Flow of Control

Chapter 3

Objectives

• Use Java branching statements
• Compare values of primitive types
• Compare objects such as strings
• Use the primitive type boolean
• Use simple enumerations in a program

Outline

• The if-else Statement
• The Type boolean
• The switch statement
Flow of Control

• Flow of control is the order in which a program performs actions.
  - Up to this point, the order has been sequential.
• A branching statement chooses between two or more possible actions.
• A loop statement repeats an action until a stopping condition occurs.

The if-else Statement: Outline

• Basic if-else Statement
• Boolean Expressions
• Comparing Strings
• Nested if-else Statements
• Multibranch if-else Statements
• The switch Statement
• (optional) The Conditional Operator
• The exit Method

The if-else Statement

• A branching statement that chooses between two possible actions.
• Syntax
  
  if (Boolean_Expression)
  
  Statement_1
  
  else
  
  Statement_2
The **if-else** Statement

**Example**

```java
if (balance <= 0)
    balance = balance + (INTEREST_RATE * balance) / 12;
else
    balance = balance - OVERDRAWN_PENALTY;
```

**Figure 3.1 The Action of the if-else Statement**

*Sample program* Listing 3.1

**Sample screen output**

Enter your checking account balance: $505.67
Original balance $505.67
After adjusting for one month of interest and penalties, your new balance is $506.51278

Enter your checking account balance: $-15.53
Original balance $-15.53
After adjusting for one month of interest and penalties, your new balance is $-25.53
Semantics of the *if-else* Statement

- Figure 3.2

![Diagram of if-else statement semantics]

Compound Statements

- To include multiple statements in a branch, enclose the statements in braces.

```java
if (count < 3) {
    total = 0;
    count = 0;
}
```

Omitting the *else* Part

- FIGURE 3.3 The Semantics of an *if* Statement without an *else*
Introduction to Boolean Expressions

- The value of a boolean expression is either true or false.
- Examples
  - time < limit
  - balance <= 0

Java Comparison Operators

- Figure 3.4 Java Comparison Operators

<table>
<thead>
<tr>
<th>Math Notation</th>
<th>Name</th>
<th>Java Notation</th>
<th>Java Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
<td>==</td>
<td>balance == 0; answer = 'y'</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>!=</td>
<td>income != tax; answer != 'y'</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&gt;</td>
<td>expenses &gt; income</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>points &gt;= 60</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>&lt;</td>
<td>pressure &lt; max</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>expenses &lt;= income</td>
</tr>
</tbody>
</table>

Compound Boolean Expressions

- Boolean expressions can be combined using the "and" (&&) operator.
- Example
  - if ((score > 0) && (score <= 100)) ...
- Not allowed
  - if (0 < score <= 100)
    ...
Compound Boolean Expressions

• Syntax
  
  (Sub_Expression_1) \&\&
  (Sub_Expression_2)

• Parentheses often are used to enhance readability.
• The larger expression is true only when both of the smaller expressions are true.

Compound Boolean Expressions

• Boolean expressions can be combined using the "or" (||) operator.
• Example
  
  if ((quantity > 5) || (cost < 10))
  ...
  
• Syntax
  
  (Sub_Expression_1) ||
  (Sub_Expression_2)

Compound Boolean Expressions

• The larger expression is true
  • When either of the smaller expressions is true
  • When both of the smaller expressions are true.
• The Java version of "or" is the inclusive or which allows either or both to be true.
• The exclusive or allows one or the other, but not both to be true.
Negating a Boolean Expression

- A boolean expression can be negated using the "not" (!) operator.
- Syntax
  
  `!(Boolean_Expression)`

- Example
  
  `(a || b) && !(a && b)`

  which is the exclusive or
Boolean Operators

- FIGURE 3.7 The Effect of the Boolean Operators && (and), || (or), and ! (not) on Boolean values

| Value of A | Value of B | Value of A && B | Value of A || B | Value of ! (A) |
|------------|------------|----------------|----------------|---------------|
| true       | true       | true           | true           | false         |
| true       | false      | false          | true           | false         |
| false      | true       | false          | true           | true          |
| false      | false      | false          | false          | true          |

Using ==

- == is appropriate for determining if two integers or characters have the same value.
  
  ```java
  if (a == 3)
  ```
  
  where `a` is an integer type
- == is not appropriate for determining if two floating point values are equal. Use `<` and some appropriate tolerance instead.
  
  ```java
  if (abs(b - c) < epsilon)
  ```
  
  where `b`, `c`, and `epsilon` are floating point types

Using ==

- == is not appropriate for determining if two objects have the same value.
  
  - if `(s1 == s2)`, where `s1` and `s2` refer to strings, determines only if `s1` and `s2` refer to a common memory location.
  - If `s1` and `s2` refer to strings with identical sequences of characters, but stored in different memory locations, `(s1 == s2)` is false.
Using ==

- To test the equality of objects of class String, use method `equals`.
  
  ```java
  s1.equals(s2)
  or
  s2.equals(s1)
  ```

- To test for equality ignoring case, use method `equalsIgnoreCase`.
  ```java
  ("Hello".equalsIgnoreCase("hello"))
  ```

equals and `equalsIgnoreCase`

- Syntax
  ```java
  String.equals(Other_String)
  String.equalsIgnoreCase(Other_String)
  ```

Testing Strings for Equality

- View sample program Listing 3.2
  ```java
class StringEqualityDemo
  ```
Lexicographic Order

- Lexicographic order is similar to alphabetical order, but it is based on the order of the characters in the ASCII (and Unicode) character set.
  - All the digits come before all the letters.
  - All the uppercase letters come before all the lower case letters.

Lexicographic Order

- Strings consisting of alphabetical characters can be compared using method `compareTo` and method `toUpperCase` or method `toLowerCase`.

```java
String s1 = "Hello";
String lowerS1 = s1.toLowerCase();
String s2 = "hello";
if (s1.compareTo(s2)) == 0
    System.out.println("Equal!");
```

Method `compareTo`

- Syntax
  ```java
  String_1.compareTo(String_2)
  ```
- Method `compareTo` returns
  - a negative number if `String_1` precedes `String_2`
  - zero if the two strings are equal
  - a positive number of `String_2` precedes `String_1`.  

Nested if-else Statements

• An if-else statement can contain any sort of statement within it.
• In particular, it can contain another if-else statement.
  • An if-else may be nested within the “if” part.
  • An if-else may be nested within the “else” part.
  • An if-else may be nested within both parts.

Nested Statements

• Syntax
  ```java
  if (Boolean_Expression_1)
      if (Boolean_Expression_2)
          Statement_1)
      else
          Statement_2)
  else
      if (Boolean_Expression_3)
          Statement_3)
      else
          Statement_4);
  ```

• Each else is paired with the nearest unmatched if.
• If used properly, indentation communicates which if goes with which else.
• Braces can be used like parentheses to group statements.
Nested Statements

- Subtly different forms

First Form
```java
if (a > b)
{
  if (c > d)
    e = f
  else
    g = h;
}
```

Second Form
```java
if (a > b)
  if (c > d)
    e = f
  else
    g = h;
// oops
```

---

Compound Statements

- When a list of statements is enclosed in braces ({}), they form a single compound statement.

Syntax
```java
{
  Statement_1;
  Statement_2;
  ...
}
```

- A compound statement can be used wherever a statement can be used.

Example
```java
if (total > 10)
{
  sum = sum + total;
  total = 0;
}
```
Multibranch if-else Statements

• Syntax

\[
\begin{align*}
\text{if} & \ (\text{Boolean\_Expression\_1}) \\
\text{Statement\_1} & \\
\text{else if} & \ (\text{Boolean\_Expression\_2}) \\
\text{Statement\_2} & \\
\text{else if} & \ (\text{Boolean\_Expression\_3}) \\
\text{Statement\_3} & \\
\vdots & \\
\text{else} & \\
\text{Default\_Statement}
\end{align*}
\]

Multibranch if-else Statements

• Figure 3.8 Semantics

Multibranch if-else Statements

• View sample program Listing 3.3
class Grader

Sample screen output:
Enter your score:
85
Score = 85
Grade = B
Multibranch if-else Statements

- Equivalent code

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

Case Study – Body Mass Index

- Body Mass Index (BMI) is used to estimate the risk of weight-related problems
- BMI = mass / height^2
  - Mass in kilograms, height in meters
- Health assessment if:
  - BMI < 18.5 Underweight
  - 18.5 ≤ BMI < 25 Normal weight
  - 25 ≤ BMI < 30 Overweight
  - 30 ≤ BMI Obese

Case Study – Body Mass Index

- Algorithm
  - Input height in feet & inches, weight in pounds
  - Convert to meters and kilograms
    - 1 lb = 2.2 kg
    - 1 inch = 0.254 meters
  - Compute BMI
  - Output health risk using if statements

View sample program Listing 3.4

class BMI
The Conditional Operator

```java
if (n1 > n2)
    max = n1;
else
    max = n2;
```
can be written as

```java
max = (n1 > n2) ? n1 : n2;
```

• The `?` and `:` together are call the conditional operator or ternary operator.

The Conditional Operator

• The conditional operator is useful with print and println statements.

```java
System.out.println("You worked " +
    ((hours > 1) ? "hours" : "hour");
```

The `exit` Method

• Sometimes a situation arises that makes continuing the program pointless.
• A program can be terminated normally by `System.exit(0)`. 
The **exit** Method

- **Example**
  
  ```java
  if (numberOfWinners == 0)
      {
      System.out.println("Error: Dividing by zero.");
      System.exit (0);
      }
  else
    {
    oneShare = payoff / numberOfWinners;
    System.out.println("Each winner will receive $" + oneShare);
    }
  ```

The Type **boolean**

- The type **boolean** is a primitive type with only two values: `true` and `false`.
- Boolean variables can make programs more readable.
  
  ```java
  if (systemsAreOK)
  instead of
  if((temperature <= 100) && (thrust >= 12000) && (cabinPressure > 30) && …)
  ```

Boolean Expressions and Variables

- Variables, constants, and expressions of type **boolean** all evaluate to either `true` or `false`.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```java
boolean isPositive = (number > 0);
...
if (isPositive) ...
```
Naming Boolean Variables

- Choose names such as `isPositive` or `systemsAreOk`.
- Avoid names such as `numberSign` or `systemStatus`.

Precedence Rules

- Parentheses should be used to indicate the order of operations.
- When parentheses are omitted, the order of operation is determined by precedence rules.

Precedence Rules

- Operations with higher precedence are performed before operations with lower precedence.
- Operations with equal precedence are done left-to-right (except for unary operations which are done right-to-left).
Precedence Rules

- Figure 3.9

**Highest Precedence**

First: the unary operators +, -, ++, --, and!
Second: the binary arithmetic operators *, /, %
Third: the binary arithmetic operators +, -
Fourth: the boolean operators <, <=, >=
Fifth: the boolean operators ==, !=
Sixth: the boolean operator &
Seventh: the boolean operator | |
Eighth: the boolean operator &&
Ninth: the boolean operator ||

**Lowest Precedence**

In what order are the operations performed?

\[
\text{score} < \text{min/2} - 10 \quad \text{||} \quad \text{score} > 90 \\
\text{score} < (\text{min/2}) - 10 \quad \text{||} \quad \text{score} > 90 \\
\text{score} < ((\text{min/2}) - 10) \quad \text{||} \quad \text{score} > 90 \\
(score < ((\text{min/2}) - 10)) \quad \text{||} \quad \text{score} > 90 \\
(score < ((\text{min/2}) - 10)) \quad \text{||} \quad (score > 90)
\]

Short-circuit Evaluation

- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
  - If the first operand associated with an || is true, the expression is true.
  - If the first operand associated with an && is false, the expression is false.
- This is called short-circuit or lazy evaluation.
Short-circuit Evaluation

- Short-circuit evaluation is not only efficient, sometimes it is essential!
- A run-time error can result, for example, from an attempt to divide by zero.
  ```java
  if ((number != 0) && (sum/number > 5))
  ```
- Complete evaluation can be achieved by substituting `&` for `&&` or `||` for `||`.

Input and Output of Boolean Values

- Example
  ```java
  boolean booleanVar = false;
  System.out.println(booleanVar);
  System.out.println("Enter a boolean value:");
  Scanner keyboard = new Scanner(System.in);
  booleanVar = keyboard.nextBoolean();
  System.out.println("You entered " + booleanVar);
  ```

Input and Output of Boolean Values

- Dialog
  ```java
  false
  Enter a boolean value: true
  true
  You entered true
  ```
The **switch** Statement

- The **switch** statement is a multiway branch that makes a decision based on an *integral* (integer or character) expression.
  - Java 7 allows String expressions
- The **switch** statement begins with the keyword **switch** followed by an integral expression in parentheses and called the **controlling expression**.

The **switch** Statement

- A list of cases follows, enclosed in braces.
- Each case consists of the keyword **case** followed by
  - A constant called the **case label**
  - A colon
  - A list of statements.
- The list is searched for a case label matching the **controlling expression**.

The **switch** Statement

- The action associated with a matching case label is executed.
- If no match is found, the case labeled **default** is executed.
  - The **default** case is optional, but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.
The switch Statement

- Syntax
  ```java
  switch (Controlling_Expression)
  {
      case Case_Label:
          Statement(s);
          break;
      case Case_Label:
          ...
      default:
          ...
  }
  ```

- View sample program Listing 3.5
  ```java
  class MultipleBirths
  
  Enter number of babies: 1
  Congratulations.
  
  Enter number of babies: 3
  Wow. Triplets.
  
  Enter number of babies: 4
  Unbelievable; 4 babies.
  
  Enter number of babies: 6
  I don't believe you.
  ```

- The action for each case typically ends with the word break.
- The optional break statement prevents the consideration of other cases.
- The controlling expression can be anything that evaluates to an integral type.
Enumerations

- Consider a need to restrict contents of a variable to certain values
- An enumeration lists the values a variable can have
- Example

```java
enum MovieRating {E, A, B}
MovieRating rating;
rating = MovieRating.A;
```

Enumerations

- Now possible to use in a `switch` statement

```java
switch (rating) {
    case E: //Excellent
        System.out.println("You must see this movie!");
        break;
    case A: //Average
        System.out.println("This movie is OK, but not great.");
        break;
    case B: //Bad
        System.out.println("Skip it!");
        break;
    default:
        System.out.println("Something is wrong.");
}
```

Enumerations

- An even better choice of descriptive identifiers for the constants

```java
enum MovieRating
    {EXCELLENT, AVERAGE, BAD}
rating = MovieRating.AVERAGE;
```

case EXCELLENT: ...
Summary

• You have learned about Java branching statements.
• You have learned about the type `boolean`. 