CHAPTER3

Selections

Objectives

- To declare boolean variables and write Boolean expressions using relational operators
- To implement selection control using one-way if statements.
- To implement selection control using two-way if-else statements.
- To implement selection control using nested if and multi-way if statements.
- To avoid common errors and pitfalls in if statements.
- To generate random numbers using the Math.random() method.
- To program using selection statements for a variety of examples (SubtractionQuiz).
- To combine conditions using logical operators (!, &&, ||, and ^).
- To program using selection statements with combined conditions (LeapYear).
- To implement selection control using switch statements.
- To write expressions using the conditional operator.
- To examine the rules governing operator precedence and associativity.

3.1 Introduction

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

3.2 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*), shown in the table below, which can be used to compare two values (assume radius is **5** in the table).

Java Operator	Math. Symbol	Name	Example	Result
<	<	**Less than**	**radius < 0**	**false**
<=	≤	**Less than or equal**	**radius <= 0**	**false**
>	>	**Greater than**	**radius > 0**	**true**
>=	≥	**Greater than equal to**	**radius >= 0**	**true**
==	=	**Equal to**	**radius == 0**	**false**
!=	≠	**Not equal to**	**radius != 0**	**true**

- **Boolean variable**

A variable that holds a Boolean value is known as a ***Boolean variable***. The **boolean** data type is used to declare Boolean variables. For example:

boolean lightsOn = **true**;

true and **false** are literals, just like a number such as **10**. They are treated as **reserved** words and cannot be used as identifiers in the program.

3.3 if Statements

An ***if- statement*** is a construct that enables a program to specify **alternative** paths of execution.

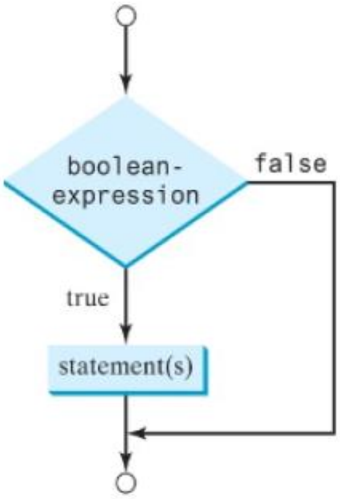
Java has several types of selection statements:

- one-way **if** statements,
- two-way **if-else** statements, nested **if** statements,
- multi-way **if-else** statements,
- **switch** statements, and
- conditional operators.

❖ 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 as follows:

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

The flowchart in Figure 3.1a illustrates how Java executes the syntax of an **if** statement.

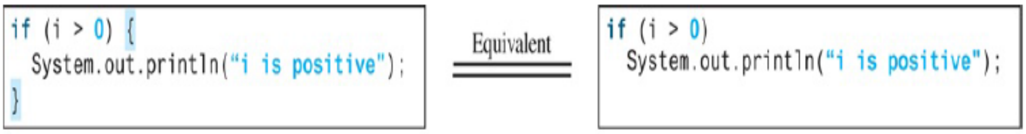


If the **boolean-expression** evaluates to **true**, the statements in the block are executed. As an example, see the following code:

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

Note

The **block braces** can be **omitted** if they enclose a **single** statement. For example, the following statements are equivalent:



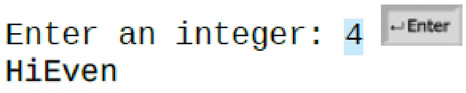
```
if (i > 0) {
  System.out.println("i is positive");
}
```

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

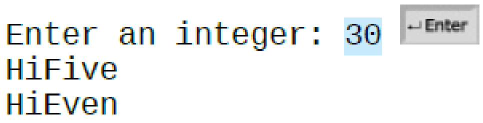
Example

Write a program that prompts the user to enter an integer. If the number is a multiple of **5**, the program displays **HiFive**. If the number is divisible by **2**, it displays **HiEven**.

```
import java.util.Scanner;

public class SimpleIfDemo {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: "
enter input    int number = input.nextInt();


check 5        if (number % 5 == 0)

               System.out.println("HiFive");


check even     if (number % 2 == 0)

               System.out.println("HiEven");
               }
               }
```



```
Enter an integer: 4 ↵Enter
HiEven
```



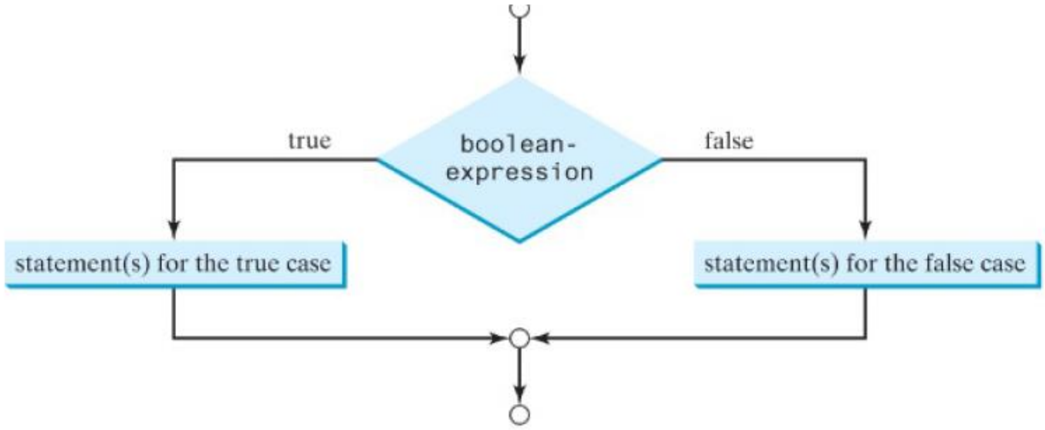
```
Enter an integer: 30 ↵Enter
HiFive
HiEven
```

❖ 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;
}
```

The flowchart of the statement is shown below:


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

The following example checks whether a number is even or odd, as follows:

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

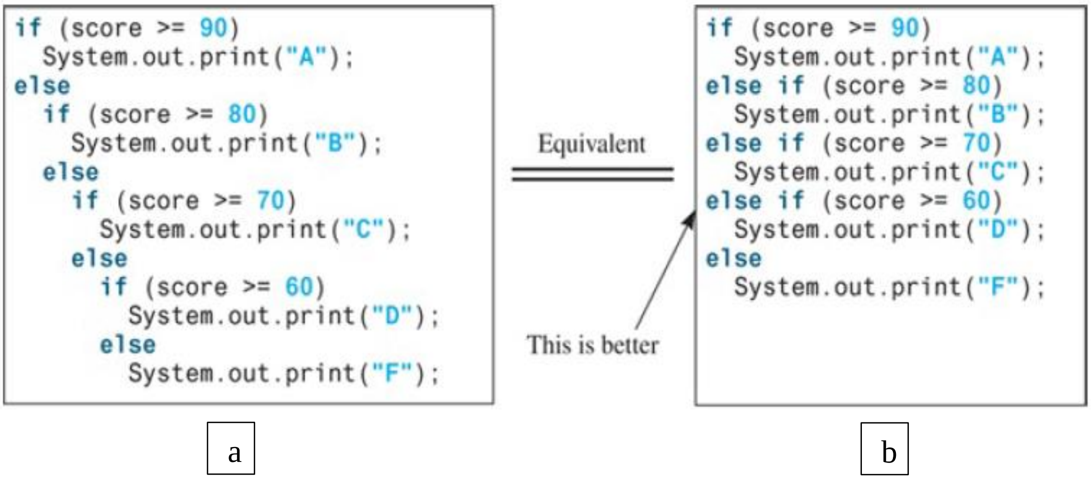
❖ Nested if and Multi-Way if-else Statements

An **if** *statement can be inside another* **if** *statement to form a nested* **if** *statement.* 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 the Figure below, for instance, prints a letter grade according to the score, with multiple alternatives.

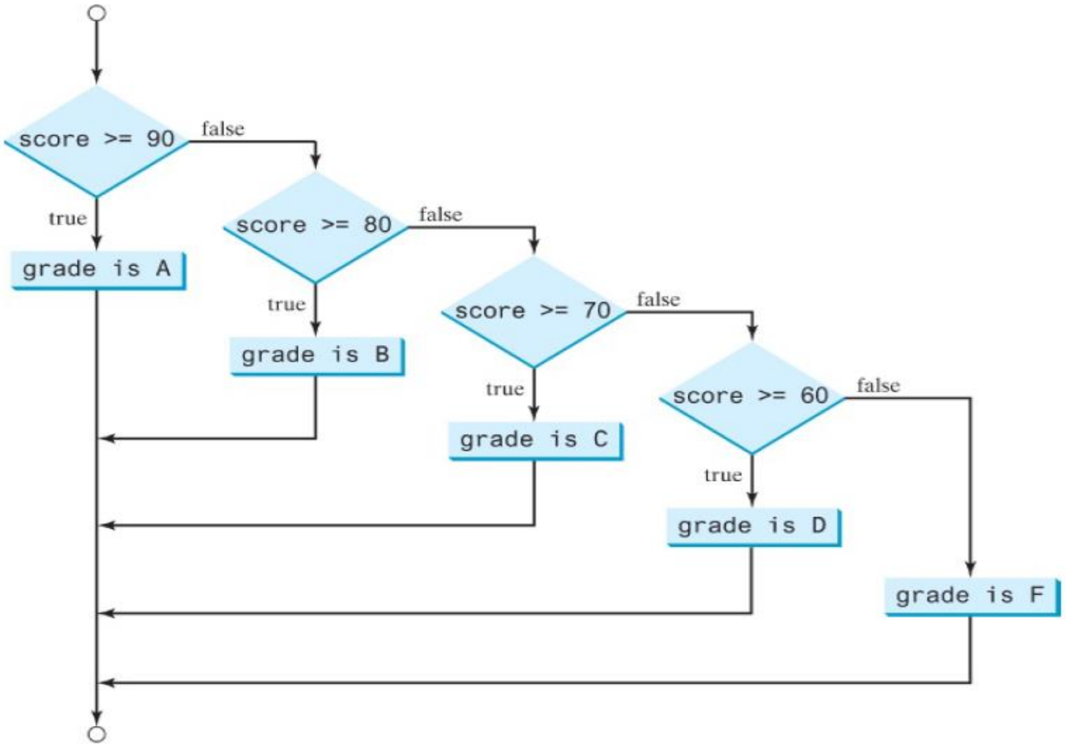


```
if (score >= 90)
  System.out.print("A");
else
  if (score >= 80)
    System.out.print("B");
  else
    if (score >= 70)
      System.out.print("C");
    else
      if (score >= 60)
        System.out.print("D");
      else
        System.out.print("F");
```

```
if (score >= 90)
  System.out.print("A");
else if (score >= 80)
  System.out.print("B");
else if (score >= 70)
  System.out.print("C");
else if (score >= 60)
  System.out.print("D");
else
  System.out.print("F");
```

In fact, Figure 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.

The execution of the above **if** statement proceeds as shown in the below flowchart:



❖ Common errors and pitfalls

- ***Common errors***:
  - ✓ Forgetting necessary braces,
  - ✓ ending an **if** statement in the wrong place,
  - ✓ mistaking == for =, and
  - ✓ dangling **else** clauses are common errors in selection statements.
- ***common Pitfalls*:**
  - ✓ Duplicated statements in **if-else** statements, and
  - ✓ testing equality of double values.

3.4 Generating Random Numbers

You can use **Math.random()** to obtain a random double value between 0.0 and 1.0, excluding 1.0.

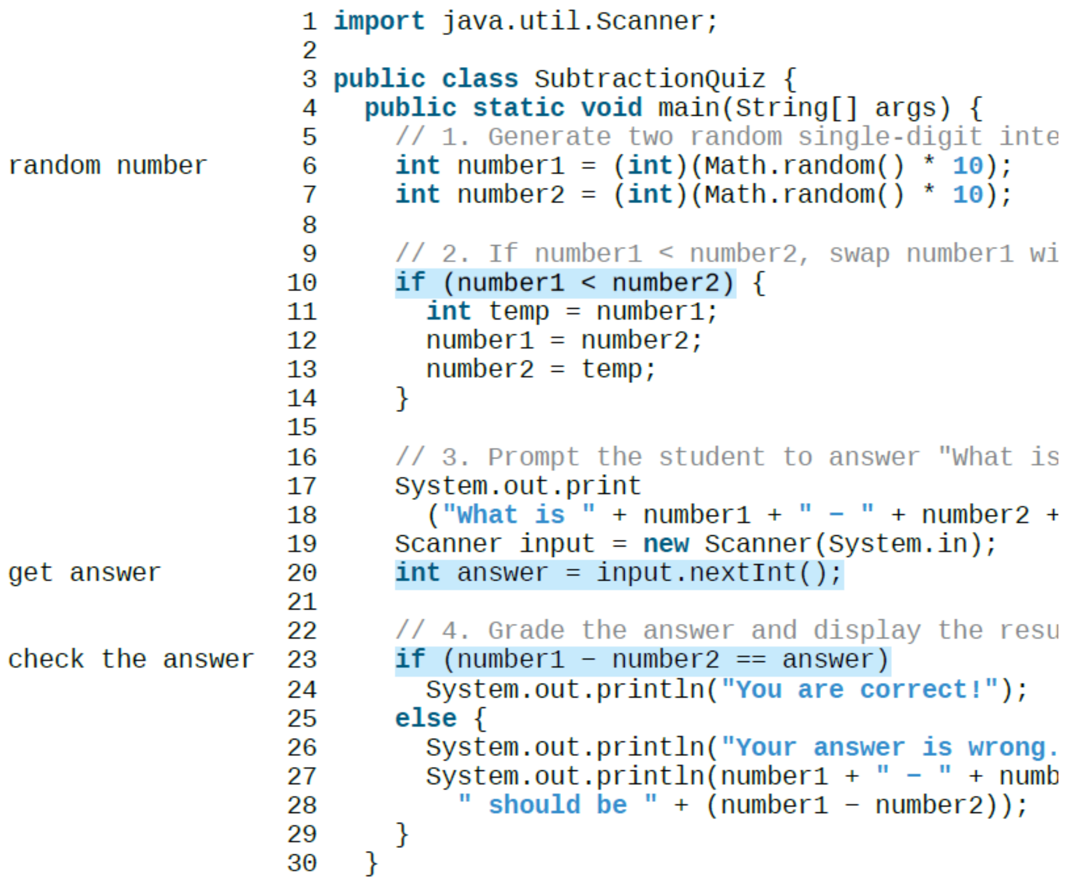
The program randomly generates **two single-digit integers**, number1 and number2, with number1 >= number2, and it displays to the student a question such as “What is 9−2?” After the student enters the answer, the program displays a message indicating whether it is correct.

A better approach is to use the **random()** method in the **Math** class. Invoking this method returns a random **double** value **d** such that **0.0** ≤ **d** < **1.0**. Thus,

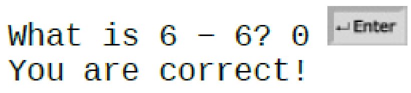
(int)(Math.random() * 10) returns a random single digit integer (i.e., a number between **0** and **9**).

The program can work as follows:

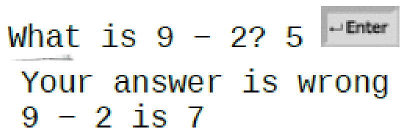
1. Generate two single-digit integers into **number1** and **number2**.
2. If **number1 < number2**, swap **number1** with **number2**.
3. Prompt the student to answer, **"What is number1 – number2?"**
4. Check the student's answer and display whether the answer is correct.



```
import java.util.Scanner;

public class SubtractionQuiz {
  public static void main(String[] args) {
    // 1. Generate two random single-digit inte
    int number1 = (int)(Math.random() * 10);
    int number2 = (int)(Math.random() * 10);

    // 2. If number1 < number2, swap number1 wi
    if (number1 < number2) {
      int temp = number1;
      number1 = number2;
      number2 = temp;
    }

    // 3. Prompt the student to answer "What is
    System.out.print
      ("What is " + number1 + " - " + number2 +
    Scanner input = new Scanner(System.in);
    int answer = input.nextInt();

    // 4. Grade the answer and display the resu
    if (number1 - number2 == answer)
      System.out.println("You are correct!");
    else {
      System.out.println("Your answer is wrong.
      System.out.println(number1 + " - " + numb
        " should be " + (number1 - number2));
    }
  }
```



```
What is 6 - 6? 0
You are correct!
```



```
What is 9 - 2? 5
Your answer is wrong
9 - 2 is 7
```

3.5 Logical Operators

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

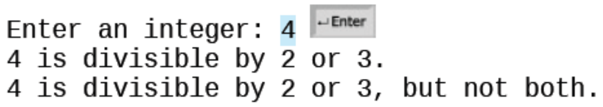
Sometimes, whether a statement is executed is determined by a combination of several conditions. You can use logical operators to combine these conditions to form a compound Boolean expression.

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

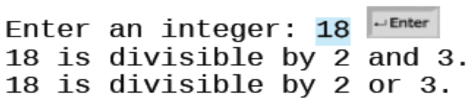
Example:

Write a program that checks whether a number is divisible by **2** and **3**, by **2** or **3**, and by **2** or **3** but not both.

```
                import class 1 import java.util.Scanner;

                 public class TestBooleanOperators {
                public static void main(String[] args) {
                // Create a Scanner
                Scanner input = new Scanner(System.in);

                // Receive an input
                System.out.print("Enter an integer: ");
input           int number = input.nextInt();

and             if (number % 2 == 0 && number % 3 == 0)
                System.out.println(number + " is divisible by 2 and 3);

or              if (number % 2 == 0 || number % 3 == 0)
                System.out.println(number + " is divisible by 2 or 3.");

exclusive or    if (number % 2 == 0 ^ number % 3 == 0)
                System.out.println(number +
                  " is divisible by 2 or 3, but not both.");
                 }
```



```
Enter an integer: 4 ↵Enter
4 is divisible by 2 or 3.
4 is divisible by 2 or 3, but not both.
```



```
Enter an integer: 18 ↵Enter
18 is divisible by 2 and 3.
18 is divisible by 2 or 3.
```

Case Study: Determining Leap Year

A year is a leap year if it is divisible by **4** but not by **100**, or if it is divisible by **400**. A leap year has **366** days. The **February** of a leap year has **29** days. You can use the following Boolean expressions to check whether a year is a leap year:

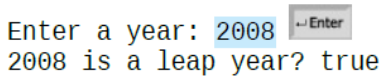
```
// A leap year is divisible by 4
boolean isLeapYear = (year % 4 == 0);
// A leap year is divisible by 4 but not by 100
isLeapYear = isLeapYear && (year % 100 != 0);
// A leap year is divisible by 4 but not by 100 or divisible by 400
isLeapYear = isLeapYear || (year % 400 == 0);
```

Or you can **combine** all these expressions into one as follows:

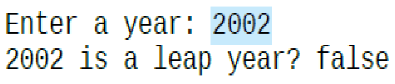
```
isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
```

EX: Write a program that lets the user enter a year and checks whether it is a leap year.

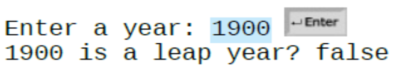
```
               import java.util.Scanner;

               public class LeapYear {
               public static void main(String[] args) {
               // Create a Scanner
               Scanner input = new Scanner(System.in);
input          System.out.print("Enter a year: ");
               int year = input.nextInt();

               // Check if the year is a leap year
leap year?     boolean isLeapYear =
               (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

               // Display the result
display result System.out.println(year + " is a leap year? "
               }
               }
```



```
Enter a year: 2008 ↵Enter
2008 is a leap year? true
```



```
Enter a year: 2002
2002 is a leap year? false
```



```
Enter a year: 1900 ↵Enter
1900 is a leap year? false
```

3.6 Switch Statements

A **switch** *statement executes statements based on the value of a* ***variable*** *or* an **expression**. Java provides a **switch** statement to simplify coding for ***multiple conditions***. the full syntax for the **switch** statement is shown below:

```
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 **switch-expression**. Note that **value1**, ..., and **valueN** are constant expressions, meaning 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.

Example, the following code displays **Weekday** for days 1–5 and **Weekend** for day 0 and day 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");
  }
```

3.7 Conditional Operators

A conditional operator evaluates an expression based on a condition.

The syntax to use the operator is as follows:

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

The result of this expression is **expression1** if **boolean-expression** is true; otherwise the result is **expression2**.

You might want to assign a value to a variable that is restricted by certain conditions. 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;
```

You can use a *conditional operator* to achieve the same result.

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

The symbols **?** and **:** appearing together is called a ***conditional operator***, also known as a ***ternary operator*** because it uses three operands.

Suppose you want to assign the larger number of variable **num1** and **num2** to **max**. You can simply write a statement using the conditional operator:

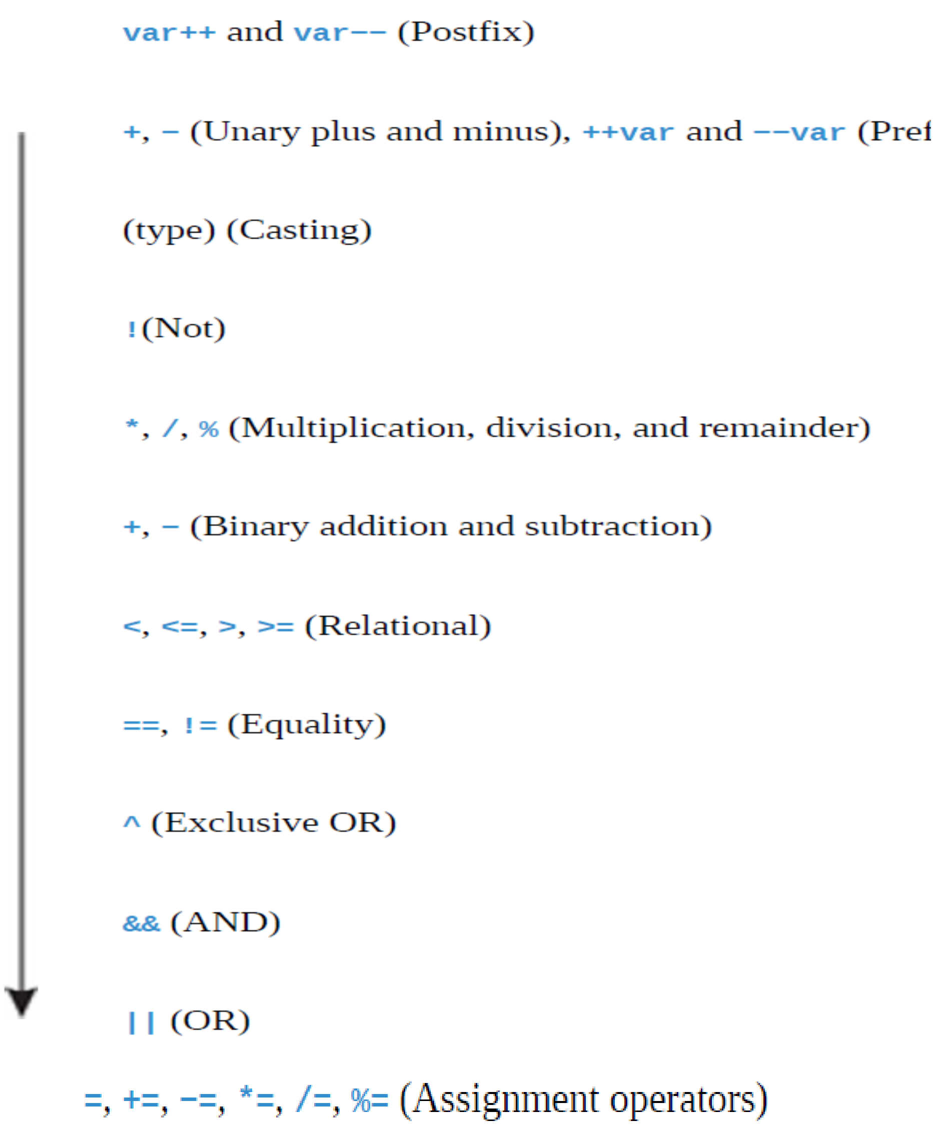
```
max = (num1 > num2) ? num1 : num2;
```

For another example, the following statement displays the message "num is even" if **num** is even, and otherwise displays “num is odd.”

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

3.8 Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated. The precedence rule defines precedence for operators, as shown below, 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.



var++ and var-- (Postfix)

+, - (Unary plus and minus), ++var and --var (Pref

(type) (Casting)

!(Not)

*, /, % (Multiplication, division, and remainder)

+, - (Binary addition and subtraction)

<, <=, >, >= (Relational)

==, != (Equality)

^ (Exclusive OR)

&& (AND)

|| (OR)

=, +=, -=, *=, /=, %= (Assignment operators)

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 is equivalent to ((a - b) + c) - d
```

Assignment operators are *right associative*. Therefore, the expression

```
a = b += c = d is equivalent to 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 left associativity for the assignment operator would not make sense.