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
- conditional statements
- comparing data
- repetition statements
- iterators
- more drawing techniques
- more GUI components
Outline

The if Statement and Conditions
Other Conditional Statements
Comparing Data
The while Statement
Iterators
Other Repetition Statements
Decisions and Graphics
More Components

Flow of Control

• Unless specified otherwise, the order of statement execution through a method is linear: one statement after another in sequence

• Some programming statements allow us to:
  ▪ decide whether or not to execute a particular statement (branching statements)
  ▪ execute a statement over and over (iterative/loop statements)

• These decisions are based on boolean expressions (or conditions) that evaluate 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 (branching) statements.
- Conditional statements give us the power to make basic decisions.
- The Java conditional statements are the:
  - if statement
  - if-else statement
  - switch statement

The 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.
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 (=)
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 isn't, it is skipped.

• Either way, the call to println is executed next

• See Age.java (page 216)

The if Statement

• What do the following statements do?
  
  ```java
  if (top >= MAXIMUM)
      top = 0;
  ```

  Sets top to zero if the current value of top is greater than or equal to the value of MAXIMUM

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

  Sets a flag to true if the value of total is not equal to the sum of stock and warehouse

• The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators
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
• Although it makes no difference to the compiler, proper indentation is crucial
  "Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."
  -- Martin Golding

Indentation Revisited

• Remember that indentation is for the human reader, and is ignored by the computer

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

    Despite what is implied by the indentation, the increment will occur whether the condition is true or not
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:
  
  ```java
  if (total > MAX)
  {
    System.out.println("Error!!");
    errorCount++;
  }
  ```

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 expressions can be shown using a truth table.
- 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 \( \neg a \) is false; if \( a \) is false, then \( \neg a \) is true

<table>
<thead>
<tr>
<th>( a )</th>
<th>( \neg 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
  \[ a \mid\mid b \]
  is true if \( a \) or \( b \) or both are true, and false otherwise
Logical Operators

• 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 |

Expressions that use logical operators can form complex conditions

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

• All logical operators have lower precedence than the relational operators

• Logical NOT has higher precedence than logical AND and logical OR
Boolean Expressions

- 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>

Outline

- The if Statement and Conditions
- Other Conditional Statements
- Comparing Data
- The while Statement
- Iterators
- Other Repetition Statements
- Decisions and Graphics
- More Components
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

Wages.java (Segment)

```java
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) {
      //set the hourly rate and get hours worked
      ...
      // Pay overtime at "time and a half"
      if (hours > STANDARD) 
          pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5); // pay
      else
          pay = hours * RATE; //pay

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

© 2007 Pearson Addison-Wesley. All rights reserved
Logic of an if-else statement

Block Statements

- In an if-else statement, the if portion, or the else portion, or both, could be block statements

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

- See Guessing.java (page 223)
import java.util.*;
… //class and main method headers

final int MAX = 10;
int answer, guess;

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

answer = generator.nextInt(MAX) + 1;

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);

…

The Conditional Operator

• Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated

• Its syntax is:

   \[ \text{condition} \ ? \text{expression1} : \text{expression2} \]

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

• The value of the entire conditional operator is the value of the selected expression
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 than num2, then num1 is assigned to larger; otherwise, num2 is assigned to larger

- The conditional operator is ternary because it requires three operands

- 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
Nested if Statements

1. The statement executed as a result of an `if` statement or `else` clause could be another `if` statement
2. These are called *nested if statements*
3. See MinOfThree.java (page 227)

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

Nested if Statements

1. An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)
   ```java
   if (code=='R')
       if (height <= 20)
           System.out.println("Situation Normal");
       else
           System.out.println("Bravo!");
   ```
2. *Braces* can be used to specify the `if` statement to which an `else` clause belongs
   ```java
   if (code=='R')
       if (height <= 20)
           System.out.println("Situation Normal");
   else
       System.out.println("Bravo!");
   ```
The switch Statement

- The *switch statement* provides another way 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 associated with the first case value that matches.

The switch Statement

- The general syntax of a *switch* statement is:

  ```java
  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 may be appropriate, but often we want to execute only the statements associated with one case.

The switch Statement

- An example of a switch statement:
  ```java
  switch (option) {
  case 'A':
    aCount++;
    break;
  case 'B':
    bCount++;
    break;
  case 'C':
    cCount++;
    break;
  }
  ```
The switch Statement

- A `switch` statement can have an optional `default` case
- 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 type`, meaning an integer (`byte, short, int, long`) or a `char`
- It cannot be a `boolean` value or a floating point value (`float` or `double`)
- The implicit boolean condition in a `switch` statement is equality
- You cannot perform relational checks with a `switch` statement
- See `GradeReport.java` (page 233)
GradeReport.java (segment)

category = grade / 10;
System.out.print ("That grade is ");
switch (category)
{
   case 10:
      System.out.println ("a perfect score. Well done.");
      break;
   case 9:
      System.out.println ("well above average. Excellent.");
      break;
   case 8:
      System.out.println ("above average. Nice job.");
      break;
   case 7:
      System.out.println ("average.");
      break;
   case 6:
      System.out.println ("below average. You should see the");
      System.out.println ("instructor to clarify the material "+"presented in class.");
      break;
   default:
      System.out.println ("not passing.");
}

Outline

The if Statement and Conditions
Other Conditional Statements
Comparing Data
The while Statement
Iterators
Other Repetition Statements
Decisions and Graphics
More Components
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
Comparing Float Values

- To determine the equality of two floats, you may want to use the following technique:
  
  ```java
  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.

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.
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

<table>
<thead>
<tr>
<th>Characters</th>
<th>Unicode Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 9</td>
<td>48 through 57</td>
</tr>
<tr>
<td>A – Z</td>
<td>65 through 90</td>
</tr>
<tr>
<td>a – z</td>
<td>97 through 122</td>
</tr>
</tbody>
</table>

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

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

• We cannot use the relational operators to compare strings

• The String class contains a method called compareTo to determine 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

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

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

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
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:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation
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

Logic of a while Loop

- **Condition evaluated**
  - **true**
  - **false**

- **Statement**
The while Statement

- An example of a while statement:
  ```java
  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

The while Statement

- 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](#) (page 237)
Average.java (main method)

```java
// 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();
    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();
    }
    if (count == 0)
        System.out.println ("No values were entered.");
    else
    {
        average = (double)sum / count;
        DecimalFormat fmt = new DecimalFormat ("0.##0");
        System.out.println ("The average is "+ fmt.format(average));
    }
}
```

The while Statement

- A loop can also be used for input validation, making a program more robust
- See WinPercentage.java on page 239 (segments below)

```java
final int NUM_GAMES = 12;
...
System.out.print ("Enter the number of games won (0 to "+
                    NUM_GAMES + ": ");
won = scan.nextInt();
while (won < 0 || won > NUM_GAMES)
{
    System.out.print ("Invalid input. Please reenter: ");
    won = scan.nextInt();
}
ratio = (double)won / NUM_GAMES;
```
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

Infinite Loops

- An example of an infinite loop:
  ```java
  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

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

```java
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 <= 20)
    {
        System.out.println("Here");
        count2++;
    }
    count1++;
}  \[10 \times 20 = 200\]
```
PalindromTester (segment)

```java
System.out.println("Enter a potential palindrome:");
str = scan.nextLine();

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

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.");
```

Outline

The if Statement and Conditions
Other Conditional Statements
Comparing Data
The while Statement
Iterators (later, after chaps. 3 - 4)
Other Repetition Statements
Decisions and Graphics
More Components
The do Statement

- A *do statement* has the following syntax:
  ```
  do
  {
    statement;
  }
  while ( condition )
  ```

- The *statement* is executed once initially, and then the *condition* is evaluated.

- The statement is executed repeatedly until the condition becomes false.

Logic of a do Loop

```
statement
```

```
true
```

```
condition evaluated
```

```
false
```
The do Statement

- An example of a do loop:
  ```java
  int count = 0;
  do
  { 
      count++;
      System.out.println (count);
  } while (count < 5);
  ```

- The body of a do loop executes at least once

- See ReverseNumber.java (page 251): understand its algorithm

ReverseNumber (main method)

```
// Reverses the digits of an integer mathematically.
public static void main (String[] args)
{
    int number, lastDigit, reverse = 0;
    Scanner scan = new Scanner (System.in);
    System.out.print ("Enter a positive integer: ");
    number = scan.nextInt();
    do 
    { 
        lastDigit = number % 10;
        reverse = (reverse * 10) + lastDigit;
        number = number / 10;
    } while (number > 0);
    System.out.println ("That number reversed is "+ reverse);
}
```
Comparing while and do

The while Loop

- condition evaluated
  - true
  - false
  - statement

The do Loop

- statement
  - true
  - false
  - condition evaluated

The for Statement

- A for statement has the following syntax:

  The initialization 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

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

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

```java
initialization;
while (condition )
{
    statement;
    increment;
}
```
The for Statement

• An example of a for loop:
  ```java
  for (int count=1; count <= 5; count++)
    System.out.println (count);
  ```

• The initialization section can be used to declare a variable

• Like a while loop, the condition of a for loop is tested prior to executing the loop body

• Therefore, the body of a for loop will execute zero or more times

The for Statement

• The increment section can perform any calculation
  ```java
  for (int num=100; num > 0; num -= 5)
    System.out.println (num);
  ```

• 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
The for Statement

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance
- See `Multiples.java` (page 255)
- See `Stars.java` (page 257)

Stars.java (main method)

```java
// Prints a ?? shape using asterisk (star) characters.
public static void main(String[] args)
{
    final int MAX_ROWS = 10;

    for (int row = 1; row <= MAX_ROWS; row++)
    {
        for (int star = 1; star <= row; star++)
            System.out.print("*");
        System.out.println();
    }
}
```
Chapter 5

• **Topics Covered in Midterm:**
  - boolean expressions (if-else)
  - conditional statements
  - comparing data
  - repetition statements

• **Topics NOT Covered in Midterm:**
  - iterators
  - more drawing techniques
  - more GUI components