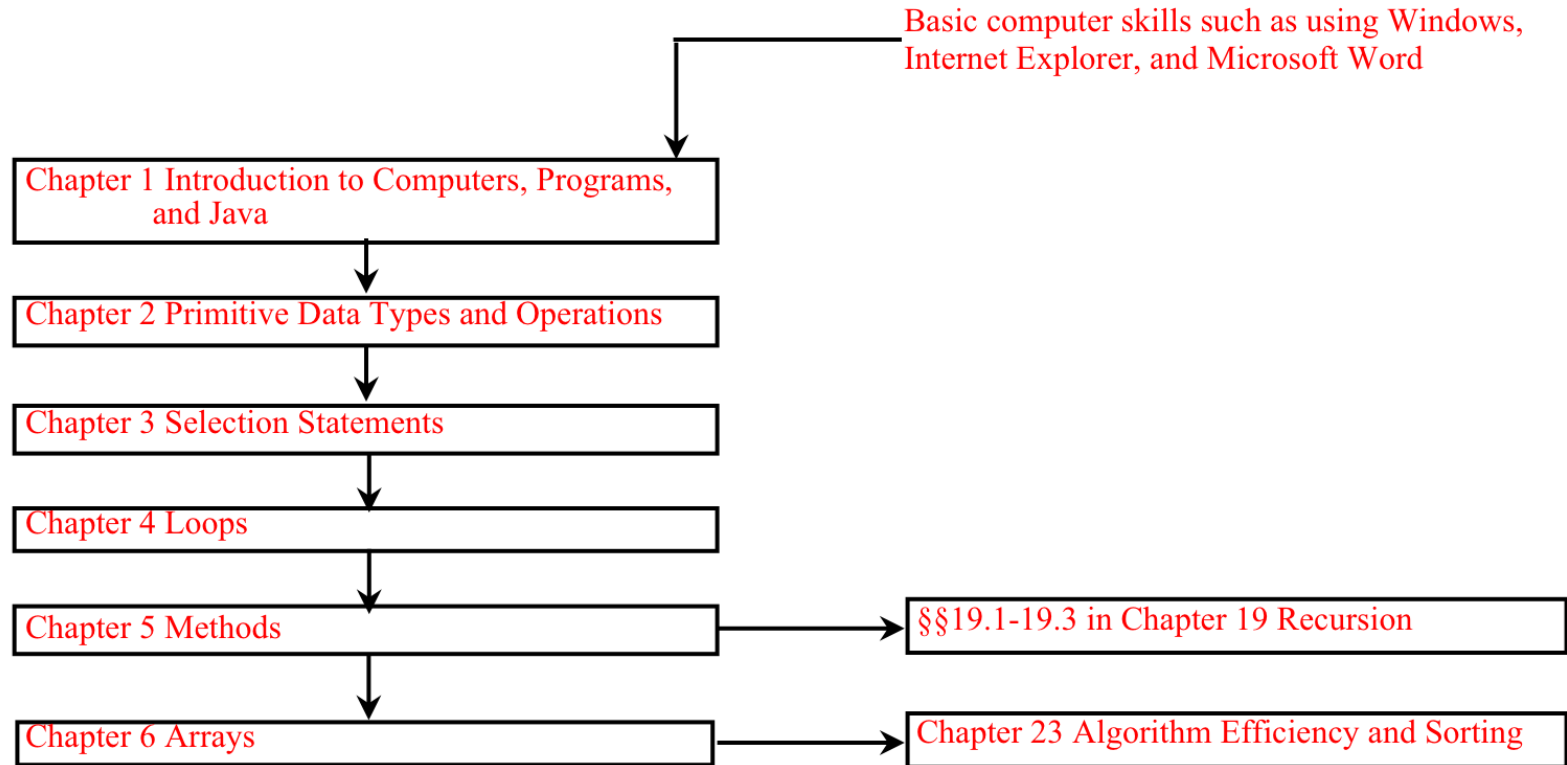


Chapter 3 Control Statements



Objectives

- ◆ To declare boolean type and write Boolean expressions (§3.2).
- ◆ To distinguish between conditional and unconditional `&&` and `||` operators (§3.2.1).
- ◆ To use Boolean expressions to control selection statements (§3.3-3.5).
- ◆ To implement selection control using if and nested if statements (§3.3).
- ◆ To implement selection control using switch statements (§3.4).
- ◆ To write expressions using the conditional operator (§3.5) .
- ◆ To display formatted output using the System.out.printf method and to format strings using the String.format method (§3.6).
- ◆ To know the rules governing operand evaluation order, operator precedence, and operator associativity (§§3.7-3.8) .

Comparison Operators

<i>Operator</i>	<i>Name</i>
-----------------	-------------

<	less than
---	-----------

<=	less than or equal to
----	-----------------------

>	greater than
---	--------------

>=	greater than or equal to
----	--------------------------

==	equal to
----	----------

!=	not equal to
----	--------------

Boolean Operators

<i>Operator</i>	<i>Name</i>
-----------------	-------------

!	not
---	-----

& &	and
-----	-----

	or
--	----

^	exclusive or
---	--------------

Truth Table for Operator !

p	!p
true	false
false	true

Example
!(1 > 2) is true, because (1 > 2) is false.
!(1 > 0) is false, because (1 > 0) is true.

Truth Table for Operator &&

p1	p2	p1 && p2
false	false	false
false	true	false
true	false	false
true	true	true

Example
(3 > 2) && (5 >= 5) is true, because (3 > 2) and (5 >= 5) are both true.
(3 > 2) && (5 > 5) is false, because (5 > 5) is false.

Truth Table for Operator ||

p1	p2	p1 p2
false	false	false
false	true	true
true	false	true
true	true	true

Example
(2 > 3) (5 > 5) is false, because (2 > 3) and (5 > 5) are both false.
(3 > 2) (5 > 5) is true, because (3 > 2) is true.

Truth Table for Operator \wedge

p1	p2	p1 \wedge p2	Example
false	false	false	$(2 > 3) \wedge (5 > 1)$ is true, because $(2 > 3)$ is false and $(5 > 1)$ is true.
false	true	true	
true	false	true	$(3 > 2) \wedge (5 > 1)$ is false, because both $(3 > 2)$ and $(5 > 1)$ are true.
true	true	false	

Examples

```
System.out.println("Is " + num + " divisible by 2 and 3? " +  
((num % 2 == 0) && (num % 3 == 0)));
```

```
System.out.println("Is " + num + " divisible by 2 or 3? " +  
((num % 2 == 0) || (num % 3 == 0)));
```

```
System.out.println("Is " + num +  
" divisible by 2 or 3, but not both? " +  
((num % 2 == 0) ^ (num % 3 == 0)));
```

Example: 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)

[LeapYear](#)

Example: 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, as shown below. After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is true or false.



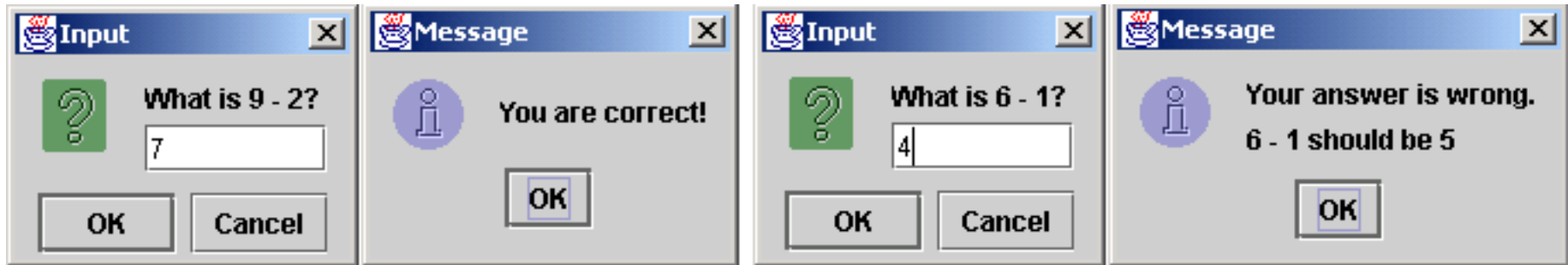
AdditionTutor

Selection Statements

- ✦ `if` Statements
- ✦ `switch` Statements
- ✦ Conditional Operators

Example: 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, as shown in the figure. After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is correct, as shown in figure.



[SubtractionTutor](#)

Example: Guessing Birth Date

The program can guess your birth date. Run to see how it works.

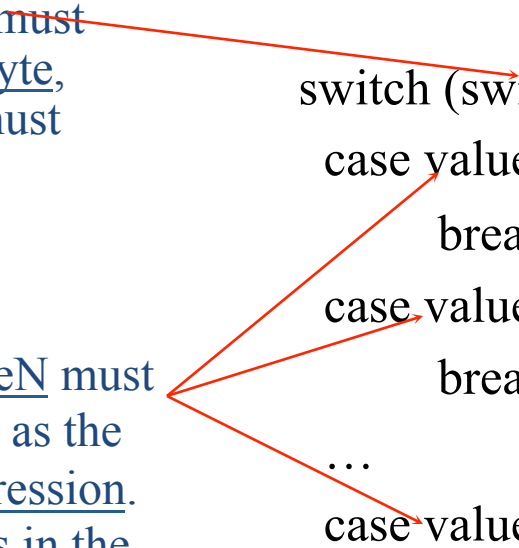
[GuessBirthDate](#)

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 $1 + \underline{x}$.

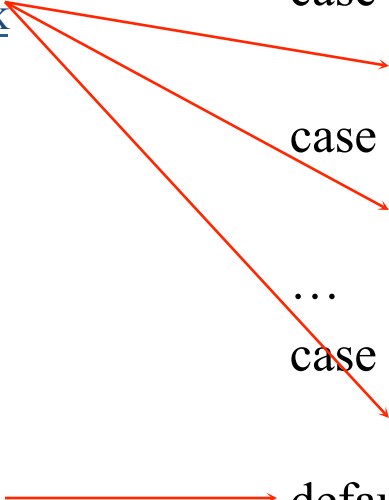
```
switch (switch-expression) {  
    case value1: statement(s)1;  
        break;  
    case value2: statement(s)2;  
        break;  
    ...  
    case valueN: statement(s)N;  
        break;  
    default: statement(s)-for-default;  
}
```



switch Statement Rules

The keyword 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.

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

Conditional Operator

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

```
    y = 1
```

```
else
```

```
    y = -1;
```

is equivalent to

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

```
(booleanExpression) ? 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.

`(booleanExp) ? exp1 : exp2`

Formatting Output

Use the new JDK 1.5 printf statement.

```
System.out.printf(format, items);
```

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
<u>%b</u>	a boolean value	true or false
<u>%c</u>	a character	'a'
<u>%d</u>	a decimal integer	200
<u>%f</u>	a floating-point number	45.460000
<u>%e</u>	a number in standard scientific notation	4.556000e+01
<u>%s</u>	a string	"Java is cool"

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000

Creating Formatted Strings

`System.out.printf(format, item1, item2, ..., item k)`

`String.format(format, item1, item2, ..., item k)`

`String s = String.format("count is %d and amount is %f", 5, 45.56));`

Operator Precedence

How to evaluate $3 + 4 * 4 > 5 * (4 + 3) - 1$?

Operator Precedence

- `var++`, `var--`
- `+`, `-` (Unary plus and minus), `++var`, `--var`
- `(type)` Casting
- `!` (Not)
- `*`, `/`, `%` (Multiplication, division, and remainder)
- `+`, `-` (Binary addition and subtraction)
- `<`, `<=`, `>`, `>=` (Comparison)
- `==`, `!=`; (Equality)
- `&` (Unconditional AND)
- `^` (Exclusive OR)
- `|` (Unconditional 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 $a = (b += (c = 5))$

Operand Evaluation Order

The precedence and associativity rules specify the order of the operators, but do not specify the order in which the operands of a binary operator are evaluated. Operands are evaluated from left to right in Java.

The left-hand operand of a binary operator is evaluated before any part of the right-hand operand is evaluated.

Operand Evaluation Order, cont.

If no operands have *side effects* that change the value of a variable, the order of operand evaluation is irrelevant. Interesting cases arise when operands do have a side effect. For example, x becomes 1 in the following code, because \underline{a} is evaluated to 0 before $++a$ is evaluated to 1.

```
int a = 0;  
int x = a + (++a);
```

But x becomes 2 in the following code, because $++a$ is evaluated to 1, then a is evaluated to 1.

```
int a = 0;  
int x = ++a + a;
```

Rule of Evaluating an Expression

Rule 1: Evaluate whatever subexpressions you can possibly evaluate from left to right.

Rule 2: The operators are applied according to their precedence.

Rule 3: The associativity rule applies for two operators next to each other with the same precedence.

Rule of Evaluating an Expression

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

