Java Programming 2
Types, Integer division

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

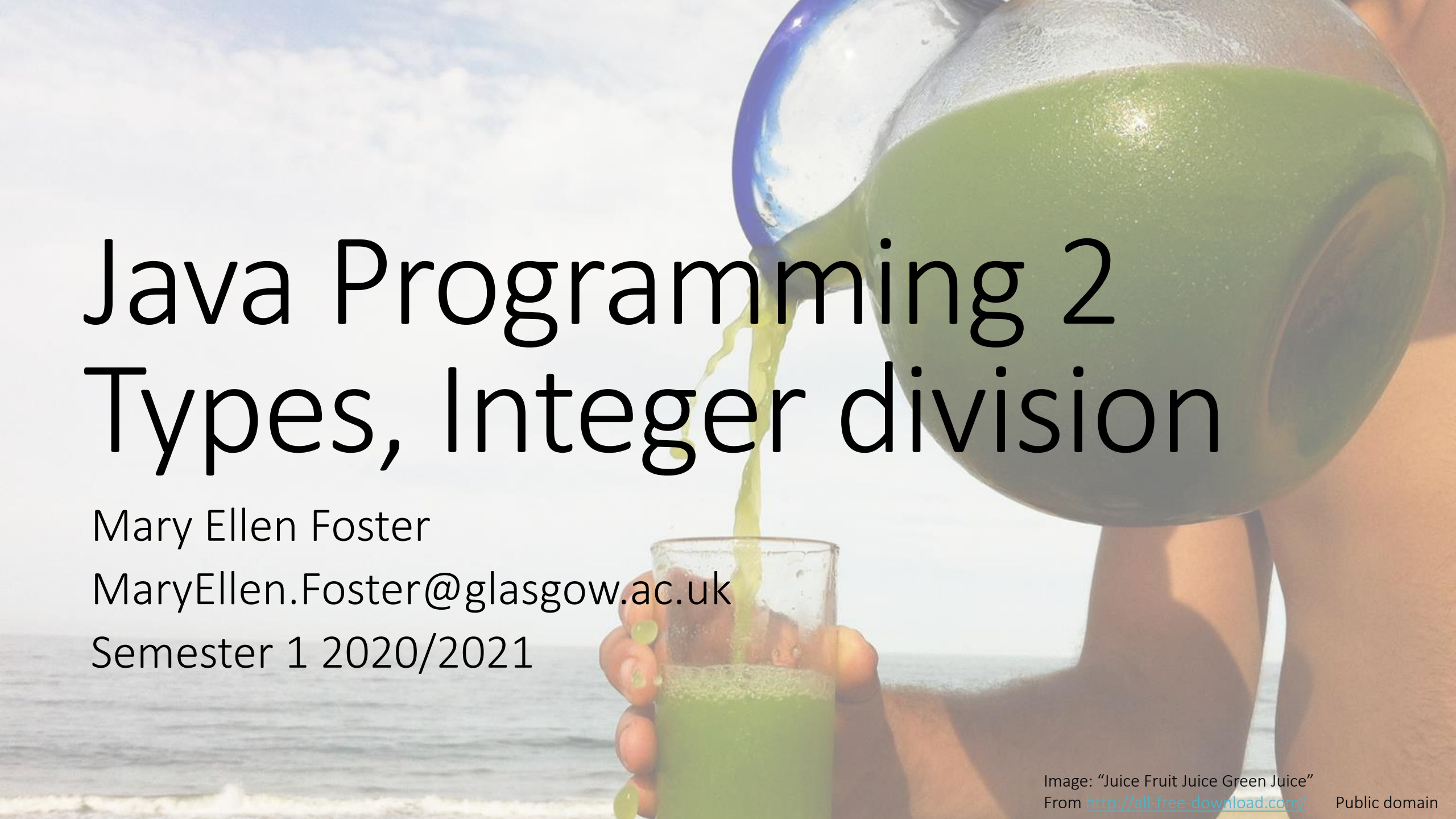

Image: "Juice Fruit Juice Green Juice"
From http://all-free-download.com/ Public domain

Primitive types in Java

"Primitive?"

- Built into the language
- Cannot be decomposed into simpler components
- Not a class
- *(technical)* Variables of this type hold the value, not a reference

See also:

- http://programmers.stackexchange.com/questions/139747/what-is-meant-by-a-primitive-data-type

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

List of Java primitive types

Type	Description	Min value	Max value	Default value
`byte`	8-bit signed integer value	-128	127	0
`short`	16-bit signed integer value	-32,768	32,767	0
`int`	32-bit signed integer value	-2^{31}	$2^{31} - 1$	0
`long`	64-bit signed integer value	-2^{63}	$2^{63} - 1$	0L
`float`	32-bit single precision IEEE 754 floating point value	2^{-149}	$(2-2^{-23})\cdot 2^{127}$	+0.0F
`double`	64-bit double precision IEEE 754 floating point value	2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	+0.0
`boolean`	Boolean value (true/false)	n/a	n/a	false
`char`	16-bit Unicode character value (use single quotes – e.g., `'c'`)	\u0000 (0)	\uffff (65,535)	\u0000

What about strings?

Technically, Java strings are of type `java.lang.String` which is an object (not primitive)

However, the Java language has special support for strings

E.g., you can create a new one by using double quotes

```
String s = "This is a string";
```

Also, they are **immutable** (will be explained later in the course)

So you will probably often tend to think of them as primitive – just don't forget that they are not!

Primitive wrapper classes

Every primitive type has an associated wrapper class

So you can treat them as objects if necessary *(relevant later)*

Primitive	Wrapper
`byte`	`Byte`
`short`	`Short`
`int`	**`Integer`**
`long`	`Long`
`float`	`Float`
`double`	`Double`
`boolean`	`Boolean`
`char`	**`Character`**

Useful operators for primitive types

Assignment = (single equals sign)

```
int i = 5;
```

Equality comparison == (two equals signs), != (not equal to)

```
if (i == 5) { … }
if (j != 5) { … }
```

Relational operators <, >, <=, >=

Boolean combinations && (and), || (or)

*Special considerations for String (because it's an **object**) – stay tuned*

Type conversions

Remember, Java is a **statically typed** language

Any variable can only hold values of a single, specific type

To store a value of type t_1 in a variable of type t_2, the value must be **converted to** t_2 **before** the assignment occurs

This is unlike Python where you can do something like this (and more!)

```
foo = 5
foo = True
foo = "bar"
foo = 0.5
```

Implicit (widening) conversions

Sometimes, type conversion can happen automatically with no extra source code

Generally, these are *widening* conversions – little or no information lost

`byte` to `long`

`long` to `double` *// potential loss of precision*

…

Precision may be lost, but the **magnitude** of the numeric value is preserved

```
int i = 5000;
long l = i;
double d = l;
```

Explicit (narrowing) conversions – casting

Some conversions would result in significant potential information loss

- `double` to `float`
- `int` to `short`
- …

These **narrowing** conversions must be made explicit in source code using *casting*: specify the target type in round brackets

```
int i = 1025;
byte b = (byte)i;
// b now has value 1
```

How does narrowing work?

`double` to `float`: loss of precision, plus …

- Out-of-range values become infinite
- Some non-zero values may become zero

`double/float` to `long/int/short/byte/char`

- Round-to-zero
- Out-of-range becomes infinity

Integer type to "smaller" integer type (e.g., `long` to `int`, `int` to `byte`, …)

- Discard all but *n* lowest-order bits
- In practice: value *mod* max

`String` ↔ primitive types

Converting from `String`

Use `parseXXX` methods of primitive wrapper classes

E.g., `int i = Integer.parseInt("42");`

Converting to `String`:

1. Just concatenate (+) with an existing `String` value
2. Use `String.valueOf(…)`

Arithmetic operators

Operator	Meaning	Precedence
–	Unary minus	Highest
+	Unary plus	Highest
*	Multiplication	Middle
/	Division	Middle
%	Remainder (mod)	Middle
+	Addition	Low
–	Subtraction	Low

Integer division

Function of division operator "/" depends on type of the two arguments

- If **both** are integers (`int, long, short, byte, char`), then it does **integer division**
- If **either** is floating-point (`float, double`), then it does **floating point division**

Example:

`7.0/4.0`	returns **`1.75`** (same result for `7.0/4` and `7/4.0`)
`7/4`	returns **`1`**

General rule: integer division throws away the remainder (so `99/100 == 0`)

More notes on integer division

Sign of the result is determined by normal rules of arithmetic

Strategy: compute as if both arguments were positive, then apply the rules

```
17/5 == 3     -17/5 == -3     17/-5 == -3     -17/-5 == 3
```

You can force Java to do floating-point division through **casting**

```
(double)17/5 == 3.4     17/(double)5 == 3.4
```

Mixing integer and floating-point in the same expression:

```
1.5 + 7/2
```


```
(1.5 + 7)/2
```