Writing boolean Expressions
- The logical operators allow us to write more complex if-conditions, to check more complex properties
- E.g., we can check if integer $x$ is between 1 and 10 (inclusive):
  ```java
  if ( (x >= 1) && (x <= 10) ) {
    System.out.println( x + " is between 1 and 10" );
  }
  ```
- Sometimes this is just for convenience: for example, both of the following pieces of code do the same thing:
  ```java
  if (x <= 0) {
    System.out.println( x + " is <= than 0" );
  }
  if (! (x > 0)) {
    System.out.println( x + " is <= than 0" );
  }
  ```

Evaluating boolean Expressions
- The simple numeric comparison operators and equality relations are easy to evaluate
- To determine whether a more complex logical formula is true or false, we use truth tables for logical operators

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P &amp;&amp; Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P || Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Step 1: evaluate the highest-priority operator ($! = \text{NOT}$)
Step 2: evaluate the lower-priority operator ($\&\& = \text{AND}$)
Evaluating boolean Expressions

- For example, suppose we have variables:
  
  \[ x = -3, \quad y = 0 \]

- Then, we can evaluate the following:

\[ ( x \leq y ) \land \neg ( y > 10 \lor y = 1 ) \]

Some Legal and Illegal Expressions

<table>
<thead>
<tr>
<th>Code</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i1 = 3; int i2 = 4; char c = 'e';</code></td>
<td></td>
</tr>
<tr>
<td><code>if ( i1 != i2 ) { ... }</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>if (!(i1 != i2)) { ... }</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>if ((0 &lt; i1) &amp;&amp; (i2 &gt;= 4)) { ... }</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td>`if ((c == 'a')</td>
<td></td>
</tr>
<tr>
<td><code>if ( (i1 != i2) ) { ... }</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>if ( i1 &lt; = i2 ) { ... }</code></td>
<td><code>ERROR</code></td>
</tr>
<tr>
<td><code>if ( 0 &lt;= i1 &lt;= 5 ) { ... }</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

Some Exercises

- Suppose `num1` and `num2` are integer variables:
  1. Write code that prints out the sum of the variables if they are both less than 7.
  2. Write code that prints out the sum of the variables if `num1` is less than 7, but `num2` is not.
  3. Write code that prints out the largest of the two variables; if they are equal, it should print out the word **EQUAL**.

Short-Cut Evaluation

- Programmers are always trying to make programs run faster, by never doing more work than is needed
- For example, the creators of Java wanted to make sure that we could get the final output value of a complicated boolean expression as fast as possible
- Therefore, when a program executes, it will use some short-cuts in evaluating these expressions:
  1. If either side of a conjunction (\&\&) is false, the entire expression is also false. **We can stop.**
  2. If either side of a disjunction (||) is true, the entire expression is also true. **We can stop.**
Short-Cut Evaluation

- Because of the short-cuts, things that are logically equivalent in do not always do the same thing in a program.
- In logical terms, e.g., these two things always mean exactly the same:

```java
if ( X && Y ){
    ...
}

if ( Y && X ){
    ...
}
```
- But in Java execution, there is a difference—since we go left-to-right in evaluating the expression, they each work differently:

```java
if ( X && Y ){
    ...
}

if ( Y && X ){
    ...
}
```

Using Short-Cut Evaluation

- We can sometimes use this feature to our advantage.
- As an example, we can order our instructions so that the first one that is false causes the evaluation of the boolean condition to stop immediately.
- This allows us to write code that is more compact and less buggy.

```java
String data = input.getText();
if ( data.charAt( 5 ) == 'a' && data.length() > 5 ){
    System.out.println( "One thing" );
}
```

```java
String data = input.getText();
if ( data.length() > 5 && data.charAt( 5 ) == 'a' ){
    System.out.println( "Another thing" );
}
```

This Week & Next

- Meetings this week:
  - Wednesday: in the CS Lab (16 Wing)
  - Monday/Tuesday/Friday: regular classroom
- Program 02: due 11:59 PM, Wednesday 19 February
- Reading: Chapter 4 due Noon, Friday 21 February
- Office Hours: Wing 212
  - Monday/Wednesday/Friday: 11:00 AM–12:00 PM
  - Tuesday: 3:15 PM–4:15 PM
- CS Lab & Tutor Hours: Posted on my webpage