Boolean Data Type

15-110 Summer 2010
Margaret Reid-Miller
Boolean values

- Named after George Boole (1815-1864), who invented mathematical logic and defined Boolean algebra.
- A variable of the primitive data type `boolean` can have two values: `true` and `false` (Boolean literals).
- Boolean variables are used to indicate whether a condition is true or not, or to represent two states, such as a light being on or off.

E.g.,

```java
boolean hasLicense;
boolean isDone = false;
boolean isTurnedOn = true;
```
Expressions

- Up to now we have seen
  - *arithmetic expressions* that use the operators + - * / % ++ --
  - *assignment expressions* that use the operators = += -= …
- *Boolean expressions* use *relational* and *logical* operators.
- The *result* of a Boolean expression is either *true* or *false*.
- Boolean expressions allow us to write programs that decide whether to execute some code or not.
- These decisions changes the *flow* of the program execution.
Relational operators

- **Relational operators** compare two arithmetic expressions and evaluate to a boolean result.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>

**Careful:** Do not confuse the assignment operator `=` with the equality operator `==`. 
Relational operators

• The relational operators determine the *relative ordering* between values.

• The relational operators may be used to compare *expressions* that evaluate to numeric and char data.

• These relational operators have *lower precedence* than the arithmetic operators.
  
  • Thus, arithmetic expressions are evaluated first, then the resulting Boolean expressions.
  
  • That is, Java does the “math” first, then the comparison.
Relational operators

Examples:

```java
int x = 15;
int y = 100;
System.out.println(x > y);
System.out.println(x < 15);
System.out.println(x <= 15);
System.out.println(x == y);
System.out.println(x != 5);
System.out.println(x * -y > 0);
boolean isBigger = x > y;
```
Logical operators

- *Logical operators* combine *boolean* values and evaluate to a *boolean* result.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Logical NOT</td>
<td>!a</td>
<td>true if a is false, false if a is true</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>a &amp;&amp; b</td>
<td>true if both a and b are true, false otherwise</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Logical OR</td>
</tr>
</tbody>
</table>
Truth Tables

• Truth tables list all possible combinations of values for the variables in an expression.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a &amp;&amp; b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Logical operators

Example:

<table>
<thead>
<tr>
<th>age &gt; 26</th>
<th>hasLicense</th>
<th>(age &gt; 26) &amp;&amp; hasLicense</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>
</tbody>
</table>

boolean canRentCar = (age > 26) && hasLicense;
Logical operators

Example:

\[
\begin{array}{|c|c|c|}
\hline
\text{age} > 26 & \text{hasLicense} & (\text{age} > 26) \land \text{hasLicense} \\
\hline
\text{true} & \text{true} & \text{true} \\
\hline
\text{true} & \text{false} & \text{false} \\
\hline
\text{false} & \text{true} & \text{false} \\
\hline
\text{false} & \text{false} & \text{false} \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{int age} & \,= \, 16; \\
\text{boolean hasLicense} & \,= \, \text{true}; \\
\text{boolean canRentCar} & \,= \, (\text{age} > 26) \land \text{hasLicense};
\end{align*}
\]
Logical operators

Examples:

```java
int x = 15;
int y = 100;
System.out.println(x > y && x >= 15);
System.out.println(x < 15 || x > 15);
System.out.println(x == y && y == 100);
System.out.println(x != 5 && x < y);
System.out.println(x + y > 100 || y <= 10);
```

&& is evaluated after relational operators. || is evaluated after &&.
Logical operators: Exercise 1

• It is time to buy a new phone when at least one of the following situations occurs:
  • the phone breaks
  • the phone is at least 3 years old

```java
int phoneAge;       // in years
Boolean isBroken;
...
// code initializes variables

boolean needPhone = _________________________________;
```
Logical operators: Exercise 1

• It is time to buy a new phone when at least one of the following situations occurs:
  • the phone breaks
  • the phone is at least 3 years old

```java
int phoneAge;       // in years
Boolean isBroken;
...
// code initializes variables

boolean needPhone = (isBroken == true) || (phoneAge >= 3);
```
Logical Operators: Exercise 2

• Assume x, y, and z are int variables that have been initialized.

```java
boolean areAllEqual = ____________________________;
```
Logical Operators: Exercise 2

- Assume $x$, $y$, and $z$ are int variables that have been initialized.

  ```java
  boolean areAllEqual = (x == y) && (y == z);
  ```
Boolean Algebra

• Double negative: \( !!a \equiv a \)

• de Morgan’s Law:
  \[
  ! (a && b) \equiv !a || !b \\
  ! (a || b) \equiv !a && !b
  \]
de Morgan’s Law (version 1)

Truth table: Consider all possible combinations of values of booleans $a$ and $b$.

$! (a \&\& b) = (\neg a \mid\mid \neg b)$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$a &amp;&amp; b$</th>
<th>$! (a &amp;&amp; b)$</th>
<th>$!a$</th>
<th>$!b$</th>
<th>$!a \mid\mid !b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**de Morgan’s Law (version 1)**

Truth table: Consider all possible combinations of values of bools `a` and `b`.

\[ !(a \&\& b) == (!a | | !b) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a &amp;&amp; b</th>
<th>!(a &amp;&amp; b)</th>
<th>!a</th>
<th>!b</th>
<th>!a</th>
<th></th>
<th>!b</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
de Morgan’s Law (version 2)

Truth table: Consider all possible combinations of values of booleans \(a\) and \(b\).

\[!(a \lor b) = (\neg a \land \neg b)\]

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(a \lor b)</th>
<th>(\neg(a \lor b))</th>
<th>(\neg a)</th>
<th>(\neg b)</th>
<th>(\neg a \land \neg b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
de Morgan’s Law (version 2)

Truth table: Consider all possible combinations of values of booleans \(a\) and \(b\).

\(\neg(a \lor b) \equiv (\neg a \land \neg b)\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(a \lor b)</th>
<th>(\neg(a \lor b))</th>
<th>(\neg a)</th>
<th>(\neg b)</th>
<th>(\neg a \land \neg b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
de Morgan’s Law

In Java:

!((age < 12) || (age >= 65))

In English: *It is not the case that age less than 12 or age greater than or equal to 65.* !!!?

Simplify using de Morgan’s Law:

!(age < 12) && !(age >= 65)

The reverse the meaning of the relational expressions:

(age >= 12) && (age < 65)

That is, *when age is at least 12 and less than 65.*
de Morgan’s Law

In English:
Words neither rhyme nor alliterate.

In Java:

```
!wordsRhyme && !wordsAlliterate
```
Words don’t rhyme and they don’t alliterate

Apply de Morgan’s Law:

```
!(wordsRhyme || wordsAlliterate)
```
It’s not the case words rhyme or alliterate.