Chapter 3: Program Statements

Presentation slides for

Java Software Solutions

Foundations of Program Design
Second Edition

by John Lewis and William Loftus

Java Software Solutions is published by Addison Wesley Longman

Program Statements

- We will now examine some other program statements
- Chapter 3 focuses on:
  - the flow of control through a method
  - decision-making statements
  - operators for making complex decisions
  - repetition statements
  - software development stages
  - more drawing techniques

Flow of Control

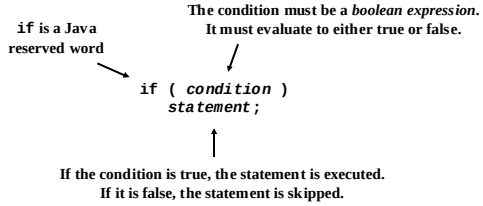
- Unless indicated otherwise, the order of statement execution through a method is linear: one after the other in the order they are written
- Some programming statements modify that order, allowing us to:
  - decide whether or not to execute a particular statement, or
  - perform a statement over and over repetitively
- 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
- Java's conditional statements are the *if statement*, the *if-else statement*, and the *switch statement*

The if Statement

- The *if statement* has the following syntax:



The if Statement

- An example of an if statement:

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

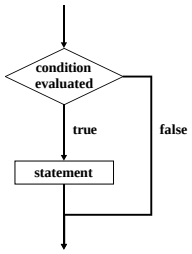
First, the condition is evaluated. The value of `sum` is either greater than the value of `MAX`, or it is not.

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 112)

Logic of an if statement



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 (=)

8

The if-else Statement

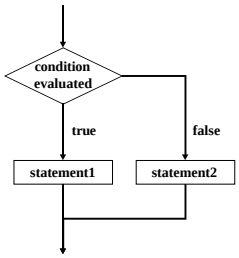
- An *else clause* can be added to an if statement to make it an *if-else statement*:

```
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 116)

9

Logic of an if-else statement



Block Statements

- Several statements can be grouped together into a *block statement*
- A block is delimited by braces ({ … })
- A block statement can be used wherever a statement is called for in 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 117)

11

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 118)
- An else clause is matched to the last unmatched if (no matter what the indentation implies)

12

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 Unicode:

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

- The uppercase alphabet (A-Z) and the lowercase alphabet (a-z) both appear in alphabetical order in Unicode

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 on a string 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 alphabetically (as determined by the Unicode character set)

Comparing Floating Point 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:

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

The switch Statement

- The *switch statement* provides another means to decide which statement to execute next
- The switch statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement list associated with the first value that matches

16

The switch Statement

- The general syntax of a switch statement is:

switch and case are reserved words

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case value3 :
      statement-list3
   case  ...

}
```

If *expression* matches *value2*, control jumps to here

The switch Statement

- Often a *break statement* is used as the last statement in each case's statement list
- A break statement causes control to transfer to the end of the switch statement
- If a break statement is not used, the flow of control will continue into the next case
- Sometimes this can be helpful, but usually we only want to execute the statements associated with one case

The switch Statement

- A switch statement can have an optional *default case* as the last case in the statement
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to it if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch

The switch Statement

- The expression of a switch statement must result in an *integral data type*, like an integer or character; it cannot be a floating point value
- Note that the implicit boolean condition in a switch statement is equality - it tries to match the expression with a value
- You cannot perform relational checks with a switch statement
- See GradeReport.java (page 121)

Logical Operators

- Boolean expressions can also 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 has one operand), but logical AND and logical OR are binary operators (they each have two operands)

21

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*

a	!a
true	false
false	true

22

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

23

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 true and false

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 in selection statements and loops can use logical operators to form complex expressions

```
if (total < MAX && !found)
   System.out.println ("Processing…");
```

- Logical operators have precedence relationships between themselves and other operators

25

Truth Tables

- Specific expressions can be evaluated using truth tables

total < MAX	found	!found	total < MAX && !found
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

26

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
  - assignment operators
  - conditional operator

27

Increment and Decrement Operators

- 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 essentially equivalent to

```
count = count + 1;
```

28

Increment and Decrement Operators

- The increment and decrement operators can be applied in *prefix form* (before the variable) or *postfix form* (after the variable)
- When used alone in a statement, the prefix and postfix forms are basically equivalent. That is,

```
count++;
```

is equivalent to

```
++count;
```

29

Increment and Decrement Operators

- When used in a larger expression, the prefix and postfix forms have a different effect
- In both cases the variable is incremented (decremented)
- But the value used in the larger expression depends on the form:

Expression	Operation	Value of Expression
count++	add 1	old value
++count	add 1	new value
count--	subtract 1	old value
--count	subtract 1	new value

30

Increment and Decrement Operators

- If `count` currently contains 45, then

```
total = count++;
```

  assigns 45 to `total` and 46 to `count`

- If `count` currently contains 45, then

```
total = ++count;
```

  assigns the value 46 to both `total` and `count`

31

Assignment Operators

- Often we perform an operation on a variable, 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;
```

32

Assignment Operators

- There are many assignment operators, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

33

Assignment Operators

- The right hand side of an assignment operator can be a complete expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

  is equivalent to

```
result = result / ((total-MIN) % num);
```

34

The Conditional Operator

- Java has a *conditional operator* that evaluates a boolean condition that determines which of two other expressions is evaluated
- The result of the chosen expression is the result of the entire conditional operator
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated

35

The Conditional Operator

- The conditional operator is similar to an if-else statement, except that it is an expression that returns a value
- For example:

```
larger = (num1 > num2) ? num1 : num2;
```

- If `num1` is greater that `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
- The conditional operator is *ternary*, meaning that it requires three operands

36

The Conditional Operator

- Another example:

```
System.out.println ("Your change is " + count +
   (count == 1) ? "Dime" : "Dimes");
```

- If `count` equals 1, then `"Dime"` is printed
- If `count` is anything other than 1, then `"Dimes"` is printed

37

Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times repetitively
- They are often simply referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements: the *while loop*, the *do loop*, and the *for loop*
- The programmer must choose the right kind of loop for the situation

The while Statement

- The *while statement* has the following syntax:

`while` is a reserved word

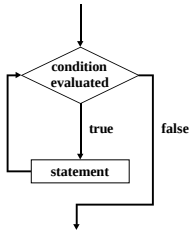
```
while ( condition )
   statement;
```

If the condition is true, the statement is executed. Then the condition is evaluated again.

The statement is executed repetitively until the condition becomes false.

39

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 133)
- See Average.java (page 134)
- See WinPercentage.java (page 136)

41

Infinite Loops

- The body of a while loop must eventually make the condition false
- If not, it is an *infinite loop*, which will execute until the user interrupts the program
- See Forever.java (page 138)
- This is a common type of logical error
- You should always double check to ensure that your loops will terminate normally

42

Nested Loops

- Similar to nested if statements, loops can be nested as well
- That is, the body of a loop could contain another loop
- Each time through the outer loop, the inner loop will go through its entire set of iterations
- See PalindromeTester.java (page 137)

The do Statement

- The *do statement* has the following syntax:

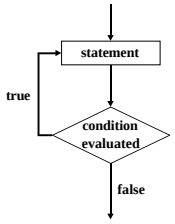
Uses both the `do` and `while` reserved words

```
do
{
   statement;
}
while ( condition )
```

The statement is executed once initially, then the condition is evaluated

The statement is repetitively executed until the condition becomes false

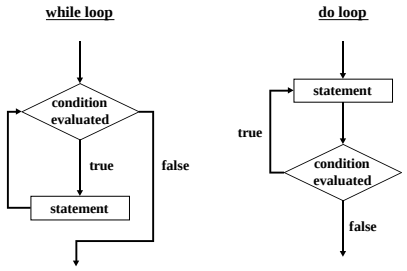
Logic of a do loop



The do Statement

- A do loop is similar to a while loop, except that the condition is evaluated after the body of the loop is executed
- Therefore the body of a do loop will execute at least one time
- See Counter2.java (page 143)
- See ReverseNumber.java (page 144)

Comparing the while and do loops



The for Statement

- The *for statement* has the following syntax:

Reserved word

The *initialization* portion is executed once before the loop begins

The statement is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

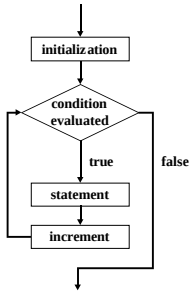
The *increment* portion is executed at the end of each iteration

The for Statement

- A for loop is equivalent to the following while loop structure:

```
initialization;
while ( condition )
{
   statement;
   increment;
}
```

Logic of a for loop



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 specific number of times that can be determined in advance
- See Counter3.java (page 146)
- See Multiples.java (page 147)
- See Stars.java (page 150)

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

- 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 basic steps are a good starting point

53

Requirements

- *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 are often provided, but usually must be critiqued, modified, and expanded
- It is often difficult to establish detailed, unambiguous, complete requirements
- Careful attention to the requirements can save significant time and money in the overall project

54

Design

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

55

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 it actually should be the least creative step
- Almost all important decisions are made during requirements analysis and design
- Implementation should focus on coding details, including style guidelines and documentation
- See ExamGrades.java (page 155)

56

Testing

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

57

More Drawing Techniques

- Conditionals and loops can greatly enhance our ability to control graphics
- See Bullseye.java (page 157)
- See Boxes.java (page 159)
- See BarHeights.java (page 162)