Java Programming 2
Arrays

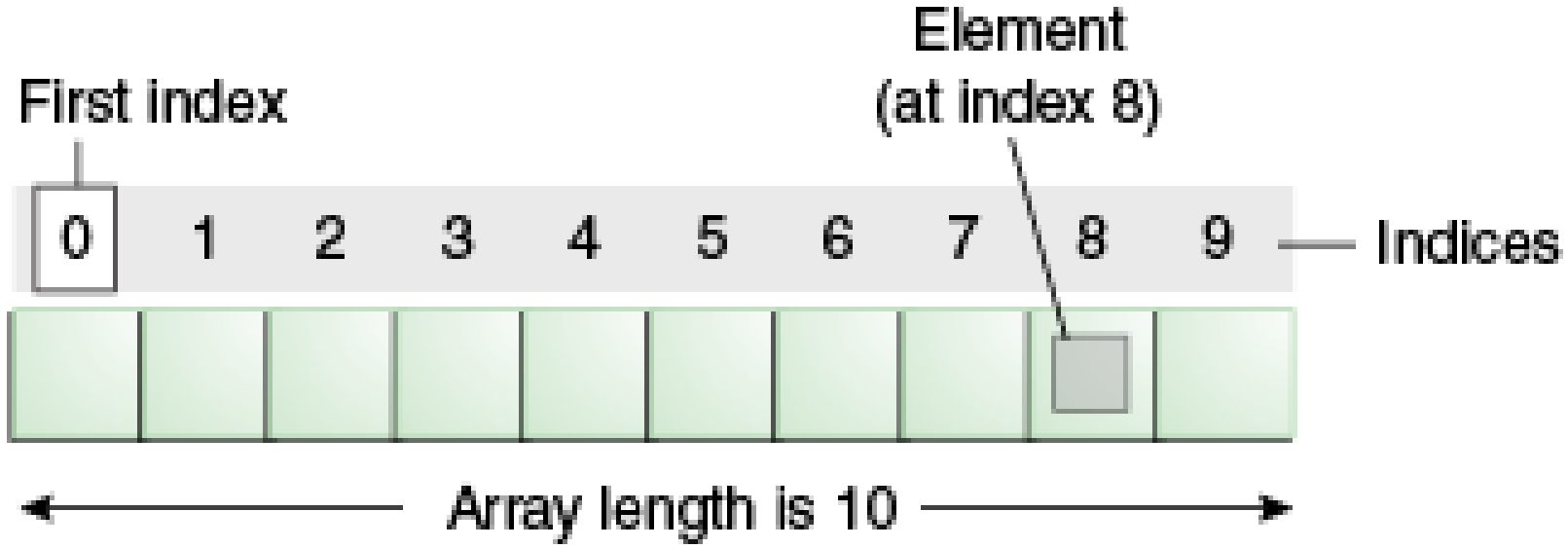
Mary Ellen Foster
MaryEllen.Foster@glasgow.ac.uk
Semester 1 2020/2021

Lego Heartlake Juice Bar product video from Lego.com

Array definition

A **fixed length** sequence of consecutive memory locations, indexed by an **integer** subscript

Supported directly by underlying Java Virtual Machine (JVM) => efficient



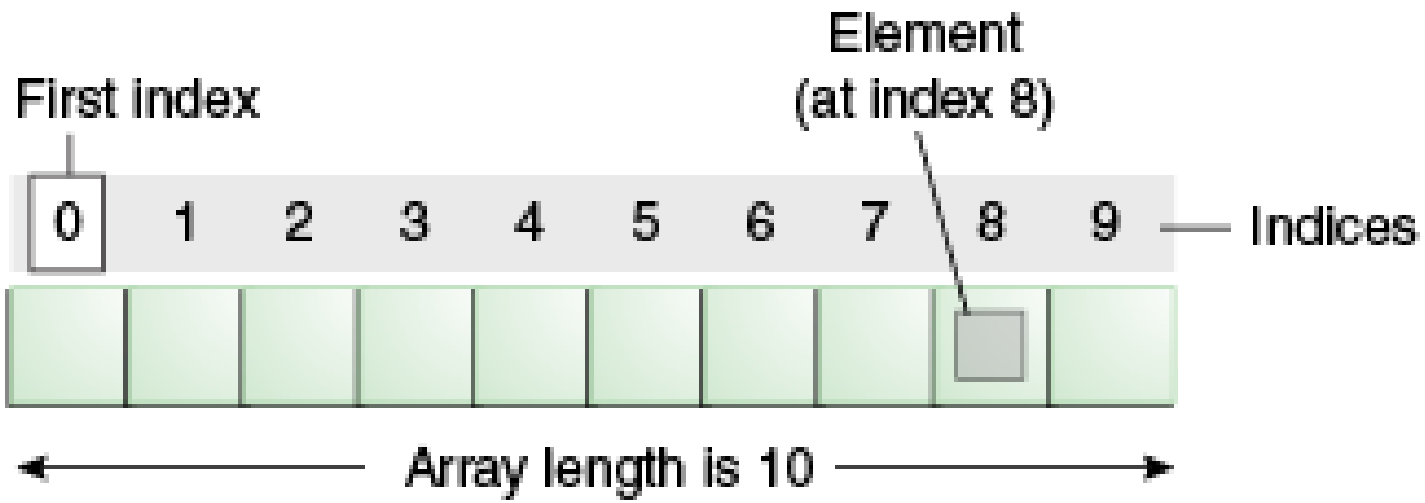
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Elements and indices

Each item in an array is called an **element**

Each element is accessed by its numeric **index** (plural **indices)**

Index numbering begins at 0 – e.g., array element #8 is the ninth element



Declaring an array

Each array has …

A **type** – the type of the individual elements in the array

A **dimension** – the dimensionality

Examples:

`int[]` – a one-dimensional array of integers

`String[][]` – a two-dimensional array of Strings

Variable declaration (method parameter, local variable, class field):

`String[] args`

(`String args[]` also valid – hang-over from C language style)

Initializing an array

Declaring an array **does not** …

- Reserve space for the array elements
- Specify the length of the array

Technically, it only creates a **reference** to point to the array

Two options for creating an array:

- Using **`new`** and giving an explicit size
- Using an "initializer" to give the initial values (shortcut syntax)

Initializing an array with `new`

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

Allocates enough space to store the specified number of elements

Fills the array with default values

- 0 for numeric types
- Null for object types

Initializing an array directly

Put a comma-separated list of elements between curly brackets

```
int[] values = { 10, 20, 30, 40, 50, };
```

Array length is determined from the length of the input list

(Nice feature: you can put a comma at the end of the list)

This syntax is only useful if you are **declaring** and **initializing** an array **at the same time**

Declaring and then initializing

~~int[] values;~~

~~values = { 10, 20, 30, 40, 50 };~~

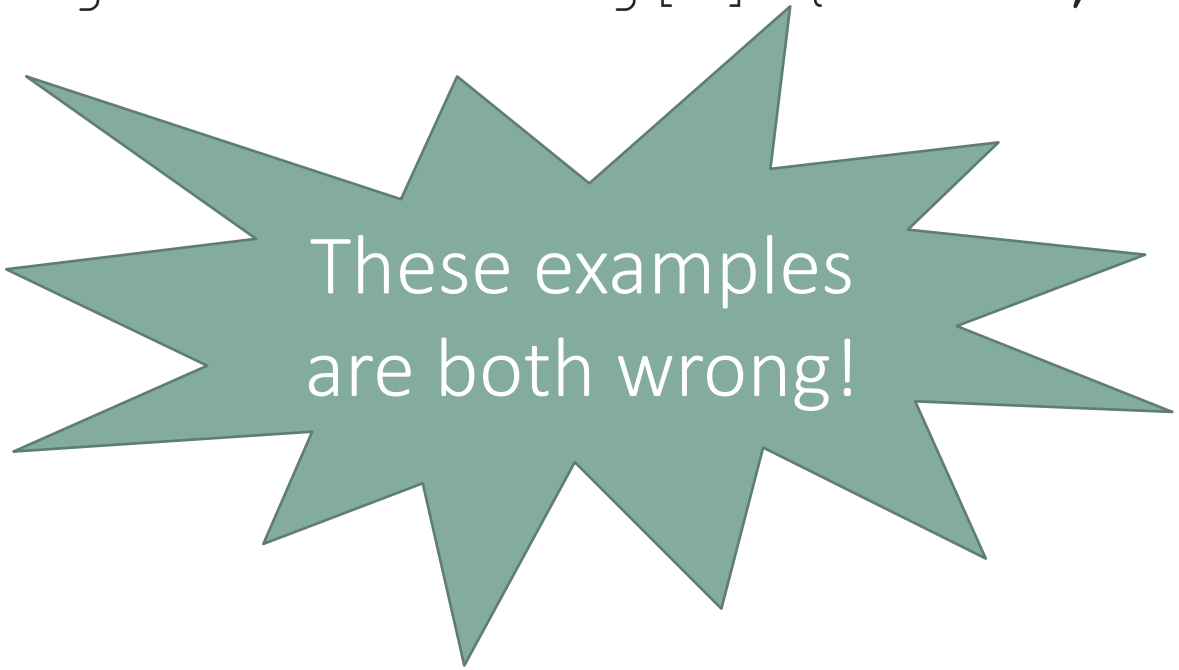
This is wrong!

```
int[] values;
values = new int[] { 10, 20, 30, 40, 50 };
```

Some other array errors

```
int[10] values;

String[] strings = new String[3] { “one”, “two”, “three” };
```



Accessing array elements

Use square brackets around index

```
int value = values[3];
```

Subscripts start at zero (as in Python – unlike other languages such as Fortran, COBOL, Matlab)

For multidimensional arrays, use multiple sets of brackets:

```
String[][] strings = // …;
System.out.println( strings[i][j] );
```

Array length

An array's length **cannot** be changed after it is initialized

Any attempt to use an out-of-range index will result in an error

Array length can be accessed through built-in field `length` (note: **not** a method)

```
String[] strings = { "Each", "peach", "pear", "plum", };
System.out.println (strings.length);
// Prints out 4
```

Iterating through an array

```
String[] fruits = new String[] { “apple”, “banana”, “cherry” ];
```

Standard Java idiom: **`for`** loop

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

More efficient option: ***for-each*** loop (since Java 1.5) – similar to Python’s **for** loop

```
for (String fruit : fruits) {
    System.out.println (fruit);
}
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Array manipulation with `Arrays` class

An array is essentially a (weird) object in Java

- One field (`length`)
- No methods except for those inherited from Object (`equals, toString`, etc)

The `java.util.Arrays` class provides a number of useful (static) methods

- `Arrays.copyOfRange` – copies (part of) an array into a newly created array
- `Arrays.equals` – compares two arrays to determine if they are equal
- `Arrays.fill` – fills an array with a specific element at each index
- `Arrays.sort` – sorts an array into ascending order
- `Arrays.toString` – returns a nicely formatted version of an array