Chapter 3: Program Statements

Now we will examine some other program statements.

Chapter 3 focuses on:

- program development stages
- the flow of control through a method
- decision-making statements
- expressions for making complex decisions
- repetition statements
- drawing with conditionals and loops

Program Development

- The creation of software involves four basic activities:
  - establishing the requirements
  - creating a design
  - implementing the code
  - testing the implementation

- The development process is much more involved than this, but these are the four basic development activities.

Requirements

- Software requirements specify the tasks a program must accomplish (what to do, not how to do it).
- They often include a description of the user interface.
- An initial set of requirements often are provided, but usually must be critiqued, modified, and expanded.
- Often it is difficult to establish detailed, unambiguous, complete requirements.
- Careful attention to the requirements can save significant time and expense in the overall project.
### Design

- A *software design* specifies how a program will accomplish its requirements.
- A design includes one or more *algorithms* to accomplish its goal.
- An *algorithm* is a step-by-step process for solving a problem.
- An algorithm may be expressed in *pseudocode*, which is code-like, but does not necessarily follow any specific syntax.
- In object-oriented development, the design establishes the classes, objects, methods, and data that are required.

### Testing

- A program should be executed multiple times with various input in an attempt to find errors.
- *Debugging* is the process of discovering the causes of problems and fixing them.
- Programmers often think erroneously that there is "only one more bug" to fix.
- Tests should consider design details as well as overall requirements.

### Implementation

- *Implementation* is the process of translating a design into source code.
- Most novice programmers think that writing code is the heart of software development, but actually it should be the least creative step.
- Almost all important decisions are made during requirements and design stages.
- Implementation should focus on coding details, including style guidelines and documentation.

### Flow of Control

- Unless specified otherwise, the order of statement execution through a method is linear: one statement after the other in sequence.
- Some programming statements modify that order, allowing us to:
  - decide whether or not to execute a particular statement, or
  - perform a statement over and over, repetitively
- These decisions are based on a *boolean expression* (also called a *condition*) that evaluates to true or false.
- The order of statement execution is called the *flow of control*.
Conditional Statements

- A conditional statement lets us choose which statement will be executed next.
- Therefore they are sometimes called selection statements.
- Conditional statements give us the power to make basic decisions.
- Some conditional statements in Java are:
  - the if statement
  - the if-else statement

The if Statement

- An example of an if statement:

```java
if (sum > MAX)
    delta = sum - MAX;
System.out.println("The sum is " + sum);
```

First, the condition is evaluated. The value of sum is either greater than the value of MAX, or it is not.

- If the condition is true, the assignment statement is executed.
- If it is not, the assignment statement is skipped.

Either way, the call to println is executed next.

- See Age.java (page 130)

Logic of an if statement

- The if statement has the following syntax:

```java
if (condition) {
    statement;
}
```

- The condition must be a boolean expression.
- It must evaluate to either true or false.

- If the condition is true, the statement is executed.
- If it is false, the statement is skipped.
Boolean Expressions

- A condition often uses one of Java's *equality operators* or *relational operators*, which all return *boolean results*:

  - `==` equal to
  - `!=` not equal to
  - `<` less than
  - `>` greater than
  - `<=` less than or equal to
  - `>=` greater than or equal to

- Note the difference between the equality operator (`==`) and the assignment operator (`=`).

The if-else Statement

- An *else clause* can be added to an *if* statement to make an *if-else statement*:

  ```java
  if ( condition )
      statement1;
  else
      statement2;
  ```

- If the *condition is true*, `statement1` is executed; if the condition is false, `statement2` is executed.
- One or the other will be executed, but not both.
- See `Wages.java` (page 134)

Block Statements

- Several statements can be grouped together into a *block statement*:

  ```java
  { ... }
  ```

- A block is delimited by braces (`{ ... }`)
- A block statement can be used wherever a statement is called for by the Java syntax
- For example, in an *if-else* statement, the *if* portion, or the *else* portion, or both, could be block statements
- See `Guessing.java` (page 136)
Nested if Statements

- The statement executed as a result of an if statement or else clause could be another if statement.
- These are called nested if statements.
- See MinOfThree.java (page 138).
- An else clause is matched to the last unmatched if (no matter what the indentation implies).
- Braces can be used to specify the if statement to which an else clause belongs.

Logical Operators

- Boolean expressions can use the following logical operators:
  - ! Logical NOT
  - && Logical AND
  - || Logical OR
- They all take boolean operands and produce boolean results.
- Logical NOT is a unary operator (it operates on one operand).
- Logical AND and logical OR are binary operators (each operates on two operands).

Logical NOT

- The logical NOT operation is also called logical negation or logical complement.
- If some boolean condition a is true, then !a is false; if a is false, then !a is true.
- Logical expressions can be shown using truth tables.

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

Logical AND and Logical OR

- The logical AND expression
  
  \[ a \&\& b \]
  
  is true if both a and b are true, and false otherwise.
- The logical OR expression
  
  \[ a \|\| b \]
  
  is true if a or b or both are true, and false otherwise.
Truth Tables

- A truth table shows the possible true/false combinations of the terms.
- Since && and || each have two operands, there are four possible combinations of conditions a and b.

| a   | b   | a && b | a || b |
|-----|-----|--------|--------|
| true| true| true   | true   |
| true| false| false  | true   |
| false| true| false  | true   |
| false| false| false  | false  |

Logical Operators

- Conditions can use logical operators to form complex expressions.
  
  ```java
  if (total < MAX && total/count > MAX)
  System.out.println("Testing.");
  ```

- Logical operators have precedence relationships among themselves and with other operators.
  - all logical operators have lower precedence than the relational or arithmetic operators.
  - logical NOT has higher precedence than logical AND and logical OR.

Short Circuited Operators

- The processing of logical AND and logical OR is "short-circuited".
- If the left operand is sufficient to determine the result, the right operand is not evaluated.
  
  ```java
  if (count != 0 && total/count > MAX)
  System.out.println("Testing.");
  ```

- This type of processing must be used carefully.

Truth Tables

- Specific expressions can be evaluated using truth tables.

<table>
<thead>
<tr>
<th>total &lt; MAX</th>
<th>Found</th>
<th>!Found</th>
<th>total &lt; MAX &amp;&amp; !Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Comparing Characters

- We can use the relational operators on character data
- The results are based on the Unicode character set
- The following condition is true because the character + comes before the character J in the Unicode character set:

  ```java
  if ('+' < 'J')
      System.out.println("+ is less than J");
  ```

- The uppercase alphabet (A-Z) followed by the lowercase alphabet (a-z) appear in alphabetical order in the Unicode character set.

Lexicographic Ordering

- Because comparing characters and strings is based on a character set, it is called a lexicographic ordering
- This is not strictly alphabetical when uppercase and lowercase characters are mixed
- For example, the string "Great" comes before the string "fantastic" because all of the uppercase letters come before all of the lowercase letters in Unicode
- Also, short strings come before longer strings with the same prefix (lexicographically)
- Therefore, "book" comes before "bookcase"

Comparing Strings

- Remember that a character string in Java is an object
- We cannot use the relational operators to compare strings
- The equals method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The String class also contains a method called compareTo to determine if one string comes before another (based on the Unicode character set)

Comparing Float Values

- We also have to be careful when comparing two floating point values (float or double) for equality
- You should rarely use the equality operator (==) when comparing two floats
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
- Therefore, to determine the equality of two floats, you may want to use the following technique:

  ```java
  if (Math.abs(f1 - f2) < 0.00001)
      System.out.println("Essentially equal.");
  ```
More Operators

➢ To round out our knowledge of Java operators, let’s examine a few more

➢ In particular, we will examine
  • the increment and decrement operators
  • the assignment operators

Increment and Decrement

➢ The increment and decrement operators are arithmetic and operate on one operand

➢ The increment operator (++) adds one to its operand

➢ The decrement operator (--) subtracts one from its operand

➢ The statement
  
  count++;

  is functionally equivalent to

  count = count + 1;

Assignment Operators

➢ Often we perform an operation on a variable, and then store the result back into that variable

➢ Java provides assignment operators to simplify that process

➢ For example, the statement

  num += count;

  is equivalent to

  num = num + count;

Assignment Operators

➢ There are many assignment operators, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>
Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore
  \[\text{result} /= (\text{total-MIN}) \% \text{num};\]
  is equivalent to
  \[\text{result} = \text{result} / ((\text{total-MIN}) \% \text{num});\]

Repetition Statements

- Repetition statements allow us to execute a statement multiple times
- Often they are referred to as loops
- Like conditional statements, they are controlled by boolean expressions
- The text covers two kinds of repetition statements:
  • the while loop
  • the for loop
- The programmer should choose the right kind of loop for the situation

The while Statement

- The while statement has the following syntax:
  \[
  \text{while ( condition )} \\
  \text{statement;}
  \]
  If the condition is true, the statement is executed. Then the condition is evaluated again.
  The statement is executed repeatedly until the condition becomes false.
Logic of a while Loop

The while Statement

- Note that if the condition of a while statement is false initially, the statement is never executed.
- Therefore, the body of a while loop will execute zero or more times.
- See Counter.java (page 147)
- See Average.java (page 148)
  - A sentinel value indicates the end of the input
  - The variable `sum` maintains a running sum.
- See WinPercentage.java (page 151)
  - A loop is used to validate the input, making the program more robust.

Infinite Loops

- The body of a while loop eventually must make the condition false.
- If not, it is an infinite loop, which will execute until the user interrupts the program.
- This is a common logical error.
- You should always double check to ensure that your loops will terminate normally.
- See Forever.java (page 152)

Nested Loops

- Similar to nested if statements, loops can be nested as well.
- That is, the body of a loop can contain another loop.
- Each time through the outer loop, the inner loop goes through its full set of iterations.
- See PalindromeTester.java (page 155)
Iterators

- An iterator is an object that has methods that allow you to process a collection of items one at a time.
- The hasNext and next methods are used to loop through the collection:
  ```java
  while (myCollection.hasNext())
  {
    System.out.println(myCollection.next());
  }
  ``
- Several classes in the Java class library define iterator objects, including Scanner.
- See URLDissector.java (page 158)

The for Statement

- A for loop is functionally equivalent to the following while loop structure:
  ```java
  initialization;
  while (condition)
  {
    statement;
    increment;
  }
  ``

Logic of a for loop

- The for statement has the following syntax:
  ```java
  for (initialization; condition; increment)
  {
    statement;
  }
  ```
- The increment portion is executed at the end of each iteration.
- The condition-statement-increment cycle is executed repeatedly.
- The initialization is executed once before the loop begins.
- The statement is executed until the condition becomes false.

Reserved word
The for Statement

- Like a while loop, the condition of a for statement is tested prior to executing the loop body.
- Therefore, the body of a for loop will execute zero or more times.
- It is well suited for executing a loop a specific number of times that can be determined in advance.
- See Counter2.java (page 161)
- See Multiples.java (page 163)
- See Stars.java (page 165)

Iterators and for Loops

- A variation of the for loop, called the foreach loop, allows us to process collections just like iterators, but without the complicated syntax.
- If bookList is an iterator object that manages Book objects, we can do the following:
  ```java
  for (Book myBook : bookList)
  {
      System.out.println(myBook);
  }
  ```
- See IceCreamShop.java (page 167)

Choosing a Loop Structure

- When you can't determine how many times you want to execute the loop body, use a while statement.
- If you can determine how many times you want to execute the loop body, use a for statement.

The for Statement

- Each expression in the header of a for loop is optional:
  - If the initialization is left out, no initialization is performed.
  - If the condition is left out, it is always considered to be true, and therefore creates an infinite loop.
  - If the increment is left out, no increment operation is performed.
- Both semi-colons are always required in the for loop header.
Program Development

- We now have several additional statements and operators at our disposal
- Following proper development steps is important
- Suppose you were given some initial requirements:
  - accept a series of test scores
  - compute the average test score
  - determine the highest and lowest test scores
  - display the average, highest, and lowest test scores

Program Development

- Requirements Analysis – clarify and flesh out specific requirements
  - How much data will there be?
  - How should data be accepted?
  - Is there a specific output format required?
- After conferring with the client, we determine:
  - the program must process an arbitrary number of test scores
  - the program should accept input interactively
  - the average should be presented to two decimal places
- The process of requirements analysis may take a long time

Program Development

- Design – determine a possible general solution
  - Input strategy? (Sentinel value?)
  - Calculations needed?
- An initial algorithm might be expressed in pseudocode
- Multiple versions of the solution might be needed to refine it
- Alternatives to the solution should be carefully considered

Program Development

- Implementation – translate the design into source code
- Make sure to follow coding and style guidelines
- Implementation should be integrated with compiling and testing your solution
- This process mirrors a more complex development model we'll eventually need to develop more complex software
- The result is a final implementation
- See ExamGrades.java (page 170)
Program Development

- Testing – attempt to find errors that may exist in your programmed solution
- Compare your code to the design and resolve any discrepancies
- Determine test cases that will stress the limits and boundaries of your solution
- Carefully retest after finding and fixing an error

Summary

- Chapter 3 has focused on:
  - program development stages
  - the flow of control through a method
  - decision-making statements
  - expressions for making complex decisions
  - repetition statements
  - drawing with conditionals and loops

More Drawing Techniques

- Conditionals and loops can greatly enhance our ability to control graphics
- See Bullseye.java (page 173)
- See Boxes.java (page 175)
- See BarHeights.java (page 177)