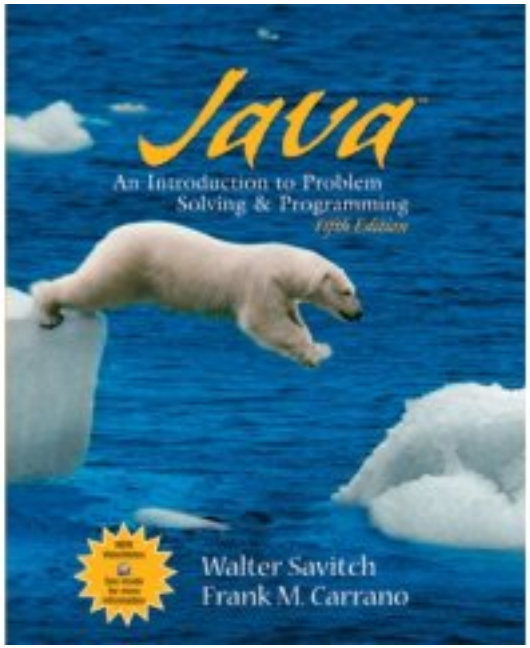
2. Primitive Types

Prof. Dr. Harald Gall
Michael Würsch

Institut für Informatik
Universität Zürich

http://seal.ifi.uzh.ch/info1



University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Learning Objectives

- Become familiar with the primitive types of Java (numbers, characters, etc.)
- Learn how to assign values to variables

Data Types in Java

Primitive types

- Atomic (non-decomposable) values
- *Examples: different kinds of numbers, characters*

Class types

- Composed of primitive types (and other class types)
- Can have instance variables and methods
- *Examples: strings, students, bank-accounts, application windows, files, etc.*

Primitive Types

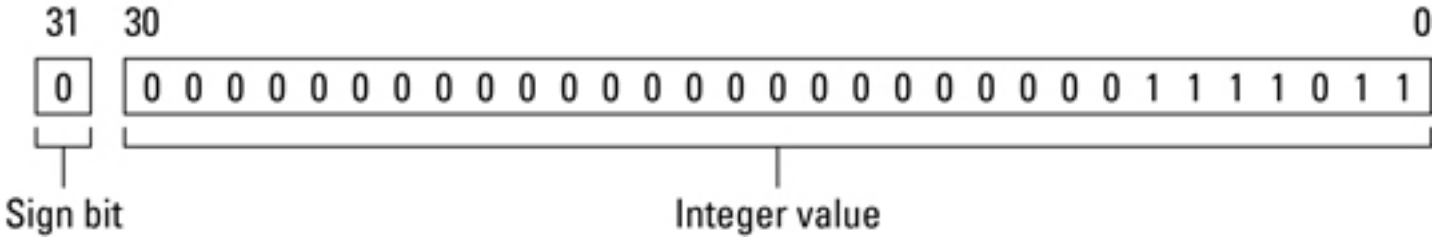
Type Name	Kind of Value	Memory Used	Size Range
`byte`	*integer*	*1 byte*	–128 to 127
`short`	*integer*	*2 bytes*	–32768 to 32767
`int`	*integer*	*4 bytes*	–2147483648 *to* 2147483647
`long`	*integer*	*8 bytes*	–9223372036854775808 *to* 9223372036854775807
`float`	*floating-point number*	*4 bytes*	$\pm 3.40282347 \times 10^{+38}$ *to* $\pm 1.40239846 \times 10^{-45}$
`double`	*floating-point number*	*8 bytes*	$\pm 1.76769313486231570 \times 10^{+308}$ *to* $\pm 4.9406564584124654 \times 10^{-324}$
`char`	*single character (Unicode)*	*2 bytes*	*all Unicode characters*
`boolean`	`true` *or* `false`	*1 bit*	*not applicable*

Display 2.2

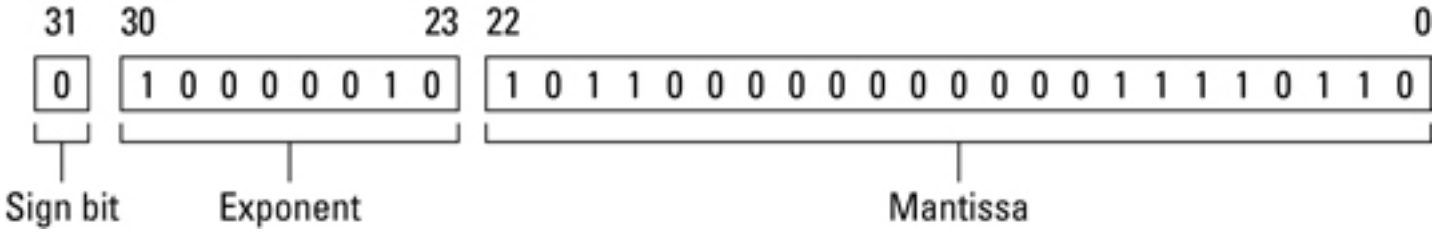
Primitive Types

Floating Point Number vs Integer

Integers can be stored as true binary values:



Floating-point numbers are stored differently



Assignments

Syntax:

```
<var name> = <value>;
```

Example:

```
int a, b;
a = 10;
b = 15;
int c_squared = a*a + b*b;
double d = 0.00483;
char firstInitial = 'M';
```

Shorthand Assignment Operators

Assignment operators can be combined with arithmetic operators (including `-`, `*`, `/`, and `%`).

```
amount = amount + 5;
```

can be written as

```
amount += 5;
```

yielding the same results.

Increment and Decrement Operators

A common situation is that of incrementing or decrementing an integer variable by one.

Shorthand operators:

```
i++;
i--;
```

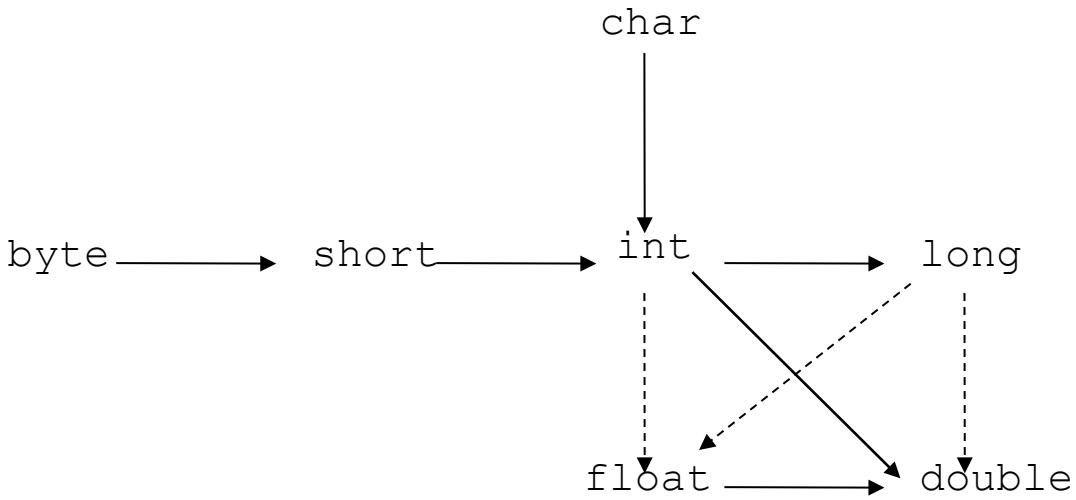
Assignment Compatibility

Since Java is strongly typed, assignments are only possible if no loss of information occurs.

```
double d = 100.5;
int i = d; // error

int i2 = 10;
double d2 = i2; // ok
```

Assignment Compatibility Chart



... Automatic Conversion without loss of information

... Automatic conversion with potential loss of information

Forced Conversion: Type Casting

A type cast temporarily changes the value of a variable from the declared type to some other type.

Warning: Any non-zero value to the right of the decimal point is truncated rather than rounded!

Example:

```
double distance = 9.5;
int points = (int) distance;
```

Automatic Conversions in Expressions

Arithmetic expressions can be formed using the `+`, `-`, `*`, and `/` operators together with variables or numbers referred to as *operands*

- When both operands are of the same type, the result is of that type.
- When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.
- if at least one of the operands is a floating-point type and the rest are integers, the result will be a floating point type.

The Division Operator

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, `99/100` has a value of `0`.

The `mod` Operator

- The `mod` (`%`) operator is used with operators of integer type to obtain the remainder after integer division
- 14 divided by 4 is 3 *with a remainder of 2*
  - Hence, `14 % 4` is equal to `2`
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:

```
(cost + tax) * discount
(cost + (tax * discount)
```

- Without parentheses, an expressions is evaluated according to the *rules of precedence.*

Precedence Rules

Highest Precedence

First: the unary operators: +, –, ++, – –, and !

Second: the binary arithmetic operators: *, /, and %

Third: the binary arithmetic operators: + and –

Lowest Precedence

Display 2.4

Precedence Rules

Precedence Rules, cont.

- The *binary* arithmetic operators `*`, `/`, and `%`, have *lower precedence* than the *unary* operators `+`, `-`, `++`, `--`, and `!`, but have *higher precedence* than the binary arithmetic operators `+` and `-`.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

Precedence Rules, cont.

- When unary operators have equal precedence, the operator on the right acts before the operation(s) on the left.
- Even when parentheses are not needed, they can be used to make the code clearer.

```
balance + (interestRate * balance)
```

- Spaces also make code clearer

```
balance + interestRate*balance
```

but spaces do not dictate precedence.

Sample Expressions

Ordinary Mathematical Expression	Java Expression (Preferred Form)	Equivalent Fully Parenthesized Java Expression
$rate^2 + delta$	`rate*rate + delta`	`(rate*rate) + delta`
$2(salary + bonus)$	`2*(salary + bonus)`	`2*(salary + bonus)`
$\frac{1}{time + 3\ mass}$	`1/(time + 3*mass)`	`1/(time + (3*mass))`
$\frac{a-7}{t+9v}$	`(a - 7)/(t + 9*v)`	`(a - 7)/(t + (9*v))`

Display 2.5

Arithmetic Expressions in Java