Chapter 3: Selections and Conditionals

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Chris Wilcox

Motivations

If you assigned a negative value for radius in Listing 2.2, ComputeAreaWithConsoleInput.java, the program would print an invalid result. If the radius is negative, you don’t want the program to compute the area. How can you deal with this situation?

Objectives

- To declare boolean variables and write Boolean expressions using relational operators (§3.2).
- To implement selection control using one-way if statements (§3.3).
- To implement selection control using two-way if-else statements (§3.4).
- To implement selection control using nested if and multi-way if statements (§3.5).
- To avoid common errors and pitfalls in if statements (§3.6).
- To generate random numbers using the Math.random() method (§3.7).
- To program using selection statements for a variety of examples (SubtractionQuiz, BMI, ComputeTax) (§§3.7–3.9).
- To combine conditions using logical operators (&&, ||, and !) (§3.10).
- To program using selection statements with combined conditions (LeapYear, Lottery) (§3.11–3.12).
- To implement selection control using switch statements (§3.13).
- To write expressions using the conditional expression (§3.14).
- To examine the rules governing operator precedence and associativity (§3.15).
- To apply common techniques to debug errors (§3.16).

The boolean Type and Operators

Often in a program you need to compare two values, such as whether i is greater than j. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

boolean b = (1 > 2);
Relational Operators

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
<th>Example (radius is 5)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less than</td>
<td>radius &lt; 0</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>less than or equal to</td>
<td>radius &lt;= 0</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater than</td>
<td>radius &gt; 0</td>
<td>true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>greater than or equal to</td>
<td>radius &gt;= 0</td>
<td>true</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>equal to</td>
<td>radius == 0</td>
<td>false</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>not equal to</td>
<td>radius != 0</td>
<td>true</td>
</tr>
</tbody>
</table>

Problem: A Simple Math Learning Tool

This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

IMPORTANT NOTE: If you cannot run the buttons, see www.cs.armstrong.edu/liang/javaslidenote.doc.

One-way if Statements

if (boolean-expression) {
    statement(s);
}

Note

(a) Equivalent
(b) Correct
Simple if Demo

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

```
if (radius >= 0) {
    area = radius * radius * 3.14159;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
} else {
    System.out.println("Negative input");
}
```

The Two-way if Statement

```
if (boolean-expression) {
    statement(s) for the true case;
} else {
    statement(s) for the false case;
}
```

Multiple Alternative if Statements

```
if (score >= 90.0) {
    System.out.print("A");
} else if (score >= 80.0) {
    System.out.print("B");
} else if (score >= 70.0) {
    System.out.print("C");
} else if (score >= 60.0) {
    System.out.print("D");
} else {
    System.out.print("F");
}
```

```
if (score >= 90.0) {
    System.out.print("A");
} else if (score >= 80.0) {
    System.out.print("B");
} else if (score >= 70.0) {
    System.out.print("C");
} else if (score >= 60.0) {
    System.out.print("D");
} else {
    System.out.print("F");
}
```
Multi-Way if-else Statements

if (score >= 90.0)
    System.out.print("A");
else if (score >= 80.0)
    System.out.print("B");
else if (score >= 70.0)
    System.out.print("C");
else if (score >= 60.0)
    System.out.print("D");
else
    System.out.print("F");

Suppose score is 70.0
The condition is false

Suppose score is 70.0
The condition is true

if (score >= 90.0)
    System.out.print("A");
else if (score >= 80.0)
    System.out.print("B");
else if (score >= 70.0)
    System.out.print("C");
else if (score >= 60.0)
    System.out.print("D");
else
    System.out.print("F");
Trace if-else statement

Suppose score is 70.0

if (score >= 90.0)
    System.out.print("A");
else if (score >= 80.0)
    System.out.print("B");
else if (score >= 70.0)
    System.out.print("C");
else if (score >= 60.0)
    System.out.print("D");
else
    System.out.print("F");

Suppose score is 70.0
Exit the if statement

Note

The else clause matches the most recent if clause in the same block.

(a) Int i = 1, j = 2, k = 3;
if (i > j)
    System.out.println("A");
else
    System.out.println("B");

(b) Int i = 1, j = 2, k = 3;
Equivalent
This is better with correct indentation
if (i > j){
    System.out.println("A");
} else
    System.out.println("B");

Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

int i = 1;
int j = 2;
int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
} else
    System.out.println("B");
This statement prints B.
Common Errors
Adding a semicolon at the end of an if clause is a common mistake.

```java
if (radius >= 0); \[\text{Wrong}\]
{
    area = radius*radius*PI;
    System.out.println(
        "The area for the circle of radius " +
        radius + " is " + area);
}
```

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error. This error often occurs when you use the next-line block style.

TIP

```java
if (number % 2 == 0)
    even = true;
else
    even = false;
```

(a) Equivalent

```java
boolean even = number % 2 == 0;
```

(b)

CAUTION

```java
if (even) \[\text{Wrong}\]
    System.out.println(
        "It is even.");
```

(a)

Equivalent

```java
if (even) \[\text{Correct}\]
    System.out.println(
        "It is even.");
```

(b)

Problem: An Improved Math Learning Tool
This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers \texttt{number1} and \texttt{number2} with \texttt{number1} \(\geq\) \texttt{number2} and displays a question such as “What is 9 – 2?” to the student. After the student types the answer, the program displays whether the answer is correct.
Problem: Body Mass Index

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

<table>
<thead>
<tr>
<th>BMI</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMI &lt; 18.5</td>
<td>Underweight</td>
</tr>
<tr>
<td>18.5 &lt;= BMI &lt; 25.0</td>
<td>Normal</td>
</tr>
<tr>
<td>25.0 &lt;= BMI &lt; 30.0</td>
<td>Overweight</td>
</tr>
<tr>
<td>BMI &gt;= 30.0</td>
<td>Obese</td>
</tr>
</tbody>
</table>

Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

<table>
<thead>
<tr>
<th>Income Range</th>
<th>Single Tax Rate</th>
<th>Married Filing Jointly Tax Rate</th>
<th>Married Filing Separately Tax Rate</th>
<th>Head of Household Tax Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 - $8,250</td>
<td>10%</td>
<td>$0 - $10,650</td>
<td>$0 - $8,250</td>
<td>$0 - $11,350</td>
</tr>
<tr>
<td>$8,251 - $13,050</td>
<td>15%</td>
<td>$10,651 - $13,050</td>
<td>$8,251 - $13,050</td>
<td>$11,351 - $14,250</td>
</tr>
<tr>
<td>$13,051 - $25,050</td>
<td>25%</td>
<td>$13,051 - $25,050</td>
<td>$13,051 - $25,050</td>
<td>$14,251 - $17,450</td>
</tr>
<tr>
<td>$25,051 - $50,000</td>
<td>28%</td>
<td>$25,051 - $50,000</td>
<td>$25,051 - $50,000</td>
<td>$17,451 - $30,200</td>
</tr>
<tr>
<td>$50,001 - $100,000</td>
<td>33%</td>
<td>$50,001 - $100,000</td>
<td>$50,001 - $100,000</td>
<td>$30,201 - $75,950</td>
</tr>
<tr>
<td>$100,001 +</td>
<td>35%</td>
<td>$100,001 +</td>
<td>$100,001 +</td>
<td>$75,951 +</td>
</tr>
</tbody>
</table>

```java
if (status == 0) {
    // Compute tax for single filers
} else if (status == 1) {
    // Compute tax for married file jointly
    // or qualifying widow(er)
} else if (status == 2) {
    // Compute tax for married file separately
} else if (status == 3) {
    // Compute tax for head of household
} else {
    // Display wrong status
}
```

Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>logical disjunction</td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>
### Truth Table for Operator `!`

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>!(age &gt; 18) is false, because (age &gt; 18) is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>!(weight == 150) is true, because (weight == 150) is false.</td>
</tr>
</tbody>
</table>

### Truth Table for Operator `&&`

<table>
<thead>
<tr>
<th>p₁</th>
<th>p₂</th>
<th>p₁ &amp;&amp; p₂</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &lt;= 18) &amp;&amp; (weight &lt; 140) is false, because both conditions are both false.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 18) &amp;&amp; (weight &gt;= 140) is true, because both (age &gt; 18) and (weight &gt;= 140) are true.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (weight &gt; 140) is false, because (weight &gt; 140) is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 18) &amp;&amp; (weight == 140) is true, because both (age &gt; 18) and (weight == 140) are true.</td>
</tr>
</tbody>
</table>

### Truth Table for Operator `||`

| p₁ | p₂ | p₁ || p₂ | Example (assume age = 24, weight = 140) |
|----|----|------|-----|------------------------------------------|
| false | false | false | false | (age > 34) || (weight <= 140) is true, because (age > 34) is false and (weight <= 140) is true. |
| false | true | true  | true  | (age > 14) || (weight >= 150) is false, because (age > 14) is true. |
| true | false | false | (age > 34) || (weight > 140) is true, because (age > 34) is false and (weight > 140) is false. |
| true | true | true  | true  | (age > 14) || (weight >= 150) is false, because (age > 14) is true and (weight >= 150) is true. |

### Truth Table for Operator `^`

<table>
<thead>
<tr>
<th>p₁</th>
<th>p₂</th>
<th>p₁ ^ p₂</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) ^ (weight &gt; 140) is true, because (age &gt; 34) is false and (weight &gt; 140) is false.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 14) ^ (weight &lt;= 140) is true, because (age &gt; 14) is true and (weight &lt;= 140) is true.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) ^ (weight &gt;= 150) is true, because (age &gt; 34) is false and (weight &gt;= 150) is true.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 14) ^ (weight &gt; 140) is true, because (age &gt; 14) is true and (weight &gt; 140) is false.</td>
</tr>
</tbody>
</table>
Examples

Here is a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:

```java
System.out.println("Is " + number + " divisible by 2 and 3? " + ((number % 2 == 0) && (number % 3 == 0)));
System.out.println("Is " + number + " divisible by 2 or 3? " + ((number % 2 == 0) || (number % 3 == 0)));
System.out.println("Is " + number + " divisible by 2 or 3, but not both? " + ((number % 2 == 0) ^ (number % 3 == 0)));
```

The & and | Operators

If x is 1, what is x after this expression?

```
(x > 1) & (x++ < 10)
```

If x is 1, what is x after this expression?

```
(1 > x) && (1 > x++)
```

How about (1 == x) | (10 > x++)?

```
(1 == x) || (10 > x++)
```
Problem: Determining Leap Year?

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

\[(\text{year} \mod 4 == 0 \text{ && } \text{year} \mod 100 != 0) \text{ || } (\text{year} \mod 400 == 0)\]

Problem: Lottery

Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- If the user input matches the lottery in exact order, the award is $10,000.
- If the user input matches the lottery, the award is $3,000.
- If one digit in the user input matches a digit in the lottery, the award is $1,000.

switch Statements

```
switch (status) {
    case 0: compute taxes for single filers;
            break;
    case 1: compute taxes for married file jointly;
            break;
    case 2: compute taxes for married file separately;
            break;
    case 3: compute taxes for head of household;
            break;
    default: System.out.println("Errors: invalid status");
             System.exit(1);
}
```

switch Statement Flow Chart
**switch Statement Rules**

The switch-expression must yield a value of `char`, `byte`, `short`, or `int` type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression.

The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as `1 + x`.

The keyword `break` is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the `break` statement is not present, the next case statement will be executed.

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a `break` statement or the end of the switch statement is reached.

---

**Trace switch statement**

Suppose day is 2:

```java
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}
```
Trace switch statement

```java
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}
```

Fall through case 3

Fall through case 4

Fall through case 5

Encounter break
Trace switch statement

switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
        System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}

Problem: Chinese Zodiac
Write a program that prompts the user to enter a year and displays the animal for the year.

Conditional Expressions
if (x > 0)
    y = 1
else
    y = -1;

is equivalent to
y = (x > 0) ? 1 : -1;

Ternary operator
Binary operator
Unary operator

Conditional Operator
if (num % 2 == 0)
    System.out.println(num + "is even");
else
    System.out.println(num + "is odd");

System.out.println(
    (num % 2 == 0) ? num + "is even":
    num + "is odd");
Conditional Operator, cont.

\[
\text{boolean-expression} \ ? \ \text{exp1} : \ \text{exp2}
\]

Operator Precedence

\[
\begin{align*}
\ast, /, \% \quad &\text{(Multiplication, division, and remainder)} \\
+, - \quad &\text{(Unary plus and minus)} \\
+, =, ++, -- \quad &\text{(Assignment operator)} \\
\text{(type)} \quad &\text{Casting} \\
! \quad &\text{(Not)} \\
\&\& \quad &\text{(Conditional AND)} \\
\text{(Exclusive OR)} \quad &\text{Short-circuit AND} \\
\text{||} \quad &\text{(Conditional OR)} \\
\Rightarrow, \Rightarrow= \quad &\text{Short-circuit OR} \\
\text{IsValid} \quad &\text{Assignment operator}
\end{align*}
\]

Operator Precedence and Associativity

The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is evaluated first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

Operator Associativity

When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation. All binary operators except assignment operators are left-associative.

\[
a - b + c - d \quad \text{is equivalent to} \quad ((a - b) + c) - d
\]

Assignment operators are right-associative. Therefore, the expression

\[
a = b += c = 5 \quad \text{is equivalent to} \quad a = (b += (c = 5))
\]
Example
Applying the operator precedence and associativity rule, the expression \(3 + 4 * 4 > 5 * (4 + 3) - 1\) is evaluated as follows:

\[
\begin{align*}
3 + 4 * 4 & \text{ (inside parentheses first)} \\
3 + 4 & \text{ (2) multiplication} \\
17 & \text{ (3) multiplication} \\
17 & \text{ (4) addition} \\
19 & \text{ (5) subtraction} \\
false & \text{ (6) greater than}
\end{align*}
\]

Operand Evaluation Order
Supplement III.A, “Advanced discussions on how an expression is evaluated in the JVM.”

Debugging
Logic errors are called bugs. The process of finding and correcting errors is called debugging. A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located. You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.

Debugger
Debugger is a program that facilitates debugging. You can use a debugger to:
- Execute a single statement at a time.
- Trace into or stepping over a method.
- Set breakpoints.
- Display variables.
- Display call stack.
- Modify variables.
Debugging in Eclipse

Supplement II.G, Learning Java Effectively with Eclipse