Chapter 3: Selections and Conditionals

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified slides by Kristina Brown
printf

printf - can format the number of decimal places that will be shown when printed to the console:

```java
System.out.printf("%5.2f", 5.12345);
double m = 6.23456;
System.out.printf("%.2f", m);
```
Scanner

A couple more methods using Scanner to get text from the keyboard:

- `next()` - reads a token
- `nextLine()` - reads a line of text
Motivations

If it rains?

If you are wearing red?

If your eyes are blue?

If you rode your bike to school?
Conditionals/Selections

Allow you to make decisions using your programs.

```
if (condition)
  statement;
```
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.

```java
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>&lt;=</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>&gt;=</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>
One-way if Statements

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

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area + " for the circle of radius " + radius + " is " + area);
}
Note

```java
if i > 0 {
    System.out.println("i is positive");
}
```

(a) Wrong

```java
if (i > 0) {   
    System.out.println("i is positive");
}
```

(b) Correct

```java
if (i > 0) {   
    System.out.println("i is positive");
}
```

(a) Equivalent

```java
if (i > 0) {
    System.out.println("i is positive");
}
```

(b)
The Two-way if Statement

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

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");
}
Multiple Alternative if Statements

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

Equivalent

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

This is better
Multi-Way if-else Statements

- If `score >= 90`, then grade is A.
- If `score >= 80`, then grade is B.
- If `score >= 70`, then grade is C.
- If `score >= 60`, then grade is D.
- If `score < 60`, then grade is 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");

The condition is false
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");

The condition is false
Trace if-else statement

Suppose score is 70.0

```java
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");
```

The condition is true
Suppose score is 70.0

grade is C

```java
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");
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:

```java
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);
{
    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.
Note

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

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

(a)

Equivalent

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

(b)
TIP

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

Equivalent

boolean even
   = number % 2 == 0;
CAUTION

if (even == true)
    System.out.println("It is even.");

(a)

Equivalent

if (even)
    System.out.println("It is even.");

(b)
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>Marginal Tax Rate</th>
<th>Single</th>
<th>Married Filing Jointly or Qualifying Widow(er)</th>
<th>Married Filing Separately</th>
<th>Head of Household</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0 – $8,350</td>
<td>$0 – $16,700</td>
<td>$0 – $8,350</td>
<td>$0 – $11,950</td>
</tr>
<tr>
<td>15%</td>
<td>$8,351 – $33,950</td>
<td>$16,701 – $67,900</td>
<td>$8,351 – $33,950</td>
<td>$11,951 – $45,500</td>
</tr>
<tr>
<td>35%</td>
<td>$372,951+</td>
<td>$372,951+</td>
<td>$186,476+</td>
<td>$372,951+</td>
</tr>
</tbody>
</table>
Problem: Computing Taxes, cont.

```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
}
```
Lecture 2
# 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></td>
<td></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>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_1 \ &amp;&amp; \ p_2 )</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>false</td>
<td></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 &gt;= 140) is true, because both (age &gt; 18) and (weight &gt;= 140) are true.</td>
</tr>
</tbody>
</table>
# Truth Table for Operator ||

| $p_1$ | $p_2$ | $p_1 \ || \ p_2$ | Example (assume age = 24, weight = 140) |
|-------|-------|------------------|-----------------------------------------|
| false | false | false            | (age > 34) || (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true. |
| false | true  | true             | (age > 14) || (weight >= 150) is false, because (age > 14) is true. |
| true  | false | true             |                                          |
| true  | true  | true             |                                          |
Truth Table for Operator $^\wedge$

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_1 \wedge p_2$</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) $^\wedge$ (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; 34) $^\wedge$ (weight $\geq$ 140) is true, because (age &gt; 34) is false but (weight $\geq$ 140) is true.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 14) $^\wedge$ (weight &gt; 140) is true, because (age &gt; 14) is true and (weight &gt; 140) is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
Examples

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)));

TestBooleanOperators
Run
The & and | Operators

Supplement III.B, “The & and | Operators”
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++)?
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

- status is 0: Compute tax for single filers, break
- status is 1: Compute tax for married jointly or qualifying widow(er), break
- status is 2: Compute tax for married filing separately, break
- status is 3: Compute tax for head of household, break
- default: Default actions
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.

```java
switch (switch-expression) {
    case value1: statement(s)1;
        break;
    case value2: statement(s)2;
        break;
    ...
    case valueN: statement(s)N;
        break;
    default: statement(s)-for-default;
}
```
switch Statement Rules

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`.

```java
switch (switch-expression) {
    case value1:  statement(s)1;
                 break;
    case value2:  statement(s)2;
                 break;
    ...
    case valueN:  statement(s)N;
                 break;
    default:      statement(s)-for-default;
}
```

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

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

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:
        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");
}
```
Trace switch statement

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");
}
```
Conditional Expressions

```java
if (x > 0)
    y = 1
else
    y = -1;
```

is equivalent to

```java
y = (x > 0) ? 1 : -1;
```

(boolean-expression) ? expression1 : expression2

- 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.

boolean-expression ? exp1 : exp2
Operator Precedence

- ()
- var++, var--
- +, - (Unary plus and minus), ++var, --var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Relational operators)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)
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 executed 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*.

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

Assignment operators are *right-associative*. Therefore, the expression

\[ a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5)) \]
Example

Applying the operator precedence and associativity rule, the expression $3 + 4 \times 4 > 5 \times (4 + 3) - 1$ is evaluated as follows:

1. $3 + 4 \times 4 > 5 \times (4 + 3) - 1$
2. $3 + 4 \times 4 > 5 \times 7 - 1$
3. $3 + 16 > 5 \times 7 - 1$
4. $3 + 16 > 35 - 1$
5. $19 > 35 - 1$
6. $19 > 34$
7. false

(1) inside parentheses first
(2) multiplication
(3) multiplication
(4) addition
(5) subtraction
(6) greater than
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.
Misc Slides
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.
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 `number1` and `number2` with `number1 >= 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 ≤ BMI &lt; 25.0</td>
<td>Normal</td>
</tr>
<tr>
<td>25.0 ≤ BMI &lt; 30.0</td>
<td>Overweight</td>
</tr>
<tr>
<td>30.0 ≤ BMI</td>
<td>Obese</td>
</tr>
</tbody>
</table>
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.
Problem: Chinese Zodiac
Write a program that prompts the user to enter a year and displays the animal for the year.

\[
\text{year} \mod 12 = \begin{cases} 
0: \text{monkey} \\
1: \text{rooster} \\
2: \text{dog} \\
3: \text{pig} \\
4: \text{rat} \\
5: \text{ox} \\
6: \text{tiger} \\
7: \text{rabbit} \\
8: \text{dragon} \\
9: \text{snake} \\
10: \text{horse} \\
11: \text{sheep} 
\end{cases}
\]
Problem: Determining Leap Year?

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

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

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