

Chapter 3 Selections



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) ;
```



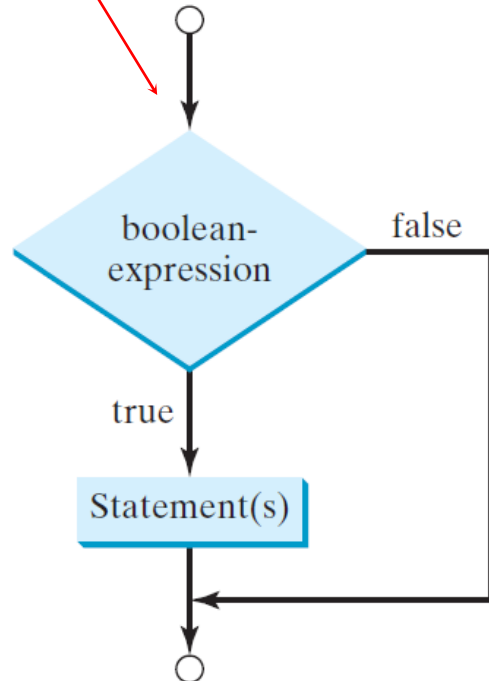
Relational Operators

Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
<	<	less than	<code>radius < 0</code>	<code>false</code>
<=	≤	less than or equal to	<code>radius <= 0</code>	<code>false</code>
>	>	greater than	<code>radius > 0</code>	<code>true</code>
>=	≥	greater than or equal to	<code>radius >= 0</code>	<code>true</code>
==	=	equal to	<code>radius == 0</code>	<code>false</code>
!=	≠	not equal to	<code>radius != 0</code>	<code>true</code>

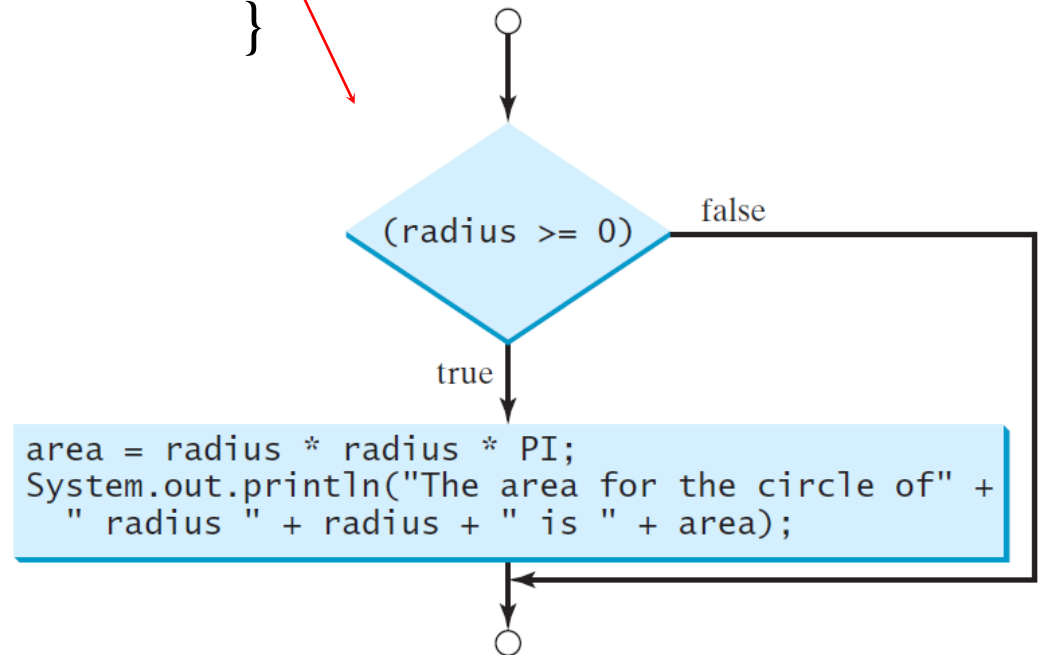


One-way if Statements

```
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

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

(a) Wrong

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

(b) Correct

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

(a)

Equivalent

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

(b)



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.

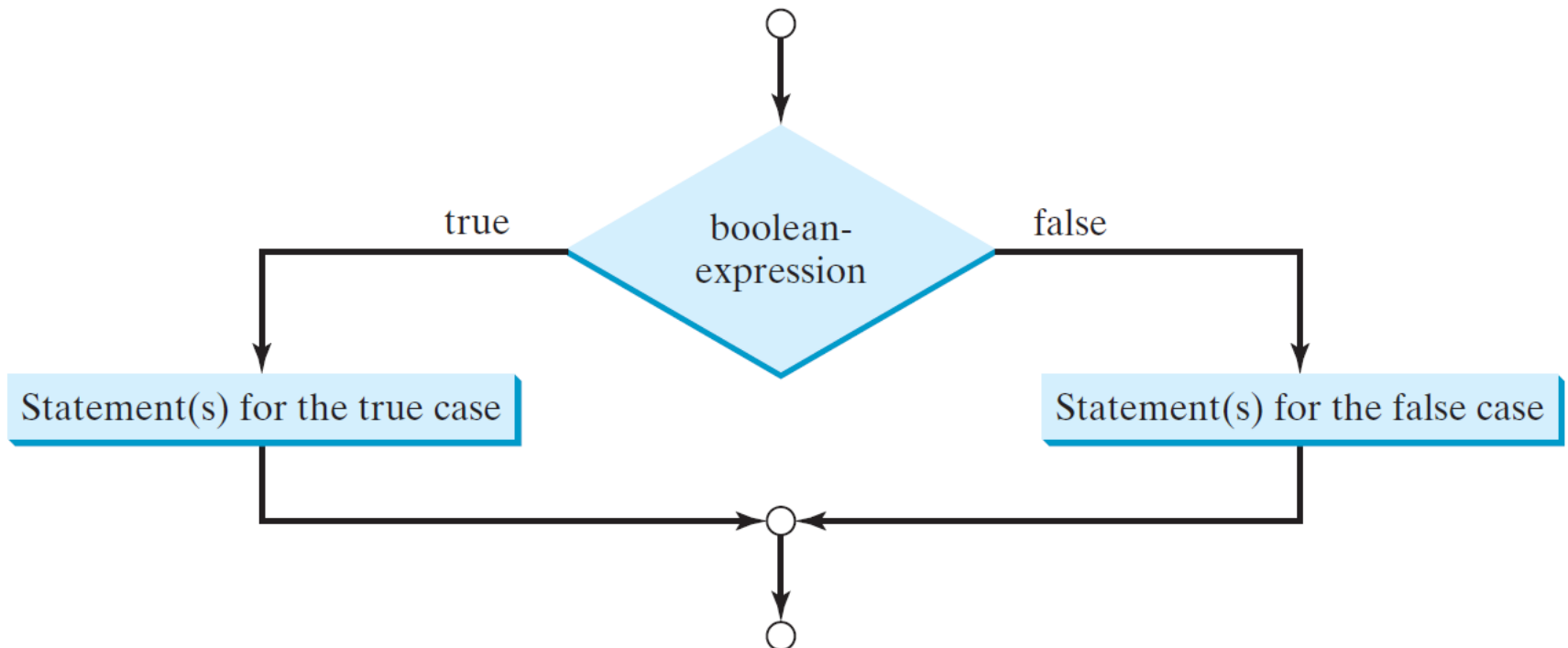
SimpleIfDemo

Run



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 >= 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)

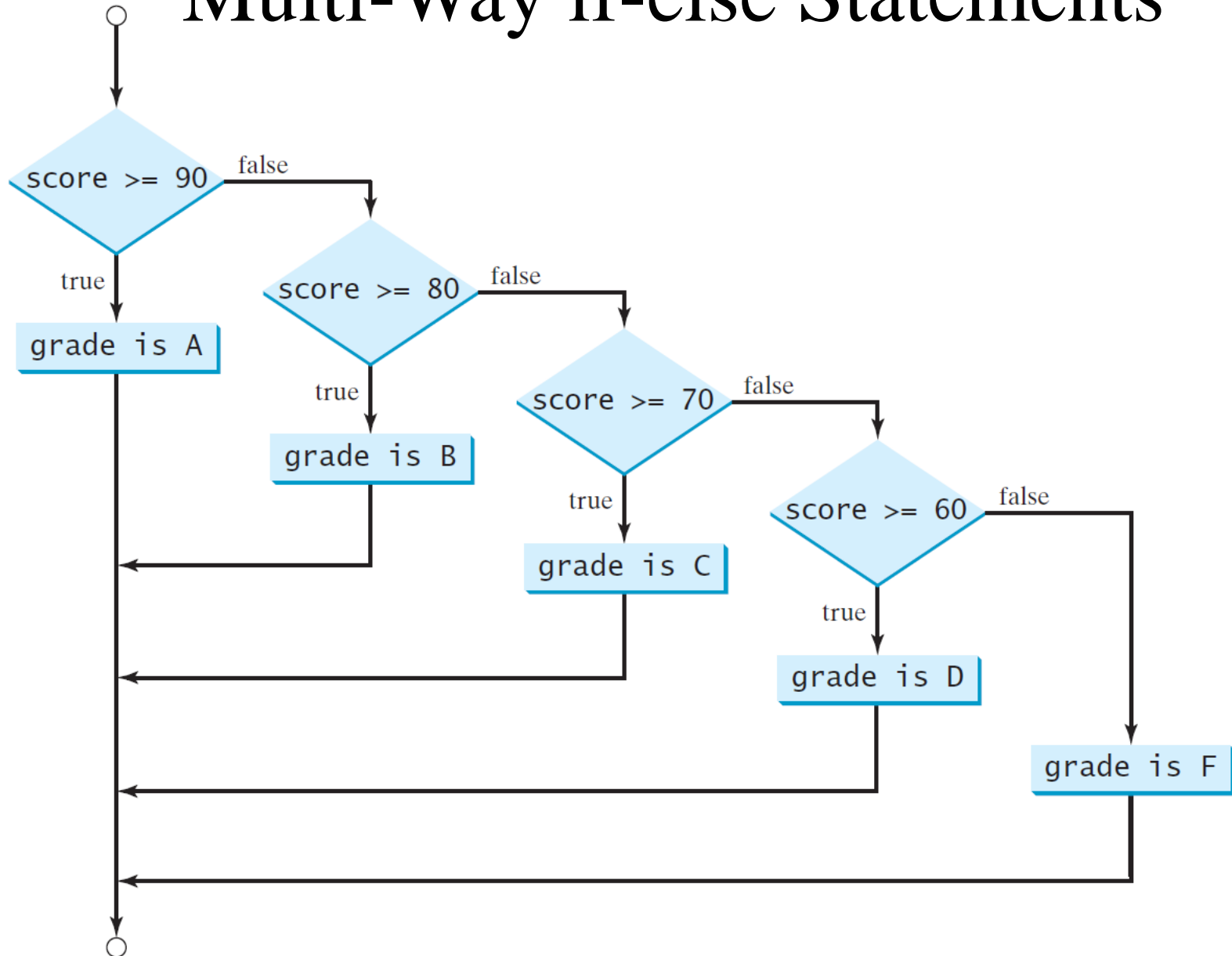
Equivalent

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



Trace if-else statement

Suppose score is 70.0

The condition is false

```
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

The condition is false

```
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

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

grade is C

```
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

Exit the if statement

```
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("E");
```



Note

The else clause matches the most recent 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)



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.

```
if (radius >= 0); ← 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

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

(a)

Equivalent

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

(b)



CAUTION

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

(a)

Equivalent

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

(b)



Logical Operators

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

Truth Table for Operator !

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 &&

p_1	p_2	$p_1 \ \&\& \ p_2$	Example (assume age = 24, weight = 140)
false	false	false	$(\text{age} \leq 18) \ \&\& \ (\text{weight} < 140)$ is false, because both conditions are both false.
false	true	false	
true	false	false	$(\text{age} > 18) \ \&\& \ (\text{weight} > 140)$ is false, because $(\text{weight} > 140)$ is false.
true	true	true	$(\text{age} > 18) \ \&\& \ (\text{weight} \geq 140)$ is true, because both $(\text{age} > 18)$ and $(\text{weight} \geq 140)$ are true.

Truth Table for Operator ||

p_1	p_2	$p_1 \parallel p_2$	Example (assume age = 24, weight = 140)
false	false	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 false, because (age > 14) is true.
true	true	true	

Truth Table for Operator \wedge

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 true, because $(\text{age} > 34)$ is false and $(\text{weight} > 140)$ is 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	

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

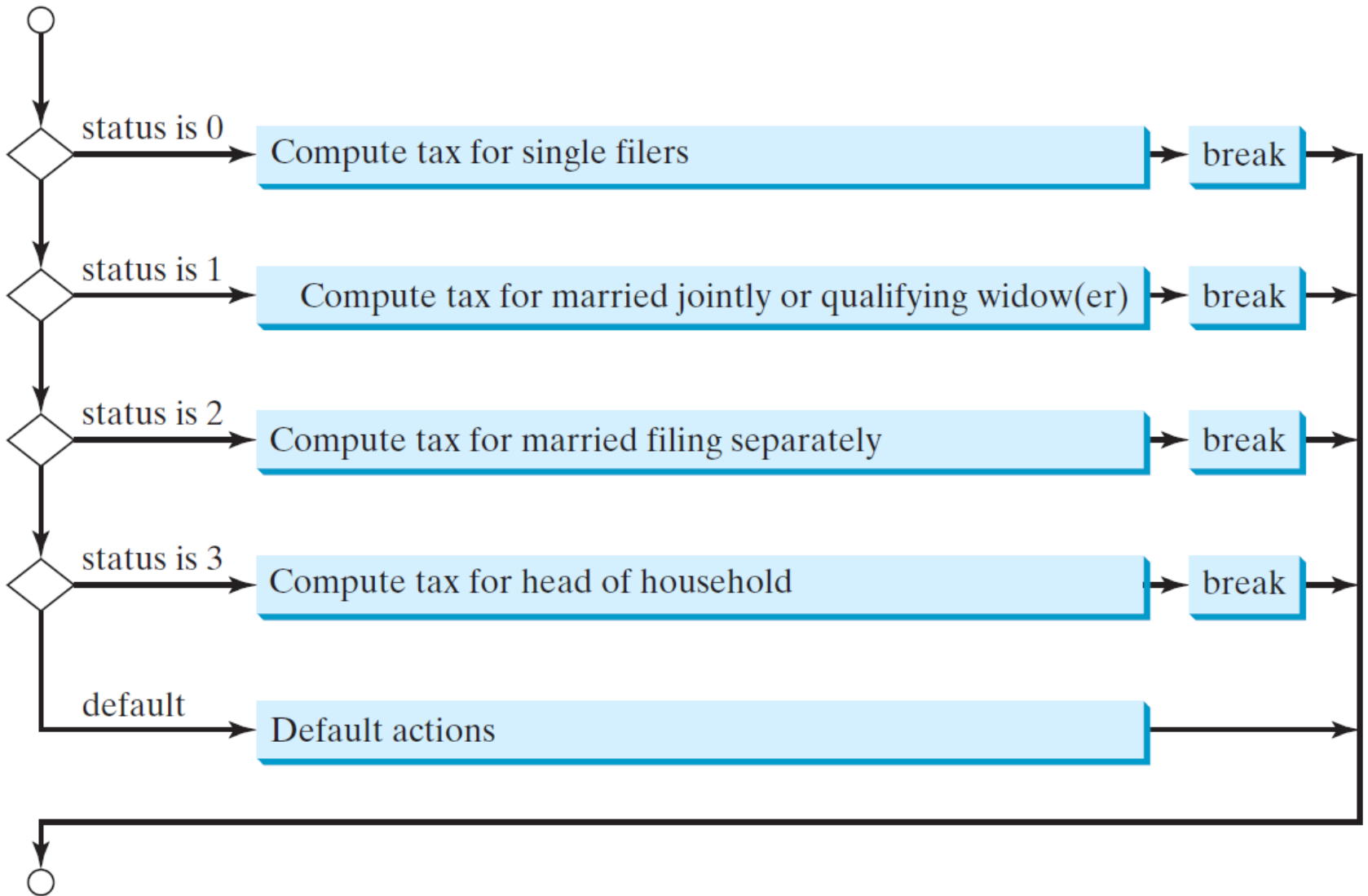
Run

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 $1 + x$.

```
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.

```
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

Fall through case 3

```
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

Fall through case 4

```
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

Fall through case 5

```
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

Encounter break

```
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

Exit the 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");
```



Conditional Operators

```
if (x > 0)
```

```
    y = 1
```

```
else
```

```
    y = -1;
```

is equivalent to

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

```
(boolean-expression) ? expression1 : expression2
```



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

- ☞ `+`, `-` (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.



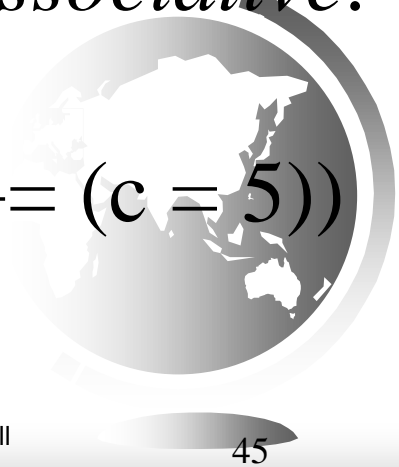
Operator Associativity

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*.

$a - b + c - d$ 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:

