Naming Abstraction and Procedural Abstraction

Nothing is "from scratch".

Languages need built-in types — primitives.

Goal:
* Simplicity but "complete"
* Support for things done well in hardware

Java primitive types (examples)

int — an integer like 3, -5, 0
double — a decimal number like 3.5, -0.25, ...
char — a single character, like 'a', '#', '
boolean — true or false

also:

class String — sequence of characters — "Hello world!"
Names are useful! Naming abstraction — simplest form of abstraction

```c
int x = 3;  // declaration & an assignment
double y;   // just a declaration
```

Languages also need "means of combination"

⇒ primitive operators
Start with `int`.

Operators: `+`, `-`, `*`, `/`

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{2}{2} )</td>
<td>( \frac{2}{8} )</td>
</tr>
<tr>
<td>( 3 + 5 )</td>
<td>8</td>
</tr>
<tr>
<td>( 17 - (3 * 5) )</td>
<td>( \frac{2}{16} ) or 22?</td>
</tr>
<tr>
<td>( 6 + 5 * 2 )</td>
<td>16</td>
</tr>
</tbody>
</table>

Java respects order of operations.

Compiler creates a parse tree for each expression.

expression tree
\[(50 + 25 \times 3) / (8 - 3)\]

Expression Tree

\[
\begin{align*}
(50 + 25 \times 3) / (8 - 3) \\
50 + 75 / 5 \\
125 / 5 \\
25
\end{align*}
\]

Substitution Model
Double type: same operators as int

int

\[
\begin{align*}
7 / 2 & \equiv 3 \\
7 / 2 & \equiv 1 \\
5 / 10 & \equiv 0
\end{align*}
\]

double

\[
\begin{align*}
7.0 / 2 & \equiv 3.5 \\
5.0 / 10 & \equiv 0.5
\end{align*}
\]

Division by 0 is an error
Division by 0 is NaN
String Java overloads '+' operator
concatenation

Expression Value
"abc" + "def" "abcdef"

123 + "abc" "123abc"

"The value is \_\_\_" + 3 + 5 => The value is 8

"The value is \_\_\_" + 3 + 5 \(\Rightarrow\) The value is 35

"The value is " + 35
"The value is" + \((a + b)\)

**Boolean operators**

**Logical operators**

- `&&` — true when both sides are true
- `||` — true when either side (or both) is true
- `!` — negates the value

**Comparison operators**

- `==` — true when the sides are the same
- `!=` — true """" different
- `<`, `>`, `<=`, `>=` — as expected
Examples:

\[ 3 = 3 \quad \text{true} \]
\[ 3 \neq 3 \quad \text{false} \]
\[ 7 > 3 \quad \text{true} \]

\[ (7 > 3) \lor (7 \leq 3) \quad \text{true} \]
\[ (7 > 3) \land (7 \leq 3) \quad \text{false} \]
Procedural Abstraction

double areaOfCircle (double radius) {
    return Math.PI * (radius * radius);
}

radius → areaOfCircle → area
int abs (int x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}

|x| = \begin{cases} 
    x & \text{when } x \geq 0 \\
    -x & \text{otherwise}
\end{cases}

(can remove because first return would end the execution of the method when x >= 0)