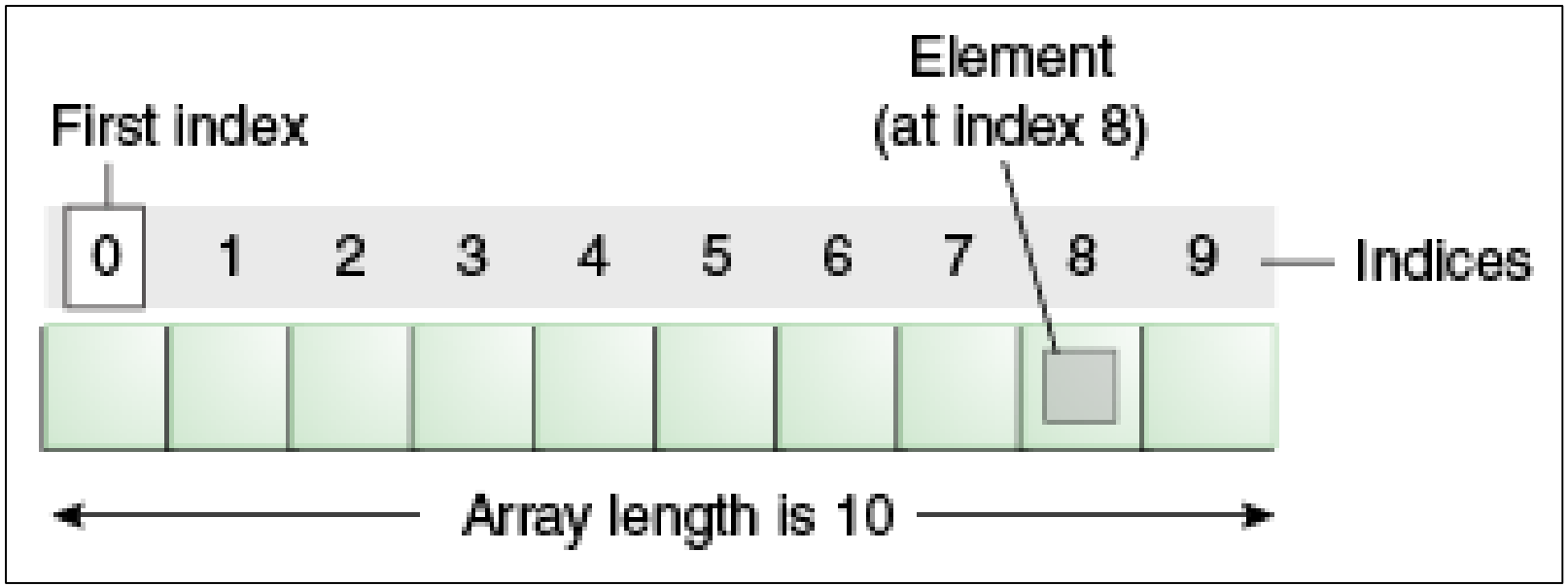
```
int num = 17;
```

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:

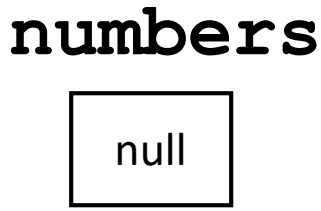
Structure of a primitive array



http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Structure of a primitive array

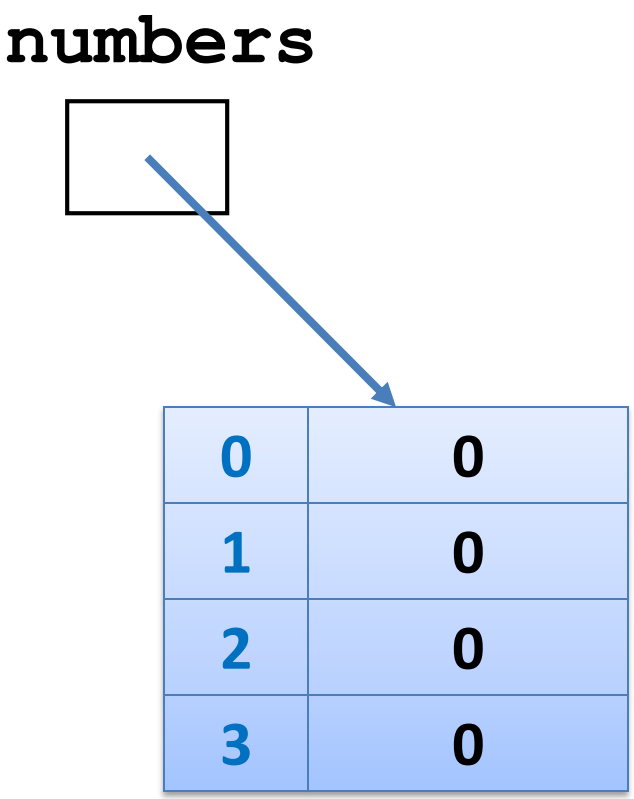
```
int[] numbers;
```



Structure of a primitive array

```
int[]  numbers;
```

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



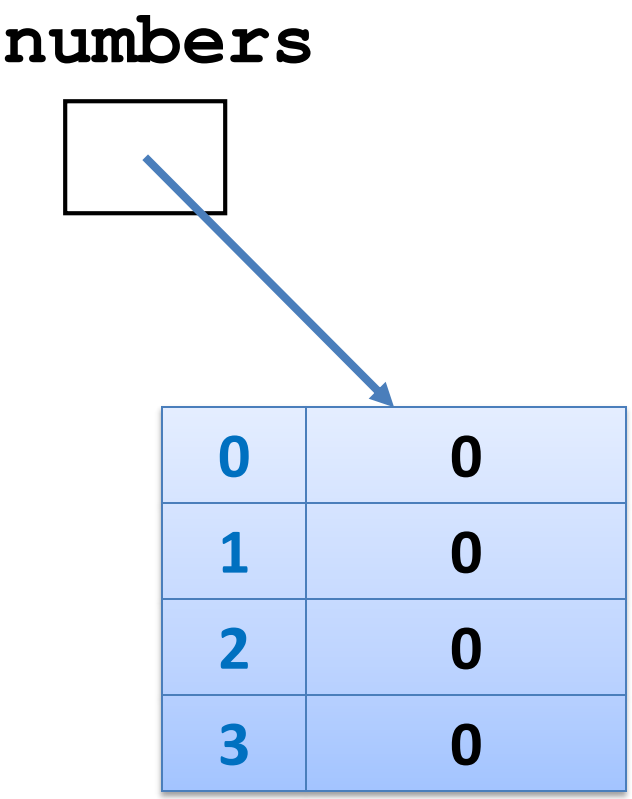
Structure of a primitive array

```
int[]  numbers;
```

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

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

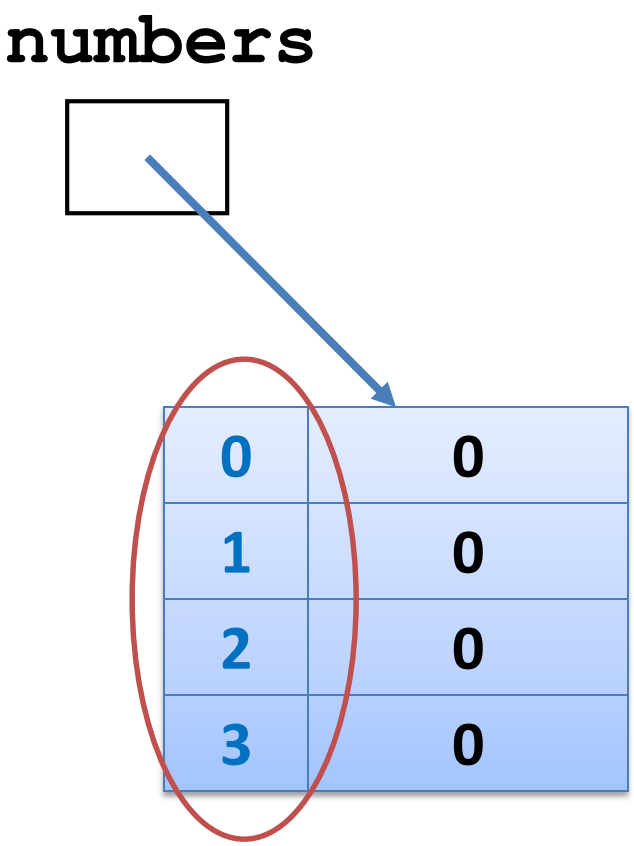
Each element is of type **int**. The array is called **numbers**.



Structure of a primitive array

```
int[] numbers;
```

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

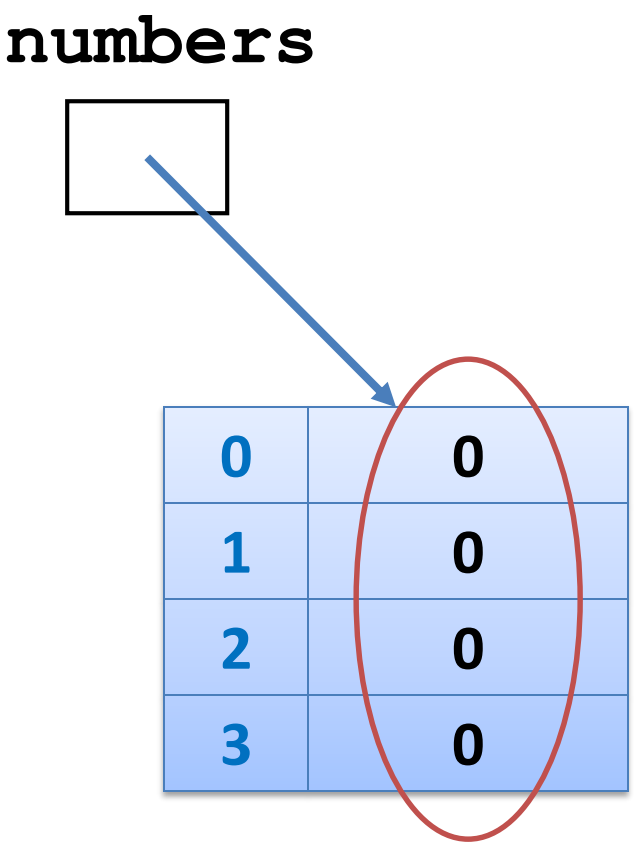


Index of each element in the array

Structure of a primitive array

```
int[] numbers;
```

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



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

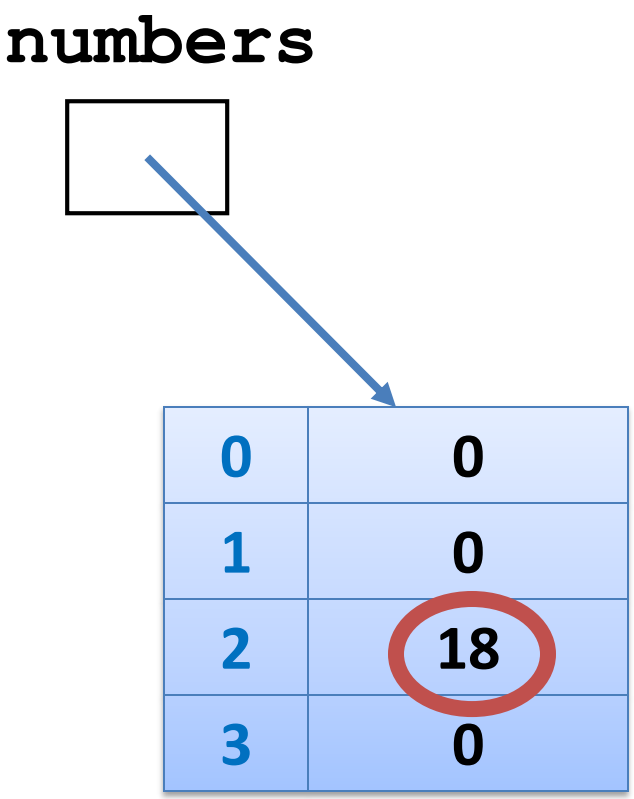
Structure of a primitive array

```
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**.



Structure of a primitive array

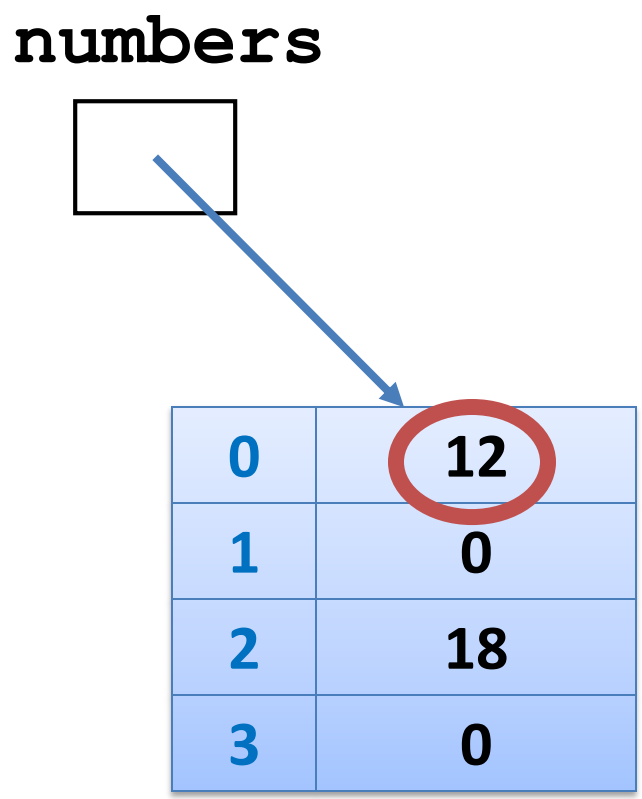
```
int[]  numbers;
```

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

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

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

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



Structure of a primitive array

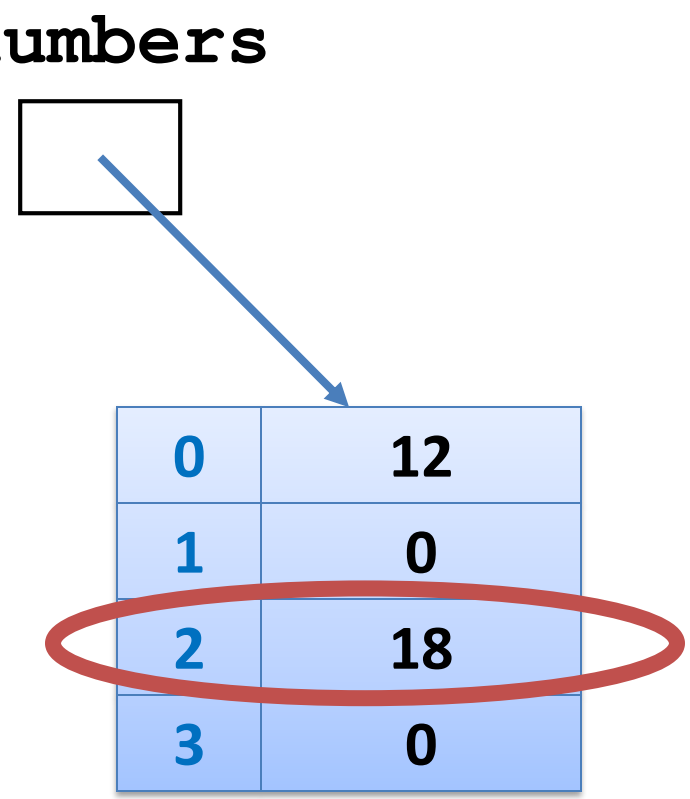
```
int[]  numbers;
```

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

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

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

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

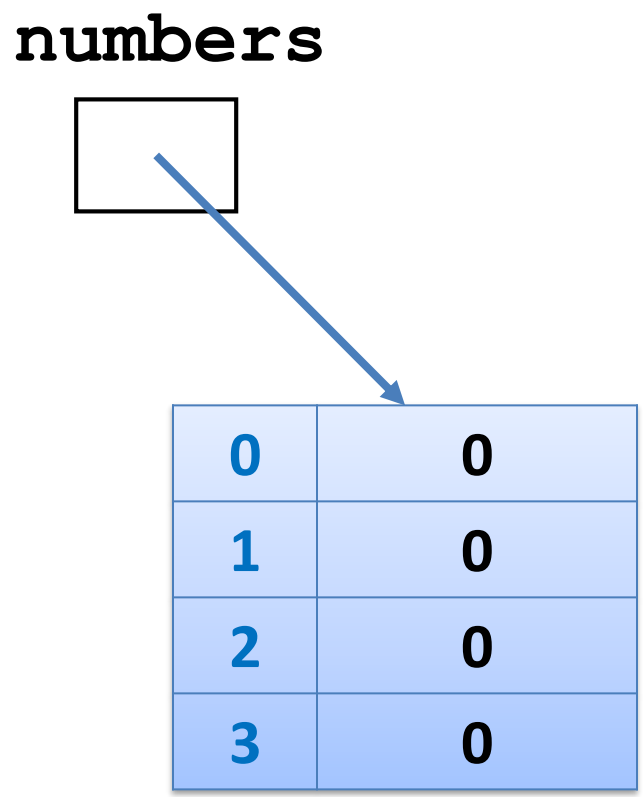


Here we are printing the contents of index location 2 i.e. 18 will be printed to the console.

Declaring a primitive array

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

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

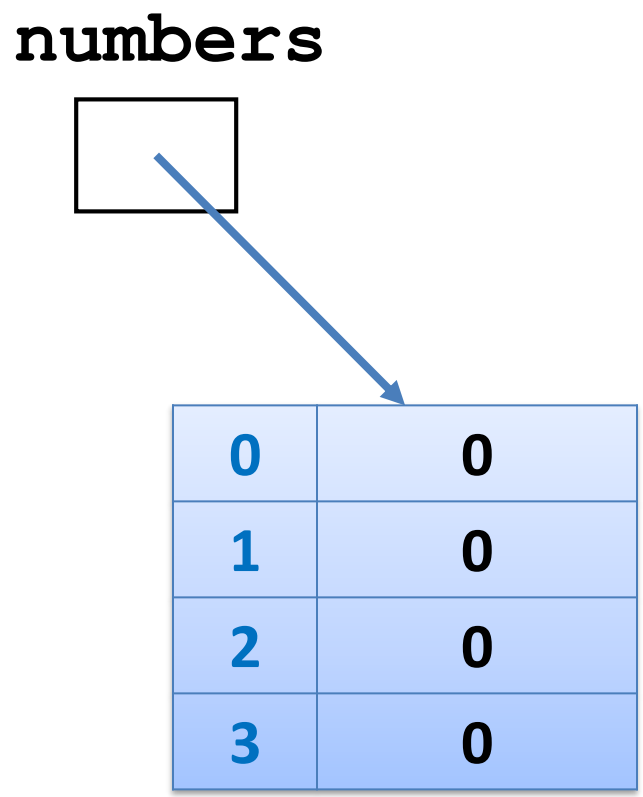


Declaring a primitive array

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

We can also declare it like this…

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



Returning to our method that reads in and sums 10 numbers (typed in from the keyboard)…

and converting 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**. This code does not remember the individual numbers typed in.

Source: Reas & Fry (2014)

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.

Source: Reas & Fry (2014)

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);
```

Q: Can we reduce the code to only have one loop? Could we move the "sum" code into the first loop?

Source: Reas & Fry (2014)

Using arrays to remember numbers

A: Yes. The functionality of our code does not change when we move the “sum” code into the first loop.

```
import javax.swing.JOptionPane;

int numbers[] = new int[10];
int sum = 0;

//read in the data and sum the values
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);
```

Source: Reas & Fry (2014)

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 to set the size of the array
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);
```

Source: Reas & Fry (2014)

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];
```

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];
```

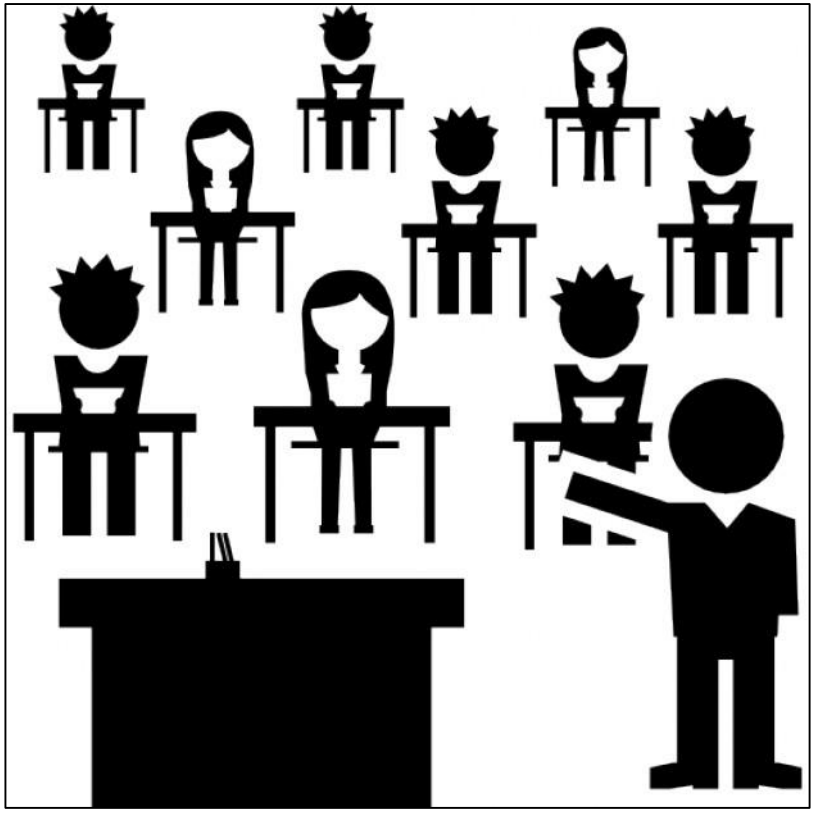
Do we have to use all the elements in the array?

Do we have to use all elements in the array?

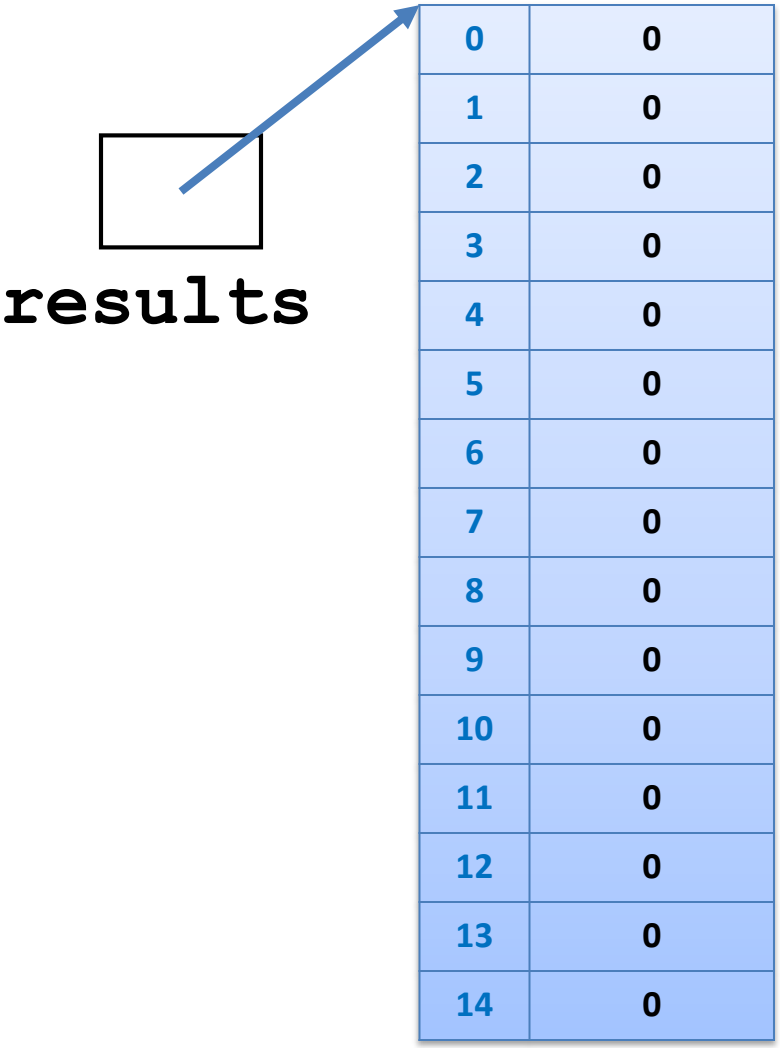
- No.
- But…this might cause logic errors, if we don't take this into consideration in our coding.
- Consider this scenario…

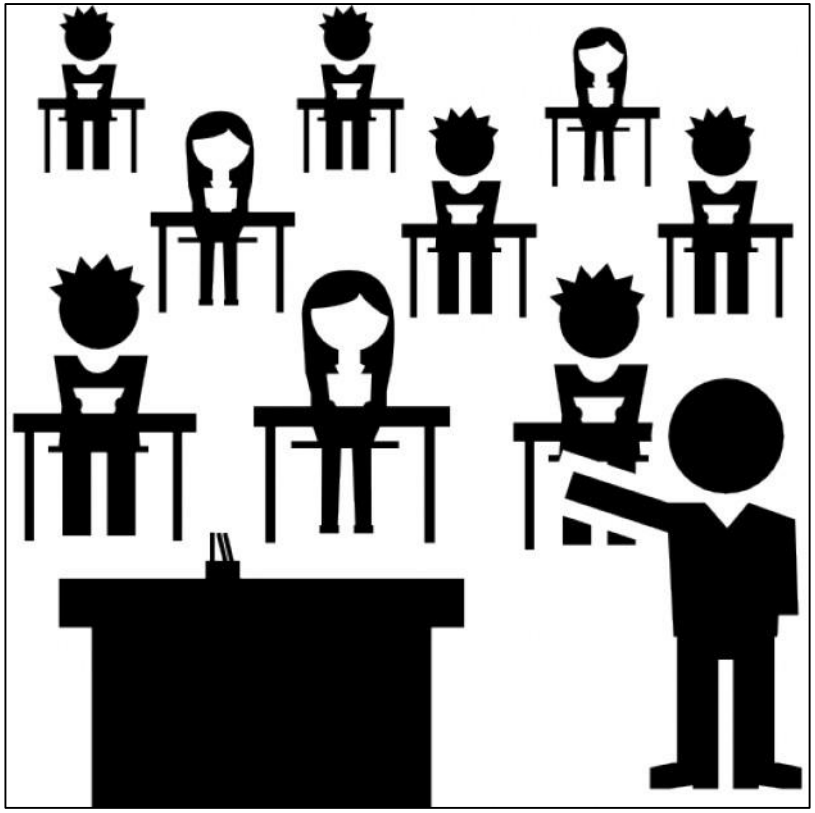
Scenario – exam results and average grade

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

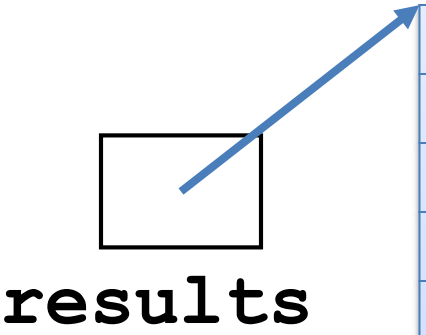


We create an array of int with a capacity of 15

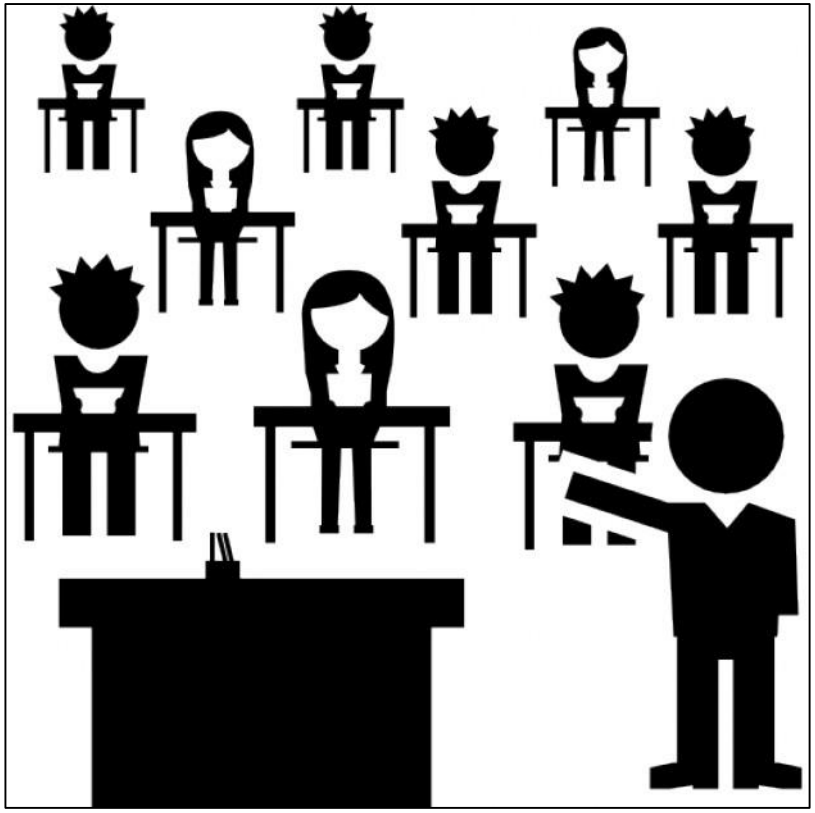




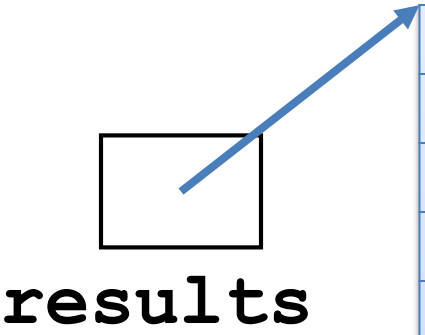
Only 12 students sat the exam and their results were recorded in the first 12 elements



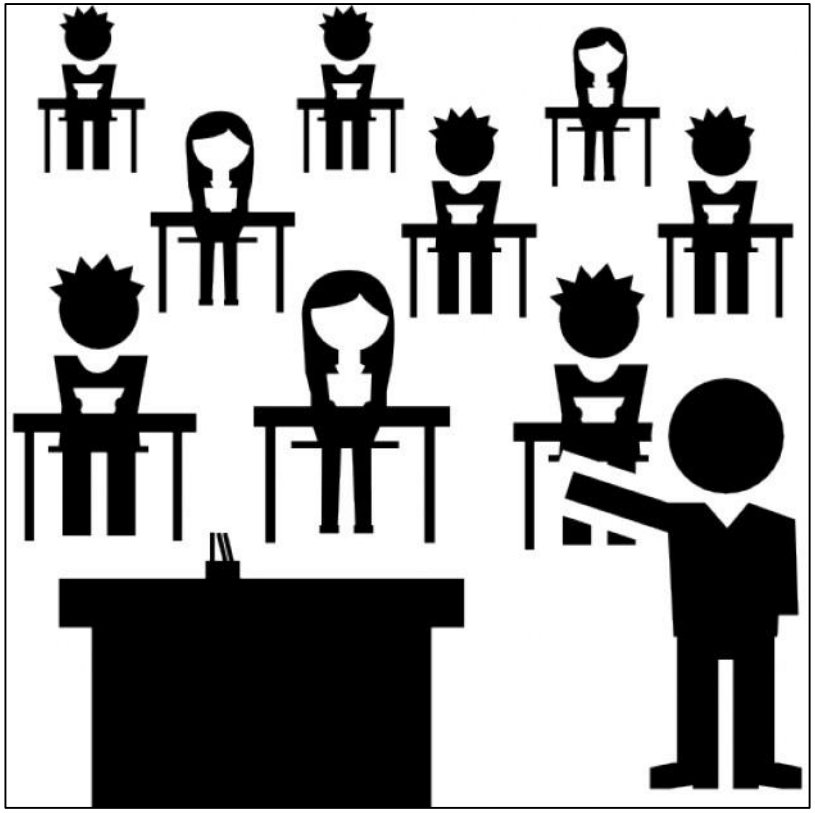
0	87
1	34
2	76
3	45
4	0
5	56
6	65
7	0
8	48
9	54
10	38
11	98
12	0
13	0
14	0



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



0	87
1	34
2	76
3	45
4	0
5	56
6	65
7	0
8	48
9	54
10	38
11	98
12	0
13	0
14	0



Do we have to use all elements in the array?

- When not all elements in an array are populated, we need to:
  - have another variable (e.g. int size) which contains the number of elements of the array is 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.

Questions?