Java Programming 2
Non-primitive types

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

Lego Heartlake Juice Bar product video from Lego.com

Non-primitive types

So far in the course we have covered **primitive types**

- In Java: all primitive types are numeric (integer, floating-point, char, Boolean)

From now on: **non-primitive types** (a.k.a. **composite**)

Non-primitive types in Java

- Strings
- Arrays
- Objects

Properties of primitive types (recap)

Built into the language

Cannot be decomposed into simpler components

Not a class

(technical) Variables of this type hold the value, not a reference

Image of a horse from the Lascaux caves.
https://commons.wikimedia.org/wiki/File:Lascaux2.jpg

Non-primitive types in Java

NOT ALWAYS built into the language

CAN be decomposed into simpler components

IS a class

(technical) Variables of this type hold **A REFERENCE, NOT THE VALUE**

Built into the language + classes

(Class: a definition of a type, including state and behaviour)

Some non-primitive types are built into Java – all are classes though

- Arrays
- Strings
- Things provided by built-in Java libraries

You can also define your own types

- Stay tuned … !

Decomposed into simpler components

Array:

int[] – made up of a set of integers

String[][] – a collection of String[]'s, each of which is a collection of String

String:

Made up of a sequence of characters

Objects:

State is represented as fields

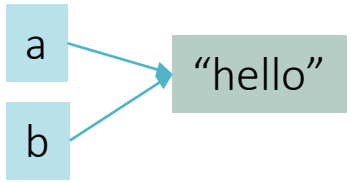
Variable of this type holds a reference

Primitive types: program stores **actual value**

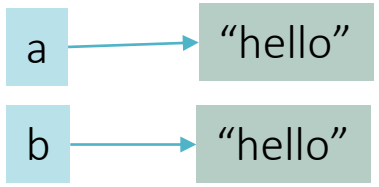
Non-primitive types: program stores **reference to the value**

Strings in Python and Java (from survey):

```
a = “hello”
b = “hello”
print (a == b)
```




```
String a = new String(“hello”);
String b = new String(“hello”);
System.out.println (a == b);
```



Comparing primitive vs non-primitive types

Primitive types:

- Variable stores **value**
- Comparing with “==“ and “!=“ compares **values**
- Other comparisons (<=, >=) are also possible on numeric values (in Java, all but **boolean**)

Non-primitive types:

- Variable stores **reference to value** (i.e., memory location)
- Comparing with “==“ and “!=“ compares **memory locations**
- To compare values, use type-specific methods
  - *Using >=, <=, etc won’t even compile if you try*

Sample code with doubles

```
double a = 5.5;
double b = 5.5;
if (a == b) {
    System.out.println("Equal with ==");
} else if (a.equals(b)) {
    System.out.println("Equal with .equals()");
}
```

Sample code with Strings

```
String a = new String(“hello”);
String b = new String(“hello”);
if (a == b) {
    System.out.println(“Equal with ==“);
} else if (a.equals(b)) {
    System.out.println(“Equal with .equals()”);
}
```

Sample code with arrays

```
int[] a = new int[] { 1, 2, 3 };
int[] b = new int[] { 1, 2, 3 };
if (a == b) {
    System.out.println("Equal with ==");
} else if (a.equals(b)) {
    System.out.println("Equal with .equals()");
} else if (Arrays.equals(a, b)) {
    System.out.println("Equal with Arrays.equals()");
}
```

Method parameters

What I said before:

"Changing the value of a parameter inside the method doesn't change its value externally"

What does that mean in the context of non-primitive types?

You can't change the **memory location**

But you can change the **contents of that location**

Arrays as method parameters

```
void doSomething (int[] values) {
    values = new int[] { 1, 2 };
}
void doSomethingElse(int[] values) {
    values[1] = 5;
}
int[] numbers = new int[] { 0, 1, 2, 3 };
doSomething(numbers);                    // What is numbers now?
doSomethingElse(numbers);                // What is numbers now?
```