Chapter Topics

Chapter 3 discusses the following main topics:

- The `if` Statement
- The `if-else` Statement
- Nested `if` statements
- The `if-else-if` Statement
- Logical Operators
- Comparing `String` Objects
Chapter Topics

Chapter 3 discusses the following main topics:

– More about Variable Declaration and Scope
– The Conditional Operator
– The `switch` Statement
– The `printf` Method
– The `DecimalFormat` Class
The `if` Statement

- The `if` statement decides whether a section of code executes or not.
- The `if` statement uses a `boolean` to decide whether the next statement or block of statements executes.

```java
if (boolean expression is true)
    execute next statement.
```
Flowcharts

- If statements can be modeled as a flow chart.

```java
if (coldOutside)
    wearCoat();
```

Is it cold outside?

Yes

Wear a coat.
Flowcharts

- A block if statement may be modeled as:

```java
if (coldOutside) {
    wearCoat();
    wearHat();
    wearGloves();
}
```

Note the use of curly braces to block several statements together.
In most cases, the boolean expression, used by the if statement, uses *relational operators*.

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>is greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>is less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
</tr>
</tbody>
</table>
Boolean Expressions

- A boolean expression is any variable or calculation that results in a true or false condition.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &gt; y</td>
<td>Is x greater than y?</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>Is x less than y?</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>Is x greater than or equal to y?</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>Is x less than or equal to y.</td>
</tr>
<tr>
<td>x == y</td>
<td>Is x equal to y?</td>
</tr>
<tr>
<td>x != y</td>
<td>Is x not equal to y?</td>
</tr>
</tbody>
</table>
if Statements and Boolean Expressions

if (x > y)
    System.out.println("X is greater than Y");

if (x == y)
    System.out.println("X is equal to Y");

if (x != y)
{
    System.out.println("X is not equal to Y");
    x = y;
    System.out.println("However, now it is.");
}

Example: AverageScore.java
Programming Style and if Statements

- An if statement can span more than one line; however, it is still one statement.

```java
if (average > 95)
    grade = 'A';
```

is functionally equivalent to

```java
if (average > 95) grade = 'A';
```
Programming Style and if Statements

- Rules of thumb:
  - The conditionally executed statement should be on the line after the if condition.
  - The conditionally executed statement should be indented one level from the if condition.
  - If an if statement does not have the block curly braces, it is ended by the first semicolon encountered after the if condition.

```c
if (expression)
    statement;
```

No semicolon here.
Semicolon ends statement here.
Block if Statements

- Conditionally executed statements can be grouped into a block by using curly braces `{}` to enclose them.
- If curly braces are used to group conditionally executed statements, the if statement is ended by the closing curly brace.

```java
if (expression)
{
    statement1;
    statement2;
}
```

Curly brace ends the statement.
Block if Statements

- Remember that when the curly braces are not used, then only the next statement after the if condition will be executed conditionally.

```java
if (expression)
    statement1;
    statement2;
    statement3;
```

Only this statement is conditionally executed.
Flags

- A flag is a boolean variable that monitors some condition in a program.
- When a condition is true, the flag is set to true.
- The flag can be tested to see if the condition has changed.
  
```java
if (average > 95)
    highScore = true;
```

- Later, this condition can be tested:
  
```java
if (highScore)
    System.out.println("That’s a high score!");
```
Comparing Characters

• Characters can be tested with relational operators.
• Characters are stored in memory using the Unicode character format.
• Unicode is stored as a sixteen (16) bit number.
• Characters are ordinal, meaning they have an order in the Unicode character set.
• Since characters are ordinal, they can be compared to each other.

```java
char c = 'A';
if(c < 'Z')
    System.out.println("A is less than Z");
```
if-else Statements

• The if-else statement adds the ability to conditionally execute code when the if condition is false.

  \[
  \text{if (expression)} \\
  \quad \text{statementOrBlockIfTrue;} \\
  \text{else} \\
  \quad \text{statementOrBlockIfFalse;}
  \]

• See example: Division.java
if-else Statement Flowcharts

Is it cold outside?

- No: Wear shorts.
- Yes: Wear a coat.
Nested if Statements

- If an if statement appears inside another if statement (single or block) it is called a nested if statement.
- The nested if is executed only if the outer if statement results in a true condition.
- See example: LoanQualifier.java
Nested if Statement Flowcharts

- Is it cold outside?
  - No: Wear shorts.
  - Yes: Is it snowing?
    - No: Wear a jacket.
    - Yes: Wear a parka.
Nested if Statements

```java
if (coldOutside)
{
    if (snowing)
    {
        wearParka();
    }
    else
    {
        wearJacket();
    }
}
else
{
    wearShorts();
}
```
if-else Matching

• Curly brace use is not required if there is only one statement to be conditionally executed.
• However, sometimes curly braces can help make the program more readable.
• Additionally, proper indentation makes it much easier to match up else statements with their corresponding if statement.
Alignment and Nested if Statements

```java
if (coldOutside) {
    if (snowing) {
        wearParka();
    } else {
        wearJacket();
    }
} else {
    wearShorts();
}
```

This if and else go together.
if-else-if Statements

if (expression_1)
{
    statement;
    statement;
    etc.
}
else if (expression_2)
{
    statement;
    statement;
    etc.
}

If expression_1 is true these statements are executed, and the rest of the structure is ignored.

Otherwise, if expression_2 is true these statements are executed, and the rest of the structure is ignored.

Insert as many else if clauses as necessary

else
{
    statement;
    statement;
    etc.
}

These statements are executed if none of the expressions above are true.
if-else-if Statements

• Nested if statements can become very complex.
• The if-else-if statement makes certain types of nested decision logic simpler to write.
• Care must be used since else statements match up with the immediately preceding unmatched if statement.
• See example: TestResults.java
if-else-if Flowchart

- True
  - score < 60
    - False
      - score < 70
        - False
          - score < 80
            - False
              - score < 90
                - False
                  - grade = 'X'
                - True
                  - grade = 'A'
            - True
              - grade = 'C'
        - True
          - grade = 'D'
    - True
      - grade = 'F'
- False
  - score < 90
    - False
      - score < 100
        - False
          - grade = 'X'
        - True
          - grade = 'A'
    - True
      - grade = 'C'
Logical Operators

• Java provides two binary logical operators (&& and ||) that are used to combine boolean expressions.

• Java also provides one unary (!) logical operator to reverse the truth of a boolean expression.
## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>Connects two boolean expressions into one. Both expressions must be true for the overall expression to be true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>NOT</td>
<td>The ! operator reverses the truth of a boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.</td>
</tr>
</tbody>
</table>
The \&\& Operator

- The logical AND operator (\&\&) takes two operands that must both be boolean expressions.
- The resulting combined expression is true if (and only if) both operands are true.
- See example: LogicalAnd.java

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression1 &amp;&amp; Expression2</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
The `||` Operator

- The logical OR operator (`||`) takes two operands that must both be boolean expressions.
- The resulting combined expression is false if (and only if) both operands are false.
- Example: [LogicalOr.java](LogicalOr.java)

| Expression 1 | Expression 2 | Expression1 || Expression2 |
|--------------|--------------|-----------------|
| true         | false        | true            |
| false        | true         | true            |
| false        | true         | true            |
| false        | false        | false           |
| true         | true         | true            |
The ! Operator

- The ! operator performs a logical NOT operation.
- If an expression is true, !expression will be false.

```java
if (!(temperature > 100))
    System.out.println("Below the maximum temperature.");
```

- If temperature > 100 evaluates to false, then the output statement will be run.

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>!Expression1</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Short Circuiting

- Logical AND and logical OR operations perform *short-circuit evaluation* of expressions.
- Logical AND will evaluate to false as soon as it sees that one of its operands is a false expression.
- Logical OR will evaluate to true as soon as it sees that one of its operands is a true expression.
Order of Precedence

- The `!` operator has a higher order of precedence than the `&&` and `||` operators.
- The `&&` and `||` operators have a lower precedence than relational operators like `<` and `>.
- Parenthesis can be used to force the precedence to be changed.
## Order of Precedence

<table>
<thead>
<tr>
<th>Order of Precedence</th>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>* unary negation !</td>
<td>Unary negation, logical NOT</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>Multiplication, Division, Modulus</td>
</tr>
<tr>
<td>3</td>
<td>+ -</td>
<td>Addition, Subtraction</td>
</tr>
<tr>
<td>4</td>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>Less-than, Greater-than, Less-than or equal to, Greater-than or equal to</td>
</tr>
<tr>
<td>5</td>
<td>== !=</td>
<td>Is equal to, Is not equal to</td>
</tr>
<tr>
<td>6</td>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td>7</td>
<td>!</td>
<td>Logical NOT</td>
</tr>
<tr>
<td>8</td>
<td>= += -= *= /= %=</td>
<td>Assignment and combined assignment operators.</td>
</tr>
</tbody>
</table>
Comparing `String` Objects

- In most cases, you cannot use the relational operators to compare two `String` objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true.
- See example: `StringCompare.java`
- See example: `StringCompareTo.java`
Ignoring Case in String Comparisons

- In the `String` class the `equals` and `compareTo` methods are case sensitive.
- In order to compare two `String` objects that might have different case, use:
  - `equalsIgnoreCase`, or
  - `compareToIgnoreCase`
- See example: `SecretWord.java`
Variable Scope

• In Java, a local variable does not have to be declared at the beginning of the method.
• The scope of a local variable begins at the point it is declared and terminates at the end of the method.
• When a program enters a section of code where a variable has scope, that variable has come into scope, which means the variable is visible to the program.
• See example: VariableScope.java
The Conditional Operator

- The *conditional operator* is a ternary (three operand) operator.
- You can use the conditional operator to write a simple statement that works like an *if-else* statement.
The Conditional Operator

• The format of the operators is:

\[ \text{BooleanExpression} \ ? \ \text{Value1} \ : \ \text{Value2} \]

• This forms a conditional expression.
• If \( \text{BooleanExpression} \) is true, the value of the conditional expression is \( \text{Value1} \).
• If \( \text{BooleanExpression} \) is false, the value of the conditional expression is \( \text{Value2} \).
The Conditional Operator

• Example:
  \[ z = x > y ? 10 : 5; \]

• This line is functionally equivalent to:
  ```
  if(x > y)
      z = 10;
  else
      z = 5;
  ```
The Conditional Operator

• Many times the conditional operator is used to supply a value.
  
  \[
  \text{number} = x > y ? 10 : 5;
  \]

• This is functionally equivalent to:
  
  ```java
  if(x > y)
    number = 10;
  else
    number = 5;
  ```

• See example: ConsultantCharges.java
The `switch` Statement

- The `if-else` statement allows you to make true / false branches.
- The `switch` statement allows you to use an ordinal value to determine how a program will branch.
- The `switch` statement can evaluate an `integer` type or `character` type variable and make decisions based on the value.
The switch Statement

• The switch statement takes the form:

```java
switch (SwitchExpression)
{
    case CaseExpression:
        // place one or more statements here
        break;
    case CaseExpression:
        // place one or more statements here
        break;
    // case statements may be repeated
    // as many times as necessary
    default:
        // place one or more statements here
}
```
The `switch` Statement

```
switch (SwitchExpression)
{
    ...
}
```

- The `switch` statement will evaluate the `SwitchExpression`, which can be a `byte`, `short`, `int`, `long`, or `char`. If you are using Java 7, the `SwitchExpression` can also be a string.

- If there is an associated `case` statement that matches that value, program execution will be transferred to that `case` statement.
The `switch` Statement

- Each `case` statement will have a corresponding `CaseExpression` that must be unique.

```java
case CaseExpression:
    // place one or more statements here
    break;
```

- If the `SwitchExpression` matches the `CaseExpression`, the Java statements between the colon and the `break` statement will be executed.
The `case` Statement

- The `break` statement ends the `case` statement.
- The `break` statement is optional.
- If a `case` does not contain a `break`, then program execution continues into the next `case`.
  - See example: `NoBreaks.java`
  - See example: `PetFood.java`
- The `default` section is optional and will be executed if no `CaseExpression` matches the `SwitchExpression`.
- See example: `SwitchDemo.java`
The `printf` Method

- You can use the `System.out.printf` method to perform formatted console output.
- The general format of the method is:

  ```java
  System.out.printf(FormatString, ArgList);
  ```
The `printf` Method

```java
System.out.printf(FormatString, ArgList);
```

**FormatString** is a string that contains text and/or special formatting specifiers.

**ArgList** is optional. It is a list of additional arguments that will be formatted according to the format specifiers listed in the format string.
The `printf` Method

• A simple example:

```java
System.out.printf("Hello World\n");
```
The `printf` Method

- Another example:

```java
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```

```
I worked 40 hours.
```
The `printf` Method

int hours = 40;
System.out.printf("I worked %d hours.\n", hours);

The `%d` format specifier indicates that a decimal integer will be printed.

The contents of the `hours` variable will be printed in the location of the `%d` format specifier.
The `printf` Method

- Another example:
```java
int dogs = 2, cats = 4;
System.out.printf("We have %d dogs and %d cats.\n",
                 dogs, cats);
```

```
We have 2 dogs and 4 cats.
```
The `printf` Method

- Another example:
  ```java
double grossPay = 874.12;
System.out.printf("Your pay is %f.\n", grossPay);
```

```
Your pay is 874.120000.
```
The `printf` Method

- Another example:
  ```java
double grossPay = 874.12;
System.out.printf("Your pay is \%f \n", grossPay);
```

The `%f` format specifier indicates that a floating-point value will be printed.

The contents of the `grossPay` variable will be printed in the location of the `%f` format specifier.
The `printf` Method

• Another example:
  
  ```java
  double grossPay = 874.12;
  System.out.printf("Your pay is \%.2f.\n", grossPay);
  ```
The `printf` Method

- Another example:
  ```java
double grossPay = 874.12;
System.out.printf("Your pay is %.2f.\n", grossPay);
```

The `% . 2f` format specifier indicates that a floating-point value will be printed, rounded to two decimal places.
The `printf` Method

- Another example:
  ```java
double grossPay = 5874.127;
System.out.printf("Your pay is %, .2f.\n", grossPay);
```

The `%,.2f` format specifier indicates that a floating-point value will be printed with comma separators, rounded to two decimal places.
The `printf` Method

- **Another example:**
  ```java
  String name = "Ringo";
  System.out.printf("Your name is %s.\n", name);
  ```

The `%s` format specifier indicates that a string will be printed.
The `printf` Method

- Specifying a field width:

```java
int number = 9;
System.out.printf("The value is %6d\n", number);
```

The `%6d` format specifier indicates the integer will appear in a field that is 6 spaces wide.
The `printf` Method

- Another example:

```java
double number = 9.76891;
System.out.printf("The value is %6.2f\n", number);
```

The `%6.2f` format specifier indicates the number will appear in a field that is 6 spaces wide, and be rounded to 2 decimal places.
The `printf` Method

- See examples:
  - Columns.java
  - CurrencyFormat.java
The `DecimalFormat` Class

- When printing out `double` and `float` values, the full fractional value will be printed.
- The `DecimalFormat` class can be used to format these values.
- In order to use the `DecimalFormat` class, the following `import` statement must be used at the top of the program:
  
  ```java
  import java.text.DecimalFormat;
  ```

- See examples:
  
  [Format1.java](#), [Format2.java](#), [Format3.java](#), [Format4.java](#)