Selection Statements (Condition)

Introduction

The program can decide which statements to execute based on a condition.

Java provides *selection statements*: statements that let you choose actions with alternative courses.

```
if (radius < 0) {
   System.out.println("Incorrect input");
}
else {
  area = radius * radius * 3.14159;
  System.out.println("Area is " + area);
}
```

Selection statements use conditions that are Boolean expressions.

A *Boolean expression* is an expression that evaluates to a *Boolean value*: true or false.

boolean Data Type

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

Java provides six *relational operators* (also known as *comparison operators*) to compare two values, shown in Table 3.1.

TABLE 3.1 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

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 and only if the condition is true.**

The syntax for a one-way if statement is:

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

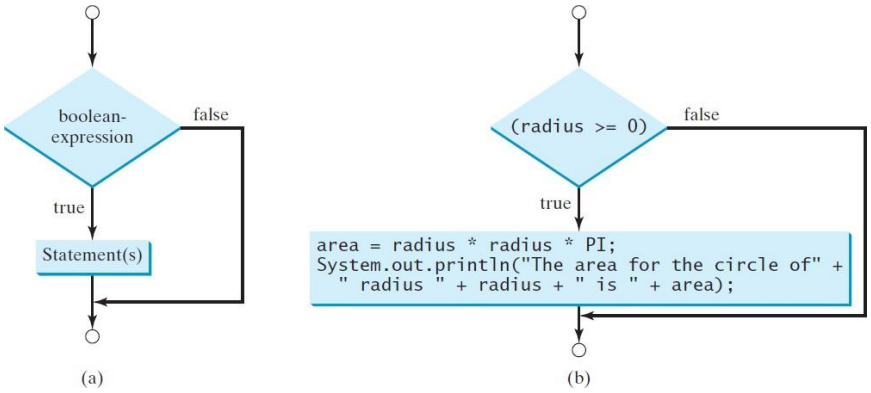


FIGURE 3.1 An if statement executes statements if the boolean-expression evaluates to true.

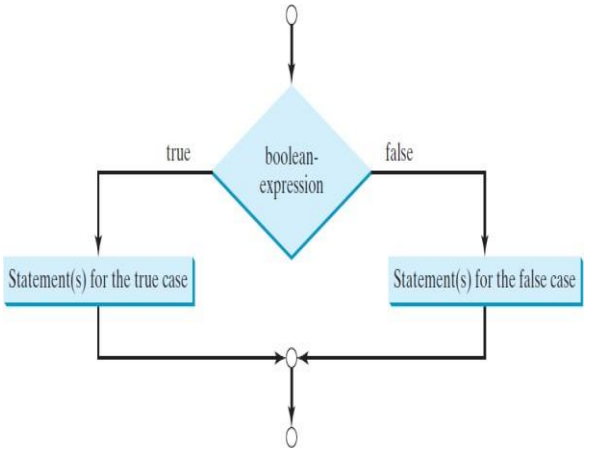
```
if (i > 0) {
   System.out.println("i is positive");
}
```

- **Two-Way if-else Statements**

An if-else statement decides the execution path based on whether the condition is true or false.

The syntax for a two-way if-else statement:

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



```
int number  =10;
if (number % 2 == 0)
   System.out.println(number + " is even.");
else
  System.out.println(number + " is odd.");
```

- **Nested if and Multi-Way if-else Statements**

An if statement can be inside another if statement to form a nested if statement.

The statement in an if or if-else statement can be any legal Java statement, including another if or if-else statement. The inner if statement is said to be *nested* inside the outer if statement. The inner if statement can contain another if statement; in fact, there is no limit to the depth of the nesting. For example, the following is a nested if statement:

```
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");
```

example:

```
double score  =67;

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");
```

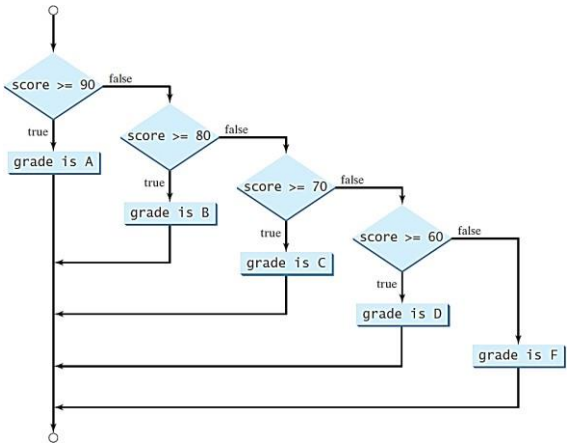


FIGURE 3.4 You can use a multi-way if-else statement to assign a grade.

You can use nested if statements to write a program that interprets body mass index.

Body Mass Index (BMI) is a measure of health based on height and weight. It can be calculated by taking your weight in kilograms and dividing it by the square of your height in meters.

$$\text{BMI} = \frac{m}{h^2}$$

BMI = body mass index
m = mass (in kilograms)
h = height (in meters)

BMI	Interpretation
BMI < 18.5	Underweight
18.5 ≤ BMI < 25.0	Normal
25.0 ≤ BMI < 30.0	Overweight
30.0 ≤ BMI	Obese

LISTING 3.4 ComputeAndInterpretBMI.java

```
1  import java.util.Scanner;
2
3  public class ComputeAndInterpretBMI {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6
7      // Prompt the user to enter weight in pounds
8      System.out.print("Enter weight in pounds: ");
9      double weight = input.nextDouble();                    input weight
10
11     // Prompt the user to enter height in inches
12     System.out.print("Enter height in inches: ");
13     double height = input.nextDouble();                    input height
14
15     final double KILOGRAMS_PER_POUND = 0.45359237; // Constant
16     final double METERS_PER_INCH = 0.0254; // Constant
17
18     // Compute BMI
19     double weightInKilograms = weight * KILOGRAMS_PER_POUND;
20     double heightInMeters = height * METERS_PER_INCH;
21     double bmi = weightInKilograms /                       compute bmi
22       (heightInMeters * heightInMeters);
23
24     // Display result
25     System.out.println("BMI is " + bmi);                   display output
26     if (bmi < 18.5)
27       System.out.println("Underweight");
28     else if (bmi < 25)
29       System.out.println("Normal");
30     else if (bmi < 30)
31       System.out.println("Overweight");
32     else
33       System.out.println("Obese");
34   }
35 }
```

Logical Operators

The logical operators !, &&, ||, and ^ can be used to create a compound Boolean expression.

TABLE 3.3 Boolean Operators

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

TABLE 3.4 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.

TABLE 3.5 Truth Table for Operator &&

p_1	p_2	p_1 && p_2	*Example (assume* age = 24, weight = 140*)*
false	false	false	
false	true	false	(age > 28) && (weight <= 140) is true, because (age > 28) is false.
true	false	false	
true	true	true	(age > 18) && (weight >= 140) is true, because (age > 18) and (weight >= 140) are both true.

TABLE 3.6 Truth Table for Operator ||

p_1	p_2	p_1 || p_2	*Example (assume* age = 24, weight = 140*)*
false	false	false	(age > 34) || (weight >= 150) is false, because (age > 34) and (weight >= 150) are both false.
false	true	true	
true	false	true	(age > 18) || (weight < 140) is true, because (age > 18) is true.
true	true	true	

TABLE 3.7 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) and (weight > 140) are both false.
false	true	true	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	false	true	
true	true	false	

Switch Statements

A switch statement executes statements based on the value of a variable or an expression.

```
switch (status) {
  case 0:  compute tax for single filers;
           break;
  case 1:  compute tax for married jointly or qualifying widow(er);
           break;
  case 2:  compute tax for married filing separately;
           break;
  case 3:  compute tax for head of household;
           break;
  default: System.out.println("Error: invalid status");
           System.exit(1);
}
```

The flowchart of the preceding switch statement is shown in Figure 3.5.

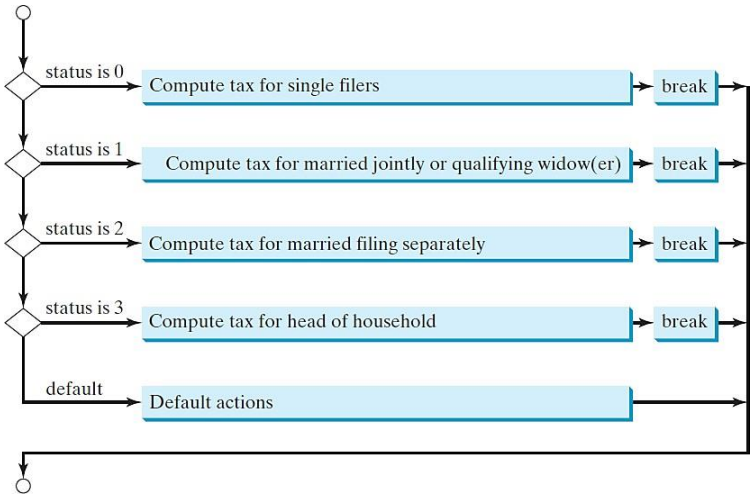


FIGURE 3.5 The switch statement checks all cases and executes the statements in the matched case.

Here is the full syntax for the switch statement:

```
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 switch statement observes the following rules:

- **The switch-expression must yield a value of char, byte, short, int, or String type and must always be enclosed in parentheses.**
  - **The value1, . . ., and valueN must have the same data type as the value of the switchexpression.**

  Note that value1, . . ., and valueN are constant expressions, meaning that they cannot contain variables, such as 1 + x.
- **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.**
- **The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.**
  - **The keyword break is optional. The break statement immediately ends the switch statement.**

Loops

Introduction

A loop can be used to tell a program to execute statements repeatedly.

The while Loop

A while loop executes statements repeatedly while the condition is true. write a loop in the following common form:

```
i = initialValue;  // Initialize loop control variable
while (i < endValue)
  // Loop body
  ...
  i++; // Adjust loop control variable
}
```

The syntax for the while loop is:

```
while (loop-continuation-condition) {
  // Loop body
  Statement(s);
}
```

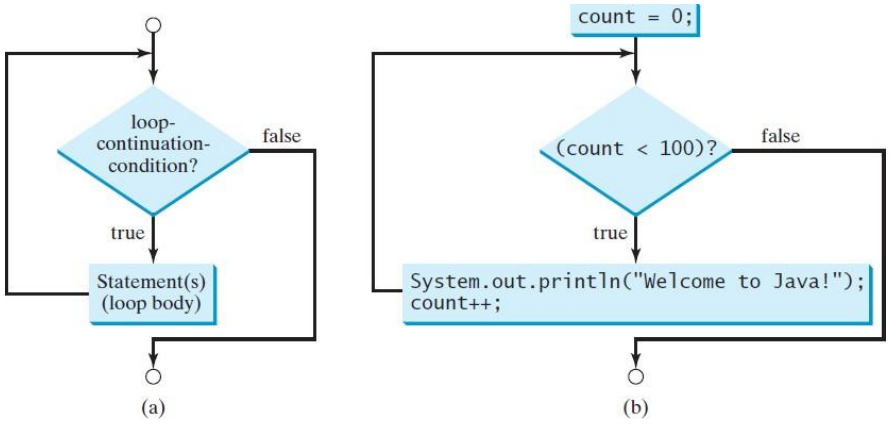


FIGURE 5.1 The while loop repeatedly executes the statements in the loop body when the loop-continuation-condition evaluates to true.

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

loop-continuation-condition

loop body

For example:

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

Write class (program) to find the great common divisor (ايجاد القاسم المشترك الاكبر)

LISTING 5.9 GreatestCommonDivisor.java

```
1  import java.util.Scanner;
2
3  public class GreatestCommonDivisor {
4    /** Main method */
5    public static void main(String[] args) {
6      // Create a Scanner
7      Scanner input = new Scanner(System.in);
8
9      // Prompt the user to enter two integers
10     System.out.print("Enter first integer: ");
11     int n1 = input.nextInt();
12     System.out.print("Enter second integer: ");
13     int n2 = input.nextInt();
14
15     int gcd = 1; // Initial gcd is 1
16     int k = 2; // Possible gcd
17     while (k <= n1 && k <= n2) {
18       if (n1 % k == 0 && n2 % k == 0)
19         gcd = k; // Update gcd
20       k++;
21     }
22
23     System.out.println("The greatest common divisor for " + n1 +
24       " and " + n2 + " is " + gcd);
25   }
26 }
```

The do-while Loop

A do-while loop is the same as a while loop except that it executes the loop body first and then checks the loop continuation condition.

```
do {
  // Loop body;
  Statement(s);
} while (loop-continuation-condition);
```

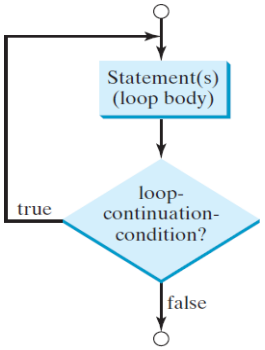


FIGURE 5.2 The do-while loop executes the loop body first, then checks the loopcontinuation- condition

to determine whether to continue or terminate the loop.

The for Loop

A for loop has a concise syntax for writing loops.

A for loop can be used to simplify the preceding loop as:

```
for (i = initialValue; i < endValue; i++)
  // Loop body
  ...
}
```

In general, the syntax of a for loop is:

```
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
  // Loop body;
    Statement(s);
}
```

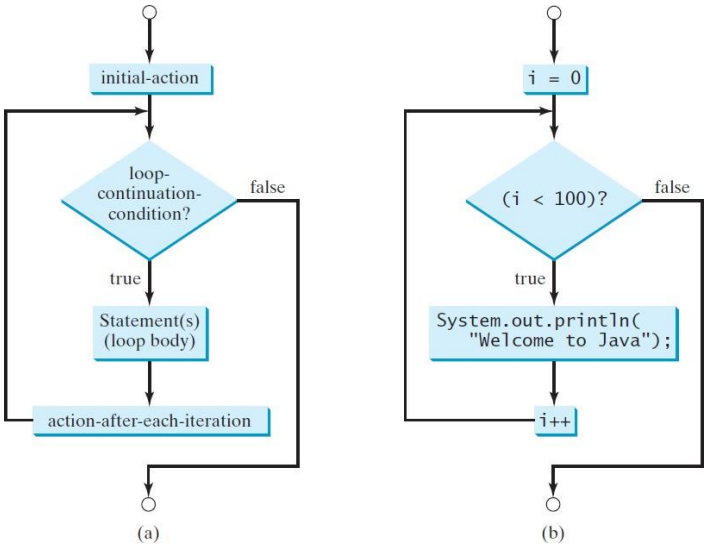
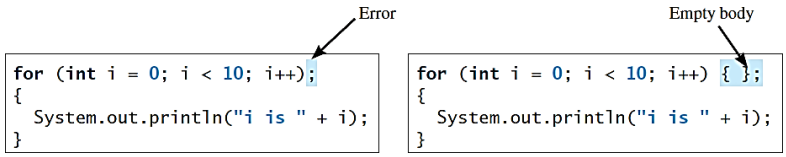


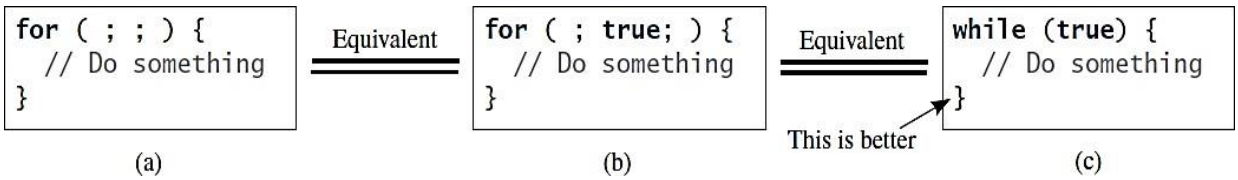
FIGURE 5.3 A for loop performs an initial action once, then repeatedly executes the statements in the loop body, and performs an action after an iteration when the loop-continuation-condition evaluates to true.

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

- **Common Errors**



- **Infinite loops**



Q1:Write class (program) ask user to enter two numbers and mathematic operation reactively. Based on entered operation the program perform either summation, subtraction, multiplication, division.

```
import java.util.Scanner;

public class SimpleCalculater {
public static void main(String[] args) {
Scanner in = new Scanner(System.in);
 System.out.println("plz Enter number1");
double number1=in.nextDouble();
 System.out.println("plz Enter number1");
double number2=in.nextDouble();
System.out.println("Choose Operation  +,  -,  *,  /,  %  ");
char ch = in.next().charAt(0);
if(ch == '+')
System.out.println(number1 + number2);
else if(ch == '-')
System.out.println(number1 - number2);
else if(ch == '/')
System.out.println(number1 / number2);
else if(ch == '*')
System.out.println(number1 * number2);
else if(ch == '%')
System.out.println(number1 % number2);
   }
}
```