Flow of Control

Chapter 3
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

• A branching statement that chooses between two possible actions.

• Syntax

```java
if (Boolean_Expression)
    Statement_1
else
    Statement_2
```
Semantics of the *if-else* Statement

- Figure 3.2
The **if-else** Statement

• Example

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

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

**Sample Program** Listing 3.1

```
if (balance >= 0) {
    balance = balance + (INTEREST_RATE * balance) / 12;
} else {
    balance = balance - OVERDRAWN_PENALTY;
}
```
The **if**-**else** Statement

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 $–23.53
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`

```
if (Boolean_Expression)
  Statement
```

Evaluate `Boolean_Expression`

- True
  - Execute `Statement`
- False
  - Continue with statement after `Statement`
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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>answer == 'y'</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal to</td>
<td>!</td>
<td>income != tax</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>answer != 'y'</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&gt;</td>
<td>expenses &gt; income</td>
</tr>
<tr>
<td>≥</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>≤</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**
  ```java
  if ((score > 0) && (score <= 100))
  ...
  ```
- **Not allowed**
  ```java
  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
Negating a Boolean Expression

• Figure 3.5 Avoiding the Negation Operator
# Java Logical Operators

- **Figure 3.6**

<table>
<thead>
<tr>
<th>Name</th>
<th>Java Notation</th>
<th>Java Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical <em>and</em></td>
<td><code>&amp;&amp;</code></td>
<td><code>(sum &gt; min) &amp;&amp; (sum &lt; max)</code></td>
</tr>
<tr>
<td>Logical <em>or</em></td>
<td>`</td>
<td></td>
</tr>
<tr>
<td>Logical <em>not</em></td>
<td><code>!</code></td>
<td><code>!(number &lt; 0)</code></td>
</tr>
</tbody>
</table>
Boolean Operators

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

<table>
<thead>
<tr>
<th>Value of A</th>
<th>Value of B</th>
<th>Value of $A &amp;&amp; B$</th>
<th>Value of $A || B$</th>
<th>Value of $! (A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
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 points 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
```

Enter two lines of text:
Java is not coffee.
Java is NOT COFFEE.
The two lines are not equal.
The two lines are not equal.
But the lines are equal, ignoring case.
Lexicographic Order

• Lexicographic order is similar to alphabetical order, but is it 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**
  
  \[\text{String}_1\text{.compareTo(}\text{String}_2\text{)}\]

- **Method `compareTo` returns**
  - a negative number if `String_1` precedes `String_2`
  - zero if the two strings are equal
  - a positive number if `String_2` precedes `String_1`