COMP20220
Programming II (Conversion)

Michael O'Mahony

Chapter 3 Selections

Motivations

Consider a program to calculate the area of a circle…

If you assigned a negative value for the radius, 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 multi-way **if** statements (§3.5).
- To program using selection statements (§§3.7–3.9).
- To combine conditions using logical operators (**&&**, ||, and **!**) (§3.10).
- To program using selection statements with combined conditions (§§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).

Relational Operators

Java provides six *relational operators* (also known as *comparison operators*) that can be used to compare two values.

Java Operator	**Mathematics Symbol**	**Name**	**Example** (radius is 5)	**Result**
`<`	$<$	less than	`radius < 0`	`false`
`<=`	$\leq$	less than or equal to	`radius <= 0`	`false`
`>`	$>$	greater than	`radius > 0`	`true`
`>=`	$\geq$	greater than or equal to	`radius >= 0`	`true`
`==`	$=$	equal to	`radius == 0`	`false`
`!=`	$\neq$	not equal to	`radius != 0`	`true`

The `boolean` Type

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

The `boolean` data type:

```
boolean b = true;
```

A *Boolean expression* is an expression that evaluates to a Boolean value – for example:

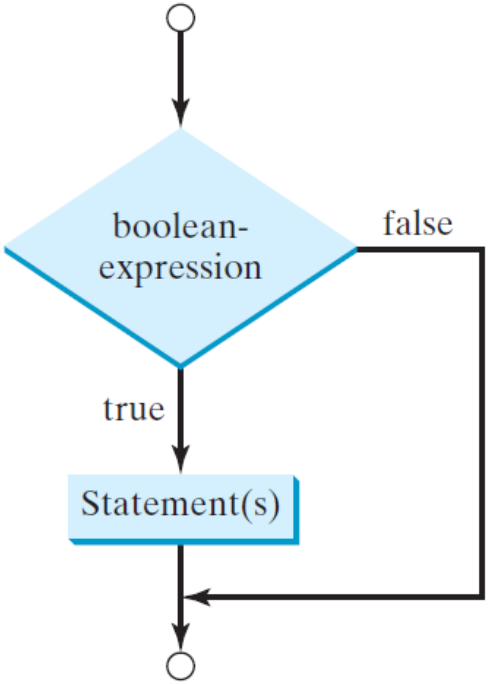
```
5 > 2
```

What is the value of `b`?

```
boolean b = 1 > 2;
```

One-way `if` Statement

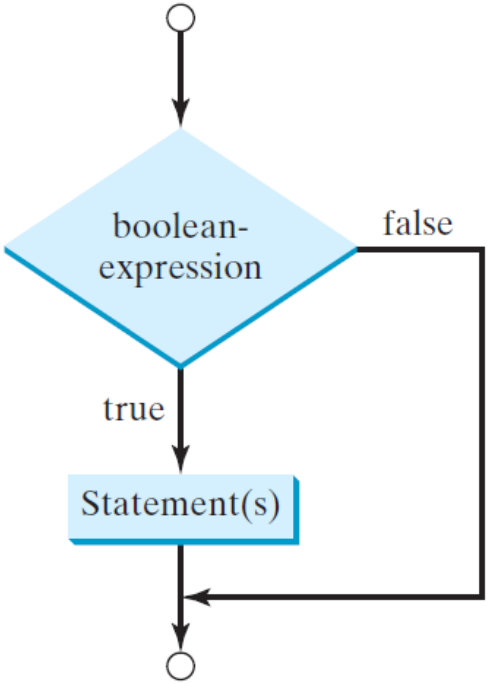
```
if (boolean-expression) {
  statement(s);
}
```

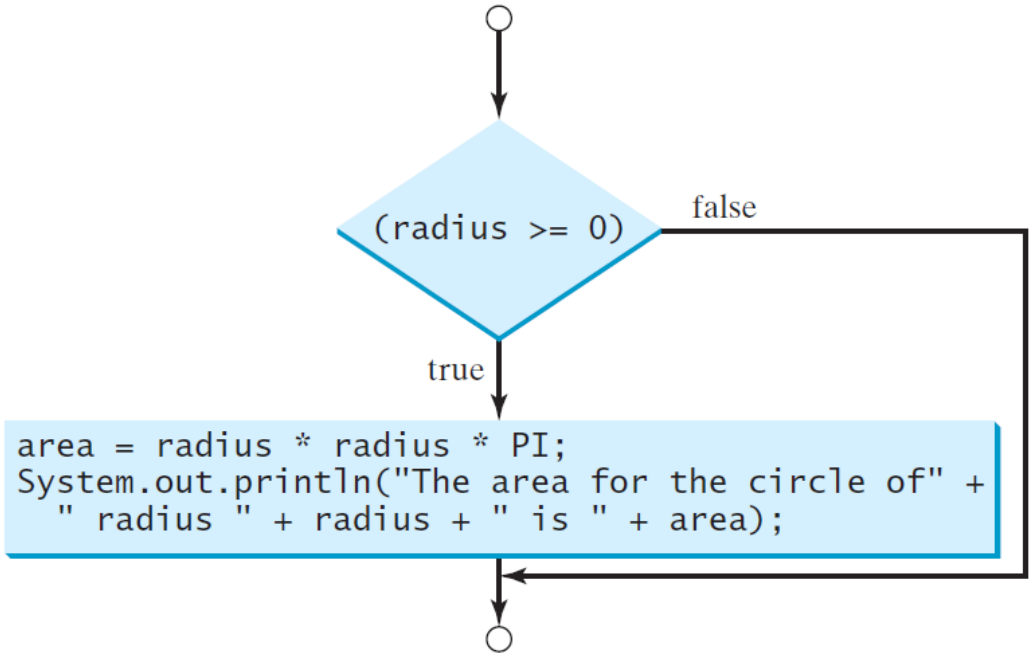


One-way `if` Statement

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

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println(…);
}
```





Notes

The boolean-expression must be enclosed in parentheses. The code in (a) is incorrect. The code in (b) is correct.

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

(a) Incorrect

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

(b) Correct

Notes

The block braces can be omitted if they enclose a *single* statement. The code shown in (a) and (b) is equivalent.

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

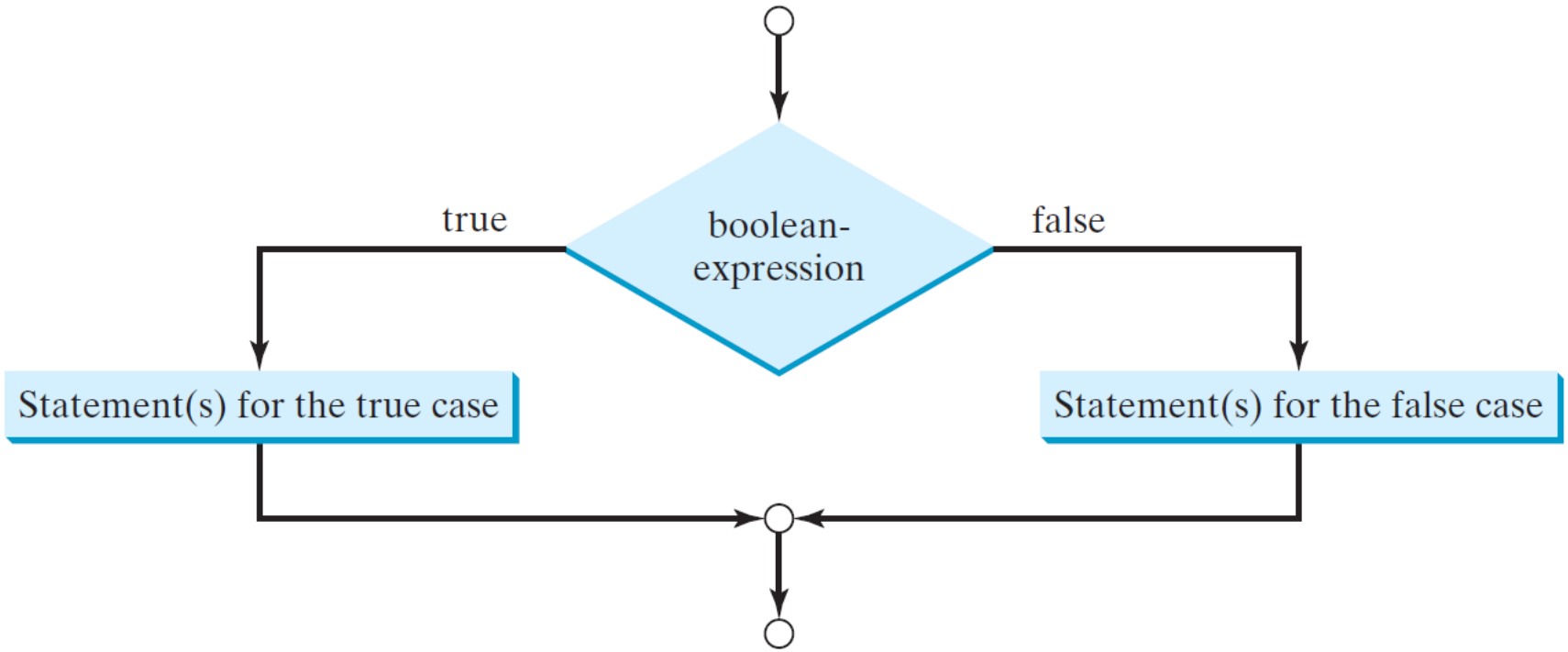
(a)

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

(b)

Two-way `if` Statement

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



`if-else` Example

Write a program to calculate the area of a circle:

- Prompt the user to enter the radius from the keyboard
- If the radius is >= 0; calculate the area and print the output
- If the radius is negative, display a suitable message to the user

ComputeArea

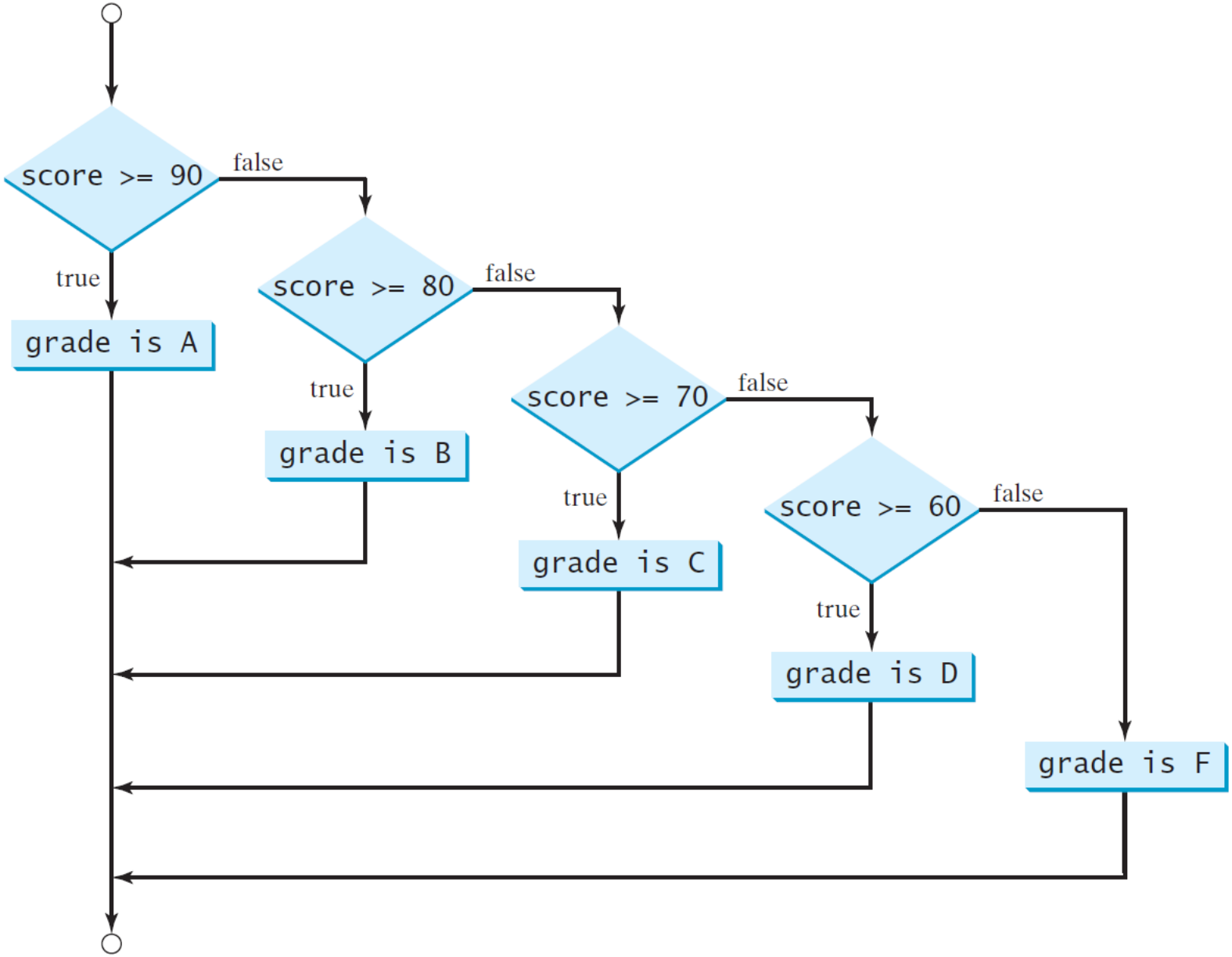
Multiple Alternative `if-else` Statements

Suppose you wish to print a letter grade corresponding to a score.

Use a multi-way `if-else` statement as follows:

```
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



animation

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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

```
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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

Execute statement – print "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");
```

animation

Trace `if-else` statement

Suppose score is 75.0

Exit the `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");
```

Equality Test: Floating-Point Values

Floating-point numbers have a limited precision and calculations involving floating-point numbers can introduce round-off errors.

Hence, equality test of two floating-point values should be avoided.

For example – is `b` equal to `true` or `false` in the below?

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
boolean b = x == 0.5;
```

Equality Test: Floating-Point Values

Floating-point numbers have a limited precision and calculations involving floating-point numbers can introduce round-off errors.

Hence, equality test of two floating-point values should be avoided.

For example – is `b` equal to `true` or `false` in the below?

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
boolean b = x == 0.5;
```

`b` is `false` because `x` is not exactly 0.5, but is 0.5000000000000001

Equality Test: Floating-Point Values

But you can compare whether two floating point values are *close enough*: i.e. two numbers `x` and `y` are close if `|x-y| < ε`.

Set ε to 10^{-14} for comparing two values of the `double` type and to 10^{-7} for comparing two values of the `float` type.

For example, the following code…

```
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");
```

…will display: 0.5000000000000001 is approximately 0.5

TIPS

Suppose we wish to check whether `number` is even or odd.

```
int number = 2;
```

TIPS

Suppose we wish to check whether `number` is even or odd.

```
int number = 2;
```

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

TIPS

Suppose we wish to check whether `number` is even or odd.

```
int number = 2;
```

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

Equivalent…

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

TIPS, Contd.

To test whether the `boolean` variable `even` is `true` or `false`, it is redundant to use the equality testing operator (==):

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

Equivalent

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

Better…

TIPS

To test whether a boolean variable is `true` or `false`, it is redundant to use the equality testing operator (==):

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

Equivalent

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

Better…

What happens here?

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

TIPS

To test whether a boolean variable is `true` or `false`, it is redundant to use the equality testing operator (==):

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

Equivalent

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

Better…

What happens here?

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

This statement assigns `true` to `even`, so that `even` is always true…

Problem: A Mathematics Learning Tool

Write a program to teach a first grade child how to learn subtractions:

- Prompt the user to enter two integers, `n1` and `n2`
- To avoid dealing with negative numbers, if `n1` < `n2` then swap the numbers
- Prompt the user to answer the question: What is `n1` – `n2` ?
- Display whether the answer is correct

SubtractionTest

Logical Operators

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

Truth Table for Operator &&

p_1	p_2	p_1 && p_2
false	false	false
false	true	false
true	false	false
true	true	true

Truth Table for Operator ||

p_1	p_2	$p_1 \| p_2$
false	false	false
false	true	true
true	false	true
true	true	true

Truth Table for Operator ^

p_1	p_2	$p_1 \wedge p_2$
false	false	false
false	true	true
true	false	true
true	true	false

Truth Table for Operator !

p	!p
true	false
false	true

Example: Logical Operators

Write a program that reads in a number and checks whether the number is:

- Divisible by both 2 and 3
- Divisible by 2 or 3 or both
- Divisible by 2 or 3 but not both

TestBooleanOperators2

Example: Leap Year

Write a program that prompts the user to enter a year as an `int` value and checks if it is a leap year.

Solution:

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

Example: Leap Year

Write a program that prompts the user to enter a year as an `int` value and checks if it is a leap year.

Solution:

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

Example: Leap Year

Write a program that prompts the user to enter a year as an `int` value and checks if it is a leap year.

Solution:

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

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

LeapYear

`switch` Statement

Overuse of multiple `if-else` statements can make a program difficult to read.

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

`switch` Statement Rules

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

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

`value1`, ..., `valueN` are constant expressions – cannot contain variables, e.g. `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)D;
}
```

`switch` Statement Rules

The keyword `break` is optional.

When the value in the `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.

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

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

animation

Trace `switch` statement

The following code displays "Weekday" for day values of 1 to 5 and "Weekend" for day values of 0 and 6.

```
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");
}
```

animation

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");
}
```

animation

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");
}
```

animation

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");
}
```

animation

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");
}
```

animation

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");
}
```

animation

Trace `switch` statement

Execute statement(s)

```
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");
}
```

animation

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");
}
```

animation

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

Conditional Expressions

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

For example:

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

is equivalent to:

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

The syntax is:

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

The symbols `?` and `:` appear together in a conditional expression. They form a conditional operator (also called a ternary operator) because three operands are involved. It is the only ternary operator in Java.

Conditional Operator

For example:

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

Conditional Operator

For example:

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

…can be written as…

```
System.out.println(
  (num % 2 == 0)? "num is even" : "num is odd");
```

Conditional Operator

For example:

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

…can be written as…

```
System.out.println(
  (num % 2 == 0)? "num is even" : "num is odd");
```

What does the following do?

```
result = (num1 > num2) ? num1 : num2;
```

Operator Precedence and Associativity

Operator *precedence* and *associativity* determine the order in which operators are evaluated.

Expressions within parentheses are evaluated first.

The *precedence rule* defines precedence for operators:

- Operators with the same precedence appear in the same group (see next slide)

Operator Precedence

Precedence	*Operator*
↓	! (Not)
	*, /, % (Multiplication, division, and remainder)
	+, − (Binary addition and subtraction)
	<, <=, >, >= (Relational)
	==, != (Equality)
	^ (Exclusive OR)
	&& (AND)
	|| (OR)
	=, +=, −=, *=, /=, %= (Assignment operator)

Operator Precedence and Associativity

If multiple operators with the same precedence occur in a statement, their *associativity* determines the order of evaluation.

All binary operators, except assignment operators, are left-associative.

For example, since + and – are of the same precedence and are left associative, the expression:

```
a - b + c - d
```

is equivalent to

```
((a - b) + c) - d
```

Assignment operators are right associative:

```
a = b += c = 5
```

is equivalent to

```
a = (b += (c = 5))
```

Example

Applying the operator precedence and associativity rule, evaluate the following expression:

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

Example

Applying the operator precedence and associativity rule, evaluate the following expression:

```
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
```

Next Topics…

Chapter 4

- Explore the `Math` class in more detail.
- Encoding characters using ASCII and Unicode, using escape characters.
- Introduce objects and instance methods.
- Represent strings using `String` objects.