Primitive Arrays

Produced by: Dr. Siobhán Drohan,
Ms. Mairead Meagher,
Ms. Siobhán Roche.

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.setu.ie/

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

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 java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int n;
int sum = 0;

for (int i = 0; i<10; i++)  {
   n = input.nextInt();
   sum += n;
}

System.out.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 java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int n;
int sum = 0;

for (int i = 0; i<10; i++)  {
    n = input.nextInt();
    sum += n;
}

System.out.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 java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int n;
int sum = 0;

for (int i = 0; i<10; i++)  {
    n = input.nextInt();
    sum += n;
}

System.out.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 java.util.Scanner;
 :
Scanner input = new Scanner(Syst
 :
int n;
int sum = 0;

for (int i = 0; i<10; i++)  {
   n = input.nextInt();
   sum += n;
}

System.out.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 = input.nextInt();
sum += n0;

n1 = input.nextInt();
//rest of code for n2 to n8

n9= input.nextInt();
sum += n9;

System.out.println("The sum of the val
```

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
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

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.

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

Primitive types

Primitive type

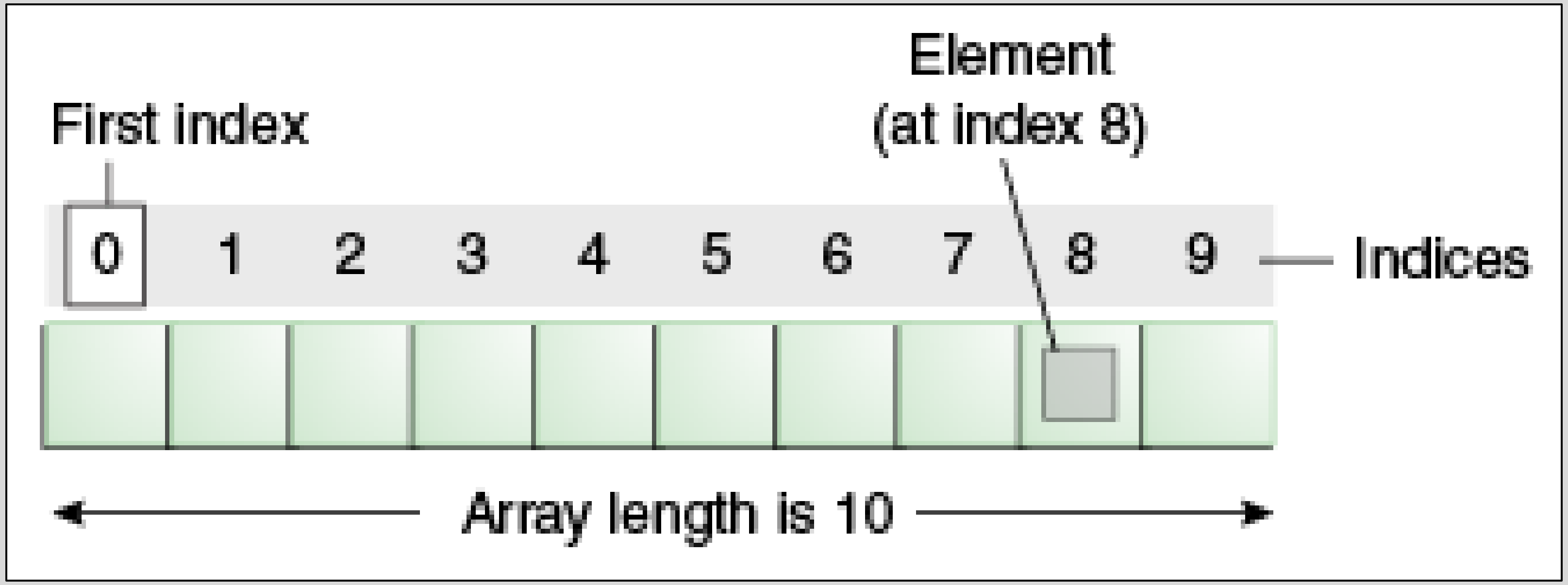
```
int num = 17;
```

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:

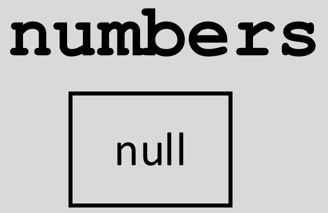
Structure of a primitive array



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

Structure of a primitive array

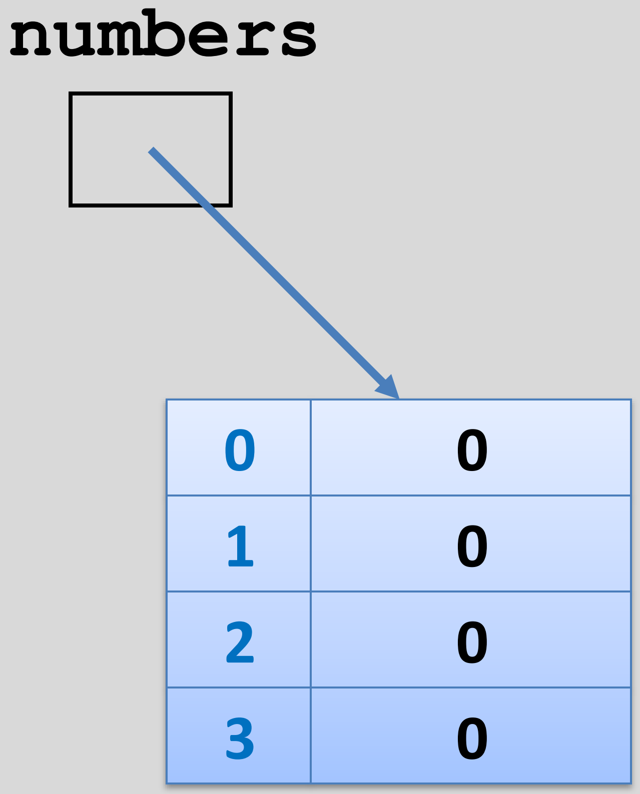
```
int[]  numbers;
```



Structure of a primitive array

```
int[]  numbers;
```

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



Structure of a primitive array

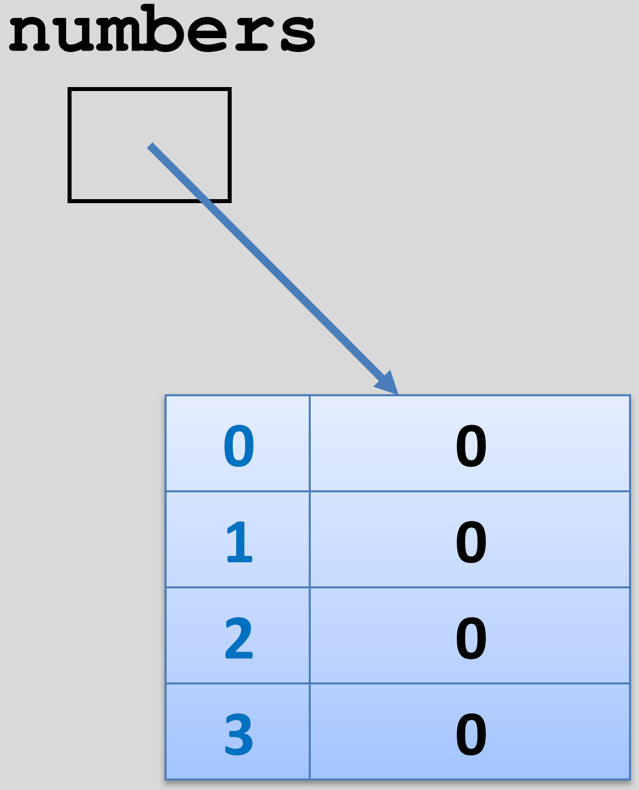
```
int[]  numbers;
```

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

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

Each element is of type **int**.

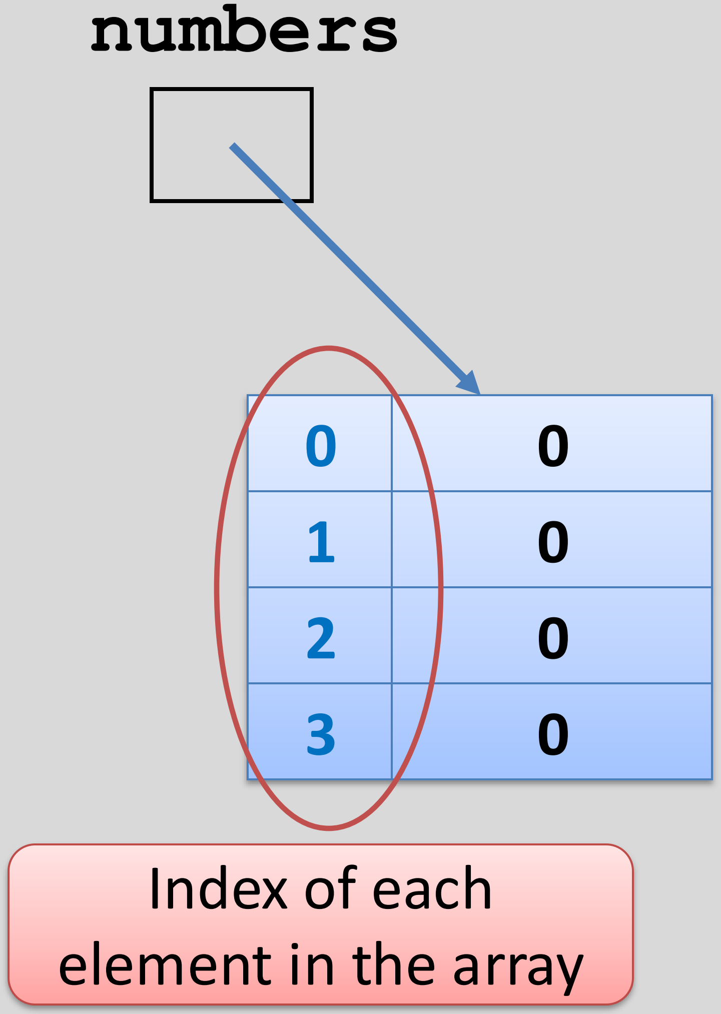
The array is called **numbers**.



Structure of a primitive array

```
int[]  numbers;
```

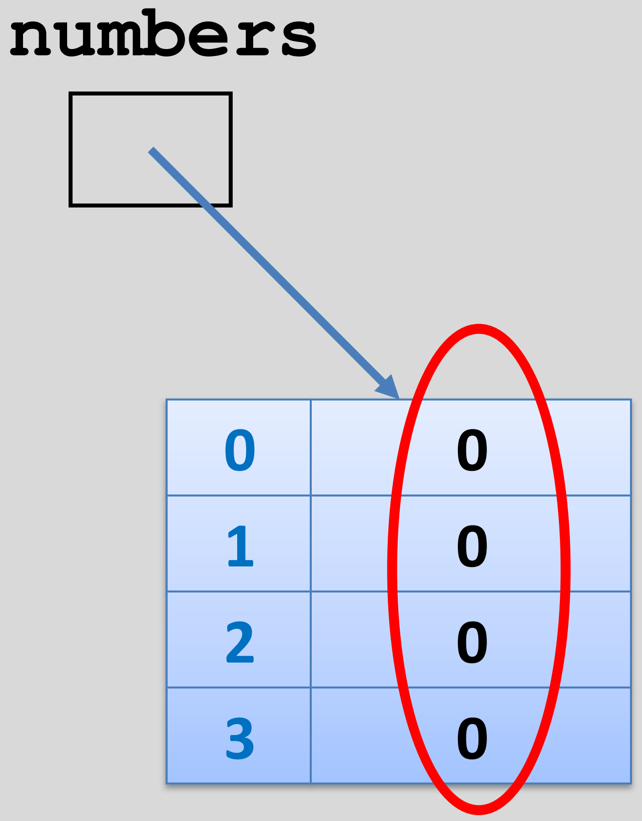
```
numbers = new int[4];
```



Structure of a primitive array

```
int[]  numbers;
```

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



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

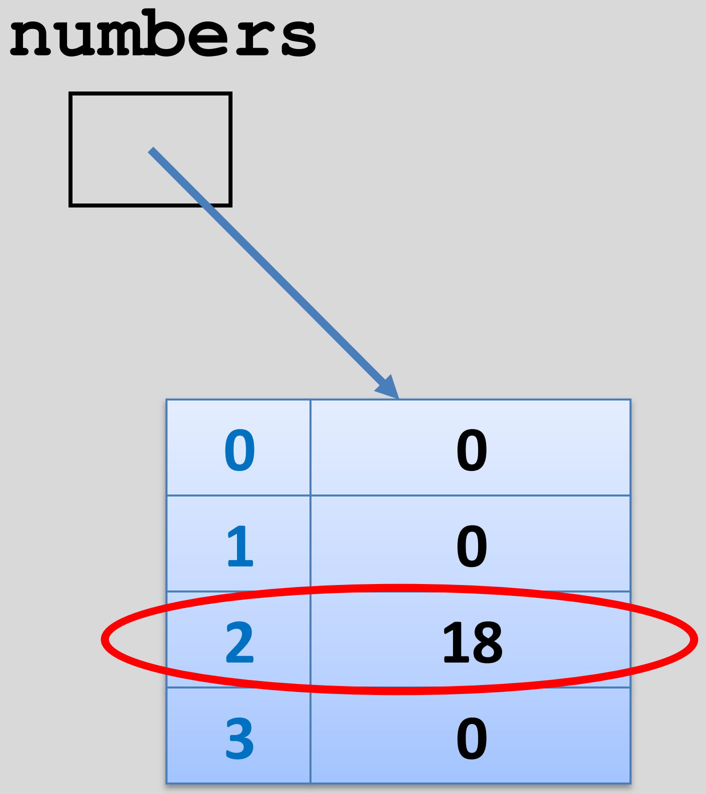
Structure of a primitive array

```
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

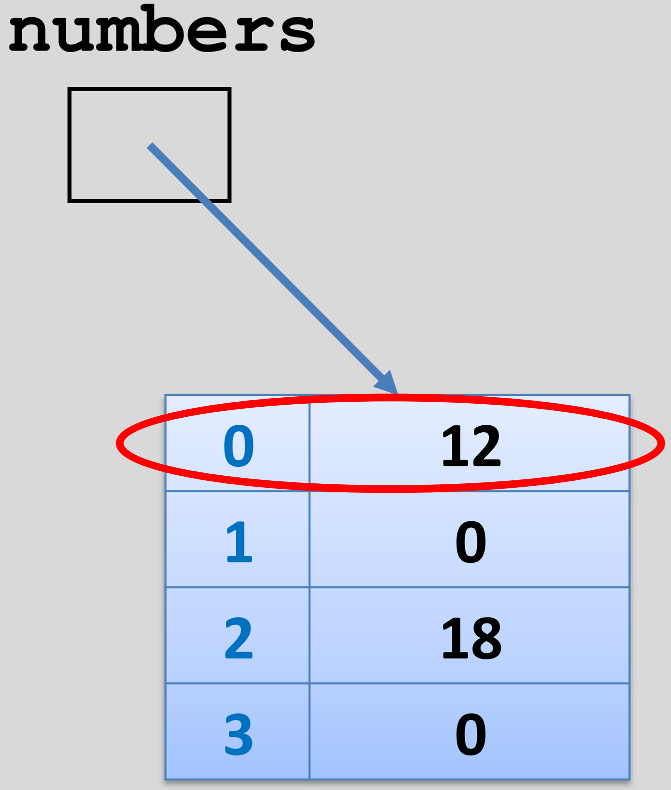
```
int[]  numbers;
```

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

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

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

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



0	12
1	0
2	18
3	0

Structure of a primitive array

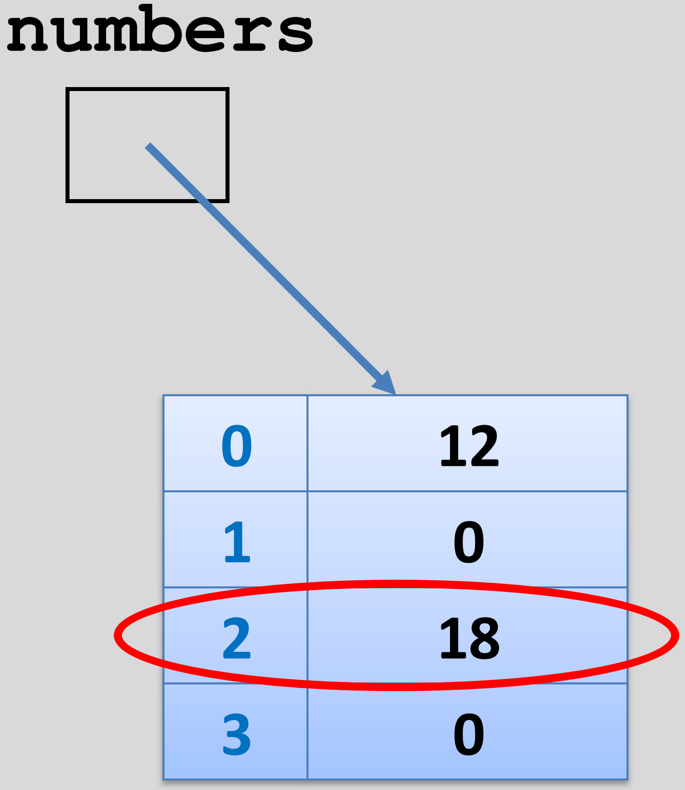
```
int[]  numbers;
```

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

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

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

```
System.out.println(numbers[2]);
```

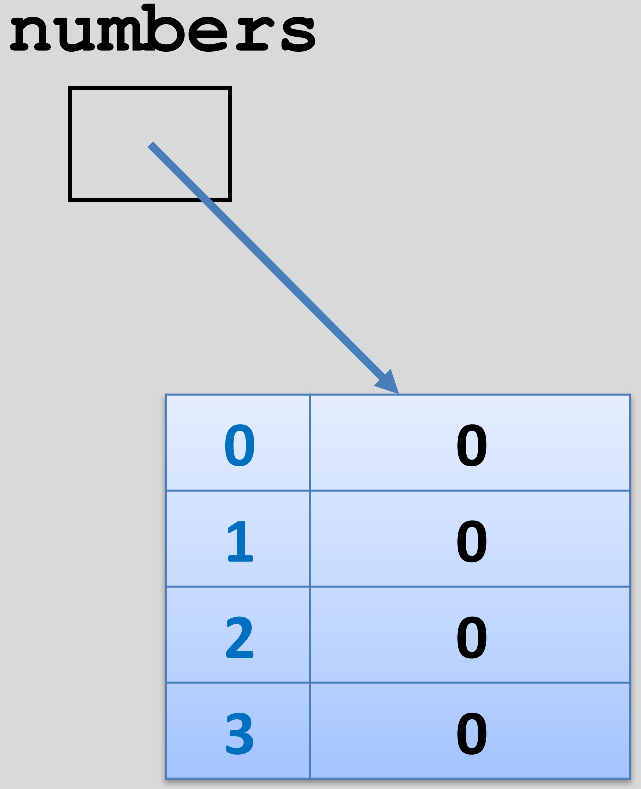


Here we are printing the contents of index location 2
i.e. 18 will be printed to the console.

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

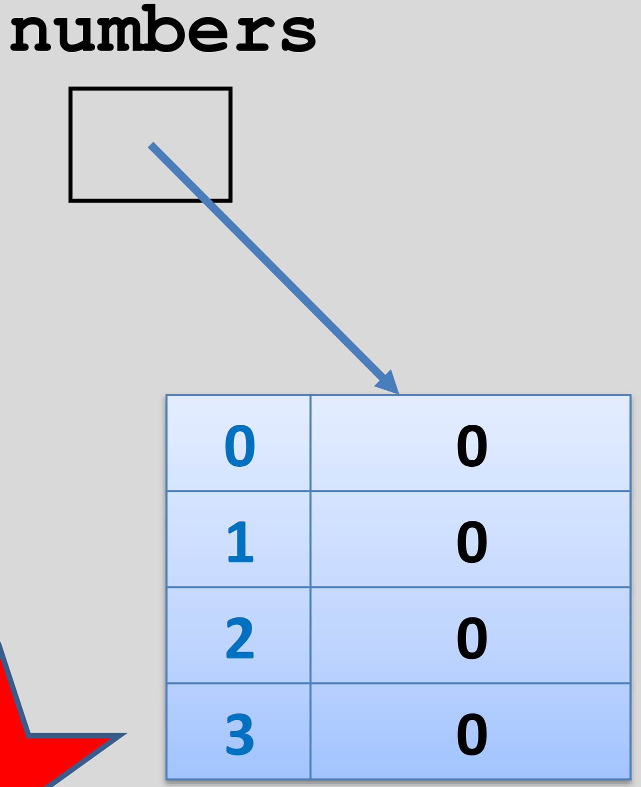
```
int[]  numbers;

//somecode

numbers = new int[4];
```

We can also declare it like this…

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

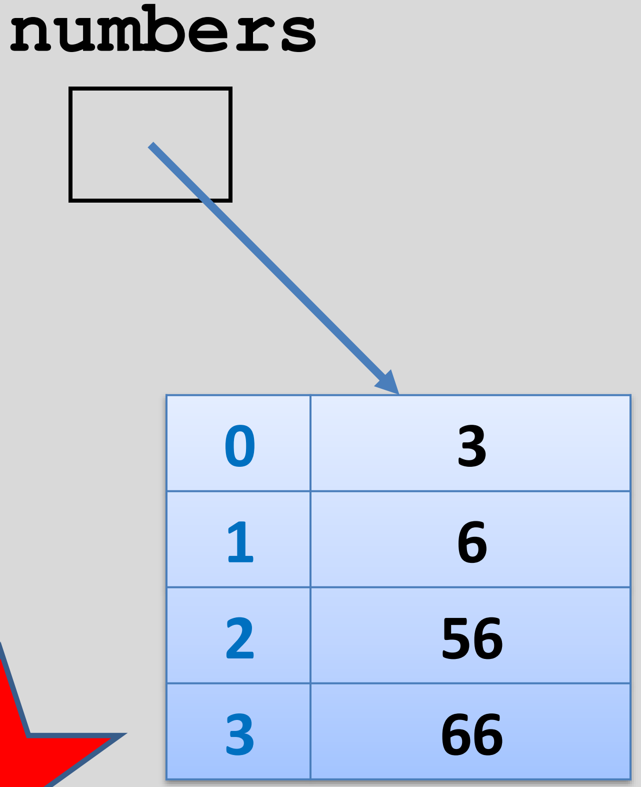


0	0
1	0
2	0
3	0

Alternative way of declaring and initialising an array

We can also declare it like this…

```
int[]  numbers = {3,6,56,66}
```



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 java.util.Scanner;
 :
Scanner input = new Scanner(Syst
 :
int n;
int sum = 0;

for (int i = 0; i<10; i++)  {
    n = input.nextInt();
    sum += n;
}

System.out.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.

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

Using arrays to remember numbers

```
import java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int numbers[] = new int[10];
int sum = 0;

//read in the data
for (int i = 0; i < 10 ; i ++)  {
        numbers[i] = input.readInt();
}

// now we sum the values
for (int i = 0; i < 10 ; i ++)   {
        sum += numbers[i];
}

System.out.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 java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int numbers[] = new int[10];
int sum = 0;                Loop 1

//read in the data
for (int i = 0; i < 10 ; i ++)  {
        numbers[i] = input.readInt();
}
                                   Loop 2
// now we sum the values
for (int i = 0; i < 10 ; i ++)   {
        sum += numbers[i];
}

System.out.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

```
import java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
 :
int numbers[] = new int[10];
int sum = 0;

//read in the data
for (int i = 0; i < 10 ; i ++)  {
        numbers[i] = input.readInt();
        sum += numbers[i];
}

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

Loop 1

A: Yes.

Move the "sum" code into the first loop.

-> functionality doesn't change

Source: Reas & Fry (2014)

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

```
import java.util.Scanner;
 :
Scanner input = new Scanner(System.in);
int sum = 0;

//Using the numData value to set the size of the array
int numbers[];
System.out.println(”How many numbers do you need?”);
int numData = input.nextInt();

numbers = new int [numData];

//read in the data and sum the values
for (int i = 0; i < numData ; i ++)  {
            numbers[i] = input.nextInt();
           sum += numbers[i];
 }

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

Source: Reas & Fry (2014)

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

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];

Person persons[] = new Person[20];
```

Topics list

- Why arrays?
- Primitive Arrays
- Array Syntax
- Loops and Arrays
- What datatypes can be stored in arrays?
- Difference between fixed size and number of populated data

Difference between fixed size and number of populated data

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

Do we have to use all elements in the array?

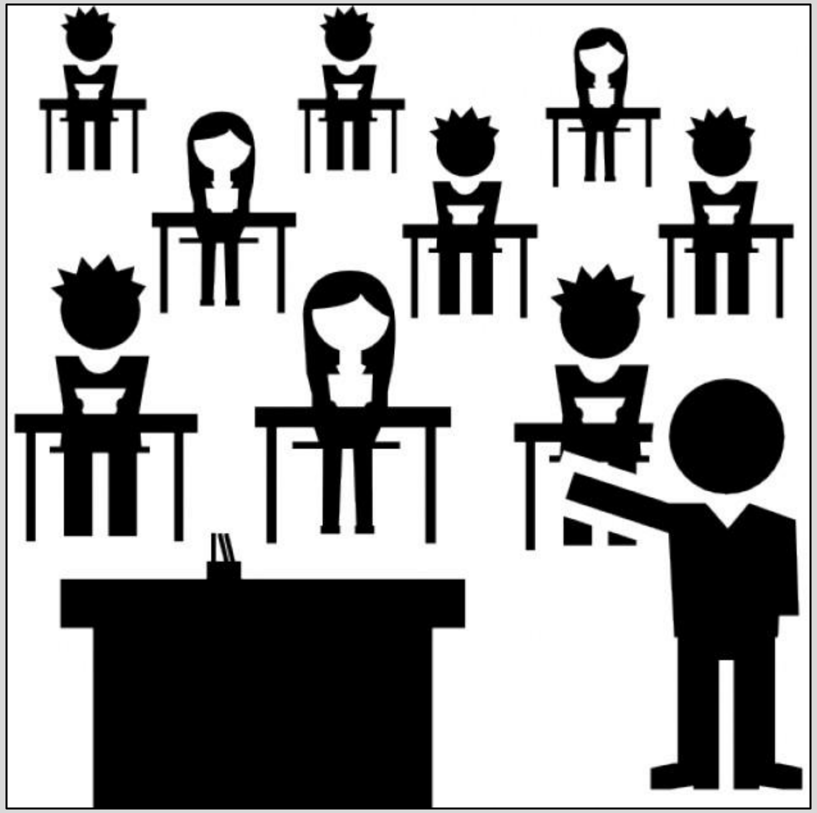
- No.

- But…this might cause logic errors,
  if we don’t take this into consideration
  in our coding.

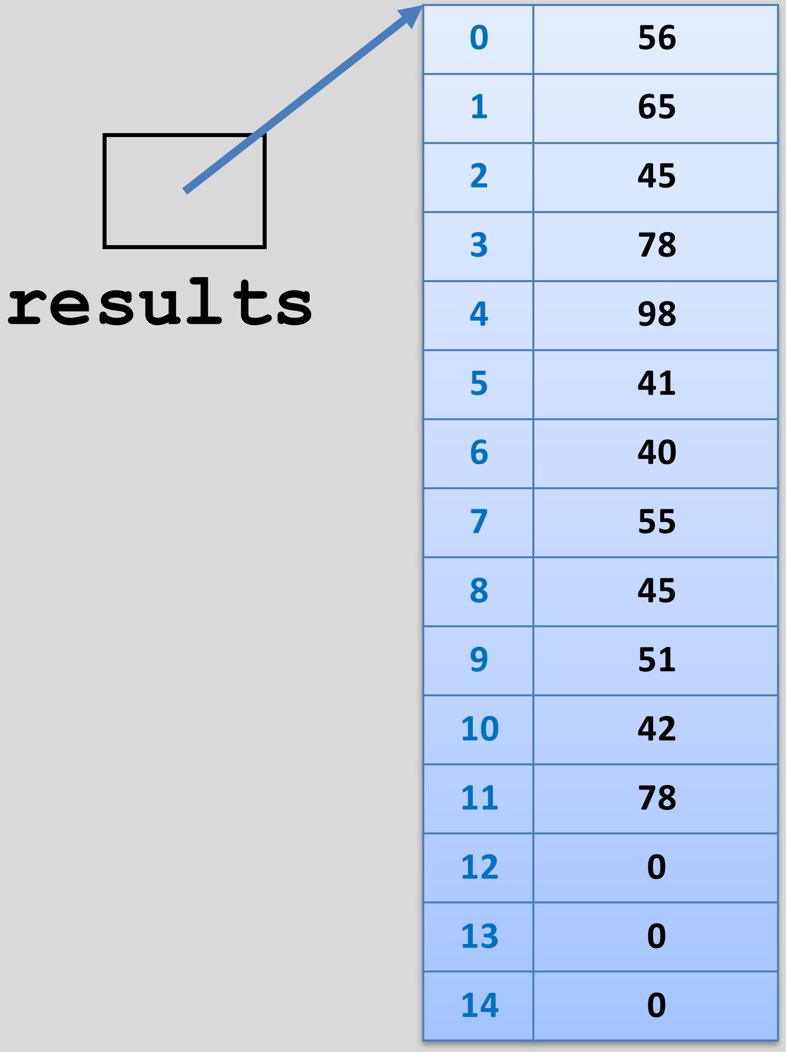
- Consider this scenario…

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



0	56
1	65
2	45
3	78
4	98
5	41
6	40
7	55
8	45
9	51
10	42
11	78
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];
```

OR

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

Questions?