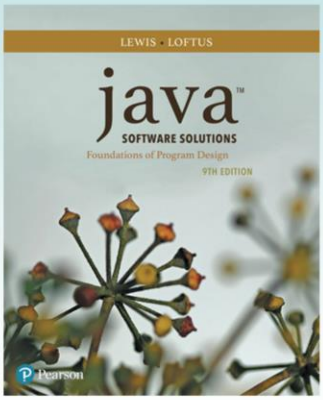
Chapter 5
Conditionals and Loops



Java Software Solutions
Foundations of Program Design
9th Edition

John Lewis
William Loftus

Outline

- Boolean Expressions
- The `if` Statement
- Comparing Data
- The `while` Statement
- Iterators
- The `ArrayList` Class

Comparing Data

- When comparing data using boolean expressions, it's important to understand the nuances of certain data types
- Let's examine some key situations:
  - Comparing floating point values for equality
  - Comparing characters
  - Comparing strings (alphabetical order)
  - Comparing object vs. comparing object references

Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (`float` or `double`)
- Two floating point values are equal only if their underlying binary representations match exactly
- Computations often result in slight differences that may be irrelevant
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal

-Consider that repeating values (e.g. 1.0/3.0) are never completely accurate represented in binary

-Slight differences in floating point values can be from round-off or precision errors

Comparing Float Values

- To determine the equality of two floats, use the following technique:

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

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal
- The tolerance could be set to any appropriate level, such as 0.000001

-The values of floats we *think* are the same are often different by some very small value

-As a result, instead of comparing **values**, we compare the **difference of the values**

-We compare this difference to some range or **tolerance**

-Note we compute the difference as the **absolute** difference

Comparing Characters

- As we've discussed, Java character data is based on the Unicode character set
- Unicode establishes a particular numeric value for each character, and therefore an ordering
- We can use relational operators on character data based on this ordering
- For example, the character `'+'` is less than the character `'J'` because it comes before it in the Unicode character set
- Appendix C provides an overview of Unicode

-Unicode ordering means that characters later in the alphabet are “greater than” those earlier

-As a result, we can say that “a is less than b” and “b is greater than a”

-We can use relational operators: a < b, b > a, or a == a

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Objects

- The == operator can be applied to objects – it returns true if the two references are aliases of each other
- The `equals` method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the == operator
- It has been redefined in the `String` class to compare the characters in the two strings
- When you write a class, you can redefine the `equals` method to return true under whatever conditions are appropriate

-Remember that object variables are object **reference** variables

-An object reference variable stores an **address** (or pointer) to where the object lives in memory

-If you compare object variables using the == operator, you compare **only** the address values!

-This just compares where each lives in memory, but does **not** compare their contents!

-Think about what it means, then, to compare two different objects in memory

-Depending on what kind of object it is, the comparison methods might be different

-For example, String object comparisons might mean to compare strings character by character

-Object comparisons for classes **you** develop might mean to compare each instance data value

-For example, recall the Die class from a previous chapter

-Maybe comparing two Die objects might mean to compare their faceValues.

-For this reason, every Java class can implement a method called **equals** to define what a comparison means

-When you develop your own class, writing the equals method defines comparison specifications

-Note if we don't write this method, it defaults to meaning the same thing as ==

Comparing Strings

- Remember that in Java a character string is an object
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The `equals` method returns a boolean result

```
if (name1.equals(name2))
   System.out.println ("Same name");
```

Comparing Strings

- We cannot use the relational operators to compare strings
- The `String` class contains the `compareTo` method for determining if one string comes before another
- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

Comparing Strings

- Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering*

```
int result = name1.comareTo(name2);
if (result < 0)
   System.out.println (name1 + "comes first");
else
   if (result == 0)
      System.out.println ("Same name");
   else
      System.out.println (name2 + "comes first");
```

Lexicographic Ordering

- Lexicographic ordering 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"`

Outline

- Boolean Expressions
- The `if` Statement
- Comparing Data
- The `while` Statement
- Iterators
- The `ArrayList` Class
- Determining Event Sources
- Check Boxes and Radio Buttons

Repetition Statements

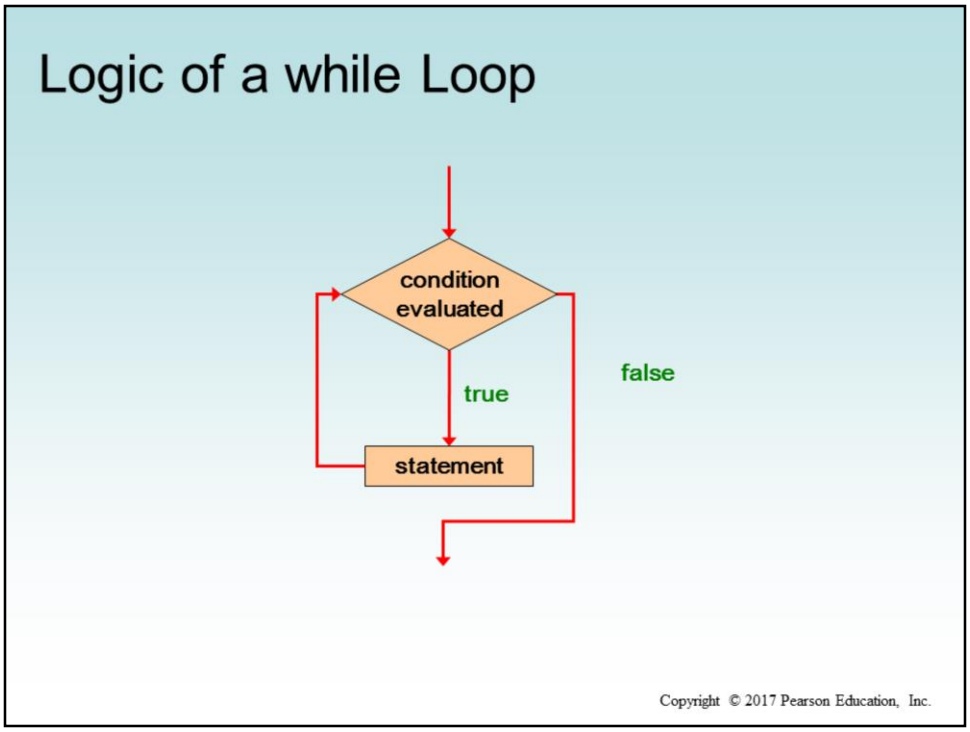
- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements: `while`, `do`, and `for` loops
- The `do` and `for` loops are discussed in Chapter 6

The while Statement

- A *while statement* has the following syntax:

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

- If the **condition** is true, the **statement** is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false



-Note the logic is the same as the if statement with the exception that the process is repeated

-This process is first the testing of the condition then the repetition of the statement if true

-This continues until the condition becomes false

The while Statement

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
   System.out.println (count);
   count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times

-Note we can use braces (as we did with the if statement) to define multiple statements

-Note that variable values defining the condition change within the while statement

Sentinel Values

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum*
- A *sentinel value* is a special input value that represents the end of input
- See `Average.java`

-Note the **sentinel** value in the Average.java program is the variable named “value”

-Note also how the programs checks for a divide by zero condition if nothing is entered

```
//********************************************************************
//  Average.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a while loop, a sentinel value, and a
//  running sum.
//********************************************************************

import java.text.DecimalFormat;
import java.util.Scanner;

public class Average
{
   //-----------------------------------------------------------------
   //  Computes the average of a set of values entered by the user.
   //  The running sum is printed as the numbers are entered.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sum = 0, value, count = 0;
      double average;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter an integer (0 to quit): ");
      value = scan.nextInt();
```

continue

continue

```
while (value != 0)  // sentinel value of 0 to terminate loop
{
   count++;

   sum += value;
   System.out.println ("The sum so far is " + sum);

   System.out.print ("Enter an integer (0 to quit): ");
   value = scan.nextInt();
}
```

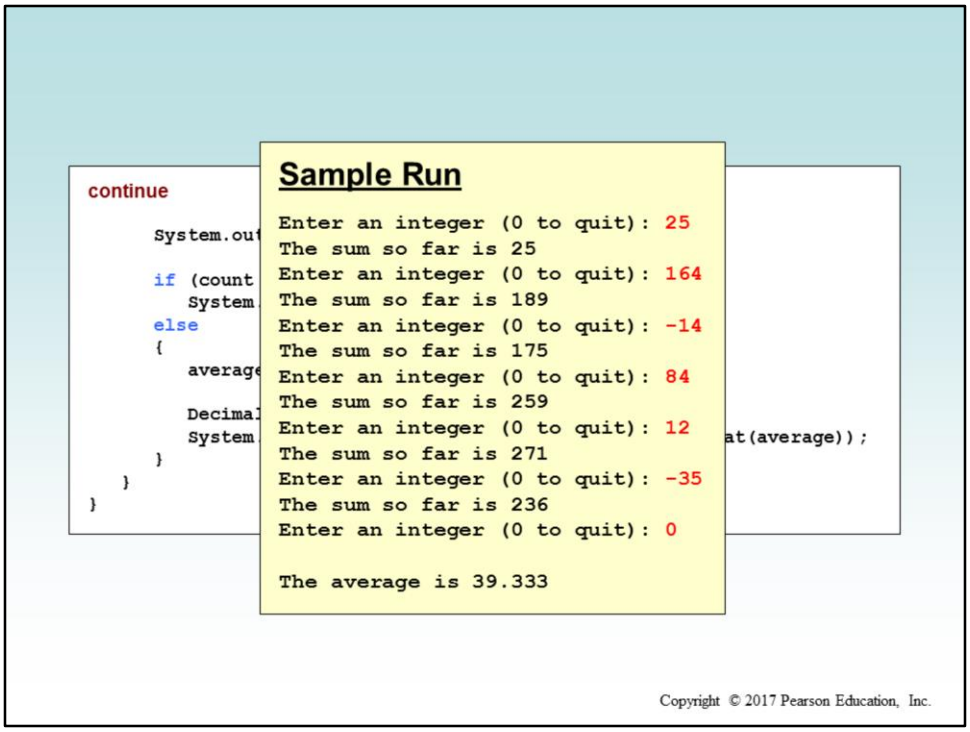
continue

continue

```
      System.out.println ();

      if (count == 0)
         System.out.println ("No values were entered.");
      else
      {
         average = (double)sum / count;

         DecimalFormat fmt = new DecimalFormat ("0.###");
         System.out.println ("The average is " + fmt.format(average));
      }
   }
}
```



Input Validation

- A loop can also be used for *input validation*, making a program more *robust*
- It's generally a good idea to verify that input is valid (in whatever sense) when possible
- See `WinPercentage.java`

```
//********************************************************************
//  WinPercentage.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a while loop for input validation.
//********************************************************************

import java.text.NumberFormat;
import java.util.Scanner;

public class WinPercentage
{
   //-----------------------------------------------------------------
   //  Computes the percentage of games won by a team.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int NUM_GAMES = 12;
      int won;
      double ratio;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the number of games won (0 to "
                        + NUM_GAMES + "): ");
      won = scan.nextInt();
```

continue

continue

```
      while (won < 0 || won > NUM_GAMES)
      {
         System.out.print ("Invalid input. Please reenter: ");
         won = scan.nextInt();
      }

      ratio = (double)won / NUM_GAMES;

      NumberFormat fmt = NumberFormat.getPercentInstance();

      System.out.println ();
      System.out.println ("Winning percentage: " + fmt.format(ratio));
   }
}
```

continue

```
        while
        {
            Sy
            wo
        }

        ratio = (double)won / NUM_GAMES;

        NumberFormat fmt = NumberFormat.getPercentInstance();

        System.out.println ();
        System.out.println ("Winning percentage: " + fmt.format(ratio));
    }
}
```

Sample Run

```
Enter the number of games won (0 to 12): -5
Invalid input. Please reenter: 13
Invalid input. Please reenter: 7

Winning percentage: 58%
```

Infinite Loops

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

-If we don't change the condition to false at some point in the statements, the loop continues indefinitely!

-We call this an **infinite** loop since it will loop forever – the program will never stop

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
   System.out.println (count);
   count = count - 1;
}
```

- This loop will continue executing until interrupted (Control-C) or until an underflow error occurs

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely
- See `PalindromeTester.java`

-When developing nested loops, work out the logic on paper before you begin programming

-Logic diagrams and pre-planning can avoid potential problems before starting coding!

```
//********************************************************************
//  PalindromeTester.java       Author: Lewis/Loftus
//
//  Demonstrates the use of nested while loops.
//********************************************************************

import java.util.Scanner;

public class PalindromeTester
{
   //-----------------------------------------------------------------
   //  Tests strings to see if they are palindromes.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String str, another = "y";
      int left, right;

      Scanner scan = new Scanner (System.in);

      while (another.equalsIgnoreCase("y"))  // allows y or Y
      {
         System.out.println ("Enter a potential palindrome:");
         str = scan.nextLine();

         left = 0;
         right = str.length() - 1;
```

continue

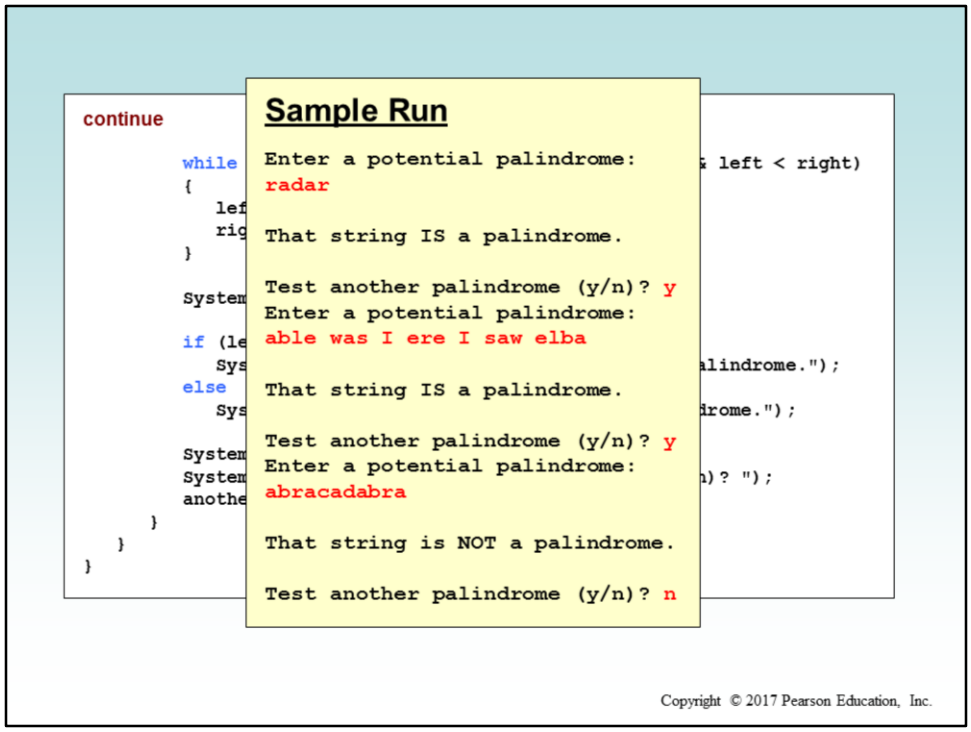
continue

```
         while (str.charAt(left) == str.charAt(right) && left < right)
         {
            left++;
            right--;
         }

         System.out.println();

         if (left < right)
            System.out.println ("That string is NOT a palindrome.");
         else
            System.out.println ("That string IS a palindrome.");

         System.out.println();
         System.out.print ("Test another palindrome (y/n)? ");
         another = scan.nextLine();
      }
   }
}
```



Quick Check

How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 < 20)
   {
      System.out.println ("Here");
      count2++;
   }
   count1++;
}
```

Quick Check

How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 < 20)
   {
      System.out.println ("Here");
      count2++;
   }
   count1++;
}
```

10 * 19 = 190