Chapter 4: Control Structures I

Java Programming:
From Problem Analysis to Program Design,
Second Edition
Chapter Objectives

- Learn about control structures.
- Examine relational and logical operators.
- Explore how to form and evaluate logical (Boolean) expressions.
- Learn how to use the selection control structures if, if...else, and switch in a program.
Control Structures

- Three methods of processing a program:
  - In sequence
  - Branching
  - Looping
- Branch: Altering the flow of program execution by making a selection or choice.
- Loop: Altering the flow of program execution by repeating statements.
Control Structures

Figure 4-1  Flow of execution
Relational Operators

- Relational operator:
  - Allows you to make comparisons in a program.
  - Binary operator.
- Condition is represented by a logical expression in Java.
- Logical expression: An expression that has a value of either true or false.
## Relational Operators

### Table 4-1  Relational Operators in Java

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<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>
</tbody>
</table>
Relational Operators and Primitive Data Types

- Can be used with integral and floating-point data types.
- Can be used with the char data type.
- Unicode Collating Sequence.
- $8 < 5$ always evaluates to false.
- $8 < '5'$ always evaluates to true. //‘5’= 53
<table>
<thead>
<tr>
<th>Expression</th>
<th>Value of the Expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>' ' &lt; 'a'</td>
<td>true</td>
<td>The Unicode value of ' ' is 32, and the Unicode value of 'a' is 97. Because 32 &lt; 97 is true, it follows that ' ' &lt; 'a' is true.</td>
</tr>
<tr>
<td>'R' &gt; 'T'</td>
<td>false</td>
<td>The Unicode value of 'R' is 82, and the Unicode value of 'T' is 84. Because 82 &gt; 84 is false, it follows that 'R' &gt; 'T' is false.</td>
</tr>
<tr>
<td>'+' &lt; '*'</td>
<td>false</td>
<td>The Unicode value of '+' is 43, and the Unicode value of '<em>' is 42. Because 43 &lt; 42 is false, it follows that '+' &lt; '</em>' is false.</td>
</tr>
<tr>
<td>'6' &lt;= '&gt;'</td>
<td>true</td>
<td>The Unicode value of '6' is 54, and the Unicode value of '&gt;' is 62. Because 54 &lt;= 62 is true, it follows that '6' &lt;= '&gt;' is true.</td>
</tr>
</tbody>
</table>
Comparing Strings

Strings are compared character by character, using the collating sequence, until one of three conditions is met:

1. A mismatch is found.
2. One string is exhausted.
3. The last characters have been compared and are equal.
Comparing Strings

For example,

- "Air" < "Big"  // because 'A' < 'B'
- "Air" < "An"   // because 'i' < 'n'
- "Hello" < "hello"  // because 'H' < 'h'
- "Bill" < "Billy"
Comparing Strings

- Strings cannot be compared with the usual `<`, `<=`, `>`, or `>=` operators,
- and the `==` and `!=` operators don't compare the characters in the strings.
Comparing Strings

class String

- Method compareTo (<0, 0, >0)

Given string str1 and str2

\[
\text{str1.compareTo(str2)} = \begin{cases} 
\text{an integer} < 0 & \text{if string str1} < \text{str2} \\
0 & \text{if string str1} \text{ is equal to string str2} \\
\text{an integer} > 0 & \text{if string str1} > \text{str2} 
\end{cases}
\]
Comparing Strings

String str1 = "Hello";
String str2 = "Hi";
String str3 = "Air";
String str4 = "Bill";
String str5 = "Bigger";

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>str1.compareTo(str2)</td>
<td>&lt; 0</td>
<td>str1 = &quot;Hello&quot; and str2 = &quot;Hi&quot;. The first character of str1 and str2 are the same, but the second character 'e' of str1 is less than the second character 'i' of str2. Therefore, str1.compareTo(str2) &lt; 0.</td>
</tr>
<tr>
<td>str1.compareTo(&quot;Hen&quot;)</td>
<td>&lt; 0</td>
<td>str1 = &quot;Hello&quot;. The first two characters of str1 and &quot;Hen&quot; are the same, but the third character 'l' of str1 is less than the third character 'n' of &quot;Hen&quot;. Therefore, str1.compareTo(&quot;Hen&quot;) &lt; 0.</td>
</tr>
</tbody>
</table>
### Comparing Strings

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>str4.compareTo(str3)</td>
<td>&gt; 0</td>
<td>str4 = &quot;Bill&quot; and str3 = &quot;Air&quot;. The first character 'B' of str4 is greater than the first character 'A' of str3. Therefore, ( \text{str4.compareTo(str3)} &gt; 0 ).</td>
</tr>
<tr>
<td>str1.compareTo(&quot;hello&quot;)</td>
<td>&lt; 0</td>
<td>str1 = &quot;Hello&quot;. The first character 'H' of str1 is less than the first character 'h' of &quot;hello&quot; because the Unicode value of 'H' is 72, and the Unicode value of 'h' is 104. Therefore, ( \text{str1.compareTo(&quot;hello&quot;)} &lt; 0 ).</td>
</tr>
<tr>
<td>str2.compareTo(&quot;Hi&quot;)</td>
<td>= 0</td>
<td>str2 = &quot;Hi&quot;. The strings str2 and &quot;Hi&quot; are of the same length and their corresponding characters are the same. Therefore, ( \text{str2.compareTo(&quot;Hi&quot;)} = 0 ).</td>
</tr>
<tr>
<td>str4.compareTo(&quot;Billy&quot;)</td>
<td>&lt; 0</td>
<td>str4 = &quot;Bill&quot; has four characters and &quot;Billy&quot; has five characters. Therefore, str4 is the shorter string. All four characters of str4 are the same as the corresponding first four characters of &quot;Billy&quot; and &quot;Billy&quot; is the larger string. Therefore, ( \text{str4.compareTo(&quot;Billy&quot;)} &lt; 0 ).</td>
</tr>
<tr>
<td>str5.compareTo(&quot;Big&quot;)</td>
<td>&gt; 0</td>
<td>str5 = &quot;Bigger&quot; has six characters and &quot;Big&quot; has three characters. Therefore, str5 is the larger string. The first three characters of str5 are the same as the corresponding first three characters of &quot;Big&quot;. Therefore, ( \text{str5.compareTo(&quot;Big&quot;)} &gt; 0 ).</td>
</tr>
</tbody>
</table>
public class Example4_2 {
    public static void main(String[] args) {
        String str1 = "Hello";                             //Line 1
        String str2 = "Hi";                                //Line 2
        String str3 = "Air";                               //Line 3
        String str4 = "Bill";                              //Line 4
        String str5 = "Bigger";                            //Line 5
        System.out.println("Line 6: " + str1.compareTo(str2) evaluates to " + str1.compareTo(str2));    //Line 6
        System.out.println("Line 7: " + str1.compareTo("Hen") evaluates to " + str1.compareTo("Hen"));  //Line 7
        System.out.println("Line 8: " + str4.compareTo(str3) evaluates to " + str4.compareTo(str3));      //Line 8
        System.out.println("Line 9: " + str1.compareTo("hello") evaluates to " + str1.compareTo("hello"));//Line 9
        System.out.println("Line 10: " + str2.compareTo("Hi") evaluates to " + str2.compareTo("Hi"));   //Line 10
        System.out.println("Line 11: " + str4.compareTo("Billy") evaluates to " + str4.compareTo("Billy"));//Line 11
        System.out.println("Line 12: " + str5.compareTo("Big") evaluates to " + str5.compareTo("Big"));   //Line 12
    }
}
Comparing Strings

Values such as $-4$, $-2$, $1$ and so on, are differences of the collating sequence of the first unmatched characters of the string.

For example:

in line 6: where, str1 = “Hello”, str2 = “Hi”

- ‘e’ $\rightarrow$ 101
- ‘i’ $\rightarrow$ 105
- $101 - 105 \rightarrow -4$
Comparing Strings

♦ In addition to the method `compareTo`, you can use the method `equals` of the class `String`.

♦ Returns `true` or `false`.

♦ Example: `str1 = "Hello", str2= "Hi"

```java
str1.equals("Hello"); // returns true
str1.equals(str2);  // returns false
```
Comparing Strings

- You should use one of the following tests to compare the contents of two strings:
  - `string1.equals(string2)`
  - `string1.compareTo(string2)`

- **Here's the wrong way to do it:**
  ```java
  string1 == string2
  ```

**Why wrong?**
A comparison of objects (such as Strings) using the `==` operator doesn't compare the contents of the Strings. Instead, it compares the *address* of the two Strings.

[http://www.javabeginner.com/java-string-comparison.htm](http://www.javabeginner.com/java-string-comparison.htm)
Comparing Strings

String s = "hi";

- s == "hi"          // true
- "hi".equals(s)     // true
- s == new String(s) // false
Logical (Boolean) Operators

- `!` is unary operator.
- `&&` is binary operator.
- `||` is binary operator.

Table 4-4  Logical (Boolean) Operators in Java

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!</code></td>
<td>not</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>and</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
</tbody>
</table>
Logical (Boolean) Operators

Table 4-5  ! (not) Operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>!(Expression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Example:

! (‘A’ > ‘B’) is true.

Because ‘A’ > ‘B’ is false ➔ ! (‘A’ > ‘B’) is true.
Logical (Boolean) Operators

Table 4-5  ! (not) Operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>!(Expression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Example:

!(‘A’ > ‘B’) is true.
Because ‘A’ > ‘B’ is false ➔ !(‘A’ > ‘B’) is true
Logical (Boolean) Operators

Table 4-6  && (and) Operator

<table>
<thead>
<tr>
<th>Expression1</th>
<th>Expression2</th>
<th>Expression1 &amp;&amp; Expression2</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 4-7  || (or) Operator

| Expression1 | Expression2 | Expression1 || Expression2 |
|-------------|-------------|-------------|----------------|
| true        | true        | true        |
| true        | false       | true        |
| false       | true        | true        |
| false       | false       | false       |
Logical (Boolean) Operators

Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(A &gt; B)</td>
<td>true</td>
<td>Because 'A' &gt; 'B' is false, !(A &gt; B) is true.</td>
</tr>
<tr>
<td>!(6 &lt;= 7)</td>
<td>false</td>
<td>Because 6 &lt;= 7 is true, !(6 &lt;= 7) is false.</td>
</tr>
<tr>
<td>(14 &gt;= 5) &amp;&amp; (A &lt; B)</td>
<td>true</td>
<td>Because (14 &gt;= 5) is true, (A &lt; B) is true, and true &amp;&amp; true is true, the expression evaluates to true.</td>
</tr>
<tr>
<td>(24 &gt;= 35) &amp;&amp; (A &lt; B)</td>
<td>false</td>
<td>Because (24 &gt;= 35) is false, (A &lt; B) is true, and false &amp;&amp; true is false, the expression evaluates to false.</td>
</tr>
</tbody>
</table>
Order of Precedence

\[11 + 5 \leq 9 \lor 6 < 15 \land 7 \geq 8\]

which to solve first:

arithmetic, relational or logical?
Order of Precedence

\[ 11 + 5 \leq 9 \mid \mid 6 < 15 \&\& 7 \geq 8 \]

which to solve first:

arithmetic, relational or logical?
## Order of Precedence

For more complex expressions:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary operators:</td>
<td>++, --, !, unary - and +, type-cast</td>
</tr>
<tr>
<td>Multiplication and division:</td>
<td>*, /, %</td>
</tr>
<tr>
<td>Addition and subtraction:</td>
<td>+, -</td>
</tr>
<tr>
<td>Relational operators:</td>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td>Equality and inequality:</td>
<td>==, !=</td>
</tr>
<tr>
<td>Boolean and:</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>Boolean or:</td>
<td></td>
</tr>
<tr>
<td>Conditional operator:</td>
<td>?:</td>
</tr>
<tr>
<td>Assignment operators:</td>
<td>=, +=, -=, *=, /=, %=</td>
</tr>
</tbody>
</table>

Operators on the same line have the same precedence. When they occur together, unary operators and assignment operators are evaluated right-to-left, and the remaining operators are evaluated left-to-right. For example:

\[ A \ast B / C \text{ means } (A \ast B) / C, \text{ while } A = B = C \text{ means } A = (B = C). \]
Precedence Rules

Example 4.6: Evaluate the following expression:

\[(17 < 4*3+5) || (8*2 == 4*4) && !(3+3 == 6)\]

= \[(17 < 12+5) || (16 == 16) && !(6 == 6)\]

= \[(17 < 17) || true && ! (true)\]

= false || true && false

= false || false

= false
**Example**: suppose the following declarations:

```java
boolean found = true;
boolean flag = false;
double x = 5.2;
```

Evaluate:

- `!found` ➞ false
- `x > 4.0` ➞ true
- `flag && found` ➞ false

**Go through example 4_8**
Suppose:

- `int age = 25;`
- `char grade='B';`

`(age >= 21) || ( 3 + 8 == 5)`
Because `(25 >= 21)` is `true` and the operator used is `||`, due to short-circuit evaluation the computer does not evaluate `( 3 + 8 == 5 )`.

`(grade == 'A') && (3 - 2 >= 7)`
Because `( 'B' == 'A' )` is `false` and the operator used is `&&`, due to short-circuit evaluation the computer does not evaluate `( 3 - 2 >= 7 )`. 
Short-Circuit Evaluation

♦ A process in which the computer evaluates a logical expression from left to right and stops as soon as the value of the expression is known.

♦ If the operators | and & are used, NO short circuit evaluation is used.
Selection

- One-way selection
- Two-way selection
- Compound (block of) statements
- Multiple selections (nested if)
- Conditional operator
- `switch` structures
One-Way Selection

- Syntax:
  \[ \text{if (expression)} \]
  \[ \text{statement ;} \]
  Must be in ( ) and no ;
  put ; after end of statement

- Expression referred to as decision maker.
- If the value of the expression is true \( \rightarrow \)
  statement executes.
- If the value of the expression is false \( \rightarrow \)
  statement does not executes.
One-Way Selection

Example:
```java
char grade=''
if ( score >= 90 )
    grade = 'A';
```
Example 4-11
//Determine the absolute value of an integer
import java.util.*;
public class AbsoluteValue
{
    static Scanner console = new Scanner(System.in);
    public static void main(String[] args)
    {
        int number;
        int temp;
        System.out.println("Enter an integer:");          //Line 1
        number = nextInt();                                  //Line 2
        temp = number;                                        //Line 3
        if (number < 0)                                        //Line 4
            number = -number;                                  //Line 5

        System.out.println("The absolute value of " + temp+ " is " + number+"Absolute Value");
    }
}
Two-Way Selection

♦ Syntax:

```plaintext
if (expression)
    statement1;
else
    statement2;
```

♦ `else` statement must be paired with an `if`. 
Two-Way Selection

Example:

```java
boolean positive, negative;
if (number >= 0)
    positive = true;
else //number < 0
    negative = true;
```
Two-Way Selection

Example 4-14

```c
if (hours > 40.0) // includes overtime payment
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);
else
    wages = hours * rate;
```

Given that rate = 100, what wages will be if:

a) hours = 50
b) hours = 30
Two-Way Selection

Example 4-15

```java
if (hours > 40.0);  //Line 1
    wages = 40.0 * rate +
    1.5 * rate * (hours - 40.0);  //Line 2
else  //Line 3
    wages = hours * rate;  //Line 4
```

- Because a semicolon follows the closing parenthesis of the `if` statement (Line 1), the `else` statement stands alone. The semicolon at the end of the `if` statement (see Line 1) ends the `if` statement, so the statement at Line 2 separates the else clause from the `if` statement. That is, `else` is by itself. Because there is no separate `else` statement in Java, this code generates a syntax error.

- For some common errors made by beginning programmers check ex 4_17, 4_18.
Compound (Block of) Statements

Syntax:

```{ {statement1
statement2
.
.
.
statementn }
}````
Compound (Block of) Statements

```java
if (age > 18)
{
    System.out.println("Eligible to vote.");
    System.out.println("No longer a minor.");
}
else
{
    System.out.println("Not eligible to vote.");
    System.out.println("Still a minor.");
}
```
Multiple Selection: Nested if

- Syntax:

```c
if (expression1)
    statement1;
else
    if (expression2)
        statement2;
    else
        statement3;
```

- `else` is associated with the most recent incomplete if.

- Multiple `if` statements can be used in place of `if...else` statements.

- May take longer to evaluate.
Multiple Selection: Nested if

Example 4_20 :

```java
if (score >= 90)
    System.out.println ("Grade is A");
else if (score >= 80)
    System.out.println ("Grade is B");
else if (score >= 70)
    System.out.println ("Grade is C");
else if (score >= 60)
    System.out.println ("Grade is D");
else System.out.println ("Grade is F");
```
Multiple Selection: Nested if

Example 4_21:

```java
if( temperature >= 50 )
    if (temperature >= 80)
        System.out.println ("Good swimming day");
    else
        System.out.println ("Good golfing day");
else
    System.out.println ("Good tennis day");
```
Multiple Selection: Nested if

Example4_22:

```java
if (temperature >= 50)
    if (temperature >= 80)
        System.out.println("Good swimming day");
    else
        System.out.println("Good golfing day");
```
Multiple Selection: Nested if

Example4_23:

```java
if ( GPA >= 2.0 )
    if (GPA >= 3.9)
        System.out.println("Dean Honor list");
else
    System.out.println("GPA below graduation requirement");
```

If GPA = 3.8 what will be printed?
Multiple Selection: Nested if

Example4_23:

```java
if ( GPA >= 2.0 )
    if ( GPA >= 3.9 )
        System.out.println("Dean Honor list");
else
    System.out.println( "GPA below graduation requirement" );
```

If GPA = 3.8 what will be printed?

GPA below graduation requirement
Multiple Selection: Nested if

Example4_23 : (rewritten)

```java
if ( GPA >= 2.0 )
{
    if (GPA >= 3.9)
        System.out.println("Dean Honor list");
}
else
    System.out.println("GPA below graduation requirement");
```

Now, if GPA = 3.8 what will be printed?
Multiple Selection: Nested if

Example4_23 : (rewritten)

if ( GPA >= 2.0 )
{
    if (GPA >= 3.9)
        System.out.println("Dean Honor list");
}
else
    System.out.println("GPA below graduation requirement");

Now, if GPA = 3.8 what will be printed?
Conditional (? :) Operator

- Ternary operator
- Syntax:
  
  expression1 ? expression2 : expression3;

- If expression1 = true, then the result of the condition is expression2. Otherwise, the result of the condition is expression3.
Conditional (? :) Operator

Example:

```c
int x = 5, y = 3, min;
if (x <= y)
    min = x;
else
    min = y;

The above stmt can be written using the conditional operator:
min = (x <= y) ? x : y;
```
Switch Structures

```
switch (expression)
{
  case value1: statements1
               break;
  case value2: statements2
               break;
  ...
  case valuen: statementsn
               break;
  default: statements
}
```

- expression is evaluated first.
- expression is also known as selector.
- expression can be an identifier or an expression and only integral.
- value can only be integral.
switch Structures

Figure 4-7  switch statement
Example 4-24

switch (grade) {
    case 'A': System.out.println("The grade is A.");
              break;
    case 'B': System.out.println("The grade is B.");
              break;
    case 'C': System.out.println("The grade is C.");
              break;
    case 'D': System.out.println("The grade is D.");
              break;
    case 'F': System.out.println("The grade is F.");
              break;
    default: System.out.println("The grade is invalid.");
}
switch Structures

- **break** is optional.
- When the value of the **switch** expression matches a **case** value, all statements execute until a **break** is encountered, and the program skips all **case** labels in between.
import java.util.*;

public class Example4_25
{
    static Scanner console = new Scanner(System.in);
    public static void main(String[] args) {

        int num;

        System.out.print("Enter an integer between 0 and 10: ");    //Line 1
        num = console.nextInt();                                    //Line 2
        System.out.println();                                        //Line 3
        System.out.println("The number you entered is "+num);        //Line 4
        switch(num)                                                  //Line 5
        {
            case 0:
            case 1: System.out.print("Hello ");                     //Line 6
            case 2: System.out.print("there. ");                    //Line 7
            case 3: System.out.print("I am ");                      //Line 8
            case 4: System.out.println("Mickey.");                  //Line 9
                break;                                              //Line 10
            case 5: System.out.print("How ");                      //Line 11
            case 6: System.out.print("How ");                      //Line 12
            case 7: System.out.print("How ");                      //Line 13
            case 8: System.out.println("are you?");                 //Line 14
                break;                                              //Line 15
            case 9: break;                                          //Line 16
            case 10: System.out.println("Have a nice day.");        //Line 17
                break;                                              //Line 18
            default: System.out.println("Sorry the number is out" +"of range.");   //Line 19
        }
        System.out.println("Out of switch structure.");           //Line 20
    }
}
switch Structures

♦ Sample Run1:

Enter an integer between 0 and 10: 0
The number you entered is 0
Hello there. I am Mickey.
Out of switch structure.

Sample Run2:

Enter an integer between 0 and 10: 9
The number you entered is 9
Out of switch structure.
Programming Example: Cable Company Billing

- **Input:** Customer’s account number, customer code, number of premium channels to which customer subscribes, number of basic service connections (in the case of business customers).
- **Output:** Customer’s account number and the billing amount.
Programming Example: Cable Company Billing

Solution:

1. Prompt user for information.
2. Use switch statements based on customer’s type.
3. Use an if statement nested within a switch statement to determine the amount due by each customer.
Chapter Summary

- Control structures are used to process programs.
- Logical expressions and order of precedence of operators are used in expressions.
- Compare strings.
- If statements.
- *if...else* statements.
- *switch* structures.
- Proper syntax for using control statements.