Chapter 2
## Java primitive data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>−128</td>
<td>+127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>−2^{15}</td>
<td>+2^{15} − 1</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>−2^{31}</td>
<td>+2^{31} − 1</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>−2^{63}</td>
<td>+2^{63} − 1</td>
</tr>
<tr>
<td>char</td>
<td>16 bits</td>
<td>Unicode 0</td>
<td>Unicode 2^{16} − 1</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>64-bit IEEE 754 floating point</td>
<td></td>
</tr>
</tbody>
</table>
Constants:

- type int: 1, -17
- type long: 1L, -17L, (long) -17
- type double: 1.0, -17.001, (double) -17
- type char: 'A', 'a', '1'
Constants:

- type int: 1, -17
- type long: 1L, -17L, (long) -17
- type double: 1.0, -17.001, (double) -17
- type char: 'A', 'a', '1'

Variables:

- must be declared before used:
  ```
  int i, j;
  i = 5;
  j = 2 * i;
  ```

- may be initialized at point of declaration:
  ```
  int i = 5, j;
  j = 2 * i;
  ```
Area0–2.java
The String type

A built-in reference type, but not a primitive type

String s1 = "hello ";
String s2 = "world";
String s  = s1 + s2;
    // plus means concatenation,
    // so now s is "hello world"

int i = 5;
String t = s1 + i;
    // plus will implicitly convert
    // numerical types to printable strings,
    // so now t is "hello 5"

System.out.println(t);  // prints "hello 5"
System.out.println(s1 + i);  // "
Area3.java
Integer Arithmetic

+ , − , * — addition, subtraction, multiplication

/ — division (truncation towards 0)

Examples:

• 5/2 = 2
• (-5)/2 = -2

% — remainder (or mod) operator

Invariant:

if q = a/b, r = a%b then a = b*q + r
Comparison with traditional “division with remainder” from mathematics

\[ [x] = \text{“floor” of } x \]
\[ = \text{greatest integer less than or equal to } x \]

\[ [2.5] = 2 \]
\[ [-2.5] = -3 \]

Invariant:
\[ \text{if } q = \lfloor \frac{a}{b} \rfloor, \ r = a \mod b \ \text{then} \ a = bq + r \]

Java integer division and “traditional” division with remainder differ only when the quotient is negative
Understanding division with remainder: clock arithmetic

\[ a = bq + r \]

- \( q \) — number of complete rotations
- \( r \) — the final offset
Understanding division with remainder: clock arithmetic

\[ a = bq + r \]

- \( q \) — number of complete rotations
- \( r \) — the final offset

Examples:

- \( a = 17, b = 7 \implies q = 2, r = 3 \)
Understanding division with remainder: clock arithmetic

\[ a = bq + r \]

- \( q \) — number of complete rotations
- \( r \) — the final offset

Examples:
- \( a = 17 \), \( b = 7 \)  \( \implies \)  \( q = 2 \), \( r = 3 \)
- \( a = -17 \), \( b = 7 \)  \( \implies \)  \( q = -3 \), \( r = 4 \) (traditional)
Understanding division with remainder: clock arithmetic

\[ a = bq + r \]

\[ \begin{array}{c}
0 \\
1 \\
2 \\
\vdots \\
\end{array} \]

\[ \begin{array}{c}
b-2 \\
b-1 \\
b \\
\vdots \\
\end{array} \]

- \( q \) — number of complete rotations
- \( r \) — the final offset

Examples:
- \( a = 17, \, b = 7 \implies q = 2, \, r = 3 \)
- \( a = -17, \, b = 7 \implies q = -3, \, r = 4 \) (traditional)
- \( a = -17, \, b = 7 \implies q = -2, \, r = -3 \) (Java)
Integer vs floating point division:

double x;
x = 1/10; // x is 0
x = 1/10.0; // x is 0.1
x = 1.0/10; // "
x = 1.0/10.0; // "

int a = 1, n = 10;
x = a/n; // x is 0
x = a/((double) n); // x is 0.1
x = ((double) a)/n; // x "