

CEN 419

Introduction to Java Programming



Dr. H. Esin ÜNAL

FALL 2021

Slides are modified from original slides of Y. Daniel Liang

Introduction

- ☞ **The program can decide which statements to execute based on a condition.**
 - *Selection statements*: statements that let you choose actions with alternative courses.
 - Selection statements use conditions that are Boolean expressions.
 - A *Boolean expression* is an expression that evaluates to a *Boolean value*: **true** or **false**.

The boolean Data Type and Operators

- ➡ *The **boolean data type** declares a variable with the value either true or false.*
- ➡ In a program you often need to compare two values, such as whether i is greater than j or not.
- ➡ Java provides six **comparison operators** (also known as **relational operators**) that can be used to compare two values.

The boolean Data Type and Operators

☞ The result of the comparison is a Boolean value: true or false.

```
double radius = 1;
```

```
System.out.println(Radius > 0);
```

☞ A variable that holds a Boolean value is known as a ***Boolean variable***.

```
boolean b = true;
```

☞ **true** and **false** are literals, just like a number such as 10. They are treated as reserved words and cannot be used as identifiers in the program.

Relational Operators

Java Operator	Mathematical Symbol	Name	Example (Radius is 5)	Result
<	<	less than	radius < 0	false
<=	?	less than or equal	radius <= 0	false
>	>	greater than	radius > 0	true
>=	?	greater than or equal	radius >= 0	true
==	=	equal to	radius==0	false
!=	?	not equal to	radius!=0	true

Question???

- Can the following conversions involving casting be allowed?

```
int i;  
boolean b = true;  
i = (int)b;
```

NO!!!

incompatible types: boolean
cannot be converted to int

```
int i = 1;  
boolean b = (boolean)i;
```

NO!!!

incompatible types: int cannot
be converted to boolean

Selection Statements

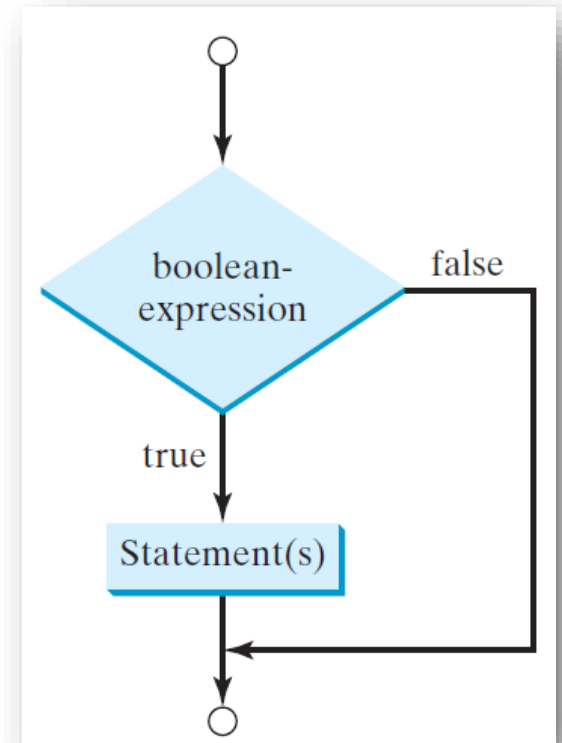
☞ Java has several types of selection statements:

- one-way if statements,
- two-way if-else statements,
- nested if statements,
- multi-way if-else statements,
- switch statements
- conditional expressions.

One-way if Statements

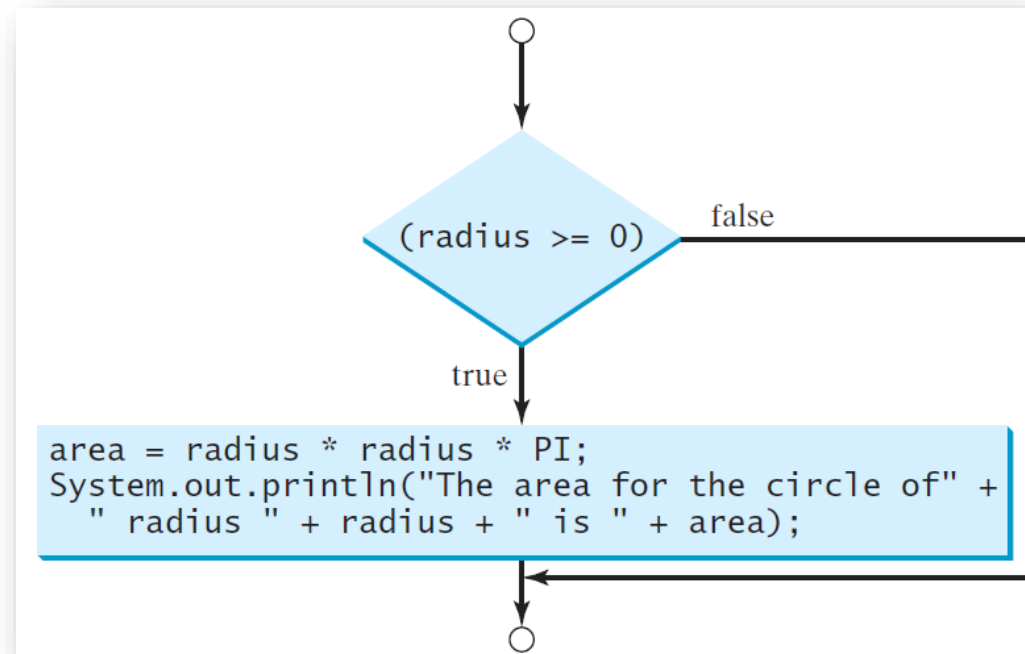
An if statement is a construct that enables a program to specify alternative paths of execution.

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



One-way if Statements

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



Simple If Demo

[Intro to Java Programming, Y. Daniel Liang - SimpleIfDemo.java \(pearsoncmg.com\)](#)

NOTE

The **boolean-expression** is enclosed in parentheses.

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

(a) Wrong

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

(b) Correct

The block braces can be omitted if they enclose a **single statement**.

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

(a)

Equivalent

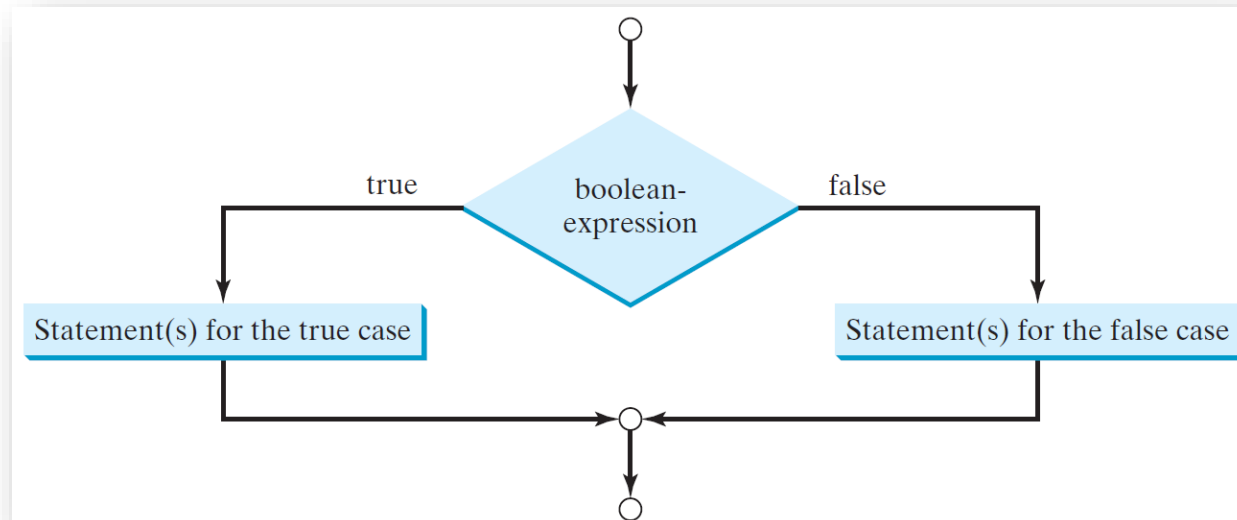
```
if (i > 0)  
    System.out.println("i is positive");
```

(b)

The Two-way if-else Statement

An if-else statement decides the execution path based on whether the condition is true or false.

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



The Two-way if-else Statement

What is the output of the code in (a) and (b) if **number** is **30**? What if **number** is **35**?

```
if (number % 2 == 0)
    System.out.println(number + " is even.");
System.out.println(number + " is odd.");
```

(a)

```
if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");
```

(b)

Nested `if` Statement

- An **if** statement can be inside another **if** statement to form a nested **if** statement.
- ☞ There is no limit to the depth of the nesting.
- ☞ The nested **if** statement can be used to implement multiple alternatives.

Examples

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");
```

(a)

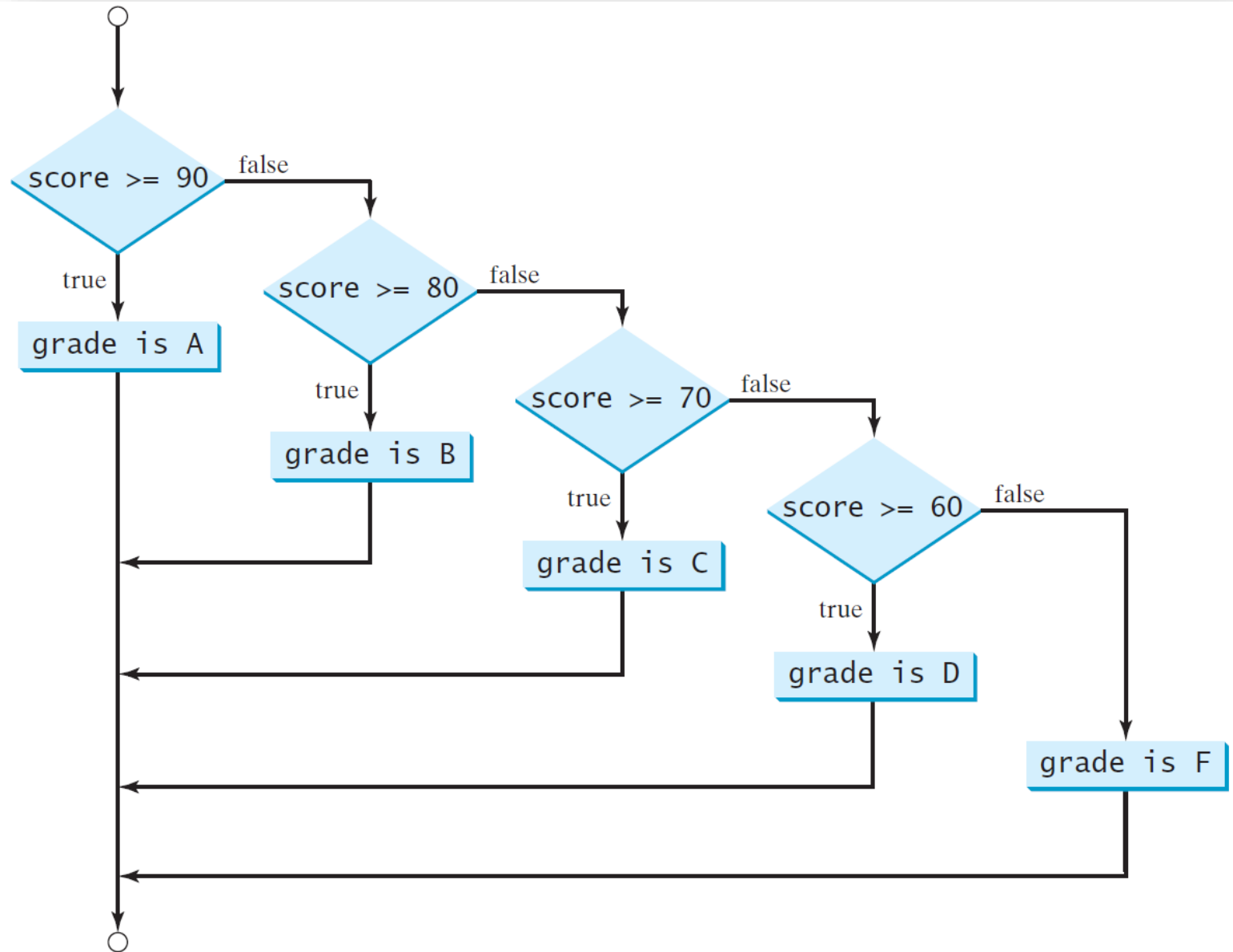
Equivalent

This is better

```
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");
```

(b)

Multi-way if-else Statements



Common Errors and Pitfalls

- *Common Errors in selection statements:*
 - ✓ forgetting necessary braces,
 - ✓ ending an **if** statement in the wrong place,
 - ✓ mistaking **==** for **=**,
 - ✓ dangling **else** clauses
 - ✓ testing equality of double values
- *Common Pitfalls in selection statements:*
 - ✓ duplicated statements in **if-else** statements
 - ✓ simplifying boolean variable assignment

Forgetting Necessary Braces

- ☞ The braces can be omitted if the block contains a single statement.
- ☞ However, forgetting the braces when they are needed for grouping multiple statements is a common programming error.

```
if (radius >= 0)
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
```

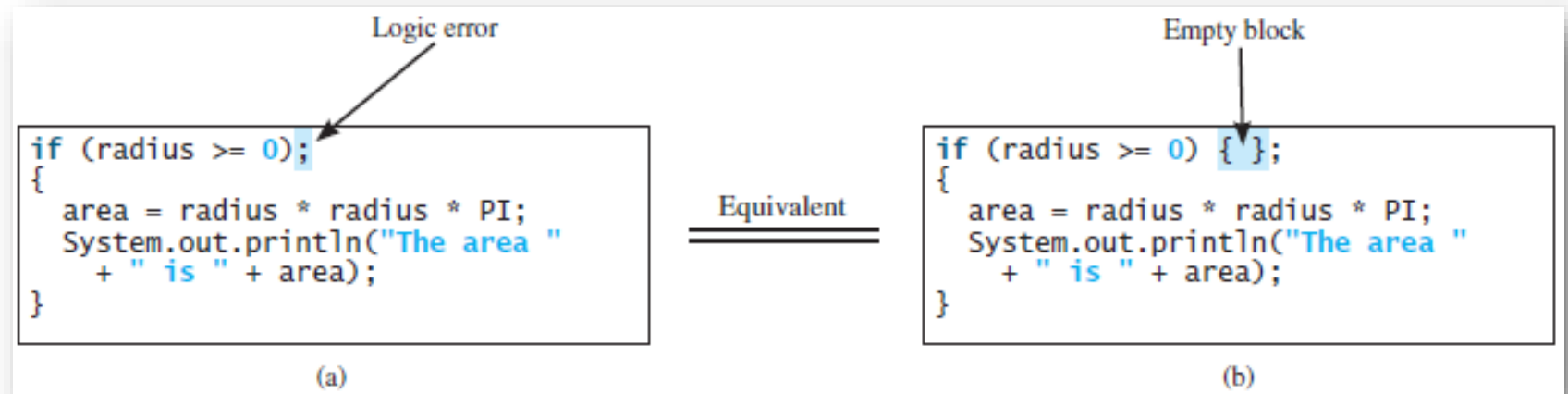
(a) Wrong

```
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area "
        + " is " + area);
}
```

(b) Correct

Wrong Semicolon at the if Line

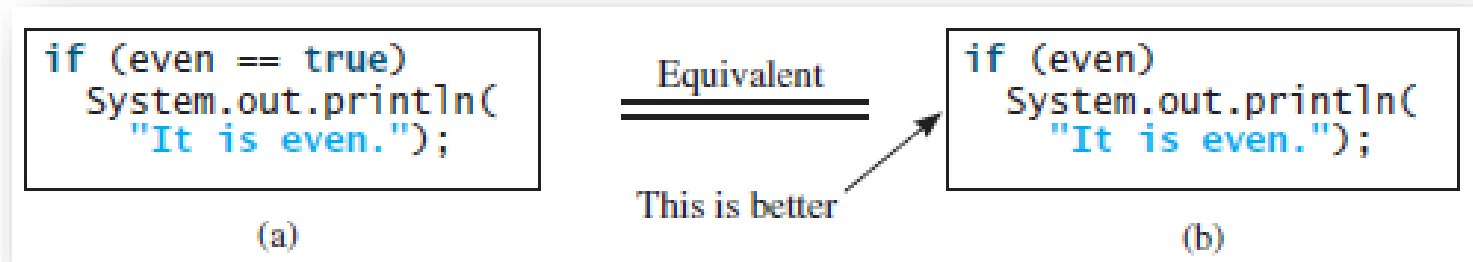
- ➡ Adding a semicolon at the end of an **if** line is a common mistake.
- ➡ This mistake is hard to find, because it is neither a compile error nor a runtime error; it is a logic error.



Using end-of-line block style can help prevent this error

Redundant Testing of Boolean Values

- ➡ To test whether a **boolean** variable is **true** or **false** in a test condition is redundant.



- ➡ Using the `=` operator instead of the `==` operator to compare the equality of two items in a test condition is a common error.

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

- ➡ This statement does not have compile errors. It assigns **true** to **even**, so that **even** is always **true**.

Dangling else Ambiguity

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

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

(a)

Equivalent

This is better
with correct
indentation

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

(b)

Dangling else Ambiguity

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, j = 2, k = 3;

if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
```

This statement prints B.

Equality Test of Two Floating- Point Numbers

- ➡ Floating-point numbers have a limited precision and calculations; involving floating-point numbers can introduce round-off errors.
- ➡ So, equality test of two floating-point values is not reliable.
- ➡ *However, you can compare whether they are close enough by testing whether the difference of the two numbers is less than some threshold.*

```
final double EPSILON = 1E-14;  
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;  
if (Math.abs(x - 0.5) < EPSILON)  
    System.out.println(x + " is approximately 0.5");
```

Simplifying Boolean Variable Assignment

- ☞ The code that assigns a test condition to a boolean variable

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

(a)

Equivalent

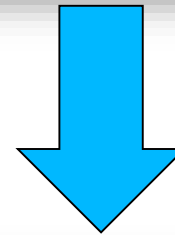
This is shorter

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

(b)

Avoiding Duplicate Code in Different Cases

```
if (inState) {  
    tuition = 5000;  
    System.out.println("The tuition is " + tuition);  
}  
else {  
    tuition = 15000;  
    System.out.println("The tuition is " + tuition);  
}
```



```
if (inState) {  
    tuition = 5000;  
}  
else {  
    tuition = 15000;  
}  
System.out.println("The tuition is " + tuition);
```


The Math.random() Method

- Generates a random double value greater than or equal to 0.0 and less than 1.0 ($0 \leq \text{Math.random()} < 1.0$).

Examples:

<code>(int)(Math.random() * 10)</code>	→	Returns a random integer between 0 and 9.
<code>50 + (int)(Math.random() * 50)</code>	→	Returns a random integer between 50 and 99.

In general,

<code>a + Math.random() * b</code>	→	Returns a random number between a and a + b, excluding a + b.
------------------------------------	---	---

Problem: A Simple Math Learning Tool

The program will work as follows:

1. Generate two numbers between 0 and 9, namely: **number1** and **number2**.
2. If **number1 < number2**, swap **number1** with **number2**.
3. Prompt the student to answer, "**What is number1 – number2?**"
4. Check the student's answer and display whether the answer is correct.

Subtraction Quiz

[Intro to Java Programming, Y. Daniel Liang - SubtractionQuiz.java \(pearsoncmg.com\)](#)

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:

BMI	Interpretation
BMI < 18.5	Underweight
18.5 ≤ BMI < 25.0	Normal
25.0 ≤ BMI < 30.0	Overweight
30.0 ≤ BMI	Obese

Body Mask Index

[Intro to Java Programming, Y. Daniel Liang - ComputeAndInterpretBMI.java \(pearsoncmg.com\)](#)

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.

<i>Marginal Tax Rate</i>	<i>Single</i>	<i>Married Filing Jointly or Qualifying Widow(er)</i>	<i>Married Filing Separately</i>	<i>Head of Household</i>
10%	\$0 – \$8,350	\$0 – \$16,700	\$0 – \$8,350	\$0 – \$11,950
15%	\$8,351 – \$33,950	\$16,701 – \$67,900	\$8,351 – \$33,950	\$11,951 – \$45,500
25%	\$33,951 – \$82,250	\$67,901 – \$137,050	\$33,951 – \$68,525	\$45,501 – \$117,450
28%	\$82,251 – \$171,550	\$137,051 – \$208,850	\$68,526 – \$104,425	\$117,451 – \$190,200
33%	\$171,551 – \$372,950	\$208,851 – \$372,950	\$104,426 – \$186,475	\$190,201 – \$372,950
35%	\$372,951+	\$372,951+	\$186,476+	\$372,951+

Problem: Computing Taxes

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

Computing Taxes

[Intro to Java Programming, Y. Daniel Liang - ComputeTax.java \(pearsoncmg.com\)](#)

Logical Operators

- The logical operators **!**, **&&**, **||**, and **^** can be used to create a compound Boolean expression.

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
 	or	logical disjunction
^	exclusive or	logical exclusion

Logical operators, also known as ***Boolean operators***, operate on Boolean values to create a new Boolean value.

Truth Table for Operator NOT (!)

p	!p	Example (assume age = 24, weight = 140)
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(weight == 150) is true, because (weight == 150) is false.

Truth Table for Operator AND (&&)

p ₁	p ₂	p ₁ && p ₂	Example (assume age = 24, weight = 140)
false	false	false	(age <= 18) && (weight < 140) is false, because (age <= 18) and (weight < 140) are both false.
false	true	false	(age > 28) && (weight <= 140) is false, because (age > 28) is false.
true	false	false	(age > 18) && (weight > 140) is false, because (weight > 140) is false.
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.

Truth Table for Operator OR (||)

p_1	p_2	$p_1 \ \ p_2$	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) (weight >= 150) is false, because both are false.
false	true	true	(age > 34) (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true.
true	false	true	(age > 14) (weight >= 150) is true, because (age > 14) is true.
true	true	true	(age > 14) (weight <= 140) is true, because each one is true.

Truth Table for Operator EXCLUSIVE OR (^)

p_1	p_2	$p_1 \wedge p_2$	Example (assume age = 24, weight = 140)
false	false	false	$(\text{age} > 34) \wedge (\text{weight} > 140)$ is false, because $(\text{age} > 34)$ and $(\text{weight} > 140)$ are both false.
false	true	true	$(\text{age} > 34) \wedge (\text{weight} \geq 140)$ is true, because $(\text{age} > 34)$ is false but $(\text{weight} \geq 140)$ is true.
true	false	true	$(\text{age} > 14) \wedge (\text{weight} > 140)$ is true, because $(\text{age} > 14)$ is true and $(\text{weight} > 140)$ is false.
true	true	false	$(\text{age} > 14) \wedge (\text{weight} \geq 140)$ is false, because $(\text{age} > 14)$ and $(\text{weight} > 140)$ are both true.

Example Test Boolean Operators

Here is a program that checks whether a number is

- ✓ divisible by 2 and 3,
- ✓ divisible by 2 or 3,
- ✓ divisible by 2 or 3 but not both

Test Boolean Operators

[Intro to Java Programming, Y. Daniel Liang - TestBooleanOperators.java \(pearsoncmg.com\)](#)

Examples

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**.

```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

Leap Year

[Intro to Java Programming, Y. Daniel Liang - LeapYear.java \(pearsoncmg.com\)](#)

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.

Lottery

[Intro to Java Programming, Y. Daniel Liang - Lottery.java \(pearsoncmg.com\)](#)

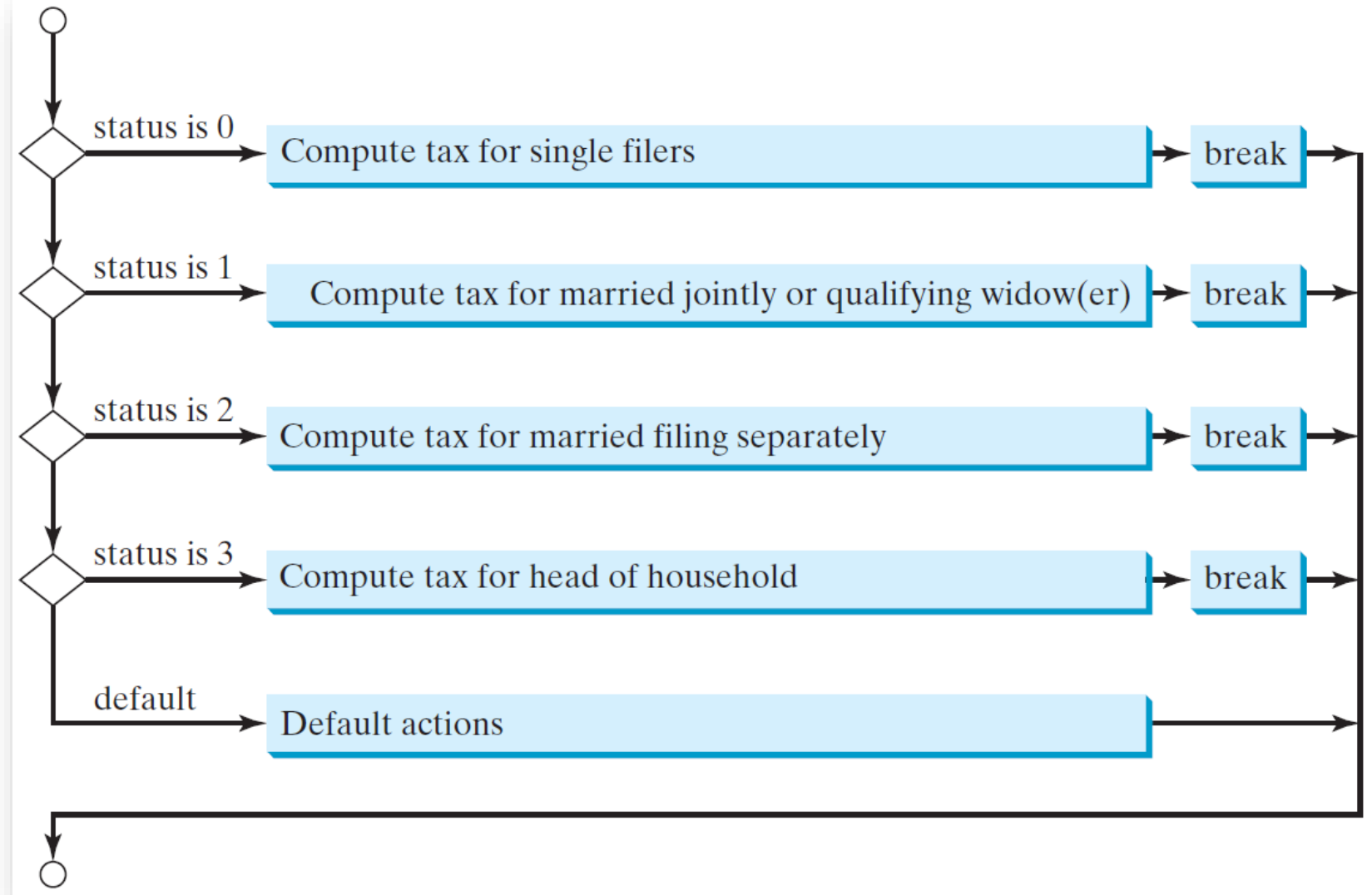
switch Statements

A **switch** statement executes statements based on the value of a variable or an expression.

✓ Java provides a **switch** statement to simplify coding for multiple conditions.

```
switch (status) {  
    case 0: compute tax for single filers;  
            break;  
    case 1: compute tax for married jointly or qualifying widow(er);  
            break;  
    case 2: compute tax for married filing separately;  
            break;  
    case 3: compute tax for head of household;  
            break;  
    default: System.out.println("Error: 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.

```
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;  
}
```

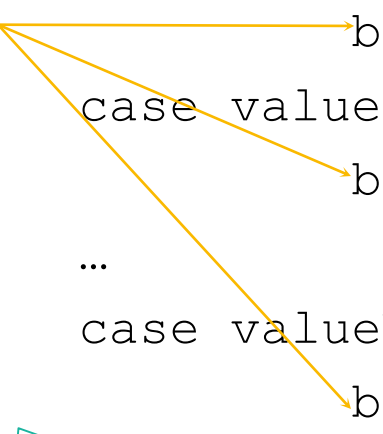
- ✓ The value1, ..., and valueN must have the same data type as the value of the switch-expression.
- ✓ The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.
- ✓ Note that value1, ..., and valueN are constant expressions,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

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.

```
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:


```
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


Match case 2

```
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

```
switch (day) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  Fall through 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");  
}
```



Fall through case 5

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");  
}
```



Encounter break

Trace switch Statement

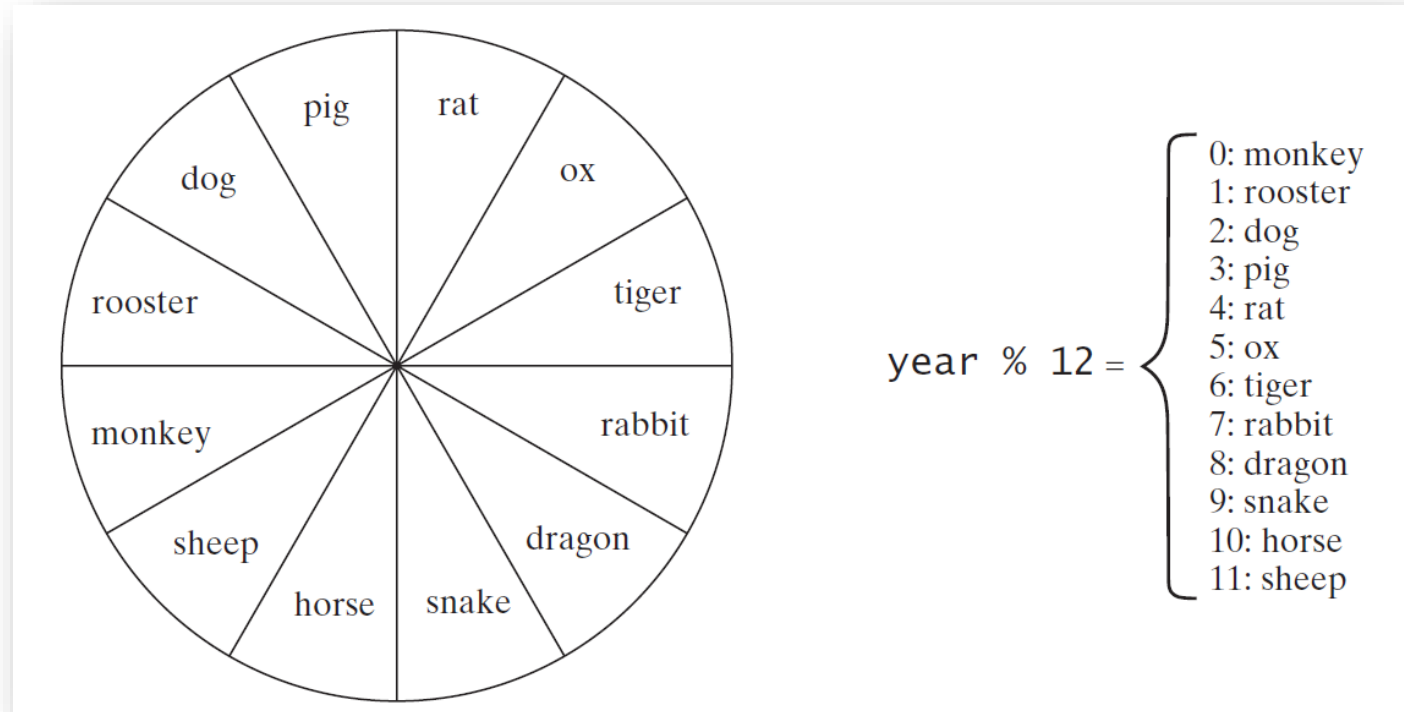
```
switch (day) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
    case 5: System.out.println("Weekday"); break;  
    case 0:  
    case 6: System.out.println("Weekend");  
}
```

Exit the statement

Example

Problem: Chinese Zodiac

Write a program that prompts the user to enter a year and displays the animal for the year.



ChineseZodiac

[Intro to Java Programming, Y. Daniel Liang - ChineseZodiac.java \(pearsoncmg.com\)](#)

Conditional Expressions

*A **conditional expression** evaluates an expression based on a condition.*

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

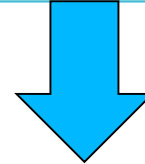
is equivalent to

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

```
(boolean-expression) ? exp1 : exp2
```

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");
```

Example

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.

Operator Precedence

<i>Precedence</i>	<i>Operator</i>
	<code>var++</code> and <code>var--</code> (Postfix)
	<code>+</code> , <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix)
	<code>(type)</code> (Casting)
	<code>!</code> (Not)
	<code>*</code> , <code>/</code> , <code>%</code> (Multiplication, division, and remainder)
	<code>+</code> , <code>-</code> (Binary addition and subtraction)
	<code><</code> , <code><=</code> , <code>></code> , <code>>=</code> (Relational)
	<code>==</code> , <code>!=</code> (Equality)
	<code>^</code> (Exclusive OR)
	<code>&&</code> (AND)
	<code> </code> (OR)
	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> (Assignment operator)

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***.

$a - b + c - d$ is equivalent to $((a - b) + c) - d$

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

$a = b += c = 5$ is equivalent to $a = (b += (c = 5))$

Example

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

$3 + 4 * 4 > 5 * (4 + 3) - 1$

(1) inside parentheses first

$3 + 4 * 4 > 5 * 7 - 1$

(2) multiplication

$3 + 16 > 5 * 7 - 1$

(3) multiplication

$3 + 16 > 35 - 1$

(4) addition

$19 > 35 - 1$

(5) subtraction

$19 > 34$

(6) greater than

false