Conditions and Loops

Examples

7 == 5 // evaluates to false.
5 > 4 // evaluates to true.
3 != 2 // evaluates to true.
6 >= 6 // evaluates to true.
5 < 5 // evaluates to false.

int a = 2;
int b = 3;
int c = 6;

a == 5 // evaluates to false since a is not equal to 5.
a*b >= c // evaluates to true since (2*3 >= 6) is true.
b+4 > a*c // evaluates to false since (3+4 > 2*6) is false.
(b+2) == a // evaluates to true.

Relational Operators

== Equal to
!= Not equal to
> Greater than
< Less than
>= Greater than or equal to
<= Less than or equal to
Logical Operators

• ! performs the Boolean operation NOT
• It has only one operand, on the right, and the only thing that it does is invert the value, producing false if its operand is true and true if its operand is false.

!(5 == 5)  // false because (5 == 5) is true.
!(6 <= 4)  // true because (6 <= 4) is false.
!true      // false
!false     // true.

Logical Operators

The operator && corresponds to Boolean logical operation AND. This operation results true if both its two operands are true and false otherwise.

<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>

The operator || corresponds to Boolean logical operation OR. This operation results true if either one of its two operands is true, false only when both operands are false.

| a     | b     | a || b |
|-------|-------|--------|
| true  | true  | true   |
| true  | false | true   |
| false | true  | true   |
| false | false | false  |

Examples

((5 == 5) && (3 > 6))  // evaluates to false ( true && false ).
((5 == 5) || (3 > 6))  // evaluates to true ( true || false ).

if-then statement

void applyBrakes()
{
    // the "if" clause: bicycle must be moving
    if (isMoving)
    {
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}

void applyBrakes()
{
    // same as above but don’t need braces for one statement
    if (isMoving)
        currentSpeed--;
if-then-else statement

```java
void applyBrakes()
{
    if (isMoving)
    {
        currentSpeed--; // Decrements current speed
    }
    else
    {
        System.out.println("The bicycle has already stopped");
    }
}
```
while statement

```java
int count = 1;
while (count < 11) {
    System.out.println("Count is: "+ count);
    count++;
}
```

do-while

- In while statement, test is performed before each loop
- In the do-while, the test is performed after statements will be performed at least once

```java
do {
    statement(s)
  } while (expression)
```

The for loop

```java
for (initialisation; termination; increment) {
    statement(s)
}
```

- **initialisation** expression initialises the loop.
  - Executed once, as the loop begins.
- **When termination** expression is false, loop terminates.
- **increment** expression invoked after each iteration through the loop.
  - It may increment or decrement a value.

Example

```java
void count() {
    for (int i = 1; i < 11; i++)
        { System.out.println("Count is: "+ i);
    }
}
```

The output of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```
for loop is a shortcut

- the for loop is exactly the same as:

```java
initialisation;
while (termination)
{
    statement(s);
    increment;
}
```