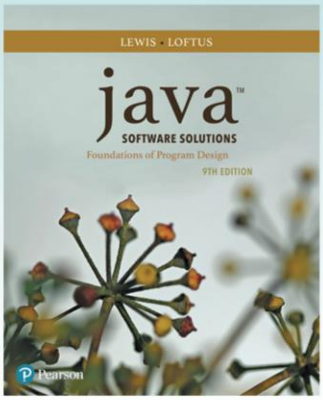
Chapter 5
Conditionals and Loops



Java Software Solutions

Foundations of Program Design

9th Edition

John Lewis
William Loftus

Conditionals and Loops

- Now we will examine programming statements that allow us to:
  - make decisions
  - repeat processing steps in a loop
- Chapter 5 focuses on:
  - boolean expressions
  - the if and if-else statements
  - comparing data
  - while loops
  - Iterators
  - the `ArrayList` class

Outline

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

Flow of Control

- Unless specified otherwise, the order of statement execution through a method is linear: one after another
- Some programming statements allow us to make decisions and perform repetitions
- These decisions are based on *boolean expressions* (also called *conditions*) that evaluate to true or false
- The order of statement execution is called the *flow of control*

-Conditionals and loops allow us to change the flow of control **within** a method

Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next
- They are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- The Java conditional statements are the:
  - `if` and `if-else` statement
  - `switch` statement
- We'll explore the `switch` statement in Chapter 6

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 (=)

-Conditionals are based on a **boolean** expression

-The boolean expression defines the **condition** that decides which statements are executed

-Boolean expressions (conditions) evaluate to one of two results (either true or false)

-Note arithmetic operators have higher order of precedence than equality and relational operators

-Arithmetic operators are evaluated first, then equality/relational operators

Boolean Expressions

- An `if` statement with its boolean condition:

```
if (sum > MAX)
   delta = sum - MAX;
```

- 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 isn't, it is skipped
- See `Age.java`

-The **condition** is (sum > MAX)

-It helps to read such statements as "**if** the sum is greater than the MAX value", **then** execute the following statement

-If this condition evaluates **true**, then the following statement is **executed**

-If this condition evaluates **false**, then the following statement is **skipped**

```java
//*******************************************************************
//  Age.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an if statement.
//*******************************************************************

import java.util.Scanner;

public class Age
{
   //-----------------------------------------------------------------
   //  Reads the user's age and prints comments accordingly.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int MINOR = 21;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter your age: ");
      int age = scan.nextInt();
```

continue

continue

```
      System.out.println ("You entered: " + age);

      if (age < MINOR)
         System.out.println ("Youth is a wonderful thing. Enjoy.");

      System.out.println ("Age is a state of mind.");
   }
}
```

Sample Run

```
Enter your age: 47
You entered: 47
Age is a state of mind.
```

continue

```
        System.out.println ("You entered: " + age);

        if (age < MINOR)
           System.out.println ("Youth is a wonderful thing. Enjoy.");

        System.out.println ("Age is a state of mind.");
    }
}
```

Another Sample Run

```
Enter your age: 12
You entered: 12
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
```

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 operates on one operand)
- Logical AND and logical OR are binary operators (each operates on two operands)

-A boolean operand is an operand that returns either true or false

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 a *truth table*:

`a`	`!a`
true	false
false	true

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

Logical AND and Logical OR

- A truth table shows all 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

- Expressions that use logical operators can form complex conditions

```
if (total < MAX+5 && !found)
   System.out.println ("Processing…");
```

- All logical operators have lower precedence than the relational operators
- The `!` operator has higher precedence than `&&` and `||`

-It helps to read this example as “if total is less than MAX+5 AND not found”

-Saying the word **not** before the variable reminds us to evaluate the **opposite** of the value stored in found

Boolean Expressions

- 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

Short-Circuited Operators

- The processing of `&&` and `||` is "short-circuited"
- If the left operand is sufficient to determine the result, the right operand is not evaluated

```
if (count != 0 && total/count > MAX)
   System.out.println ("Testing.");
```

- This type of processing should be used carefully

Outline

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

The if Statement

- Let's now look at the `if` statement in more detail
- The *if statement* has the following syntax:

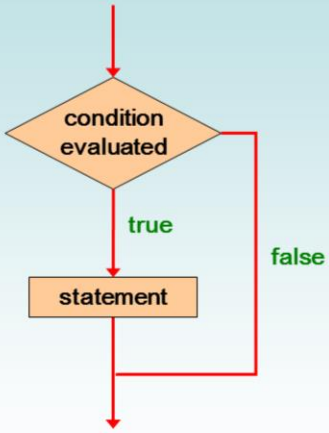
The ***condition*** must be a boolean expression. It must evaluate to either true or false.

`if` is a Java reserved word

```
if ( condition )
   statement;
```

If the ***condition*** is true, the ***statement*** is executed. If it is false, the ***statement*** is skipped.

Logic of an if statement



Indentation

- The statement controlled by the `if` statement is indented to indicate that relationship
- The use of a consistent indentation style makes a program easier to read and understand
- The compiler ignores indentation, which can lead to errors if the indentation is not correct

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding

Quick Check

What do the following statements do?

```
if (total != stock + warehouse)
   inventoryError = true;
```

```
if (found || !done)
   System.out.println("Ok");
```

Quick Check

What do the following statements do?

```
if (total != stock + warehouse)
   inventoryError = true;
```

Sets the boolean variable to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

```
if (found || !done)
   System.out.println("Ok");
```

Prints "Ok" if `found` is true or `done` is false

The if-else Statement

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

```java
//*******************************************************************
//  Wages.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an if-else statement.
//*******************************************************************

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

public class Wages
{
   //-----------------------------------------------------------------
   //  Reads the number of hours worked and calculates wages.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final double RATE = 8.25;  // regular pay rate
      final int STANDARD = 40;   // standard hours in a work week

      Scanner scan = new Scanner (System.in);

      double pay = 0.0;
```

continue

continue

```
      System.out.print ("Enter the number of hours worked: ");
      int hours = scan.nextInt();

      System.out.println ();

      // Pay overtime at "time and a half"
      if (hours > STANDARD)
         pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
      else
         pay = hours * RATE;

      NumberFormat fmt = NumberFormat.getCurrencyInstance();
      System.out.println ("Gross earnings: " + fmt.format(pay));
   }
}
```

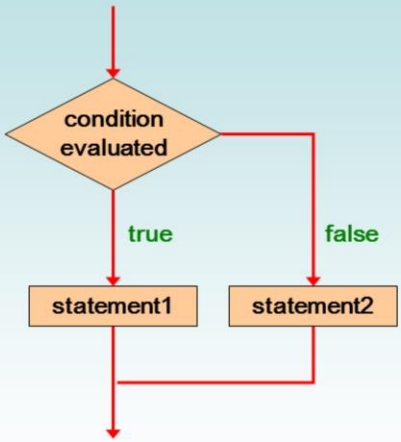
continue

```
      System.o                                                    ");
      int hour

      System.o

      // Pay overtime at "time and a half"
      if (hours > STANDARD)
         pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
      else
         pay = hours * RATE;

      NumberFormat fmt = NumberFormat.getCurrencyInstance();
      System.out.println ("Gross earnings: " + fmt.format(pay));
   }
}
```

Sample Run

```
Enter the number of hours worked: 46

Gross earnings: $404.25
```

Logic of an if-else statement



The Coin Class

- Let's look at an example that uses a class that represents a coin that can be flipped
- Instance data is used to indicate which face (heads or tails) is currently showing
- See `CoinFlip.java`
- See `Coin.java`

```java
//*******************************************************************
//  CoinFlip.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an if-else statement.
//*******************************************************************

public class CoinFlip
{
   //-----------------------------------------------------------------
   //  Creates a Coin object, flips it, and prints the results.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Coin myCoin = new Coin();

      myCoin.flip();

      System.out.println (myCoin);

      if (myCoin.isHeads())
         System.out.println ("You win.");
      else
         System.out.println ("Better luck next time.");
   }
}
```

```
//*****************************************************************
//  CoinFlip.java
//
//  Demonstrates the
//*****************************************************************

public class CoinFlip
{
   //-----------------------------------------------------------------
   //  Creates a Coin object, flips it, and prints the results.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Coin myCoin = new Coin();

      myCoin.flip();

      System.out.println (myCoin);

      if (myCoin.isHeads())
         System.out.println ("You win.");
      else
         System.out.println ("Better luck next time.");
   }
}
```

Sample Run

```
Tails
Better luck next time.
```

```java
//*******************************************************************
//  Coin.java       Author: Lewis/Loftus
//
//  Represents a coin with two sides that can be flipped.
//*******************************************************************

public class Coin
{
   private final int HEADS = 0;
   private final int TAILS = 1;

   private int face;

   //-----------------------------------------------------------------
   //  Sets up the coin by flipping it initially.
   //-----------------------------------------------------------------
   public Coin ()
   {
      flip();
   }
```

continue

continue

```
//-----------------------------------------------------------------
//  Flips the coin by randomly choosing a face value.
//-----------------------------------------------------------------
public void flip ()
{
   face = (int) (Math.random() * 2);
}

//-----------------------------------------------------------------
//  Returns true if the current face of the coin is heads.
//-----------------------------------------------------------------
public boolean isHeads ()
{
   return (face == HEADS);
}
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns the current face of the coin as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String faceName;

      if (face == HEADS)
         faceName = "Heads";
      else
         faceName = "Tails";

      return faceName;
   }
}
```

Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the compiler

```
if (depth >= UPPER_LIMIT)
   delta = 100;
else
   System.out.println("Reseting Delta");
   delta = 0;
```

- Despite what the indentation implies, `delta` will be set to 0 no matter what

Block Statements

- Several statements can be grouped together into a *block statement* delimited by braces
- A block statement can be used wherever a statement is called for in the Java syntax rules

```
if (total > MAX)
{
   System.out.println ("Error!!");
   errorCount++;
}
```


-We block statements between braces when we have more than one line of statements

-It's good practice to use braces even if we only have a single statement:

```
if(sum > MAX)
{
   System.out.println("Sum is greater than MAX");
}
else
{
   System.out.println("Sum is less than or equal to MAX");
}
```

Block Statements

- The `if` clause, or the `else` clause, or both, could govern block statements

```
if (total > MAX)
{
   System.out.println ("Error!!");
   errorCount++;
}
else
{
   System.out.println ("Total: " + total);
   current = total*2;
}
```

- See `Guessing.java`

```java
//*******************************************************************
//  Guessing.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a block statement in an if-else.
//*******************************************************************

import java.util.*;

public class Guessing
{
   //-----------------------------------------------------------------
   //  Plays a simple guessing game with the user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int MAX = 10;
      int answer, guess;

      Scanner scan = new Scanner (System.in);
      Random generator = new Random();

      answer = generator.nextInt(MAX) + 1;
```

continue

continue

```
      System.out.print ("I'm thinking of a number between 1 and "
                        + MAX + ". Guess what it is: ");

      guess = scan.nextInt();

      if (guess == answer)
         System.out.println ("You got it! Good guessing!");
      else
      {
         System.out.println ("That is not correct, sorry.");
         System.out.println ("The number was " + answer);
      }
   }
}
```


-Note if we didn't use braces under the else clause, the third print statement would always print whether the condition is true or false

Sample Run

```
I'm thinking of a number between 1 and 10. Guess what it is: 6
That is not correct, sorry.
The number was 9
```

```
      if (guess == answer)
         System.out.println ("You got it! Good guessing!");
      else
      {
         System.out.println ("That is not correct, sorry.");
         System.out.println ("The number was " + answer);
      }
   }
}
```

Nested if Statements

- The statement executed as a result of an `if` or `else` clause could be another `if` statement
- These are called *nested if statements*
- 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
- See `MinOfThree.java`

```java
//*******************************************************************
//  MinOfThree.java       Author: Lewis/Loftus
//
//  Demonstrates the use of nested if statements.
//*******************************************************************

import java.util.Scanner;

public class MinOfThree
{
   //-----------------------------------------------------------------
   //  Reads three integers from the user and determines the smallest
   //  value.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int num1, num2, num3, min = 0;

      Scanner scan = new Scanner (System.in);

      System.out.println ("Enter three integers: ");
      num1 = scan.nextInt();
      num2 = scan.nextInt();
      num3 = scan.nextInt();
```

continue

continue

```
      if (num1 < num2)
         if (num1 < num3)
            min = num1;
         else
            min = num3;
      else
         if (num2 < num3)
            min = num2;
         else
            min = num3;

      System.out.println ("Minimum value: " + min);
   }
}
```

continue

```
      if (num1 < num2)
         if (num1 < num3)
            min = num1;
         else
            min = num3;
      else
         if (num2 < num3)
            min = num2;
         else
            min = num3;

      System.out.println ("Minimum value: " + min);
   }
}
```

Sample Run

```
Enter three integers:
84   69   90
Minimum value: 69
```


-Using braces in the example above can help the readability of such nested if statements

-It also helps follow the logic and flow of control