Complex conditions can be formed with the Boolean operators: and (&&), or (||), not (!)

Assume that b1 and b2 are predicate methods:

- \( b1() \land b2() \) is true only if \( b1() \) and \( b2() \) are both true
- \( b1() \lor b2() \) is true, if either \( b1() \) is true or \( b2() \) is true
- \( \neg b1() \) is true if \( b1() \) is false

The Boolean operators \&\& and || are binary infix operators, this simply means that they are placed between two Boolean operands, which for now are just predicate methods.

The Boolean operator ! is a unary prefix operator, it is placed immediately preceding one Boolean operand, (predicate method).
Truth Tables

The semantics of the Boolean operators are defined by the following "truth tables" (where A and B represent Boolean expressions (predicate methods):

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp;&amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

| A   | B   | A || B |
|-----|-----|-------|
| true| true| true  |
| true| false| true  |
| false| true| true  |
| false| false| false |
Boolean Operators

Equivalent IFs

- Operators provide for the formation of logically equivalent code:

```java
if (frontIsClear()) {
    move();
} else {
    jumpHurdle();
}
```

```java
if (! frontIsClear()) {
    jumpHurdle();
} else {
    move();
}
```

- Can be used to eliminate empty IF clauses:

```java
if (<test>) {
    doNothing();
} else {
    <instruction>;
}
```

```java
if (! <test>) {
    <instruction>;
}
```
Equivalence of Expressions

- Some Boolean expressions are equivalent

\[ !!b() == b() \]
\[ !(b1() && b2()) == !b1() || !b2() \]
\[ !(b1() || b2()) == !b1() && !b2() \]

- Nested IF Equivalence

```java
if (frontIsClear()) {
    if (wallOnRight()) {
        move();
    }
}
```

```java
if (frontIsClear() && wallOnRight()) {
    move();
}
```
Complex logical expressions involving Boolean operators are evaluated according to the following hierarchy:

- ()  // evaluation works from the innermost to outermost parenthesis
- !  // unary negation operators are evaluated next
- &&  // logical AND operators are evaluated next
- ||  // finally logical OR operators are evaluated