CS3231
Object Oriented Programming I

Name: _________________________________ (   ) Date: ________________

Chapter 5: Conditionals

Lesson Objectives

After completing the lesson, the student will be able to

- *declare boolean variables and*
- *write Boolean expressions using relational operators*
- *write Boolean expressions using logical operators*
- *implement selection control using one-way if statements*
- *implement selection control using two-way if-else statements*
- *implement selection control using nested if and multi-way if statements*
- *implement selection control using switch statements*
- *examine the rules governing operator precedence and associativity*

5.1 Control Structure

All programs could be written in terms of only three control structure: the **sequence structure**, the **selection structure** and the **repetition structure**. See Figure 5.1

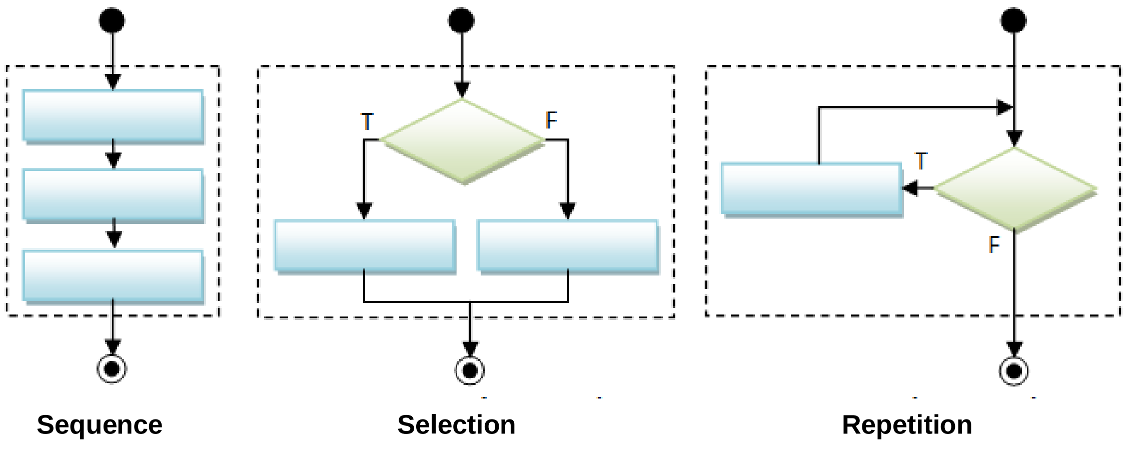


Figure 5.1 Control Structure

5.1.1 Sequential

The sequence structure is built into Java. A program is a sequence of instructions. Unless directed otherwise, **the computer executes Java statements one after the other in the order in which they are written**—that is, from top to bottom in sequence.

5.1.2 Selection

The selection structure is used to test a condition. A sequence of statements is executed depending on whether or not the condition it true or false. This means the program chooses between two or more alternative paths. **Condition refers to any expression or value that returns a Boolean value, meaning true or false.**

Java has several types of selection statements: one-way `if` statements, two-way `if-else` statements, nested `if` statements, multi-way `if-else` statements, `switch` statements, and conditional expressions.

5.1.3 Repetition

The repetition structure (iteration) repeatedly executes a series of statements as long as the condition is true. The condition may be predefined or open-ended. **`while`, `do..while` and `for` loop are the three types of iteration statements**. A loop can either be event controlled or counter controlled. **An event-controlled loop executes a sequence of statements till an event occurs while a counter-controlled loop executes the statements a predetermined number of times.**

5.2 The Boolean Expressions

Selection statements use conditions that are Boolean expressions. A Boolean expression evaluates to a Boolean value. Before we can proceed to write selection statements we need to know two category of operators that would evaluate to Boolean expressions, namely; **relational operators** and **logical operators**.

5.2.1 Relational Operators

Java provides six relational operators (also known as comparison operators), shown in Table 5.1, which can be used to compare two values.

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

Table 5.1 Relational Operators

5.2.1 Logical Operators

The logical operators, `!` , `&&`, `||`, and `^` can be used to create a compound Boolean expression. Table 5.2 lists the Boolean operators.

Operator	Name	Description
!	NOT	logical negation
&&	AND	logical conjunction
||	OR	logical disjunction
^	EXCLUSIVE OR	logical exclusion

Table 5.2 Logical Operators

Sometimes, whether a statement is executed is determined by a combination of several conditions. You can use logical operators to combine these conditions to form a compound Boolean expression. Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.

Table 5.3 defines the NOT operator. **The NOT operator negates true to false and false to true.**

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 5.3 NOT Operator

Table 5.4 defines the AND operator. **The AND of two Boolean operands is true if and only if both operands are true.**

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~~ f, 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 5.4 AND Operator

Table 5.5 defines the OR operator. **The OR of two Boolean operands is true if at least one of the operands is true.**

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 5.5 OR Operator

Table 5.6 defines the EXCLUSIVE OR ^ operator. **The EXCLUSIVE OR ^ of two Boolean operands is true if and only if the two operands have different Boolean values**. Note that `p1 ^ p2` is the same as `p1!= p2`.

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	

Table 5.6 EXCLUSIVE OR Operator

5.3 `if` Statement

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);
}
```

The flowchart in Figure 5.1(a) illustrates how Java executes the syntax of an `if` statement. **A flowchart is a diagram that describes an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows.** Process operations are represented in these boxes, and arrows connecting them represent the flow of control. A diamond box denotes a Boolean condition and a rectangle box represents statements.

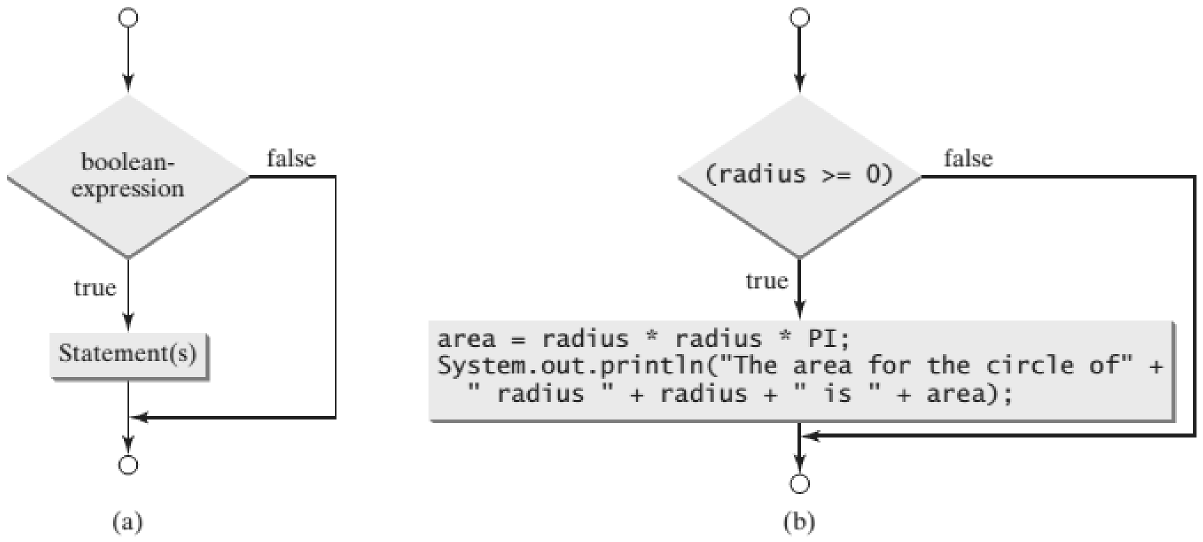

boolean-expression
false
true
Statement(s)
(a)
(radius >= 0)
false
true
area = radius * radius * PI;
System.out.println("The area for the circle of" +
" radius " + radius + " is " + area);
(b)

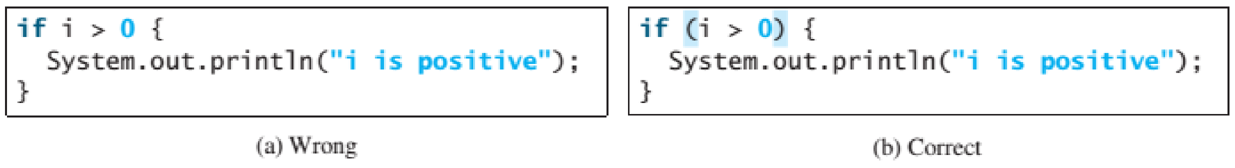
Figure 5.1 An `if` statement executes statements if the boolean-expression evaluates to true.

If the boolean-expression evaluates to true, the statements in the block are executed. As an example, see the following code:

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

The flowchart of the preceding statement is shown in Figure 5.1(b). If the value of radius is greater than or equal to 0, then the area is computed and the result is displayed; otherwise, the two statements in the block will not be executed.

The boolean-expression is enclosed in parentheses. For example, the code in Figure 5.2(a) is wrong. It should be corrected, as shown in Figure 5.2 (b).



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

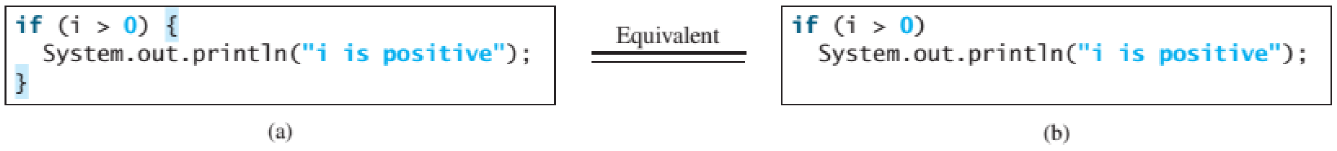
(a) Wrong

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

(b) Correct

Figure 5.2 Boolean Expression Must Be Enclosed In Parentheses

The block braces can be omitted if they enclose a single statement. Figure 5.3 shows the statements are equivalent.



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

(a)

Equivalent

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

(b)

Figure 5.3 Braces Can Be Omitted If There Is Only A Single Statement

PROGRAM 5-1 prompts the user to enter an integer (lines 8–9). If the number is a multiple of 5, the program displays "Hi Five" (lines 9–10). If the number is divisible by 2, it displays "Hi Even" (lines 11–12).

PROGRAM 5-1

```
1   import java.util.Scanner;
2   public class CheckEvenOrFive {
3    public static void main(String[] args) {
4      Scanner input = new Scanner(System.in);
5      System.out.println("Enter an integer: ");
6      int number = input.nextInt();
7
8      if (number % 5 == 0)  // check multiples of 5
9         System.out.println("Hi Five");
10
11     if (number % 2 == 0) // check even
12        System.out.println("Hi Even");
13   }
14  }
```

PROGRAM OUTPUT 1

```
Enter an integer: 4
Hi Even
```

PROGRAM OUTPUT 2

```
Enter an integer: 30
Hi Five
Hi Even
```

5.4 `if..else` Statement

An `if-else` statement executes statements for the true case if the Boolean expression evaluates to true; otherwise, statements for the false case are executed.

A one-way `if` statement performs an action if the specified condition is true. If the condition is false, nothing is done. You can use a two-way `if-else` statement, if you want to take alternative actions when the condition is false. The actions that a two-way `if-else` statement specifies differ based on whether the condition is true or false.

Here is 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;
}
```

The flowchart of the statement is shown in Figure 5.4.

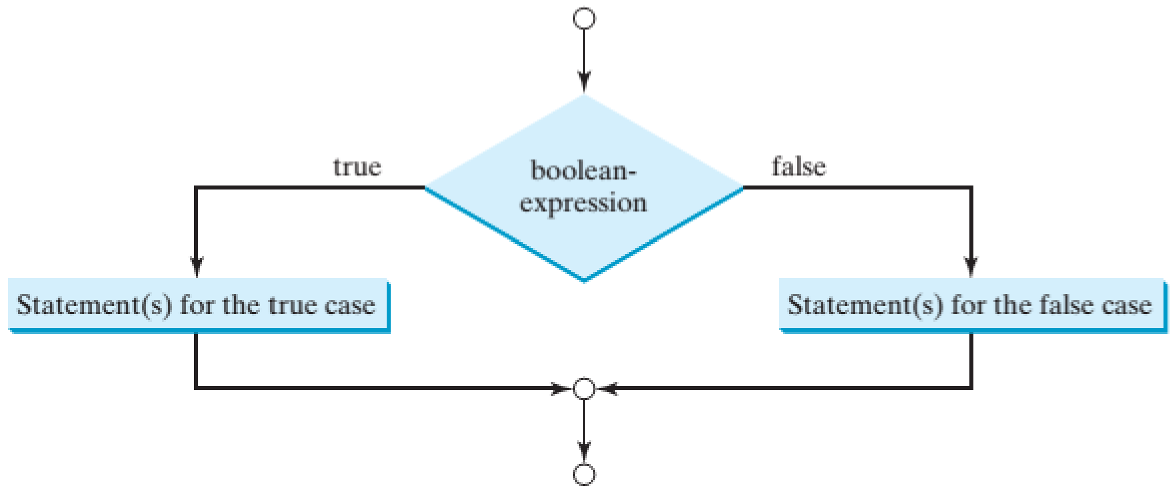

true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

Figure 5.4 An if-else statement executes statements for the true case if the Boolean expression evaluates to true; otherwise, statements for the false case are executed.

If the boolean-expression evaluates to true, the statement(s) for the true case are executed; otherwise, the statement(s) for the false case are executed. PROGRAM 5-2 illustrates the use of `if-else` statement.

PROGRAM 5-2

```
import java.util.Scanner;
public class CircleArea {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Enter an radius: ");
   int radius = input.nextInt();
   if (radius >= 0) {
     double area = radius * radius * 3.14159;
     System.out.println("The area for the circle of radius " +
     radius + " is " + area);
   }
   else {
     System.out.println("Negative input");
 }
}
```

PROGRAM OUTPUT 1

```
Enter an radius: -5
Negative input
```

PROGRAM OUTPUT 2

```
Enter an radius: 3
The area for the circle of radius 3 is 28.27431
```

If radius >= 0 is true, area is computed and displayed; if it is false, the message "`Negative input`" is displayed. As usual, the braces can be omitted if there is only one statement within them. The braces enclosing the `System.out.println("Negative input")` statement can therefore be omitted in the preceding example.

PROGRAM 5-3 is another example of using the `if-else` statement. The example checks whether a number is even or odd, as follows:

PROGRAM 5-3

```
import java.util.Scanner;
public class CircleArea {
 public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Enter a number: ");
      int number = input.nextInt();

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

PROGRAM OUTPUT 1

```
Enter a number: 7
7 is odd.
```

PROGRAM OUTPUT 2

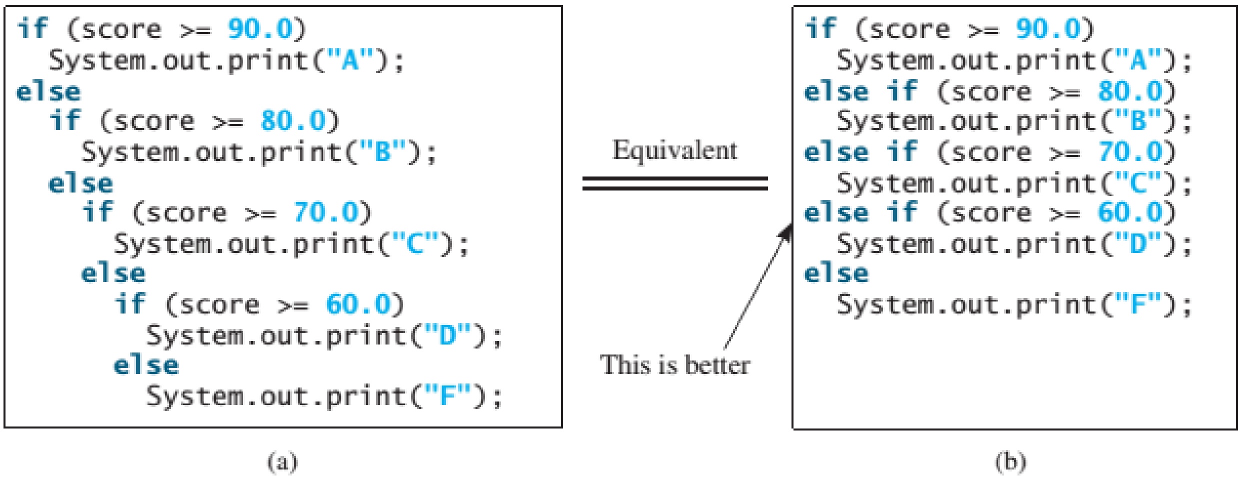
```
Enter a number: 8
8 is even.
```

5.5 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");
```

The `if(j > k)` statement is nested inside the `if(i > k)` statement. The nested `if` statement can be used to implement multiple alternatives. The statement given in Figure 5.5a, for instance, prints a letter grade according to the score, with multiple alternatives.



```
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");
```

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");
```

(a)
(b)

Figure 5.5 A preferred format for multiple alternatives is shown in (b) using a multi-way if-else statement

Figure 5.6 shows the execution of this `if` statement proceeds. The first condition (score >= 90.0) is tested. If it is true, the grade is A. If it is false, the second condition (score >= 80.0) is tested. If the second condition is true, the grade is B. If that condition is false, the third condition and the rest of the conditions (if necessary) are tested until a condition is met or all of the conditions prove to be false. If all of the conditions are false, the grade is F. Note that a condition is tested only when all of the conditions that come before it are false.

The `if` statement in Figure 5.5(a) is equivalent to the `if` statement in Figure 5.5(b). In fact, Figure 5.5(b) is the preferred coding style for multiple alternative if statements. **This style, called multi-way `if-else` statements, avoids deep indentation and makes the program easy to read.**

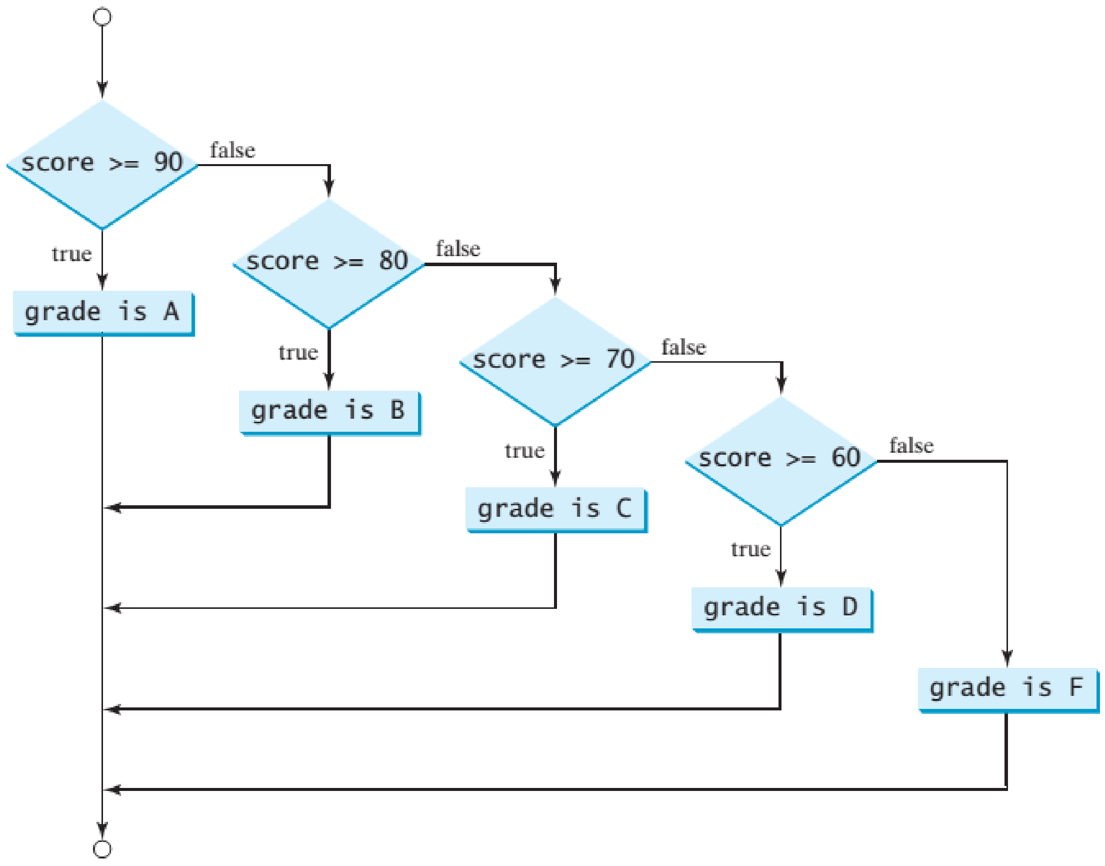


Figure 5.6 You can use a multi-way if-else statement to assign a grade.

5.6 `switch` Statement

A `switch` statement executes statements based on the value of a variable or an expression enclosed within parentheses. Here is the 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;
}
```

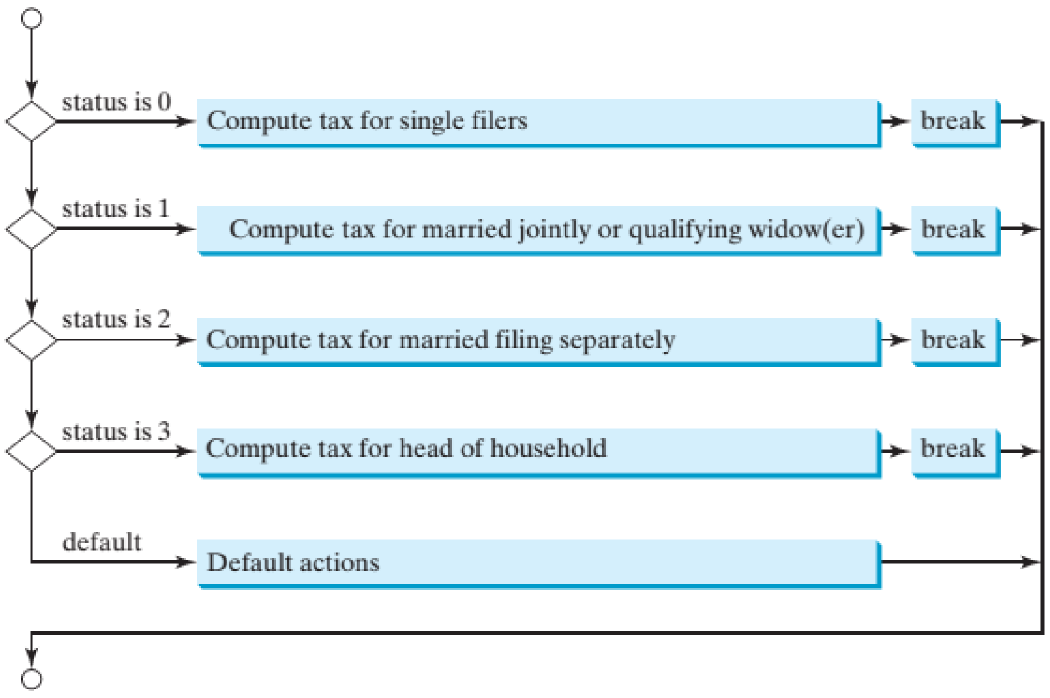
The `switch` statement observes the following rules:

- The switch-expression must yield a value of char, byte, short, int, or String type and must always be enclosed in parentheses.
- The `value1`, `value2`, .. and `valueN` must have the same data type as the value of the switch expression. Note that `value1`, `value2`, .. 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.

Assuming, you are to write a program to compute personal income tax. There are four cases for computing taxes, which depend on the value of status. The filing status can be determined using if statements outlined as follows:

```
if (status == 0 || status == 5) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married filing jointly or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married filing separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

The logic expressed above is perfectly fine. However, overuse of `if` or nested `if` statements makes a program difficult to read. Java provides a `switch` statement to simplify coding for multiple conditions. The sample flowchart for computing tax using `switch` statement is shown in Figure 5.7.



status is 0
Compute tax for single filers
break
status is 1
Compute tax for married jointly or qualifying widow(er)
break
status is 2
Compute tax for married filing separately
break
status is 3
Compute tax for head of household
break
default
Default actions

Figure 5.7 You can use a multi-way if-else statement to assign a grade

You can write the following `switch` statement to replace the `if` statement.

PROGRAM 5-4

```java
import java.util.*;
public class ScannerInput {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Enter status: ");
   int status = input.nextInt();
   switch (status){
       case 0: case 5: System.out.println("Compute tax for single
filers");
               break;
       case 1: System.out.println("Compute tax for married jointly or "
                          + "qualifying widow(er)");
               break;
       case 2: System.out.println("Compute tax for married filing "
                          + "separately");
               break;
       case 3: System.out.println("Compute tax for head of household");
               break;
       default: System.out.println("Error: invalid status");
               System.exit(1);

    }
  }
}
```

PROGRAM OUTPUT 1

```
Enter status: 2
Compute tax for married filing separately
```

PROGRAM OUTPUT 2

```
Enter status: 7
Error: invalid status
```

This statement checks to see whether the status matches the value 0, 1, 2, or 3, in that order. If matched, the corresponding tax is computed; if not matched, a message is displayed.

Note: Do not forget to use a `break` statement when one is needed. **Once a case is matched, the statements starting from the matched case are executed until a `break` statement or the end of the switch statement is reached. This is referred to as fall-through behavior**. PROGRAM 5-5 displays Weekdays for day of 1 to 5 and Weekends for day 0 and 6.

PROGRAM 5-5

```
import java.util.*;
public class ScannerInput {
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.print("Enter day: ");
   int day = input.nextInt();
   switch (day) {
      case 1:
      case 2:
      case 3:
      case 4:
      case 5: System.out.println("\nWeekday"); break;
      case 0:
      case 6: System.out.println("Weekend"); break;
      default: System.out.println("\nError: invalid status");
               System.exit(1);
   }
  }
}
```

PROGRAM OUTPUT 1

```
Enter day: 4
Weekday
```

PROGRAM OUTPUT 2

```
Enter day: 6
Weekend
```

5.7 Short Circuit Evaluation of Boolean Operators

The && and || logical operators are computed using short-circuit evaluation. In other words, **logical expressions are evaluated from left to right, and evaluation stops as soon as the truth value is determined**. **When an && or || is evaluated and the first condition is false, the second condition is not evaluated**, because it does not matter what the outcome of the second test is. **This process is called short-circuit evaluation, also known as lazy evaluation.**

PROGRAM 5-6 shows an example of a short circuit evaluation. When a number that is entered is not even, the second condition `number % 3 ==` 0 in the Boolean expression will be skipped and not evaluated. There is no point in evaluating further because the whole expression will evaluate to a false value.

PROGRAM 5-6

```
import java.util.*;
public class ScannerInput {
 public static void main(String[] args){
   Scanner input = new Scanner(System.in);
   System.out.print("Enter an integer: ");
   int number = input.nextInt();
   if (number % 2 == 0 && number % 3 == 0)
     System.out.println(number + " is divisible by 2 and 3.");
   else
     System.out.println(number + " is not divisible by 2 and 3");
 }
}
```

5.8 Conditional Expressions

A conditional expression evaluates an expression based on a condition. You might want to assign a value to a variable that is restricted by certain conditions. Conditional expressions are in a completely different style, with no explicit if in the statement.

The syntax is:

```
boolean-expression ? expression1 : expression2;
```

The result of this conditional expression is expression1 if boolean-expression is true; otherwise the result is expression2. For example, the following statement assigns 1 to `y` if `x` is greater than 0, and -1 to `y` if `x` is less than or equal to 0.

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

Alternatively, as in the following example, you can use a conditional expression (also known as ternary operator because they uses 3 operators) to achieve the same result.

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

Java supports this type of old programming style but you are advice not to use it as it makes code difficult to read.

5.9 Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated. The precedence rule defines precedence for operators. Operators are listed in decreasing order of precedence from top to bottom. The logical operators have lower precedence than the relational operators and the relational operators have lower precedence than the arithmetic operators. Operators with the same precedence appear in the same group.

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

$$a - b + c - d \quad \overset{\text{is equivalent to}}{=\!=\!=\!=} \quad ((a - b) + c) - d$$

Assignment operators are right associative. Therefore, the expression

$$\texttt{a = b += c = 5} \quad \overset{\text{is equivalent to}}{=\!=\!=\!=\!=\!=} \quad \texttt{a = (b += (c = 5))}$$

Suppose a, b, and c are 1 before the assignment; after the whole expression is evaluated, a becomes 6, b becomes 6, and c becomes 5. Note that left associativity for the assignment operator would not make sense.

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)
	`<`, `<=`, `>`, `>=` (Relational)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

For a complete list of Java operators and their precedence refer to APPENDIX B-2.

5.10 Common Errors (Optional Reading)

Forgetting necessary braces, ending an `if` statement in the wrong place, mistaking == for =, and dangling else clauses are common errors in selection statements.

Common Error 1: Forgetting Necessary Braces

The braces can be omitted if the block contains a single statement. However, forgetting the braces when they are needed for grouping multiple statements is a common programming error. If you modify the code by adding new statements in an if statement without braces, you will have to insert the braces. For example, the following code in (a) is wrong. It should be written with braces to group multiple statements, as shown in (b).

```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

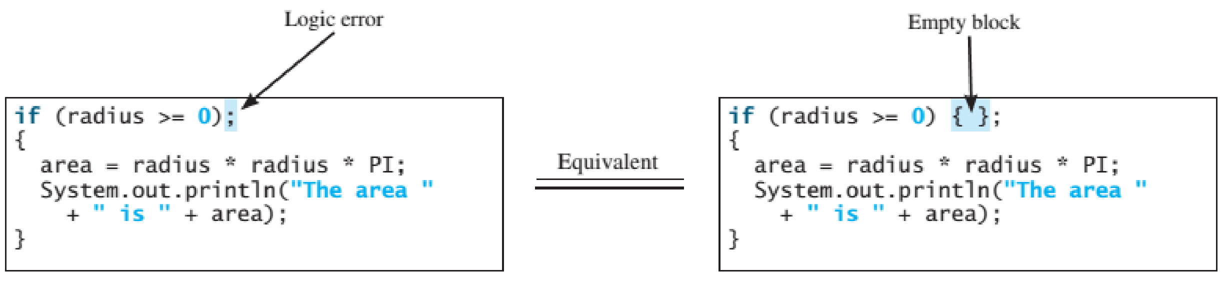
(a) Wrong

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

(b) Correct

Common Error 2: Wrong Semicolon at the if Line

Adding a semicolon at the end of an if line, as shown in (a) below, is a common mistake.



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

(a)

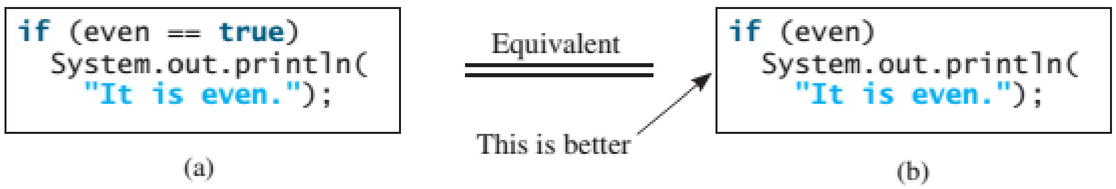
```java
if (radius >= 0) { };
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b)

This mistake is hard to find, because it is neither a compile error nor a runtime error; it is a logic error. The code in (a) is equivalent to that in (b) with an empty block. This error often occurs when you use the next-line block style. Using the end-of-line block
style can help prevent this error.

Common Error 3: Redundant Testing of Boolean Values

To test whether a boolean variable is true or false in a test condition, it is redundant to use the equality testing operator like the code in (a):



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

(a)

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

(b)

Instead, it is better to test the boolean variable directly, as shown in (b). Another good reason for doing this is to avoid errors that are difficult to detect. Using the = operator instead of the == operator to compare the equality of two items in a test condition is a common error. It could lead to the following erroneous statement:

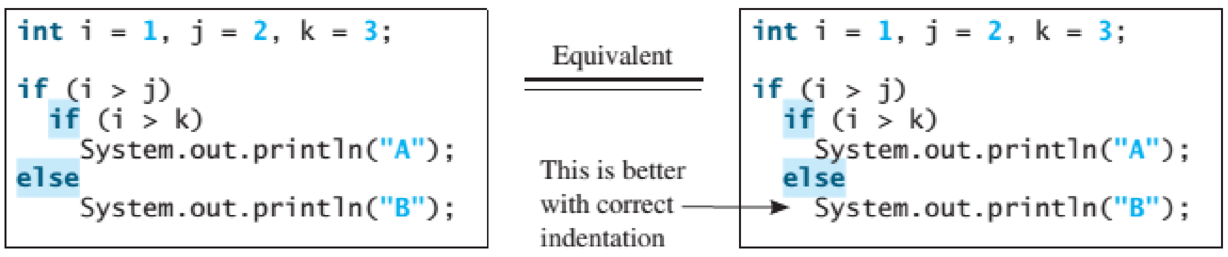
```java
if (even = true)
 System.out.println("It is even.");
```

This statement does not have compile errors. It assigns true to even, so that even is always true.

Common Error 4: Dangling else Ambiguity

The code in (a) below has two if clauses and one else clause. Which if clause is matched by the else clause? The indentation indicates that the else clause matches the first if clause.

However, the else clause actually matches the second if clause. This situation is known as the dangling else ambiguity. The else clause always matches the most recent unmatched if clause in the same block. So, the statement in (a) is equivalent to the code in (b).



```java
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

```java
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Since (i > j) is false, nothing is displayed from the statements in (a) and (b). To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1, j = 2, k = 3;
if (i > j) {
if (i > k)
 System.out.println("A");
}
else
 System.out.println("B");
```

This statement displays B.

Common Error 5: Equality Test of Two Floating-Point Values

Floating-point numbers have a limited precision and calculations; involving floating-point numbers can introduce round-off errors. So, equality test of two floating-point values is not reliable. For example, you expect the following code to display true, but surprisingly it displays false.

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

Here, x is not exactly 0.5, but is 0.5000000000000001. You cannot reliably test equality of two floating-point values. However, you can compare whether they are close enough by testing whether the difference of the two numbers is less than some threshold. That is, two numbers x and y are very close if $|x-y| < \varepsilon$ for a very small value, ε. ε, a Greek letter pronounced epsilon, is commonly used to denote a very small value. Normally, you 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 that 0.5000000000000001 is approximately 0.5.
The Math.abs(a) method can be used to return the absolute value of a.

De Morgan’s Law

Humans generally have a hard time comprehending logical conditions with not operators applied to AND/OR expressions. De Morgan’s Law, named after the logician Augustus De Morgan (1806–1871), can be used to simplify these Boolean expressions.

The law states:

```
!(A && B) is the same as !A || !B
!(A || B) is the same as !A && !B
```

Pay particular attention to the fact that the `AND` and `OR` operators are reversed by moving the `NOT` inward.

[Reference]

[1] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[2] Java How To Program 10th Ed, Paul Deitel, Harvey Deitel, 2016.

[Self-Review Question]

1. Write an `if` statement that increases pay by 3% if score is greater than 90.

2. What is the output of the code in (a) and (b) if number is 30? What if number is 35?

```java
if (number % 2 == 0)
  System.out.println(number + " is even.");

System.out.println(number + " is odd.");
```

(a)

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

(b)

3. Suppose x = 2 and y = 3. Show the output, if any, of the following code. What is the output if x = 3 and y = 2? What is the output if x = 3 and y = 3?

```java
if (x > 2)
  if (y > 2){
  int z = x + y;
              System.out.println("z is " + z);
        }
     else
        System.out.println("x is " + x);
```

4. What is y after the following switch statement is executed?

```java
x = 3; y = 3;
switch (x + 3) {
  case 6: y = 1;
  default: y += 1;
}
```