Introduction to Computers and Programming

Lecture 6

Chris Conway
New York University
Road Map

• if / else continued
  – Nested if / else statements

• Logical operators
  – &&, ||, ^
  – !
  – &,

• Reading:
  – Liang: chapter 3: 3.3 - 3.5
What is the output of the following code fragment?

```java
int a = 100;
if (a = 100)
System.out.println("a is equal to " + a);
```

What is the output of the following code fragment?

```java
int a = 100, b = 1;
if (a < b)
System.out.println("a is less than b");
```
review

• What is the output of the following code fragment?
int a = 100;
if (a != 100);
System.out.println (" a is equal to " + a);

• What is the output of the following code fragment?
int a = 100, b = 1;
if (a < b)
System.out.println ("a is less than b");
System.out.println ("Thank you");
Java Provides a shortcut if/else operator:

- This is Java’s only ternary operator (i.e. it takes three operands)

```java
System.out.println ( (sales > 100) ? "Federal Express" : "US Mail");
```
nested if statements

• When one if/else structure is contained inside another if/else structure it is called a nested if/else.

```java
if (grade > 60)
{
    if (grade > 70)
        System.out.println("You passed");
    else
        System.out.println("You passed but need a tutor");
} else
    System.out.println("You failed");
```
else if

• Usually you try to nest within the else statement. Note the indentation.

```java
if (grade > 70)
    System.out.println("You passed");
else if (grade > 60)
    System.out.println("You passed but need a tutor");
else
    System.out.println("You failed");
```
Choosing the flavor of your \texttt{if}

• When you have multiple possible branches, you can have any of the following situations:
  1. You want to execute exactly one of the branches
  2. You want to execute zero or one of the branches
  3. You want to execute zero or more branches

• Which would be appropriate for each of the situations above:
  a) A series of \texttt{ifs}
  b) A series of \texttt{if / else ifs ending with an else if}
  c) A series of \texttt{if / else ifs ending with an else}

\begin{tabular}{|c|}
\hline
1 – c \\
2 – b \\
3 – a \\
\hline
\end{tabular}
Using boolean variables as flags

• You may want to use boolean variables to hold the value of a boolean expression.

• For example:

```java
boolean passed = (grade >= 65)
```

• Then you can use the variable later in a conditional statement:

```java
if (passed)
    System.out.println("You passed");
else
    System.out.println("You failed");
```
&& - Logical AND

```
((boolean exp a) && (boolean exp b))
```

- When using the and operator (&&), both expression a and b must be true for the compound statement to be true.

**truth table**

<table>
<thead>
<tr>
<th>Logical AND</th>
<th>false</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

- For example:

```
((total > 50) && (status == 0))
```
|| - Logical OR

((boolean exp a) || (boolean exp b))

- When using the or operator (||), at least one expression a or b must be true for the compound statement to be true.

truth table

<table>
<thead>
<tr>
<th>Logical OR</th>
<th>false</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

- For example:

  ((total > 50) || (status == 0))
^ - Exclusive OR

\(((\text{boolean exp a}) \, ^\, (\text{boolean exp b}))\)

• When using the exclusive or operator (\(^\)), at least one expression a or b must have opposite Boolean values for the compound statement to be true.

truth table

<table>
<thead>
<tr>
<th>Exclusive OR</th>
<th>false</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

• For example:

\(((\text{person1} == 1) \, ^\, (\text{person2} == 1))\)
logical negation !

! (a)

• Reverses the truth or falsity of expression a Boolean expression
• ! has high precedence so you must use parenthesis
• You can avoid using the logical negation by expressing the condition differently with an appropriate relational operator. However, in the case of complex expressions, it is sometimes easier to use negation.
• Note: it is a unary operator
Unconditional vs. Conditional Boolean Operators

• Java provides us with a second “and” operator and a second “or” operator.

• The unconditional operators guarantee that both expressions will be evaluated.

• In this class, you should just use the conditional operators.
Style Considerations
Initialization (revisited)

• When you declare a primitive variable, you do not know the value stored in that variable until you place something there. The language specification does not guarantee any initial value.

• Until the user initializes the value, the value stored in that memory is called a garbage value.

• Java will not allow you to use the garbage value in a calculation or for output. If it is possible that a variable has not been initialized when you try to use it, you will get a compilation error.

• So you must initialize the variable before you use it on the right hand side of a calculation or output it to the screen.
Principle of Proximity

• Advanced tip for initialization
• Initialize your variables close to where you use them if possible
  – Avoid mistakes about value when using late in code (something modified it?)
Line Continuations

• Pick good spots to break up a long line
  – Break so it’s obvious from reading a line there should be more
    • E.g.

```java
if ( (totalSaleBeforeTax > 100) &&
    (isPreferredCustomer) )
System.out.print("\n");
```

  – Note, the dangling && should alert reader to a break
Braces

• Whenever adding a left brace to denote a block or method start, add the right brace right away and then fill between them

• Put a comment after the right brace!

```java
public static void main(String[] args) {
    ... 
    if ( MyGrade < YourGrade ) {
    } /* end if (MyGrade < YourGrade) */
    else {
        } /* end else of (MyGrade < YourGrade) */
    } /* end main */
}
```

• Use curly braces even if you only have one statement as the body of a control structure.
indentation

• Place the brace associated with a control statement (or method / class header) on the next line, indented to the same level as the control statement.

• Statements within the braces are indented to the next level.

• For example:

```java
public class Foo {
    public static void main(String[] args) {
        /* statements go here */
        ...
    }
}
```