Operators, Expressions, Statements, Control Flow
Operators

• An *operator* is a symbol (or combination of a couple symbols) that is used with variables and values to simplify how you write certain program *expressions*
  – Usually, operators are designed to mimic mathematical notation—but *do not be fooled* into confusing programming and mathematics!
# Most Common Operators

<table>
<thead>
<tr>
<th>String</th>
<th>boolean</th>
<th>char</th>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>++  --</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td></td>
<td>+  - +  -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td></td>
<td>+  - +  -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;  * /  % * /</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;  &gt;  &lt;=  &gt;=</td>
<td>&lt;  &gt;  &lt;=  &gt;=</td>
<td>&lt;  &gt;  &lt;=  &gt;=</td>
<td>&lt;  &gt;  &lt;=  &gt;=</td>
</tr>
</tbody>
</table>
# Most Common Operators

<table>
<thead>
<tr>
<th>String</th>
<th>boolean</th>
<th>char</th>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>++</td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;=</td>
<td>&lt;=</td>
<td>&gt;=</td>
<td>==</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
<td>!=</td>
<td></td>
<td>!=</td>
</tr>
<tr>
<td>&lt;</td>
<td>&gt;</td>
<td></td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>&lt;</td>
<td></td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
<td>==</td>
<td>!=</td>
<td>==</td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td>!=</td>
<td></td>
<td>!=</td>
</tr>
</tbody>
</table>

**Best Practice:** do not use `==` or `!=` with `Strings`, but rather the `equals` method; details later.
## Most Common Operators

<table>
<thead>
<tr>
<th>String</th>
<th>boolean</th>
<th>char</th>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>++</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operators for **or** (||) and **and** (&&) use **short-circuit evaluation**.
Most Common Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Best Practice**: be careful with the *remainder* (%) operator: the second operand must be positive; this is, unfortunately, not “clock arithmetic”; details later.
Most Common Operators

<table>
<thead>
<tr>
<th>String</th>
<th>boolean</th>
<th>char</th>
<th>int</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>++</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;=</td>
<td>&gt;=</td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>!=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>&gt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>!=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>&gt;=</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>!=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Best Practice**: do not check **doubles** for equality; details later.
Expressions

• An *expression* is a “syntactically well-formed and meaningful fragment” (roughly analogous to a *word* in natural language)

• Meaningful?
  – It has a value (of some type, of course)
Some Expressions

• Examples of code fragments that are expressions:

```java
i
j + 7
"Hello" + " World!"
keyboardIn.nextLine()
n == 0
new SimpleWriter1L()
```
Some Expressions

• Examples of code fragments that are expressions:

    i
    j + 7
    "Hello" + " World!"
    keyboardIn.nextLine()
    n == 0
    new SimpleWriter1L()
Some Expressions

• Examples of code fragments that are expressions:

```java
i
j + 7
"Hello" + " World!
keyboardIn.nextLine()
n == 0
new SimpleWriter1LL()
```

This fragment creates a new object of type `SimpleWriter1LL`, and its value is a `reference` to that object; details later.
Statements

• A **statement** is a “smallest complete unit of execution” (roughly analogous to a **sentence** in natural language)

• A simple statement is terminated with a semi-colon `;`
Simple Statements

• Some examples of simple statements:

\[
i = 12; \\
j += 7; \\
k++; \\
\text{SimpleWriter } fileOut = \text{new SimpleWriter1L("foo.txt");} \\
fileOut.print("Hi, Mr. Foo.");
\]
Simple Statements

• Some examples of simple statements:

\[
i = 12; \\
j += 7; \\
k++; \\
\text{SimpleWriter fileOut = new SimpleWriter1L("foo.txt");} \\
\text{fileOut.print("Hi, Mr. Foo.");} \\
\]

This is the same as

\[
j = j + 7; \\
\]
Assignment Statement

• Assignment statement form:
  \[
  \text{variable} = \text{expression};
  \]

• **Copies** the value of the expression on the right side of the assignment operator \(=\) to the variable on the left side

• The \(=\) in Java code does not mean “equals” like in math!
  – Recall the tracing table earlier?
Compound Statements/Blocks

• Any sequence of zero or more statements enclosed in {...} is a *block*

• Example:

```java
{
    String s = in.nextLine();
    out.println("s = " + s);
}
```
Compound Statements/Blocks

• Any sequence of zero or more statements enclosed in {...} is a **block**

• Example:

```java
{ 
    String s = in.nextLine();
    out.println ("s = " + s);
}
```

The **scope** of variable `s` is just the block in which it is declared.
Compound Statements/Blocks

• Any sequence of zero or more statements enclosed in {...} is a **block**

• Example:

```java
{  
  String s = in.nextLine();
  out.println("s = " + s);
}
```

There is no semi-colon after a block.
Control Flow

• **Conditional** or **selection** statements
  – if
  – if-else
  – if-else-if
  – switch

• **Loop** or **iteration** statements
  – while
  – for
  – do-while
Control Flow

• **Conditional or selection** statements
  – if
  – if-else
  – if-else-if
  – switch

• **Loop or iteration** statements
  – while
  – for
  – do-while

We will normally use these, but you may use a switch statement if you like; details later.
Control Flow

• Conditional or selection statements
  – if
  – if-else
  – if-else-if
  – switch

• Loop or iteration statements
  – while
  – for
  – do-while

We will normally use while loops, but you may use the others if you like.
if (test) {
    then_block
}

if Statement
**if** Statement

```java
if (test) {
    then_block
}
```

Any **boolean** expression may go here.
if Statement

```java
if (test) {
    then_block
}
```

**Best Practice**: even a single statement here should be in a block.
if-else Statement

if (test) {
    then_block
} else {
    else_block
}
if (test) {
    then_block
} else {
    else_block
}
### if-else-if Statement

```cpp
if (test_1) {
    then_block_1
} else if (test_2) {
    then_block_2
} else {
    else_block
}
```

The `else if` part may be repeated.
**if-else-if** Statement

```plaintext
if (test_1) {
    then_block_1
} else if (test_2) {
    then_block_2
} else {
    else_block
}
```

Can you draw a **flow-chart** for this statement?
while Statement

while (test) {
    while_block
}

while block diagram
while (test) {
    while_block
}

Control flow here can go backward, which creates a loop in the flow chart.
if-else

if (test) {
    then_block
} else {
    else_block
}
Expressions and Statements

```java
public static void main(String[] args) {
    SimpleWriter output = new SimpleWriter1L();

    int x = 1, count = 0, n = 12345;

    while (x < n) {
        if (n % x == 0) {
            output.println(x);
            count = count + 1;
        }
        x++;
    }

    output.println("Number of factors: " + count);
    output.close();
}
```
### Best Practices for boolean

**If you want to say this, e.g., in an if or while condition:**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Say this instead:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b == true</code></td>
<td><code>b</code></td>
</tr>
<tr>
<td><code>b == false</code></td>
<td><code>!b</code></td>
</tr>
<tr>
<td><code>if (b) {</code></td>
<td></td>
</tr>
<tr>
<td>return true;</td>
<td></td>
</tr>
<tr>
<td>} else {</td>
<td></td>
</tr>
<tr>
<td>return false;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>return b;</code></td>
</tr>
</tbody>
</table>
Resources

• *Java for Everyone*, Chapter 3
• *Java for Everyone*, Chapter 4