CANADIAN UNIVERSITY OF BANGLADESH

Inspiring Applied Knowledge

School of Science & Engineering
Department of CSE
Canadian University of Bangladesh

Lecture-4: Object-Oriented Problem Solving (Part-II)

Prerequisite: CSE 1101
Semester: Summer 2024

Object-Oriented Problem Solving

Programming Fundamentals (Part II)

Based on sections from chapters 3 & 5 of “Introduction to Java Programming” by Y. Daniel Liang.

Outline

- Selections
  - One-way if statements (3.3)
  - Two-way if-else statements (3.4)
  - Nested If and Multi-Way if-else Statements (3.5)
  - Logical Operators (3.10)
  - Switch statements (3.13)
  - Conditional expressions (3.14)
- Loops
  - While loops (5.2)
  - The do-while loops (5.3)
  - For loops (5.4)
  - Nested Loops (5.6)
  - Keywords break and continue (5.9)

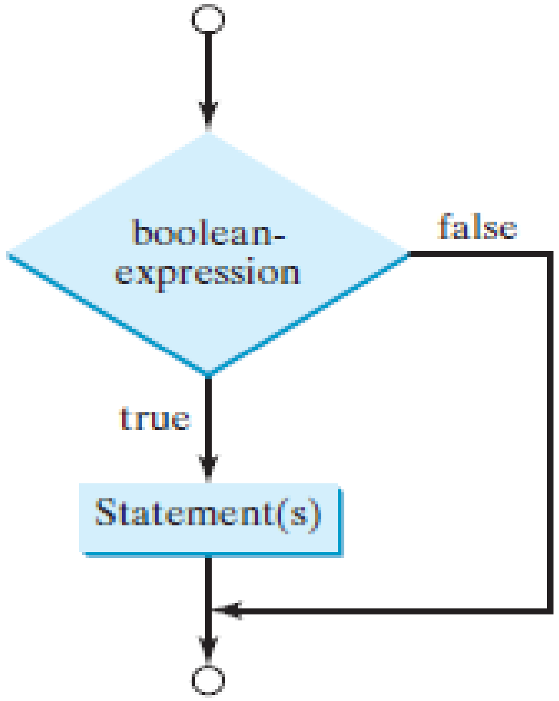
Selections

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

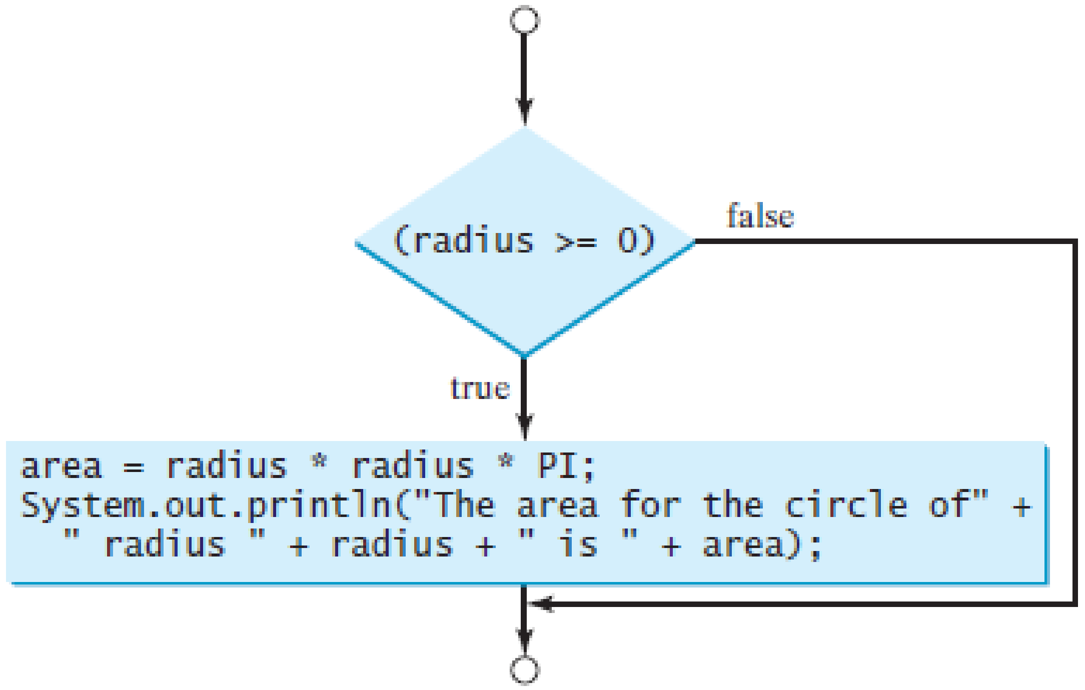
One-way If Statements

- An *if* statement is a construct that enables a program to specify alternative paths of execution.
- A *one-way if* statement executes an action *if an only if* the condition is *true*.
  - If the condition is *false*, nothing is done.
- The syntax for a one-way if statement is:

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



One-way If Statements (Example)



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

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

One-way If Statements to find Circle Area:

```
public class CircleAreaCalculator {

    public static void main(String[] args) {
        // Declare and initialize the radius
        double radius = 5.0; // You can change this value to test with different radii
        double area;
        final double PI = 3.14159; // Constant value for PI

        // One-way if statement to calculate the area if the radius is non-negative
        if (radius >= 0) {
            area = radius * radius * PI;
            System.out.println("The area for the circle of radius " + radius + " is " + area);
        } else {
            System.out.println("The radius cannot be negative.");
        }
    }
}
```

One-way If Statements (Cont.)

- 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");
}
```

Equivalent

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

An example of a Java program that demonstrates the use of a "one-way if statement." This type of if statement executes a block of code only if the specified condition is true. If the condition is false, the code block is skipped.

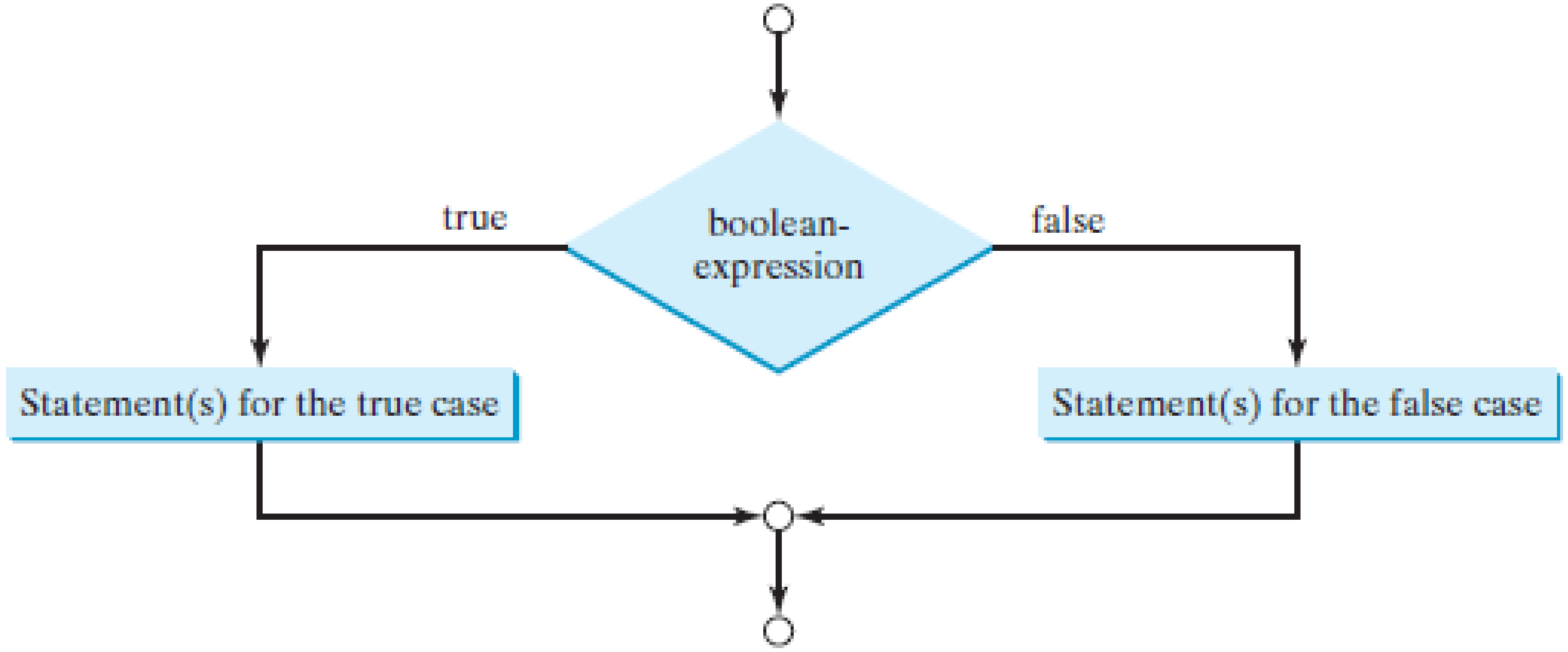
```
public class OneWayIfExample {

    public static void main(String[] args) {
        int number = 10;

        // One-way if statement
        if (number > 5) {
            System.out.println("The number is greater than 5.");
        }
        System.out.println("This statement is always executed.");
    }
}
```

Two-way If-else Statements

- A *two-way if-else* statement executes an action if the condition is *true* and another action if the condition is *false*.
- The syntax for a two-way if-else statement is:

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

Two-way If-else Statements (Cont.)


true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

Two-way If-else Statements (Example)

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

```
// Two-way if-else statement
public class CircleAreaCalculator {

    public static void main(String[] args) {
        // Declare and initialize the radius
        double radius = -3.0; // You can change this value to test with
different radii
        double area;
        final double PI = 3.14159; // Constant value for PI
        // If-else statement to calculate the area if the radius is non-negative
        if (radius >= 0) {
            area = radius * radius * PI;
            System.out.println("The area for the circle of radius " + radius +
" is " + area);
        } else {
            System.out.println("Negative input");
        }
    }
}
```

Nested If and Multi-Way if-else Statements

- An if statement can be inside another if statement to form a *nested* if statement.
- Example:

```
if (i > k){
    if (j > k)
        System.out.println("i and j are greater than k");
}
else
    System.out.println("i is less than or equal to k");
```

Executed only if i>k and j>k

Executed if i<=k

Nested If and Multi-Way if-else Statements (Example)

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

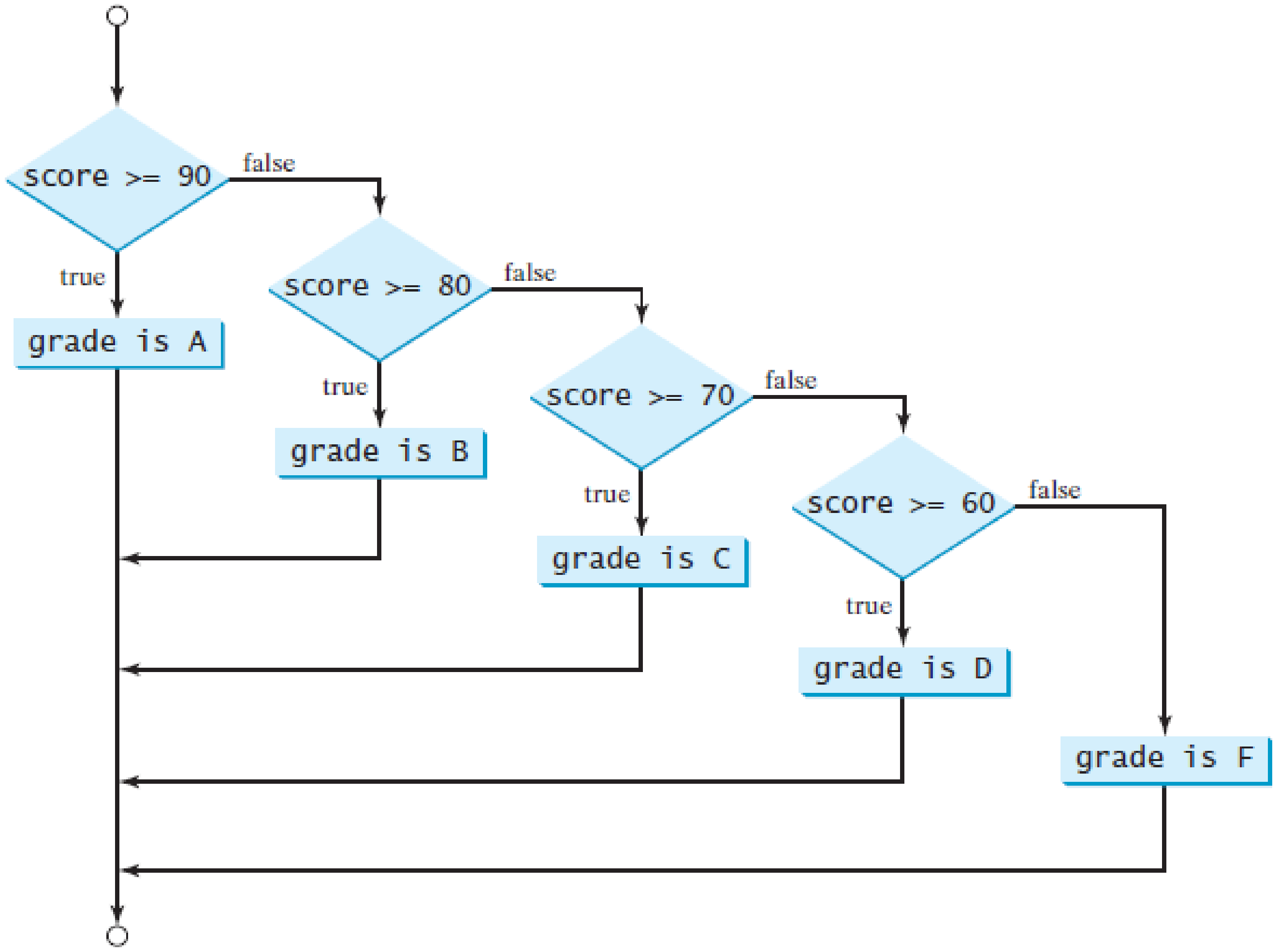
Equivalent

This is better

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

```
// Nested If and Multi-Way if-else Statements
public class GradeCalculator {

    public static void main(String[] args) {
        // Declare and initialize the score
        double score = 85.5; // You can change this value to test with different scores
        char grade;

        // If-else if-else statement to determine the grade based on the score
        if (score >= 90.0) {
            grade = 'A';
        } else if (score >= 80.0) {
            grade = 'B';
        } else if (score >= 70.0) {
            grade = 'C';
        } else if (score >= 60.0) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        // Print the result
        System.out.println("The grade for the score of " + score + " is " + grade);
    }
}
```

Nested If and Multi-Way if-else Statements (Example)

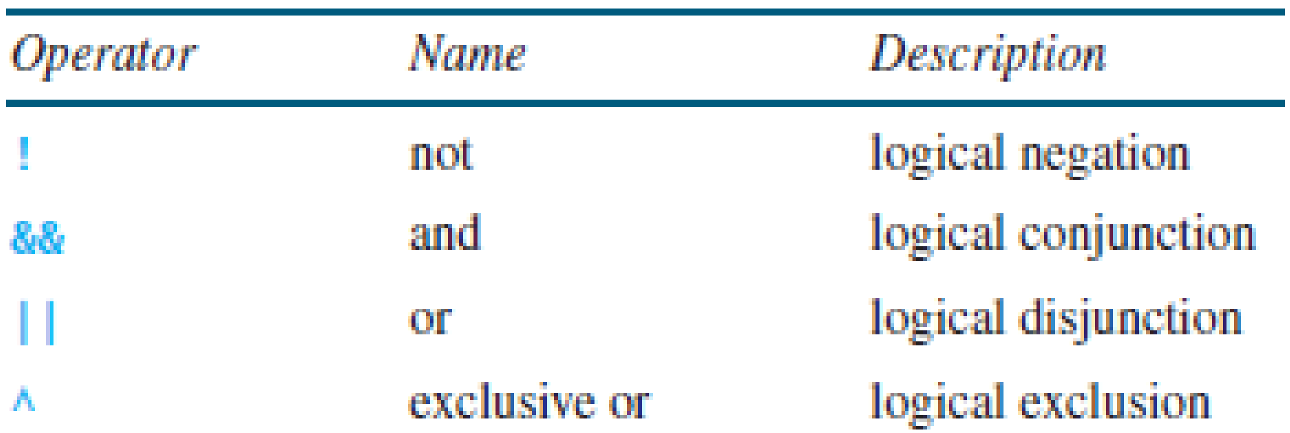


Note

- Check section 3.6 (Common Errors and Pitfalls).

Logical Operators

- *Logical operators* can be used to create a *compound* Boolean expression.



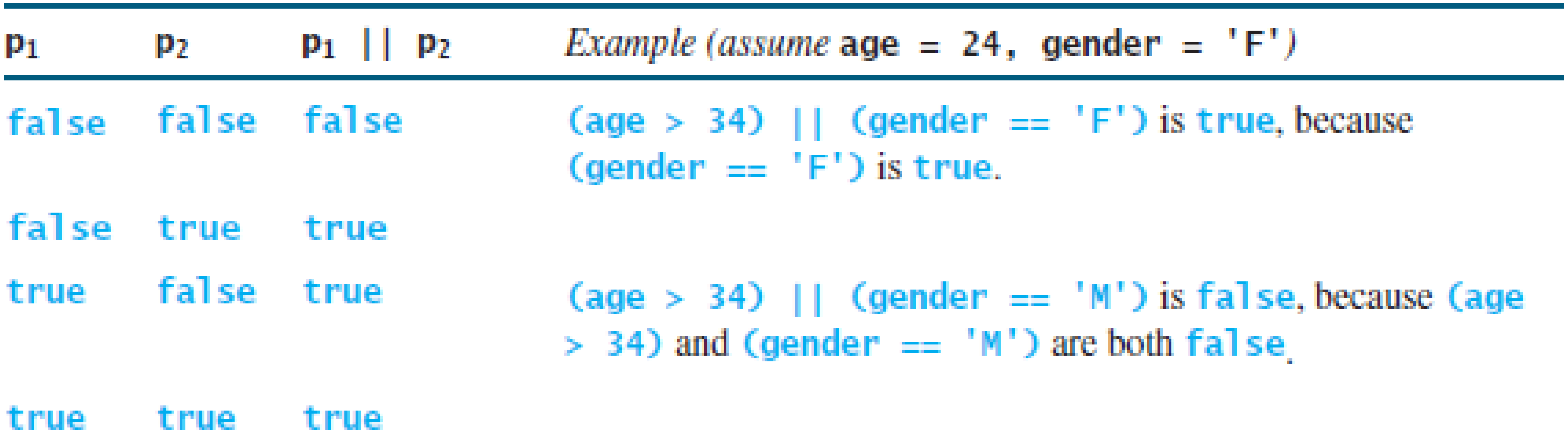
Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

Logical Operators (Cont.)

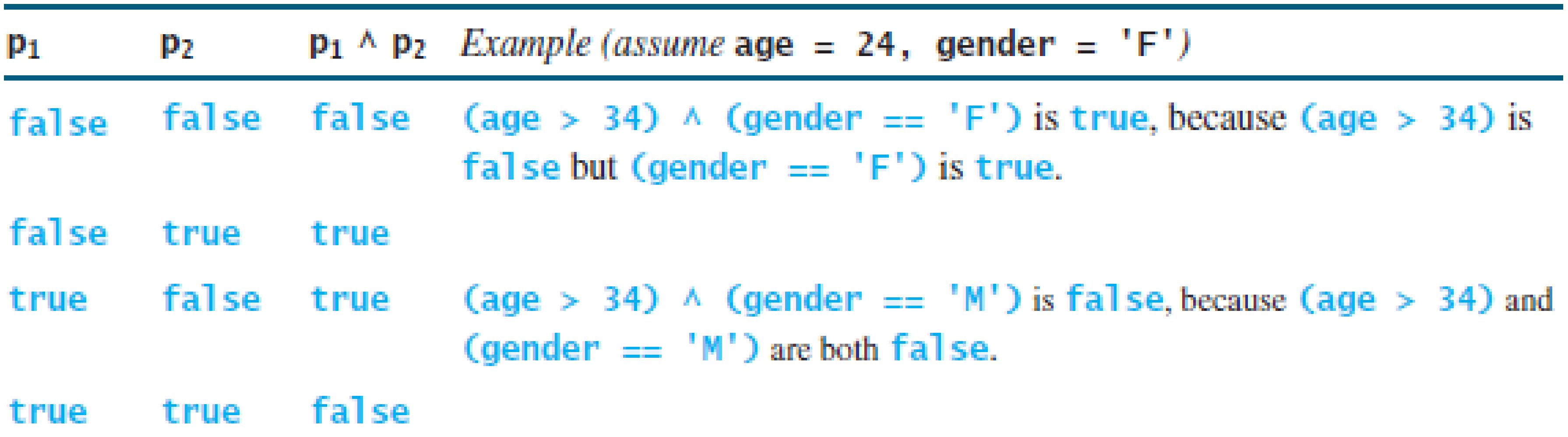
p	!p	*Example (assume* age = 24, gender = 'F'*)*
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(gender == 'M') is true, because (gender == 'M') is false.

p_1	p_2	p_1 && p_2	*Example (assume* age = 24, gender = 'F'*)*
false	false	false	(age > 18) && (gender == 'F') is true, because (age > 18) and (gender == 'F') are both true.
false	true	false	
true	false	false	(age > 18) && (gender != 'F') is false, because (gender != 'F') is false.
true	true	true	

Logical Operators (Cont.)



p_1	p_2	p_1 `||` p_2	*Example (assume* `age = 24, gender = 'F'`*)*
`false`	`false`	`false`	`(age > 34) || (gender == 'F')` is `true`, because `(gender == 'F')` is `true`.
`false`	`true`	`true`	
`true`	`false`	`true`	`(age > 34) || (gender == 'M')` is `false`, because `(age > 34)` and `(gender == 'M')` are both `false`.
`true`	`true`	`true`	



p_1	p_2	p_1 ^ p_2	*Example (assume* `age = 24, gender = 'F'`*)*
`false`	`false`	`false`	`(age > 34) ^ (gender == 'F')` is `true`, because `(age > 34)` is `false` but `(gender == 'F')` is `true`.
`false`	`true`	`true`	
`true`	`false`	`true`	`(age > 34) ^ (gender == 'M')` is `false`, because `(age > 34)` and `(gender == 'M')` are both `false`.
`true`	`true`	`false`	

switch Statements

- Nested if can be used to write code for multiple conditions.
  - However, it makes the program difficult to read.
- A *switch* statement simplifies coding for multiple conditions.
- A *switch* statement executes statements based on the value of a variable or an expression.

switch Statements (Cont.)

- The syntax for the switch statement is:

```
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;
}
```

Must yield a value of char, byte, short, int, or string

Constant expressions of the same type as the value of switch-expression

When the value in a case statement matches the value of the switch-expression, statements starting from this case are executed until either a break statement or the end of the switch statement is reached

Statements of the default case are executed when none of the specified cases matches the switch-expression.

An example of a Java program that uses a switch statement. This program prompts the user to enter a day of the week and then prints out a message corresponding to the day entered.

```java
import java.util.Scanner;

public class DayOfWeek {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter a day of the week (1 for Monday, 2 for Tuesday, etc.): ");
        int day = scanner.nextInt();

        switch (day) {
            case 1:
                System.out.println("Monday - Start of the work week!");
                break;
            case 2:
                System.out.println("Tuesday - Keep going!");
                break;
            case 3:
                System.out.println("Wednesday - Hump day!");
                break;
            case 4:
                System.out.println("Thursday - Almost there!");
                break;
            case 5:
                System.out.println("Friday - Weekend is coming!");
                break;
            case 6:
                System.out.println("Saturday - Enjoy your weekend!");
                break;
            case 7:
                System.out.println("Sunday - Rest well for the week ahead!");
                break;
            default:
                System.out.println("Invalid day! Please enter a number between 1 and 7.");
                break;
        }

        scanner.close();
    }
}
```

Explanation:
1. **Importing Scanner:** The program imports the Scanner class from the java.util package to read user input.

2. **Main Method:** The main method is the entry point of the program.

3. **Prompting User Input:** The program prompts the user to enter a number corresponding to a day of the week.

4. **Switch Statement:** The switch statement evaluates the variable day and executes the corresponding case block based on the user input.

5. **Cases and Default:** Each case corresponds to a day of the week and prints a specific message. The default case handles invalid input.

6. **Closing Scanner:** The scanner.close() method is called to close the scanner and prevent resource leaks.

Conditional Expressions

- A *conditional expression* evaluates an expression based on a condition.
- The syntax is:
  - *boolean-expression ? expression1 : expression2;*
  - The result of the conditional expression is *expression1* if *boolean-expression* is *true*, otherwise the result is *expression2*.
- Example:

  max = (num1 > num2) ? num1 : num2;

Operators Precedence Revisited

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

Loops

- A *loop* can be used to tell a program to execute statements *repeatedly*.
- Three types of loop statements:
  - While loops.
  - Do-while loops.
  - For loops.

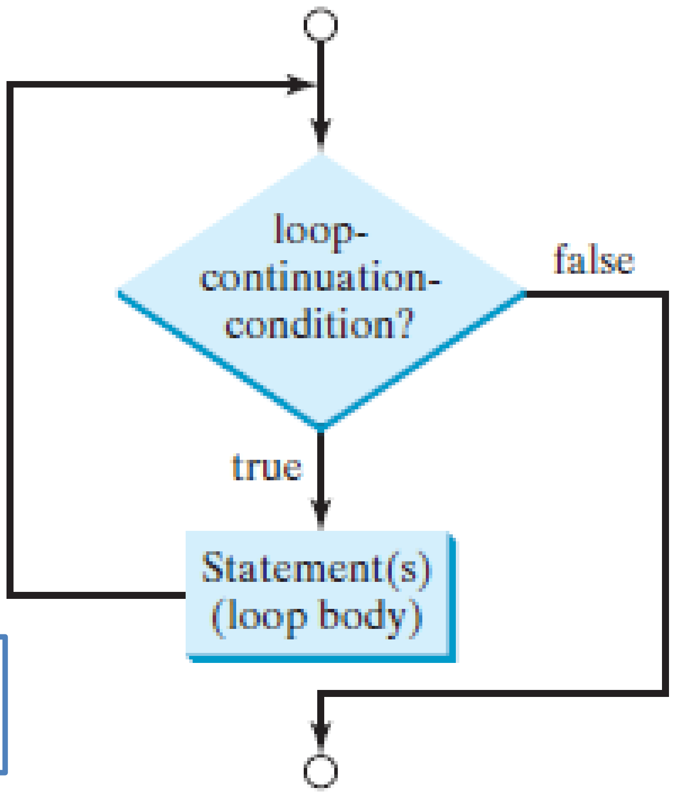
While Loops

- A *while* loop executes statements repeatedly while the condition is *true*.
- The syntax for the *while* loop is:

```
while (loop-continuation-condition){
    statement(s);
}
```

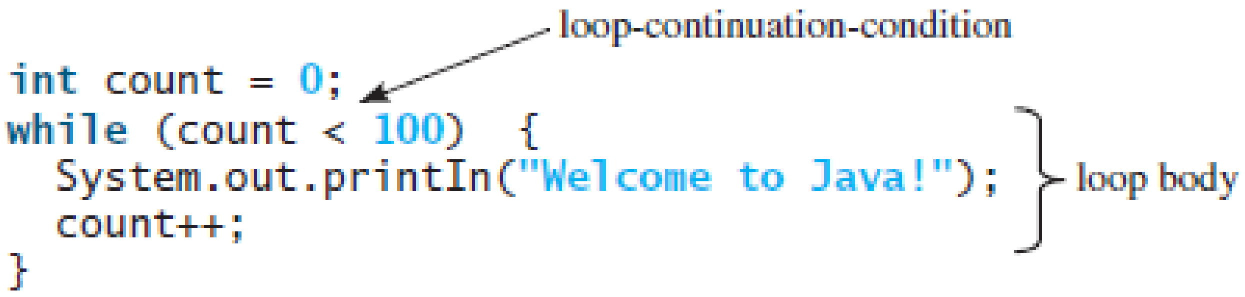
Loop body

Evaluated each time to determine whether to execute the loop body



While Loops (Cont.)

- A *while* loop that displays “Welcome to Java!” a hundred times:



```
int count = 0;
while (count < 100)  {
  System.out.printIn("Welcome to Java!");
  count++;
}
```

- Two types of loops:
  - *Counter-controlled* loops
    - A control variable is used to count the number of iterations.
  - *Sentinel-controlled* loops
    - A special input value signifies the end of the iterations.

While Loops (Examples)

```
int sum = 0, i = 1;
while (i < 10) {
   sum = sum + i;
   i++;
}
System.out.println("sum is " + sum); // sum is 45
```

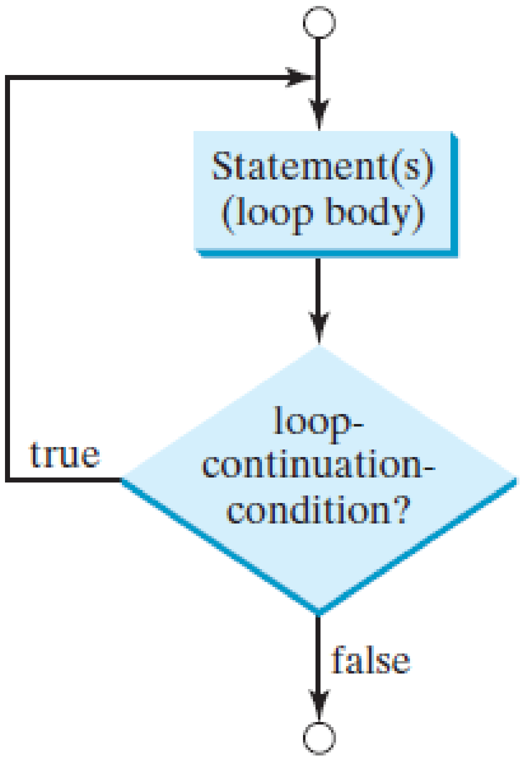
- Wrong implementation of a loop:

```
int sum = 0, i = 1;
while (i < 10) {
   sum = sum + i;
}
```

The do-While Loops

- Same as the *while* loop except that it executes the loop body first then checks the loop continuation condition.
- The syntax for the *do-while* loop:

```
do {
   statement(s);
} while (loop-continuation-condition);
```



```java
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 1;

        // While loop
        while (i <= 5) {
            System.out.println("Count: " + i);
            i++;
        }
    }
}
```

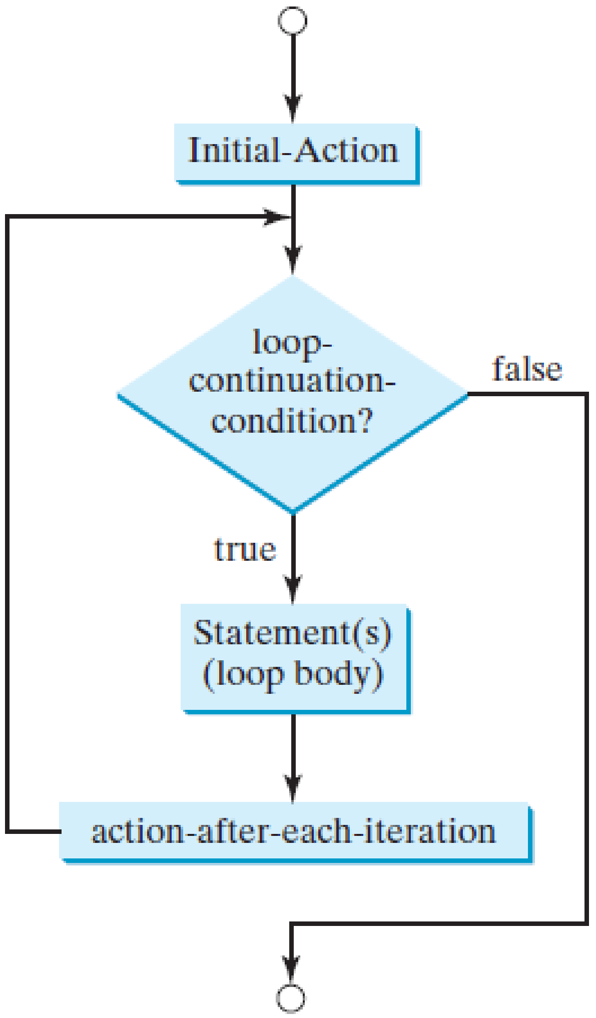
```
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int i = 1;

        // Do-while loop
        do {
            System.out.println("Count: " + i);
            i++;
        } while (i <= 5);
    }
}
```

The for Loop

- A for loop has a concise syntax for writing loops.
- The syntax for the for loop is:

```
for (initial-action;  loop-continuation-condition;
action-after-each-iteration){
    statement(s);
}
```

The for Loop (Cont.)



The for Loop (Cont.)

- A *for* loop that displays “Welcome to Java!” a hundred times:

```
for (int i = 0; i < 100; i++){
    System.out.println(“Welcome to Java!”);
}
```

- The *initial-condition* in a *for* loop can be a list of zero or more comma-separated variable declaration/assignment statements:

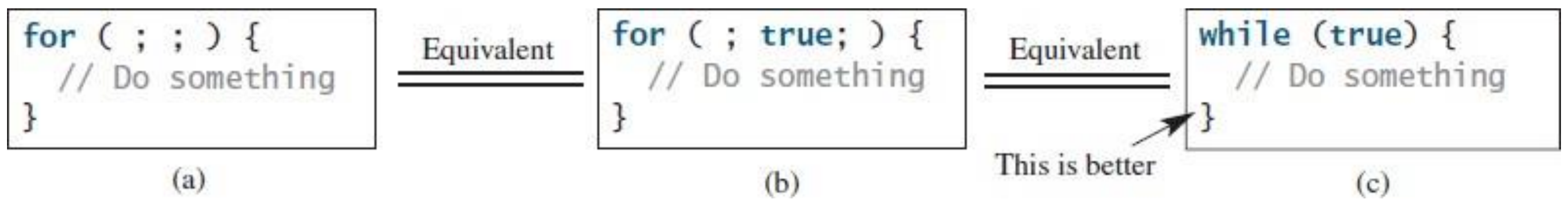
```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
        //Do something
}
```

- The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements:

```
for (int i = 1; i < 100; System.out.println(i), i++);
```

Infinite Loops

- Examples of *infinite* loops



```
for ( ; ; ) {
  // Do something
}
```

(a)

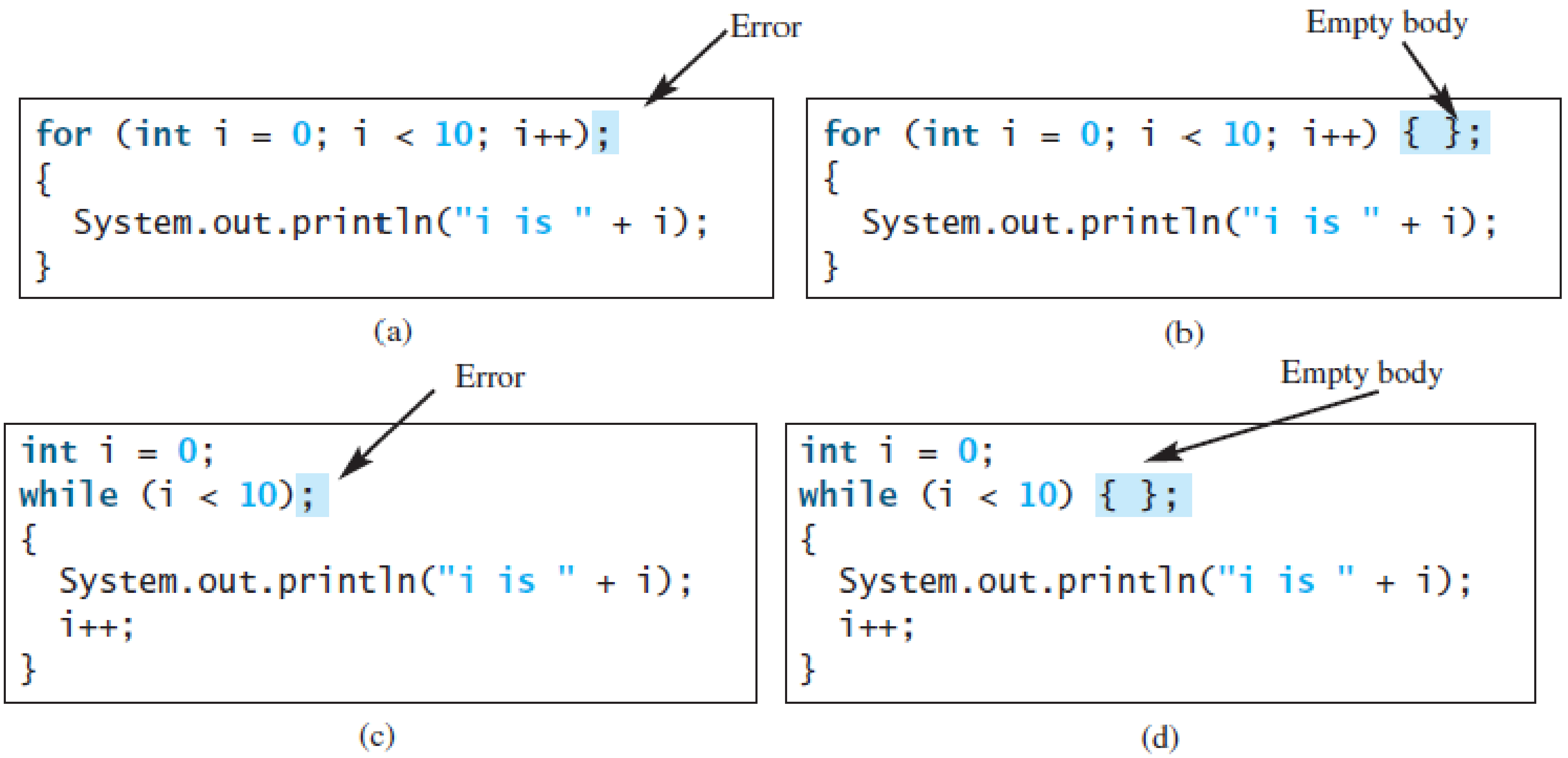
```
for ( ; true; ) {
  // Do something
}
```

(b)

```
while (true) {
  // Do something
}
```

(c)

Common Errors


Error
Empty body
Error
Empty body

```
for (int i = 0; i < 10; i++);
{
  System.out.println("i is " + i);
}
```

(a)

```
for (int i = 0; i < 10; i++) { };
{
  System.out.println("i is " + i);
}
```

(b)

```
int i = 0;
while (i < 10);
{
  System.out.println("i is " + i);
  i++;
}
```

(c)

```
int i = 0;
while (i < 10) { };
{
  System.out.println("i is " + i);
  i++;
}
```

(d)

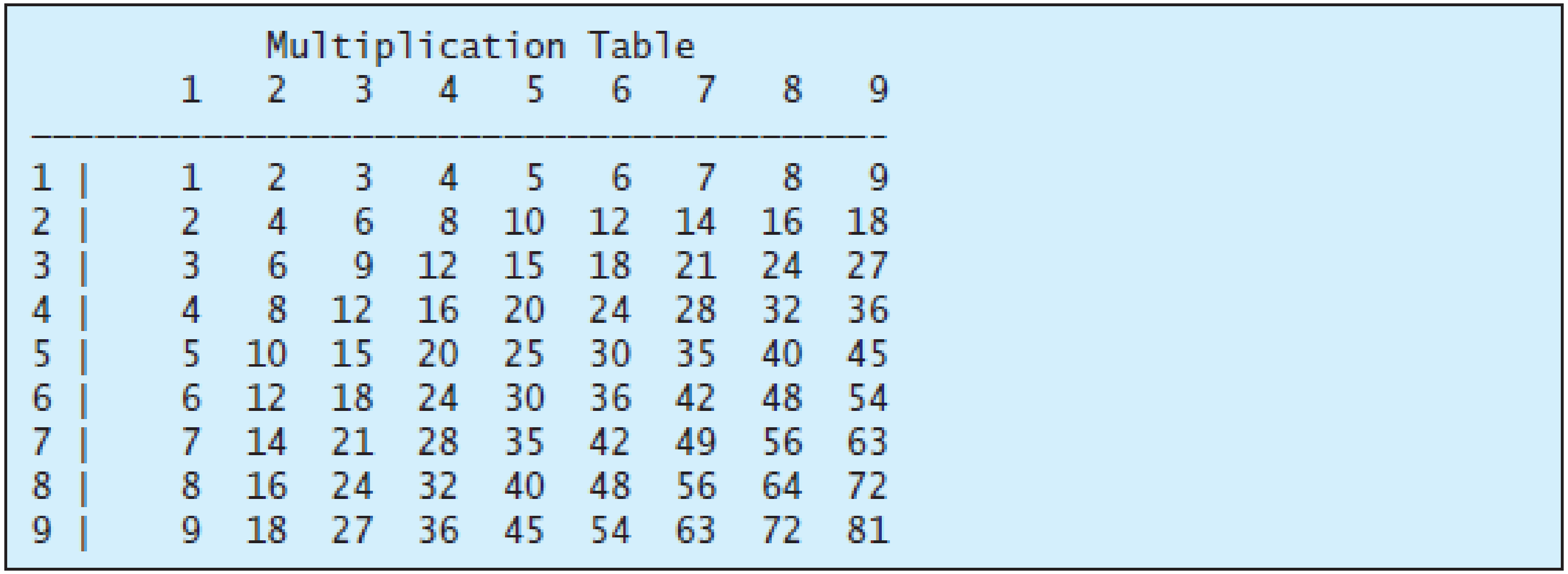
Nested Loops

- Nested loops consist of an *outer* loop and one or more *inner* loops.
- Each time, the outer loop is repeated, the inner loops are reentered.

Nested Loops (Example)

```java
public class MultiplicationTable {
  /** Main method */
  public static void main(String[] args) {
    // Display the table heading
    System.out.println("           Multiplication Table");

    // Display the number title
    System.out.print("    ");
    for (int j = 1; j <= 9; j++)
      System.out.print("   " + j);

    System.out.println("\n---------------------------------------");

    // Display table body
    for (int i = 1; i <= 9; i++) {
      System.out.print(i + " | ");
      for (int j = 1; j <= 9; j++) {
        // Display the product and align properly
        System.out.printf("%4d", i * j);
      }
      System.out.println();
    }
  }
}
```

Nested Loops (Example)



```
           Multiplication Table
       1   2   3   4   5   6   7   8   9
----------------------------------------
1 |    1   2   3   4   5   6   7   8   9
2 |    2   4   6   8  10  12  14  16  18
3 |    3   6   9  12  15  18  21  24  27
4 |    4   8  12  16  20  24  28  32  36
5 |    5  10  15  20  25  30  35  40  45
6 |    6  12  18  24  30  36  42  48  54
7 |    7  14  21  28  35  42  49  56  63
8 |    8  16  24  32  40  48  56  64  72
9 |    9  18  27  36  45  54  63  72  81
```

```java
// Yearly Calendar

import java.time.LocalDate;
import java.time.YearMonth;
import java.time.format.TextStyle;
import java.util.Locale;

public class YearlyCalendarExample {
/** Main method */
public static void main(String[] args) {
// Display the calendar for each month
for (int month = 1; month <= 12; month++) {
printMonth(month);
}
}

/** Print the calendar for a specific month */
public static void printMonth(int month) {
LocalDate date = LocalDate.of(LocalDate.now().getYear(), month, 1);
YearMonth yearMonth = YearMonth.of(LocalDate.now().getYear(), month);
int daysInMonth = yearMonth.lengthOfMonth();

// Display the month name and year
System.out.printf("\n%s %d\n", date.getMonth().getDisplayName(TextStyle.FULL,
Locale.ENGLISH), date.getYear());
// Display the days of the week
System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
// Print leading spaces for the first week
int dayOfWeek = date.getDayOfWeek().getValue();
if (dayOfWeek != 7) { // Adjust if the first day of the month is not
Sunday
for (int i = 0; i < dayOfWeek; i++) {
System.out.print("  ");
}
}
// Print the days of the month
for (int day = 1; day <= daysInMonth; day++) {
System.out.printf("%4d", day);
dayOfWeek++;
if (dayOfWeek == 7) {
dayOfWeek = 0;
System.out.println();
}
}
if (dayOfWeek != 0) {
System.out.println();
}
}
}
```

This code will print the calendar for the entire year, with each month's days correctly aligned according to the days of the week.

1. Imports: Added `import java.time.LocalDate;`, `import java.time.YearMonth;`, and other required imports to handle dates.
2. Main Method: Iterates through each month (from 1 to 12) and calls `printMonth` to print each month's calendar.
3. printMonth Method:
    - Uses `LocalDate` and `YearMonth` to get information about the current month and year.
    - Prints the month's name and year.
    - Prints the days of the week header.
    - Adds leading spaces for the first day of the month to align the days correctly.
    - Iterates through the days of the month, printing each day and breaking lines appropriately to maintain the calendar format.

Keywords *break* and *continue*

- The *break* and *continue* keywords provide additional controls in a loop.
- The *break* keyword is used in a loop to immediately terminate the loop.
- Example of using the *break* keyword:

```
for (int n=0, sum=0; n<20; n++){
        sum += n;
        if (sum >= 100) break;
}
```

Keywords *break* and *continue* (Cont.)

- The *continue* keyword is used in a loop to end the current iteration and program control goes to the end of the loop body.
- Example of using the *continue* keyword:

```
for (int n=0, sum=0; n<20; n++){
      if (n == 10 || n == 11) continue;
     sum += n;
}
```