Conditionals

Topics

- Relational operators.
- Logical operators.
- Conditional statements
  - if
  - if/else
  - switch
Relational Operators

- Relational operators produce results of type boolean (true or false).
- These operators can be used to compare values of numerical types (byte, short, char, int, long, float, and double):
  - Equality operator: ==
  - Less than operator: <
  - Greater than operator: >
  - Less than or equal to operator: <=
  - Greater than or equal to operator: >=
  - Not equal operator: !=

Relational Operators (2)

- Booleans and compatible reference types can be compared with the equality (==) and inequality (!=) operators.
  - In the case of reference types, these operators compare the references, not the object values.
- The instanceof operator tests if an object belongs to a particular class or one of its subclasses.
Relational Operator Examples

```java
int i = 10, j = 20, k = 20;
String s1 = "hello", s2 = "hello", s3 = "goodbye";
JFrame frame = new JFrame();
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i &lt; j</td>
<td>true</td>
</tr>
<tr>
<td>i &lt;= j</td>
<td>true</td>
</tr>
<tr>
<td>i == j</td>
<td>false</td>
</tr>
<tr>
<td>j &gt; i</td>
<td>true</td>
</tr>
<tr>
<td>j &gt; i + 10</td>
<td>false</td>
</tr>
<tr>
<td>j &gt;= i + 10</td>
<td>true</td>
</tr>
<tr>
<td>i != j</td>
<td>true</td>
</tr>
<tr>
<td>frame instanceof Container</td>
<td>true</td>
</tr>
<tr>
<td>s1 == s3</td>
<td>false</td>
</tr>
<tr>
<td>s1 &lt; s3</td>
<td>error</td>
</tr>
<tr>
<td>s1 == s2</td>
<td>???</td>
</tr>
</tbody>
</table>

Logical Operators

- Operate on boolean values to produce new boolean values.
  - AND: &&, 
  - OR: ||, |
  - XOR (exclusive or): ^
  - NOT: !
Logical Operators (2)

For boolean expressions p and q:

<table>
<thead>
<tr>
<th>Expression</th>
<th>True When</th>
</tr>
</thead>
<tbody>
<tr>
<td>p &amp;&amp; q</td>
<td>p and q are both true</td>
</tr>
<tr>
<td>p &amp; q</td>
<td>p and q are both true</td>
</tr>
<tr>
<td>p</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>p ^ q</td>
<td>p is true or q is true, but not both</td>
</tr>
<tr>
<td>!p</td>
<td>p is false</td>
</tr>
</tbody>
</table>

&& and || vs. & and |

- The && and || operators always evaluate their first operand, but only evaluate the second when it is necessary to determine the overall value of the expression.
  - In (p && q), q is not evaluated if p is false.
  - In (p || q), q is not evaluated if p is true.
- & and | always evaluate both operands.
  - In practice, these operators are rarely used.
- The difference is significant if evaluating q may produce side effects, e.g. if q involves sending a message or performing an assignment.
Logical Operator Examples

```java
int i = 10, j = 20, k = 30;
boolean p = false, q = true;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(i &lt; j) &amp;&amp; (j &lt; k)</code></td>
<td>true</td>
</tr>
<tr>
<td>`(i &lt; j)</td>
<td></td>
</tr>
<tr>
<td><code>(i &lt; j) ^ (j &lt; k)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>(i &lt; j) &amp;&amp; (j &gt; k)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>(i &lt; j) &amp;&amp; !(j &gt; k)</code></td>
<td>true</td>
</tr>
<tr>
<td>`(i &lt; j)</td>
<td></td>
</tr>
<tr>
<td><code>(i &lt; j) ^ (j &gt; k)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>p == q</code></td>
<td>false</td>
</tr>
<tr>
<td><code>p = q</code></td>
<td>true (now p is true)</td>
</tr>
<tr>
<td><code>i &gt; j &amp;&amp; j &lt; (k = 5)</code></td>
<td>false (k is unchanged)</td>
</tr>
<tr>
<td><code>i &gt; j &amp;&amp; j &lt; (k = 5)</code></td>
<td>false (k is changed to 5)</td>
</tr>
</tbody>
</table>

More Precedence Rules

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical NOT</td>
<td>!</td>
</tr>
<tr>
<td>multiplicative</td>
<td>* / %</td>
</tr>
<tr>
<td>additive</td>
<td>+ -</td>
</tr>
<tr>
<td>relational</td>
<td>&lt; &gt; &lt;= &gt;= instanceof</td>
</tr>
<tr>
<td>equality</td>
<td>== !=</td>
</tr>
<tr>
<td>logical AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>logical exclusive OR</td>
<td>^</td>
</tr>
<tr>
<td>logical OR</td>
<td></td>
</tr>
<tr>
<td>logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>logical inclusive OR</td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>= += -= *= /= %=</td>
</tr>
</tbody>
</table>
**If Statement**

if (expression)
    statement;

- The expression is evaluated, and the statement is executed only if the value is true.

- Must be boolean
- Usually a compound statement.

**If-Else Statement**

if (expression)
    statement1;
else
    statement2;

- The expression is evaluated, and statement1 is executed if the value is true. Otherwise, statement2 is executed.

- Must be boolean
- statement1 and statement2 are usually compound statements.
Common Errors - Missing Brackets

```java
int n = 0;

// if n is too big, print a warning
// and decrease the value of n by 50
if (n > 100)
    System.out.println("n is too big.");
    n = n - 50;
System.out.println(n);
```

- When n is less than or equal to 100, the warning is not printed, but the value is still decreased by 50.

Common Errors - Extra Semicolon

```java
int n = 0;
if (n > 100);
    System.out.println("n is too big");
```

- The warning is always printed!
- A semicolon by itself is a valid statement called a null statement.
Common Errors - Dead Code

public char finalGrade(double average) {
    if (average >= 60)
        return 'D';
    if (average >= 70)
        return 'C';
    if (average >= 80)
        return 'B';
    if (average >= 90)
        return 'A';
    else
        return 'F';
}

Dead Code Fixed (1)

public char finalGrade(double average) {
    if (average >= 60 && average < 70)
        return 'D';
    if (average >= 70 && average < 80)
        return 'C';
    if (average >= 80 && average < 90)
        return 'B';
    if (average >= 90)
        return 'A';
    else
        return 'F';
}
Dead Code Fixed (2)

```java
public char finalGrade(double average) {
    if (average >= 90)
        return 'A';
    if (average >= 80)
        return 'B';
    if (average >= 70)
        return 'C';
    if (average >= 60)
        return 'D';
    else
        return 'F';
}
```

Common Errors -
All return statements are conditional

```java
public char finalGrade(double average) {
    if (average < 60)
        return 'F';
    if (average < 70)
        return 'D';
    if (average < 80)
        return 'C';
    if (average < 90)
        return 'B';
    if (average <= 100)
        return 'A';
}
```
Fixed Problem with
All return statements are conditional

```java
public char finalGrade(double average) {
    if (average < 60)
        return 'F';
    if (average < 70)
        return 'D';
    if (average < 80)
        return 'C';
    if (average < 90)
        return 'B';
    else
        return 'A';
}
```

Using Nested If-Else Statements

```java
public char finalGrade(double average) {
    if (average < 60)
        return 'F';
    else
        if (average < 70)
            return 'D';
        else
            if (average < 80)
                return 'C';
            else
                if (average < 90)
                    return 'B';
                else
                    return 'A';
    }
```
Formatting Nested If-Else Statements

```java
public char finalGrade(double average) {
    if (average < 60)
        return 'F';
    else if (average < 70)
        return 'D';
    else if (average < 80)
        return 'C';
    else if (average < 90)
        return 'B';
    else
        return 'A';
}
```

- Only the formatting has changed.
- There is no such thing as an else-if statement!

Using Brackets is Less Error Prone

```java
public char finalGrade(double average) {
    if (average < 60) {
        return 'F';
    }
    else if (average < 70) {
        return 'D';
    }
    else if (average < 80) {
        return 'C';
    }
    else if (average < 90) {
        return 'B';
    }
    else {
        return 'A';
    }
}
```
Which is the if-else statement?

```
if (p) {
  if (q) {
    s1;
  } else {
    s2;
  }
}
≡

if (p) {
  if (q) {
    s1;
  } else {
    s2;
  }
}
```

Which is the correct truth table for this code?

<table>
<thead>
<tr>
<th>p</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>s2</td>
<td></td>
</tr>
</tbody>
</table>

```
OR
```

```
if (p) {
  if (q) {
    s1;
  } else {
    s2;
  }
}
```

Which is the if-else statement?

```
if (p) {
  if (q) {
    s1;
  } else {
    s2;
  }
}
```

Format Doesn’t Change Meaning

- Formatting with proper indentation makes the code more readable but doesn’t change the meaning.

Use brackets to change the meaning or make the code more clear:

```
if (p) {
  if (q) {
    s1;
  }
} else {
  s2;
}
```

```
if (p) {
  if (q) {
    s1;
  }
} else {
  s2;
}
```

```
if (p) {
  if (q) {
    s1;
  }
} else {
  s2;
}
```

```
if (p) {
  if (q) {
    s1;
  }
} else {
  s2;
}
```
Switch Statement

public void printMonthName(int n) {
    switch(n) {
        case 1:
            System.out.println("January");
            break;
        case 2:
            System.out.println("February");
            break;
        case 12:
            System.out.println("December");
            break;
        default:
            System.out.println("Invalid Month");
            break;
    }
}

Jump to case with matching value. Continue executing statements in the switch block, until encountering a break or return statement.

Common Error: Omit break statements

public void printMonthName(int n) {
    switch(n) {
        case 1:
            System.out.println("January");
        case 2:
            System.out.println("February");
        case 3:
            System.out.println("March");
        case 4:
            System.out.println("April");
            ...
        case 12:
            System.out.println("December");
        default:
            System.out.println("Invalid Month");
    }
}

What will print when n is 2?