Chapter 3
Selections

Motivations

If you assigned a negative value for <u>radius</u> in 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

The boolean data type declares a variable with the value either true or false.

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	radius < 0	false
<=	?	less than or equal to	radius <= 0	false
>	>	greater than	radius > 0	true
>=	?	greater than or equal to	radius >= 0	true
==	=	equal to	radius == 0	false
!=	?	not equal to	radius != 0	true

Problem: Addition Question

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.
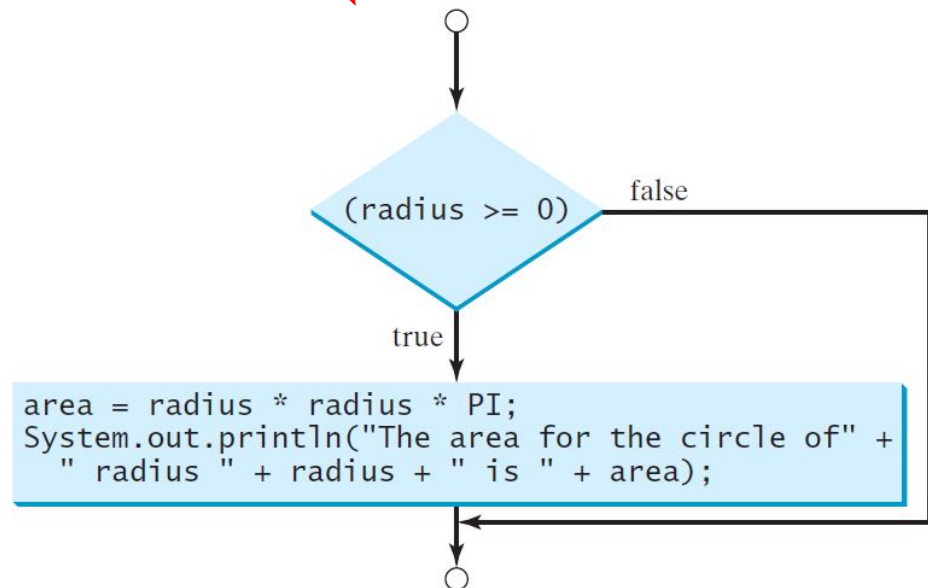
Animation

AdditionQuiz

if Statements (One Condition)

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

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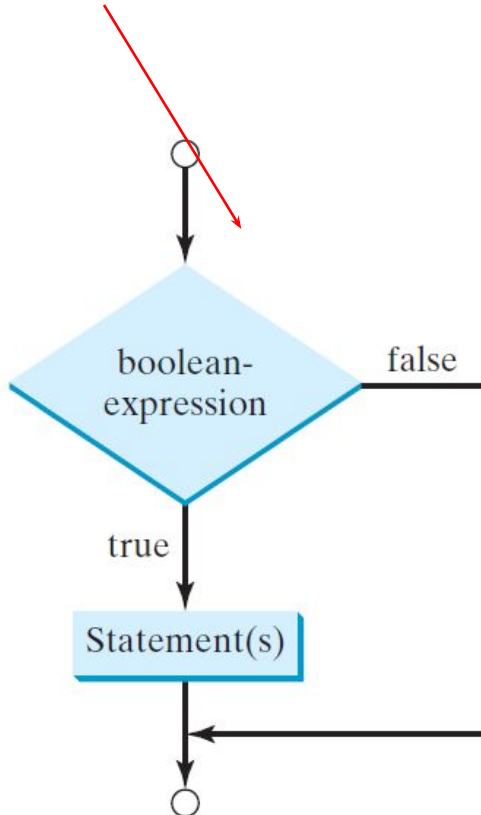
```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area"
    + " for the circle of radius "
    + radius + " is " + area);
}
```

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.
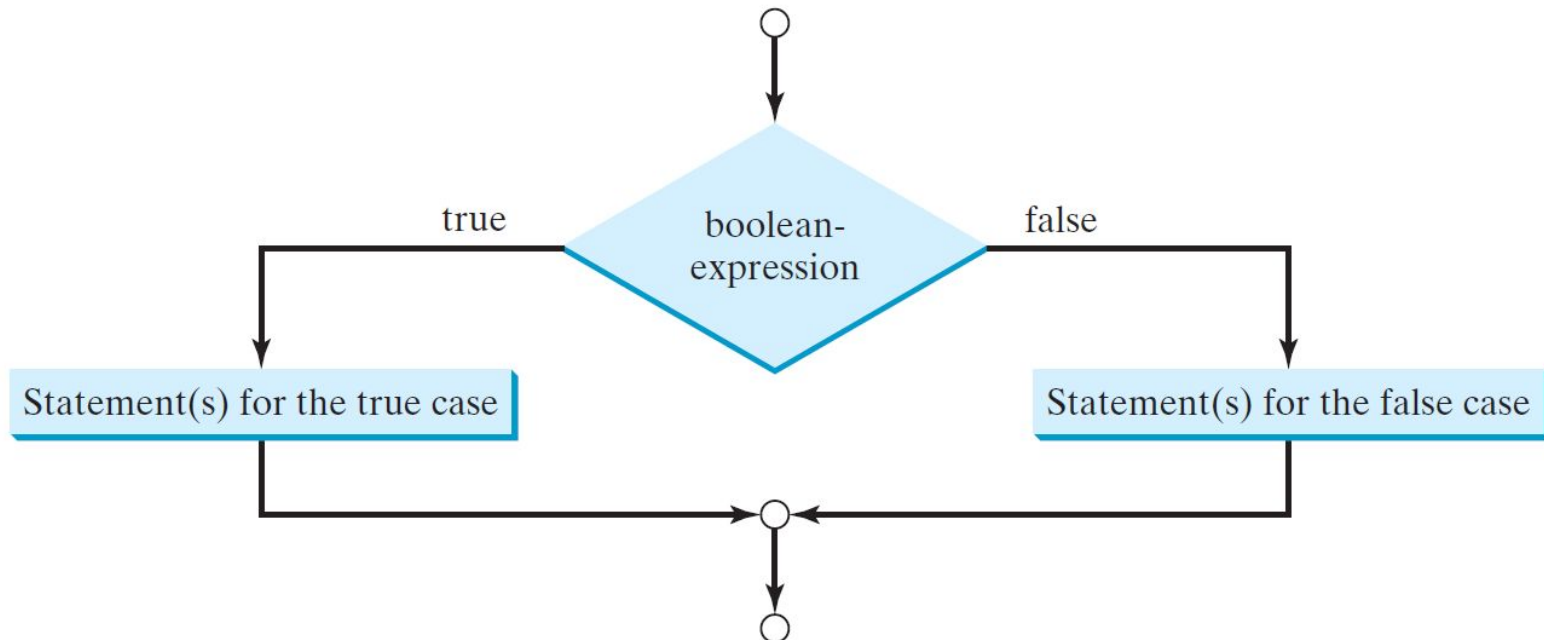
Animation

SimpleIfDemo

`if-else` Statement (2 conditions)

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;
}
```


true   boolean-expression   false

Statement(s) for the true case

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");
}
```

if Statements (more than 2 conditions)

```
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");
```
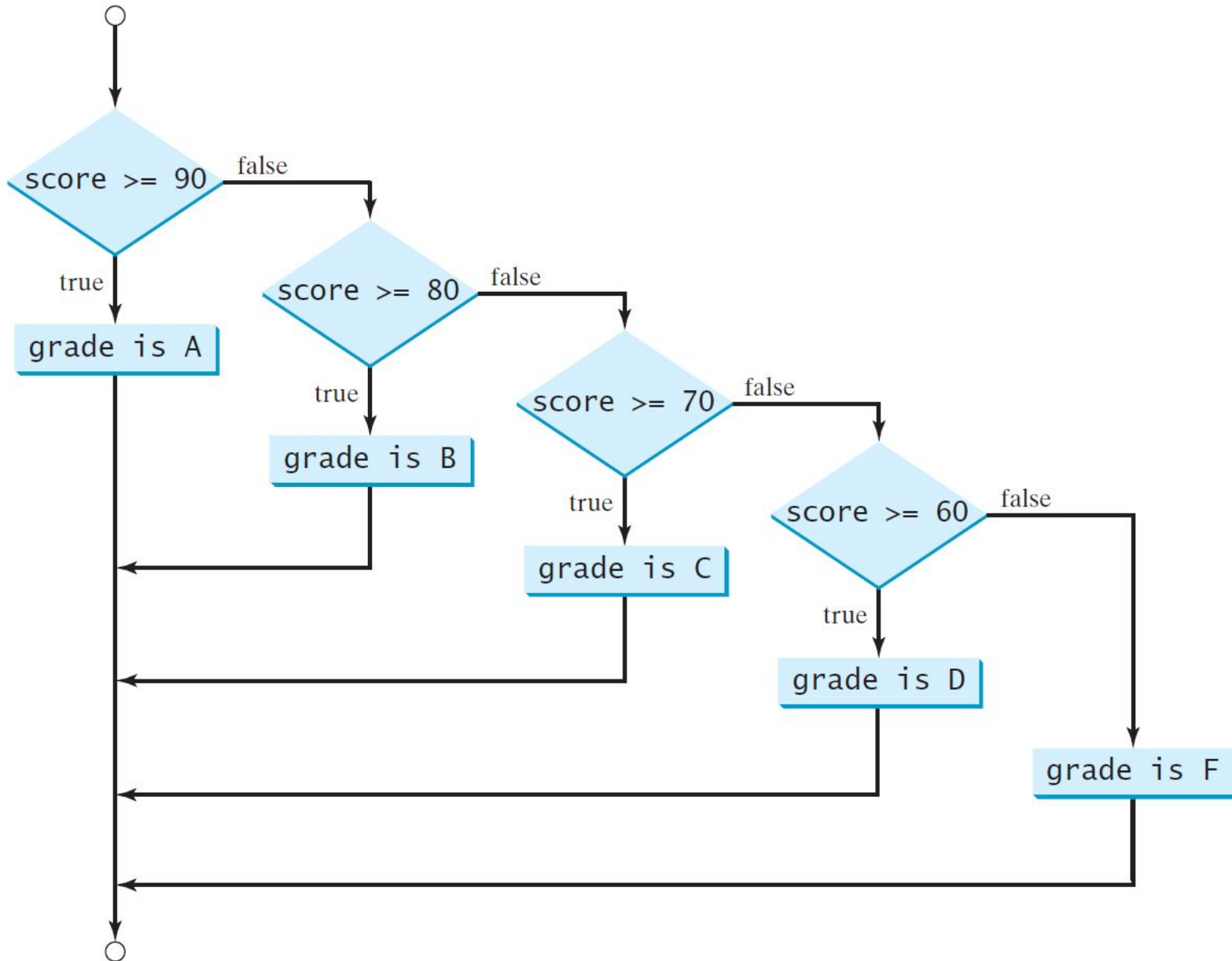
(a)

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)

12

if Statement (more than 2 conditions)

Trace if-else 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("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

The condition is true

```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

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
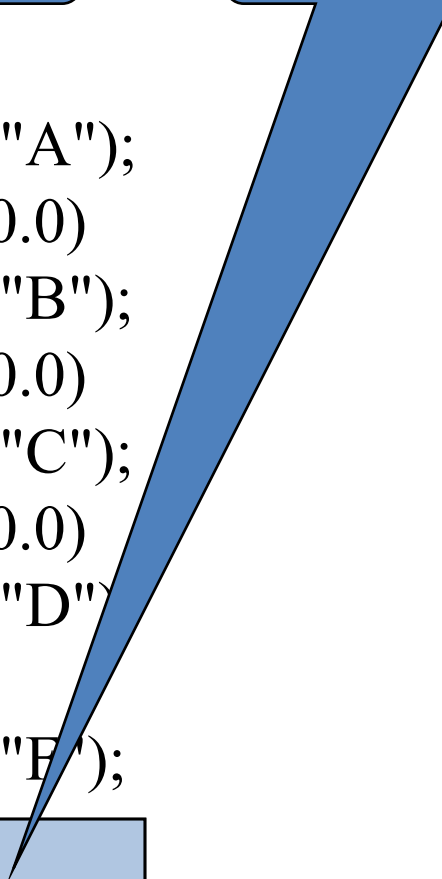
Suppose score is 70.0

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("F");
```

Note

The <u>else</u> clause matches the most recent <u>if</u> 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)

19

Note, cont.

Nothing is printed from the preceding statement. To force the <u>else</u> clause to match the first <u>if</u> 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.

Without block anything is printed.

Common Errors

Adding a semicolon at the end of an <u>if</u> 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)

Problem: Subtraction Question

This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers <u>number1</u> and <u>number2</u> with <u>number1 >= number2</u> 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.

SubtractionQuiz

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

ComputeAndInterpretBMI

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.

Marginal Tax Rate	Single	Married Filing Jointly or Qualifying Widow(er)	Married Filing Separately	Head of Household
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, cont.

```
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
}
```

[ComputeTax]

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

28

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	(age <= 18) && (weight < 140) is false, because (age > 18) and (weight <= 140) are both false.
false	true	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 ||

p_1	p_2	$p_1 \mid\mid p_2$	Example (assume age = 24, weihgt = 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 ^

p_1	p_2	p_1 ^ p_2	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) ^ (weight > 140) is false, because (age > 34) is false and (weight > 140) is 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 > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	

Examples

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

TestBooleanOperators

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)));

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 <u>int</u> 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)

LeapYear

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

`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
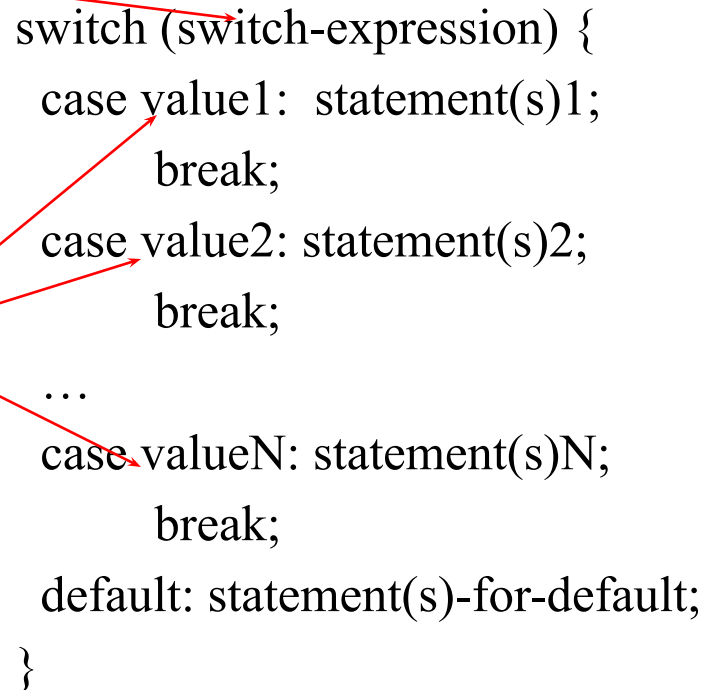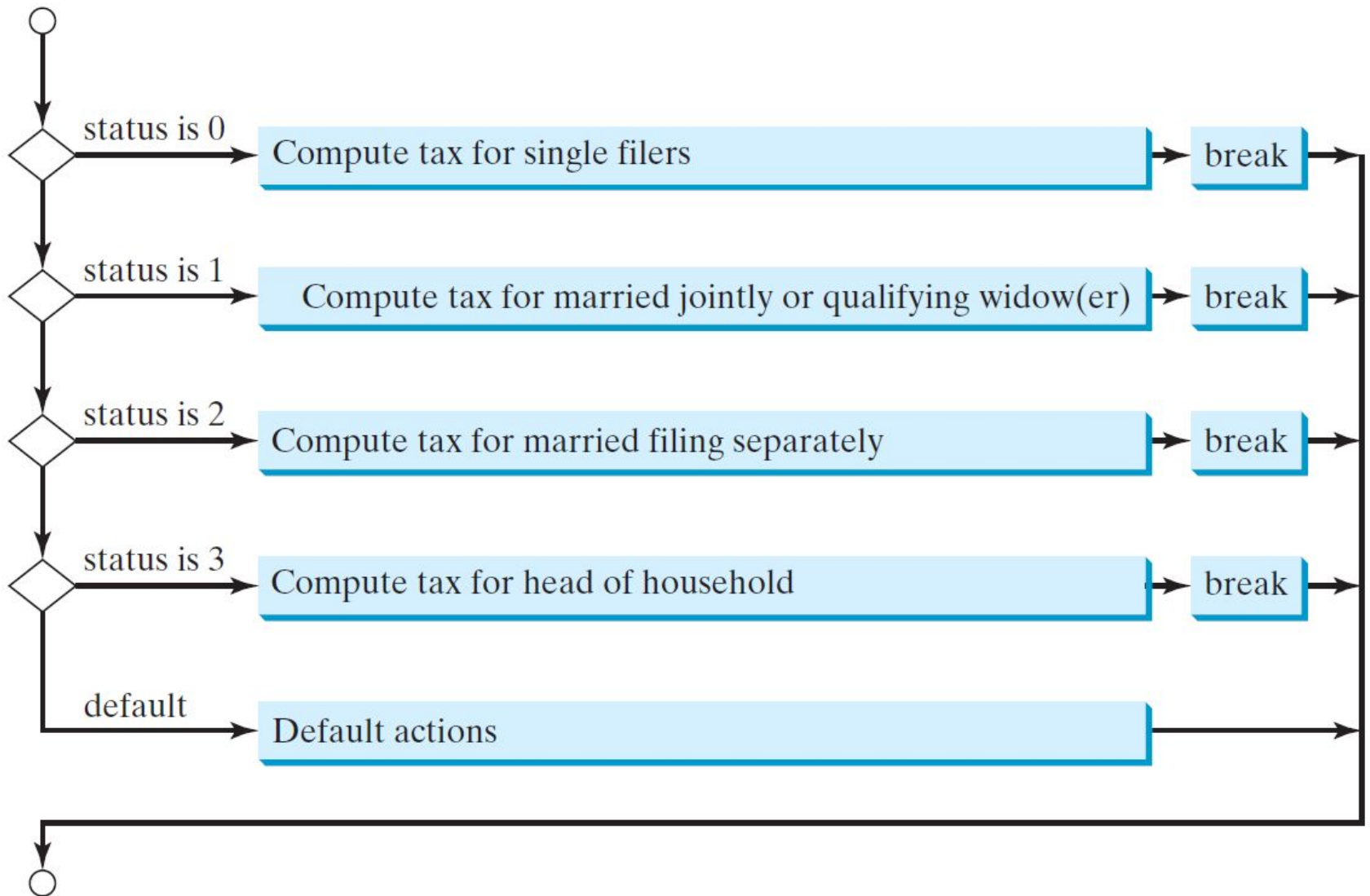
`switch` Statement Rules

The switch-expression must
be a value of <u>char</u>, <u>byte</u>,
<u>short</u>, or <u>int</u> type and must
always be enclosed in
parentheses.

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

```
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;
}
```
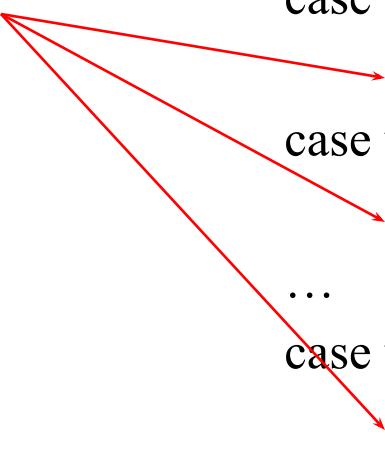
`switch` Statement Rules

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

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

```
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.

41

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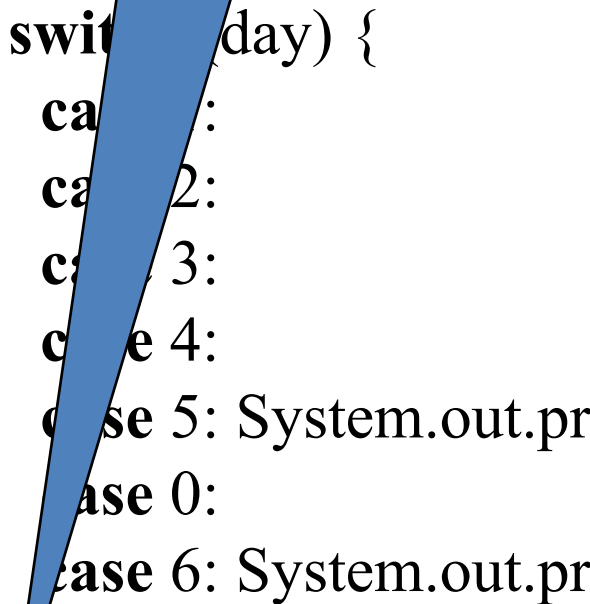
Trace switch statement

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");
}
```

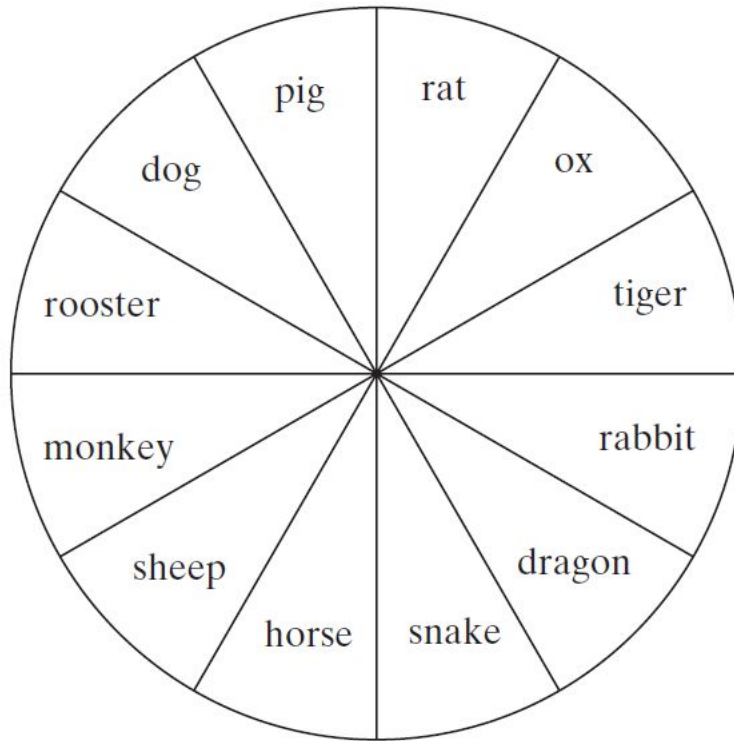# Trace switch statement

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");
}
```

Problem: Chinese Zodiac

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

$$year \% 12 = \begin{cases} 0: monkey \\ 1: rooster \\ 2: dog \\ 3: pig \\ 4: rat \\ 5: ox \\ 6: tiger \\ 7: rabbit \\ 8: dragon \\ 9: snake \\ 10: horse \\ 11: sheep \end{cases}$$

ChineseZodiac

Conditional Operator (?)

Syntax of conditional operator:

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

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 Expressions

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

is equivalent to

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

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*.**
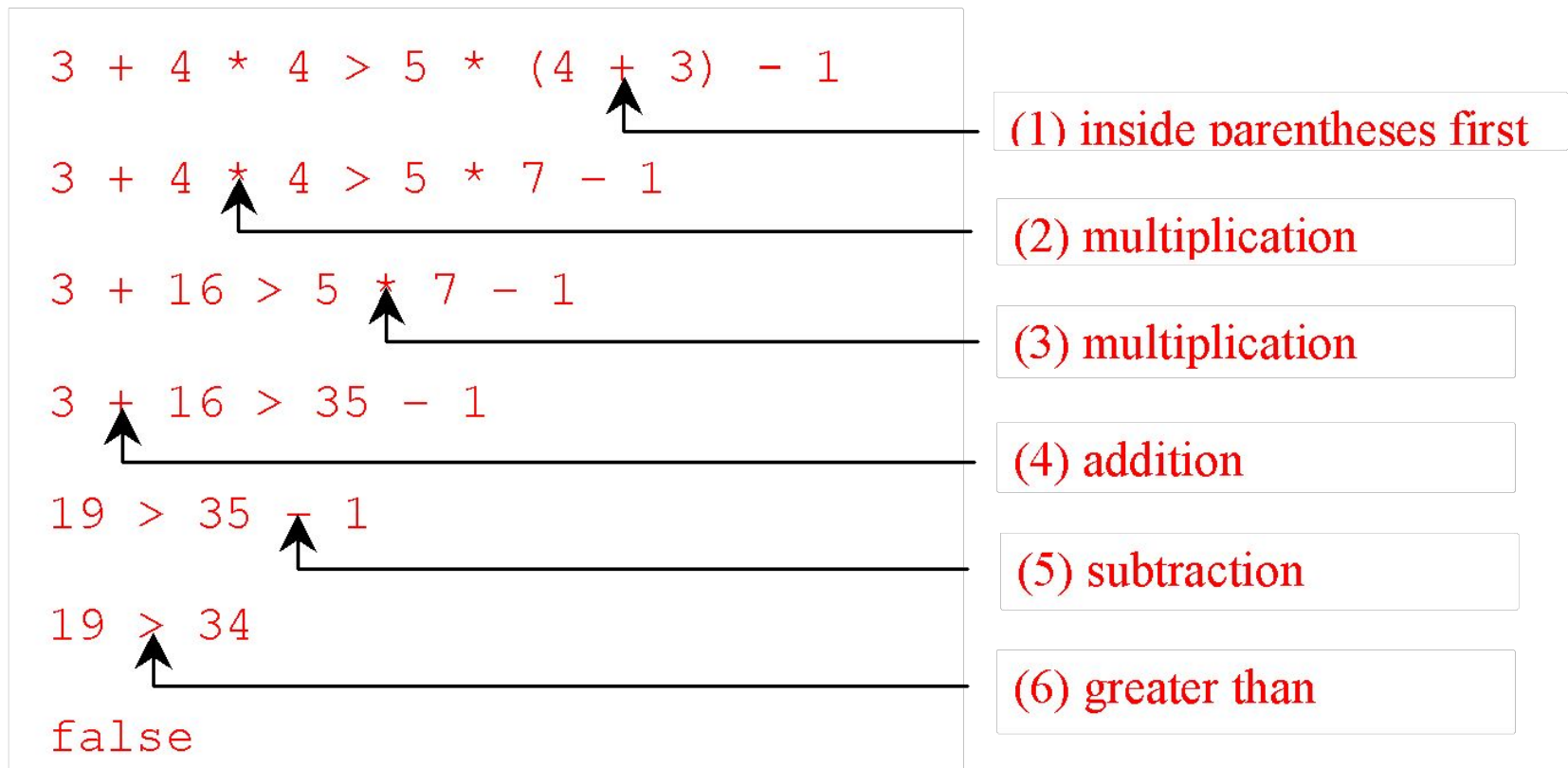
a – b + c – d is equivalent to ((a – b) + c) – d

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
```

Debugging

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 NetBeans/Eclipse