Chapter 3
Selection Statements

3.1 Introduction

- Java provides selection statements that let you choose actions with two or more alternative courses.
- Selection statements use conditions. Conditions are Boolean expressions.
- Java has several types of selection statements:
  - if Statements, if … else statements, nested if statements
  - switch Statements
  - Conditional Expressions

3.2 boolean Data Type and Operations

- Often in a program you need to compare two values, such as whether i is greater than j. Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

TABLE 3.1 Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

- Examples
  `System.out.println(1 < 2);   // Displays true`
  `boolean b = (1 > 2);`
  `System.out.println("b is " + b);   // Displays b is false`

TABLE 3.2 Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>

- Examples
  `&& (and)  (1 < x) && (x < 100)`
  `|| (or) (lightsOn) || (isDayTime)`
### TABLE 3.3 Truth Table for Operator ！

<table>
<thead>
<tr>
<th>p</th>
<th>！p</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

**Example**

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

### TABLE 3.4 Truth Table for Operator &&

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 &amp;&amp; p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**Example**

(2 > 3) && (5 > 5) is false, because either (2 > 3) and (5 > 5) is false.
(3 > 2) && (5 > 5) is false, because (5 > 5) is false.
(3 > 2 && (5 >= 5) is true, b/c (3 > 2) and (5 >= 5) are both true.

### TABLE 3.5 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.

### TABLE 3.6 Truth Table for Operator ^

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 ^ p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

**Example**

(2 > 3) ^ (5 > 1) is true, because (2 > 3) is false and (5 > 1) is true.
(3 > 2) ^ (5 > 1) is false, because both (3 > 2) and (5 > 1) are true.
• Examples: Page 70

```java
import javax.swing.JOptionPane;

public class TestBoolean {
    public static void main(String[] args) {
        int number = 18;

        JOptionPane.showMessageDialog(null,
                        "Is " + number + 
                        "\ndivisible by 2 and 3? " +
                        (number % 2 == 0 && number % 3 == 0) +
                        "\ndivisible by 2 or 3? " +
                        (number % 2 == 0 || number % 3 == 0) +
                        "\ndivisible by 2 or 3, but not both? " +
                        (number % 2 == 0 ^ number % 3 == 0));
    }
}
```

Output:

![Message dialog showing the result of the boolean expressions]

3.2.1 Unconditional vs. Conditional Boolean Operators

&&: conditional (short-circuit) AND operator
&: unconditional AND operator
||: conditional (short-circuit) OR operator
|: unconditional OR operator

exp1 && exp2
(l < x) && (x < 100)
(l < x) & (x < 100)

If x is 1, what is x after this expression?
(x > 1) & (x++ < 10) \( \Rightarrow \) x = 2
(l > x) && (l > x++) \( \Rightarrow \) x = 1

How about
(l == x) | (10 > x++) \( \Rightarrow \) x = 2
(l == x) || (10 > x++) \( \Rightarrow \) x = 1

• Tip: Avoid using the & and |. For example:
(x != 0) & (100 / x < 10)? ➔ runtime error if x is 0

- **Note**: The & and | operators can also apply to **bitwise** operations. See Appendix F
- **Note**: A Boolean value, however, cannot be cast into a value of other types, nor can a value of other types be cast into a Boolean value.
- **Note**: true and false are literals, just like a number such as 10, so they are not keywords, but you cannot use them as identifiers, just as you cannot use 10 as an identifier.

### 3.2.2 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.

\[(\text{year} \% 4 == 0 && \text{year} \% 100 != 0) || (\text{year} \% 400 == 0)\]

- **Examples**: Page 71

    ```java
    import javax.swing.JOptionPane;
    public class LeapYear {
        public static void main(String args[]) {
            // Prompt the user to enter a year
            String yearString = JOptionPane.showInputDialog("Enter a year");

            // Convert the string into an int value
            int year = Integer.parseInt(yearString);

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

            // Display the result in a message dialog box
            JOptionPane.showMessageDialog(null,
                year + " is a leap year? " + isLeapYear);
        }
    }
    ```
3.3 if Statements

- Java has several types of selection statements:
  o if Statements, if ... else statements, nested if statements
  o switch Statements
  o Conditional Expressions

3.3.1 Simple if Statements

```java
if (booleanExpression) {
    statement(s);
}    // execution flow chart is shown in Figure (A)
```

Example
```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
    radius + " is " + area);
} // if the Boolean expression evaluates to T, the statements in the
block are executed as shown in figure (B)
```

![Flowchart](image)

**FIGURE 3.3** An if statement executes statements if the Boolean Expression evaluates as true
• **Note:**
  - The Boolean expression is enclosed in **parentheses** for all forms of the if statement. Thus, the outer parentheses in the previous if statements are required.
  - The braces can be omitted if they enclose a **single** statement.

  ![Image of boolean expression with outer parentheses required and braces can be omitted if they enclose a single statement]

• **Caution:**
  - Adding a semicolon at the end of an if clause is a **common mistake**.
  - This mistake is hard to find, because it is not a compilation error or a runtime error, it is a **logic error**.
  - This error often occurs when you use the next-line block style.

```java
if (radius >= 0); {  
  area = radius*radius*PI;  
  System.out.println("The area for the circle of radius "+
                    + "integer between 0 and 10");  
}
```

3.3.2 if...else Statements

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

![Diagram of if...else statements]

**FIGURE 3.2** An if … else executes statements for the true case if the Boolean expression evaluations are true; otherwise, statements for the false case are executed.
• if...else Example

```java
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"); // braces may be omitted
}
```

- If radius >= 0 is true, area is computed and displayed; if it is false, the message “Negative input” is printed.
- Using the if … else statement, you can rewrite the following code for determining whether a number is even or odd, as follows:

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

// rewriting the code using else

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

- This is more efficient because whether number % 2 is 0 is tested only once.

**3.3.3 Nested if Statements**

- 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.
- There is no limit to the depth of the nesting.

```java
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) is nested inside the if (i > k)
```
• The nested if statement can be used to implement multiple alternatives.

```java
if (score >= 90)
    grade = 'A';
else
    if (score >= 80)
        grade = 'B';
    else
        if (score >= 70)
            grade = 'C';
        else
            if (score >= 60)
                grade = 'D';
            else
                grade = 'F';
```

• The preceding if statement is equivalent to the following preferred format because it is easier to read:

```java
if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'F';
```

**Note:**
• The else clause matches the most recent unmatched if clause in the same block. For example, the following statement:

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

is equivalent to:

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

• Nothing is printed from the preceding statement because the compiler ignores indentation. To force the else clause to match the first if clause, you must add a pair of braces:
int i = 1; int j = 2; int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
    } else
        System.out.println("B");

This statement prints B.

Tip
- Often new Programmers write that assigns a test condition to a Boolean variable like the code in (a).

| Equivalent | (a) if (number % 2 == 0)  
|            | even = true;  
|            | else  
|            | even = false; |
| (b) boolean even  
|            | = number % 2 == 0; |

- The code can be simplified by assigning the test value directly to the variable, as shown in (b).

Caution
- To test whether a Boolean variable is true or false in a test condition, it is redundant to use the equality comparison operator like this:

| Equivalent | (a) if (even == true)  
|            | System.out.println(  
|            | "It is even."); |
| (b) if (even)  
|            | System.out.println(  
|            | "It is even."); |

- What’s wrong with the following?

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

This statement does not have syntax errors. It assigns true to even so that even is always true.
3.3.4 Example: Computing Taxes (Page 78)

- Write a program that prompts the user to enter the filing status and taxable income and computes the tax for the year 2002.
- The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2002 are shown in Table 3.7.

<table>
<thead>
<tr>
<th>Tax rate</th>
<th>Single filers</th>
<th>Married filing jointly or qualifying widow/widower</th>
<th>Married filing separately</th>
<th>Head of household</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>Up to $6,000</td>
<td>Up to $12,000</td>
<td>Up to $6,000</td>
<td>Up to $10,000</td>
</tr>
<tr>
<td>15%</td>
<td>$6,001 - $27,950</td>
<td>$12,001 - $46,700</td>
<td>$6,001 - $23,350</td>
<td>$10,001 - $37,450</td>
</tr>
<tr>
<td>30%</td>
<td>$67,701 - $141,250</td>
<td>$112,851 - $171,950</td>
<td>$56,426 - $85,975</td>
<td>$96,701 - $156,600</td>
</tr>
<tr>
<td>35%</td>
<td>$141,251 - $307,050</td>
<td>$171,951 - $307,050</td>
<td>$85,976 - $153,525</td>
<td>$156,601 - $307,050</td>
</tr>
<tr>
<td>38.6%</td>
<td>$307,051 or more</td>
<td>$307,051 or more</td>
<td>$153,526 or more</td>
<td>$307,051 or more</td>
</tr>
</tbody>
</table>

```
import javax.swing.JOptionPane;
public class ComputeTaxWithSelectionStatement {
    public static void main(String[] args) {
        // Prompt the user to enter filing status
        String statusString = JOptionPane.showInputDialog(null,
                "Enter the filing status:
                (0-single filer, 1-married jointly,
                2-married separately, 3-head of household)",
                "Example 3.1 Input", JOptionPane.QUESTION_MESSAGE);
        int status = Integer.parseInt(statusString);
        // Prompt the user to enter taxable income
        String incomeString = JOptionPane.showInputDialog(null,
                "Enter the taxable income:",
                "Example 3.1 Input", JOptionPane.QUESTION_MESSAGE);
        int income = Integer.parseInt(incomeString);
        // Compute tax
        double tax = 0;
        if (status == 0) { // Compute tax for single filers
            if (income <= 6000)
                tax = income * 0.10;
            else if (income <= 27950)
                tax = 6000 * 0.10 + (income - 6000) * 0.15;
            else if (income <= 67700)
                tax = 6000 * 0.10 + (27950 - 6000) * 0.15 +
                (income - 27950) * 0.27;
            else if (income <= 141250)
                tax = 6000 * 0.10 + (27950 - 6000) * 0.15 +
                (67700 - 27950) * 0.27 + (income - 67700) * 0.30;
            else if (income <= 307050)
                tax = 6000 * 0.10 + (27950 - 6000) * 0.15 +
                (67700 - 27950) * 0.27 + (141250 - 67700) * 0.30 +
                (income - 141250) * 0.35;
```
else
    tax = 6000 * 0.10 + (27950 - 6000) * 0.15 +
        (67700 - 27950) * 0.27 + (141250 - 67700) * 0.30 +
        (307050 - 141250) * 0.35 + (income - 307050) * 0.386;
}  
else if (status == 1) {  // Compute tax for married file jointly,
    // married separately, and head of household Left as exercise
}
else if (status == 2) { }
else if (status == 3) { }
else {
    System.out.println("Error: invalid status");
    System.exit(0);
}

    // Display the result
    JOptionPane.showMessageDialog(null, "Tax is " +
        (int)(tax * 100) / 100.0,
        "Example 3.1 Output", JOptionPane.INFORMATION_MESSAGE);
}
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.

```java
import javax.swing.JOptionPane;

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

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

        // 3. Prompt the student to answer “what is number1 – number2?”
        String answerString = JOptionPane.showInputDialog
            ("What is " + number1 + " - " + number2 + "?");
        int answer = Integer.parseInt(answerString);

        // 4. Grade the answer and display the result
        String replyString;
        if (number1 - number2 == answer)
            replyString = "You are correct!";
        else
            replyString = "Your answer is wrong. \n" + number1 + " - " + number2 + " should be " + (number1 - number2);
        JOptionPane.showMessageDialog(null, replyString);
    }
}
```
3.4 switch Statements

- One can write a `switch` statement to replace a nested `if` statement. For example,

```
switch (status) {
    case 0: compute taxes for single filers;
            break;
    case 1: compute taxes for married file jointly;
            break;
    case 2: compute taxes for married file separately;
            break;
    case 3: compute taxes for head of household;
            break;
    default: System.out.println("Errors: invalid status");
            System.exit(0);
}
```

/ checks if status matches the values 0, 1, 2, or 3 respectively.

![Diagram of switch statement]

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

The `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`. `value1...` and `valueN` are constant expressions, meaning that they cannot contain variables in the expression, 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 keyword break is optional. The break statement immediately ends the switch statement.
• The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.
• The cases statements are checked in sequential order, but the order of the cases (including the default case) does not matter. However, it is a good programming style to follow the logical sequence of the cases and place the default case at the end.

Caution
• Do not forget to use a break statement when one is needed. For example, the following code prints character a three times if ch is ‘a’:

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
    case 'd': System.out.println(ch);
    // default case
}
```
3.5 Conditional Expressions

- Conditional expressions are in different style, which no explicit if in the statement. The syntax is shown below:

\[ \text{BooleanExpression} \ ? \ \text{expression1} : \ \text{expression2}; \]

The result of this conditional expression expression1 if BooleanExpression is true; otherwise the result is expression2.

- For example:

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

is equivalent to

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

- For example:

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

is equivalent to

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

- For example:

```java
Max = (num1 > num2)? num1 : num2;
```

**Note**
- The symbols `?` and `:` appear together in a **conditional expression**. They form a condition operator. The operator is called a **ternary** operator because it uses three operands.
3.6 Formatting Console Output and Strings

- Use the new JDK 1.5 printf statement.

```java
System.out.printf(format, item);
```

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.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>'a'</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in standard scientific notation</td>
<td>4.556000e+01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>&quot;Java is cool&quot;</td>
</tr>
</tbody>
</table>

```java
int count = 5;
double amount = 45.56;
System.out.printf("count is %2d %4.2f", count, amount);
```

displays

```
count is 5 45.56
```

- Items must match the specifiers in order, on number, and in exact type. By default, a floating-point value is displayed with 6 digits after the decimal points.

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

displays

```
count is 5 and amount is 45.560000
```

- Creating Formatted Strings

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

How to evaluate?

\[ 3 + 4 \ast 4 > 5 \ast (4 + 3) - 1 \]

- The precedence rule defines precedence for operators as shown below.
- 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:
  
  \[ a - b + c - d \text{ is equivalent to } ((a - b) + c) - d \]

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

\[ a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5)) \]

**TABLE 3.10 Operator Precedence Chart**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++, --</td>
<td>var++, var--</td>
</tr>
<tr>
<td>+, -</td>
<td>(Unary plus and minus), ++var, --var</td>
</tr>
<tr>
<td>(type)</td>
<td>Casting</td>
</tr>
<tr>
<td>!</td>
<td>(Not)</td>
</tr>
<tr>
<td>*, /, %</td>
<td>(Multiplication, division, and modulus)</td>
</tr>
<tr>
<td>+, -</td>
<td>(Binary addition and subtraction)</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>(Comparison)</td>
</tr>
<tr>
<td>==, !=</td>
<td>(Equality)</td>
</tr>
<tr>
<td>&amp;</td>
<td>(Unconditional AND)</td>
</tr>
<tr>
<td>^</td>
<td>(Exclusive OR)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>(Conditional AND) Short-circuit AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=, +=, -=, *=, /=, %=</td>
<td>(Assignment operator)</td>
</tr>
</tbody>
</table>

- Example

Applying the operator precedence and associativity rule, the expression \( 3 + 4 \ast 4 > 5 \ast (4 + 3) - 1 \) is evaluated as follows:

\[ 3 + 4 \ast 4 > 5 \ast (4 + 3) - 1 \]

(1) inside parentheses first

\[ 3 + 4 \ast 4 > 5 \ast 7 - 1 \]

(2) multiplication

\[ 3 + 16 > 5 \ast 7 - 1 \]

(3) multiplication

\[ 3 + 16 > 35 - 1 \]

(4) addition

\[ 19 > 35 - 1 \]

(5) subtraction

\[ 19 > 34 \]

(6) greater than

false
3.8 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.
- 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 a is evaluated to 0 before ++a is evaluated to 1.

```java
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.

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

- The order for evaluating operands takes precedence over the operator precedence rule.
- (++a) has higher precedence than (+), but since it is a left-hand operand of the addition (+), it is evaluated before any part of its right-hand operand.
- Rule of Evaluating an Expression:
  - Rule 1: Evaluate whatever sub-expressions you can possibly evaluate from left to right.
  - Rule 2: The operators are applied according to their precedence, as shown in Table 2.11.
  - Rule 3: The associativity rule applies for two operators next to each other with the same precedence.
- Applying the rule, the expression $3 + 4 * 4 > 5 * (4 + 3) - 1$ is evaluated as follows:

```
3 + 4 * 4 > 5 * (4 + 3) - 1
3 + 16 > 5 * (4 + 3) - 1
19 > 5 * (4 + 3) - 1
19 > 5 * 7 - 1
19 > 35 - 1
19 > 34
false
```