Chapter 5: Conditionals and Loops
Lab Exercises

Topics	Lab Exercises
Boolean expressions	PreLab Exercises
The **if** statement	Computing a Raise
The **switch** statement	A Charge Account Statement
	Activities at Lake LazyDays
	Rock, Paper, Scissors
	Date Validation
Conditional Operator	Processing Grades
The **while** statement	PreLab Exercises
	Counting and Looping
	Powers of 2
	Factorials
	A Guessing Game
Iterators & Reading Text Files	Baseball Statistics
The **do** statement	More Guessing
	Election Day
The **for** statement	Finding Maximum and Minimum Values
	Counting Characters
	Using the Coin Class
Drawing with loops and conditionals	A Rainbow Program
Determining Event Sources	Vote Counter, Revisited
Dialog Boxes	Modifying EvenOdd.java
	A Pay Check Program
Checkboxes & Radio Buttons	Adding Buttons to StyleOptions.java

Prelab Exercises
Sections 5.1-5.3

1. Rewrite each condition below in valid Java syntax (give a boolean expression):
   a. x > y > z
   b. x and y are both less than 0
   c. neither x nor y is less than 0
   d. x is equal to y but not equal to z

2. Suppose *gpa* is a variable containing the grade point average of a student. Suppose the goal of a program is to let a student know if he/she made the Dean's list (the gpa must be 3.5 or above). Write an *if... else...* statement that prints out the appropriate message (either "Congratulations—you made the Dean's List" or "Sorry you didn't make the Dean's List").

3. Complete the following program to determine the raise and new salary for an employee by adding if ... else statements to compute the raise. The input to the program includes the current annual salary for the employee and a number indicating the performance rating (1=excellent, 2=good, and 3=poor). An employee with a rating of 1 will receive a 6% raise, an employee with a rating of 2 will receive a 4% raise, and one with a rating of 3 will receive a 1.5% raise.

```
// ************************************************************
//   Salary.java
//   Computes the raise and new salary for an employee
// ************************************************************
import java.util.Scanner;

public class Salary
{
   public static void main (String[] args)
   {
      double currentSalary;  // current annual salary
      double rating;         // performance rating
      double raise;          // dollar amount of the raise

      Scanner scan = new Scanner(System.in);

      // Get the current salary and performance rating
      System.out.print ("Enter the current salary: ");
      currentSalary = scan.nextDouble();
      System.out.print ("Enter the performance rating: ");
      rating = scan.nextDouble();

      // Compute the raise -- Use if ... else ...

      // Print the results
      System.out.println ("Amount of your raise: $" + raise);
      System.out.println ("Your new salary: $" + currentSalary + raise);
   }
}
```

Computing A Raise

File *Salary.java* contains most of a program that takes as input an employee's salary and a rating of the employee's performance and computes the raise for the employee. This is similar to question #3 in the pre-lab, except that the performance rating here is being entered as a String—the three possible ratings are "Excellent", "Good", and "Poor". As in the pre-lab, an employee who is rated excellent will receive a 6% raise, one rated good will receive a 4% raise, and one rated poor will receive a 1.5% raise.

Add the *if... else...* statements to program Salary to make it run as described above. Note that you will have to use the *equals* method of the String class (not the relational operator ==) to compare two strings (see Section 5.3, Comparing Data).

```
// ********************************************************************
//   Salary.java
//
//   Computes the amount of a raise and the new
//   salary for an employee.  The current salary
//   and a performance rating (a String: "Excellent",
//   "Good" or "Poor") are input.
// ********************************************************************

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

public class Salary
{
   public static void main (String[] args)
   {
       double currentSalary;  // employee's current  salary
       double raise;          // amount of the raise
       double newSalary;      // new salary for the employee
       String rating;         // performance rating

       Scanner scan = new Scanner(System.in);

       System.out.print ("Enter the current salary: ");
       currentSalary = scan.nextDouble();
       System.out.print ("Enter the performance rating (Excellent, Good, or Poor): ");
       rating = scan.nextLine();

       // Compute the raise using if ...

       newSalary = currentSalary + raise;

       // Print the results
       NumberFormat money = NumberFormat.getCurrencyInstance();
       System.out.println();
       System.out.println("Current Salary:       " + money.format(currentSalary));
       System.out.println("Amount of your raise: " + money.format(raise));
       System.out.println("Your new salary:      " + money.format(newSalary));
       System.out.println();
    }
}
```

A Charge Account Statement

Write a program to prepare the monthly charge account statement for a customer of CS CARD International, a credit card company. The program should take as input the previous balance on the account and the total amount of additional charges during the month. The program should then compute the interest for the month, the total new balance (the previous balance plus additional charges plus interest), and the minimum payment due. Assume the interest is 0 if the previous balance was 0 but if the previous balance was greater than 0 the interest is 2% of the total owed (previous balance plus additional charges). Assume the minimum payment is as follows:

```
      new balance          for a new balance less than $50
        $50.00             for a new balance between $50 and $300 (inclusive)
20% of the new balance     for a new balance over $300
```

So if the new balance is $38.00 then the person must pay the whole $38.00; if the balance is $128 then the person must pay $50; if the balance is $350 the minimum payment is $70 (20% of 350). The program should print the charge account statement in the format below. Print the actual dollar amounts in each place using currency format from the NumberFormat class—see Listing 3.4 of the text for an example that uses this class.

```
CS CARD International Statement
===============================

Previous Balance:      $
Additional Charges:    $
Interest:              $

New Balance:           $

Minimum Payment:       $
```

Activities at Lake LazyDays

As activity directory at Lake LazyDays Resort, it is your job to suggest appropriate activities to guests based on the weather:

```
temp >= 80:        swimming
60 <= temp < 80:   tennis
40 <= temp < 60:   golf
temp < 40:         skiing
```

1. Write a program that prompts the user for a temperature, then prints out the activity appropriate for that temperature. Use a cascading if, and be sure that your conditions are no more complex than necessary.

2. Modify your program so that if the temperature is greater than 95 or less than 20, it prints "Visit our shops!". (Hint: Use a boolean operator in your condition.) For other temperatures print the activity as before.

Rock, Paper, Scissors

Program *Rock.java* contains a skeleton for the game Rock, Paper, Scissors. Open it and save it to your directory. Add statements to the program as indicated by the comments so that the program asks the user to enter a play, generates a random play for the computer, compares them and announces the winner (and why). For example, one run of your program might look like this:

```
$ java Rock
Enter your play: R, P, or S
r
Computer play is S
Rock crushes scissors, you win!
```

Note that the user should be able to enter either upper or lower case r, p, and s. The user's play is stored as a string to make it easy to convert whatever is entered to upper case. Use a switch statement to convert the randomly generated integer for the computer's play to a string.

```
// ****************************************************************
//   Rock.java
//
//   Play Rock, Paper, Scissors with the user
//
// ****************************************************************
import java.util.Scanner;
import java.util.Random;

public class Rock
{
    public static void main(String[] args)
    {
      String personPlay;    //User's play -- "R", "P", or "S"
      String computerPlay;  //Computer's play -- "R", "P", or "S"
      int computerInt;      //Randomly generated number used to determine
                            //computer's play

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

      //Get player's play -- note that this is stored as a string
      //Make player's play uppercase for ease of comparison
      //Generate computer's play (0,1,2)
      //Translate computer's randomly generated play to string
      switch (computerInt)
      {

      }

      //Print computer's play
      //See who won.  Use nested ifs instead of &&.
      if (personPlay.equals(computerPlay))
          System.out.println("It's a tie!");
      else if (personPlay.equals("R"))
          if (computerPlay.equals("S"))
            System.out.println("Rock crushes scissors.  You win!!");
          else

            //...  Fill in rest of code
    }
}
```

Date Validation

In this exercise you will write a program that checks to see if a date entered by the user is a valid date in the second millenium. A skeleton of the program is in *Dates.java*. Open this program and save it to your directory. As indicated by the comments in the program, fill in the following:

1. An assignment statement that sets monthValid to true if the month entered is between 1 and 12, inclusive.
2. An assignment statement that sets yearValid to true if the year is between 1000 and 1999, inclusive.
3. An assignment statement that sets leapYear to true if the year is a leap year. Here is the leap year rule (there's more to it than you may have thought!):

   > If the year is divisible by 4, it's a leap year UNLESS it's divisible by 100, in which case it's not a leap year UNLESS it's divisible by 400, in which case it is a leap year. If the year is not divisible by 4, it's not a leap year.

   Put another way, it's a leap year if a) it's divisible by 400, or b) it's divisible by 4 and it's *not* divisible by 100. So 1600 and 1512 are leap years, but 1700 and 1514 are not.

4. An if statement that determines the number of days in the month entered and stores that value in variable daysInMonth. If the month entered is not valid, daysInMonth should get 0. Note that to figure out the number of days in February you'll need to check if it's a leap year.
5. An assignment statement that sets dayValid to true if the day entered is legal for the given month and year.
6. If the month, day, and year entered are all valid, print "Date is valid" and indicate whether or not it is a leap year. If any of the items entered is not valid, just print "Date is not valid" without any comment on leap year.

```
// ****************************************************************
// Dates.java
//
// Determine whether a 2nd-millenium date entered by the user
// is valid
// ****************************************************************
import java.util.Scanner;

public class Dates
{
    public static void main(String[] args)
    {
      int month, day, year;   //date read in from user
      int daysInMonth;        //number of days in month read in
      boolean monthValid, yearValid, dayValid;  //true if input from user is valid
      boolean leapYear;       //true if user's year is a leap year

      Scanner scan = new Scanner(System.in);

      //Get integer month, day, and year from user

      //Check to see if month is valid

      //Check to see if year is valid

      //Determine whether it's a leap year

      //Determine number of days in month

      //User number of days in month to check to see if day is valid

      //Determine whether date is valid and print appropriate message

    }
}
```

Processing Grades

The file *Grades.java* contains a program that reads in a sequence of student grades and computes the average grade, the number of students who pass (a grade of at least 60) and the number who fail. The program uses a loop (which you learn about in the next section).

1. Compile and run the program to see how it works.
2. Study the code and do the following.
   - Replace the statement that finds the sum of the grades with one that uses the += operator.
   - Replace each of three statements that increment a counting variable with statements using the increment operator.

3. Run your program to make sure it works.
4. Now replace the "if" statement that updates the pass and fail counters with the conditional operator.

```
// ********************************************************************
//   Grades.java
//
//   Read in a sequence of grades and compute the average
//   grade, the number of passing grades (at least 60)
//   and the number of failing grades.
// ********************************************************************
import java.util.Scanner;

public class Grades
{
    //-----------------------------------------------------------------
    //  Reads in and processes grades until a negative number is entered.
    //-----------------------------------------------------------------
    public static void main (String[] args)
    {
      double grade;  // a student's grade
      double sumOfGrades; // a running total of the student grades
      int numStudents; // a count of the students
      int numPass;     // a count of the number who pass
      int numFail;     // a count of the number who fail

      Scanner scan = new Scanner(System.in);

      System.out.println ("\nGrade Processing Program\n");

      // Initialize summing and counting variables
      sumOfGrades = 0;
      numStudents = 0;
      numPass = 0;
      numFail = 0;

      // Read in the first grade
      System.out.print ("Enter the first student's grade: ");
      grade = scan.nextDouble();

      while (grade >= 0)
          {
            sumOfGrades = sumOfGrades + grade;
            numStudents = numStudents + 1;

            if (grade < 60)
                numFail = numFail + 1;
            else
                numPass = numPass + 1;
```

```
            // Read the next grade
            System.out.print ("Enter the next grade (a negative to quit): ");
            grade = scan.nextDouble();
          }

      if (numStudents > 0)
          {
            System.out.println ("\nGrade Summary: ");
            System.out.println ("Class Average: " + sumOfGrades/numStudents);
            System.out.println ("Number of Passing Grades: " + numPass);
            System.out.println ("Number of Failing Grades: " + numFail);
          }
      else
          System.out.println ("No grades processed.");
    }
}
```

Prelab Exercises
Section 5.5

In a while loop, execution of a set of statements (the *body* of the loop) continues until the boolean expression controlling the loop (the *condition*) becomes false. As for an if statement, the condition must be enclosed in parentheses. For example, the loop below prints the numbers from 1 to to LIMIT:

```
final int LIMIT = 100;          // setup
int count = 1;

while (count <= LIMIT)          // condition
{                               // body
  System.out.println(count);    //   -- perform task
  count = count + 1;            //   -- update condition
}
```

There are three parts to a loop:

- The *setup*, or *initialization*. This comes before the actual loop, and is where variables are initialized in preparation for the first time through the loop.
- The *condition*, which is the boolean expression that controls the loop. This expression is evaluated each time through the loop. If it evaluates to true, the body of the loop is executed, and then the condition is evaluated again; if it evaluates to false, the loop terminates.
- The *body* of the loop. The body typically needs to do two things:
  - Do some work toward the task that the loop is trying to accomplish. This might involve printing, calculation, input and output, method calls—this code can be arbitrarily complex.
  - Update the condition. Something has to happen inside the loop so that the condition will eventually be false—otherwise the loop will go on forever (an *infinite* loop). This code can also be complex, but often it simply involves incrementing a counter or reading in a new value.

  Sometimes doing the work and updating the condition are related. For example, in the loop above, the print statement is doing work, while the statement that increments count is both doing work (since the loop's task is to print the values of count) and updating the condition (since the loop stops when count hits a certain value).

The loop above is an example of a *count-controlled* loop, that is, a loop that contains a counter (a variable that increases or decreases by a fixed value—usually 1—each time through the loop) and that stops when the counter reaches a certain value. Not all loops with counters are count-controlled; consider the example below, which determines how many even numbers must be added together, starting at 2, to reach or exceed a given limit.

```
final int LIMIT = 16;                              TRACE
int count = 1;                          sum        nextVal     count
int sum = 0;                            ---        -------     -----
int nextVal = 2;

while (sum < LIMIT)
{
  sum = sum + nextVal;
  nextVal = nextVal + 2;
  count = count + 1;
}

System.out.println("Had to add together " + (count-1) + " even numbers " +
                   "to reach value " + LIMIT + ".  Sum is " + sum);
```

Note that although this loop counts how many times the body is executed, the condition does not depend on the value of count.

Not all loops have counters. For example, if the task in the loop above were simply to add together even numbers until the sum reached a certain limit and then print the sum (as opposed to printing the number of things added together), there would

be no need for the counter. Similarly, the loop below sums integers input by the user and prints the sum; it contains no counter.

```
int sum = 0;                                          //setup
String keepGoing = "y";
int nextVal;

while (keepGoing.equals("y") || keepGoing.equals("Y"))
{
  System.out.print("Enter the next integer: ");       //do work
  nextVal = scan.nextInt();
  sum = sum + nextVal;

  System.out.println("Type y or Y to keep going");    //update condition
  keepGoing = scan.next();
}

System.out.println("The sum of your integers is " + sum);
```

Exercises

1. In the first loop above, the println statement comes before the value of count is incremented. What would happen if you reversed the order of these statements so that count was incremented before its value was printed? Would the loop still print the same values? Explain.

2. Consider the second loop above.
   a. Trace this loop, that is, in the table next to the code show values for variables nextVal, sum and count at each iteration. Then show what the code prints.
   b. Note that when the loop terminates, the number of even numbers added together before reaching the limit is count-1, not count. How could you modify the code so that when the loop terminates, the number of things added together is simply count?

3. Write a while loop that will print "I love computer science!!" 100 times. Is this loop count-controlled?

4. Add a counter to the third example loop above (the one that reads and sums integers input by the user). After the loop, print the number of integers read as well as the sum. Just note your changes on the example code. Is your loop now count-controlled?

5. The code below is supposed to print the integers from 10 to 1 backwards. What is wrong with it? (Hint: there are two problems!) Correct the code so it does the right thing.

```
count = 10;
while (count >= 0)
{
 System.out.println(count);
 count = count + 1;
}
```

Counting and Looping

The program in *LoveCS.java* prints "I love Computer Science!!" 10 times. Copy it to your directory and compile and run it to see how it works. Then modify it as follows:

```
// ************************************************************
//   LoveCS.java
//
//   Use a while loop to print many messages declaring your
//   passion for computer science
// ************************************************************

public class LoveCS
{
    public static void main(String[] args)
    {
      final int LIMIT = 10;

      int count = 1;

      while (count <= LIMIT){
          System.out.println("I love Computer Science!!");
          count++;
      }
    }
}
```

1. Instead of using constant LIMIT, ask the user how many times the message should be printed. You will need to declare a variable to store the user's response and use that variable to control the loop. (Remember that all caps is used only for constants!)
2. Number each line in the output, and add a message at the end of the loop that says how many times the message was printed. So if the user enters 3, your program should print this:

```
1 I love Computer Science!!
2 I love Computer Science!!
3 I love Computer Science!!
Printed this message 3 times.
```

3. If the message is printed N times, compute and print the sum of the numbers from 1 to N. So for the example above, the last line would now read:

   ```
   Printed this message 3 times.  The sum of the numbers from 1 to 3 is 6.
   ```

   Note that you will need to add a variable to hold the sum.

Powers of 2

File *PowersOf2.java* contains a skeleton of a program to read in an integer from the user and print out that many powers of 2, starting with 20.

1. Using the comments as a guide, complete the program so that it prints out the number of powers of 2 that the user requests. **Do not use Math.pow to compute the powers of 2!** Instead, compute each power from the previous one (how do you get 2^n from 2^{n-1}?). For example, if the user enters 4, your program should print this:

   ```
   Here are the first 4 powers of 2:
   1
   2
   4
   8
   ```

2. Modify the program so that instead of just printing the powers, you print which power each is, e.g.:

   ```
   Here are the first 4 powers of 2:
   2^0 = 1
   2^1 = 2
   2^2 = 4
   2^3 = 8
   ```

```
// ****************************************************************
//   PowersOf2.java
//
//   Print out as many powers of 2 as the user requests
//
// ****************************************************************
import java.util.Scanner;

public class PowersOf2
{
    public static void main(String[] args)
    {
        int numPowersOf2;        //How many powers of 2 to compute
        int nextPowerOf2 = 1;    //Current power of  2
        int exponent;            //Exponent for current power of 2 -- this
                                 //also serves as a counter for the loop
        Scanner scan = new Scanner(System.in);

        System.out.println("How many powers of 2 would you like printed?");
        numPowersOf2 = scan.nextInt();

        //print a message saying how many powers of 2 will be printed
        //initialize exponent -- the first thing printed is 2 to the what?

        while (  )
        {
            //print out current power of 2

            //find next power of 2 -- how do you get this from the last one?

            //increment exponent

        }
    }
}
```

Factorials

The *factorial* of n (written n!) is the product of the integers between 1 and n. Thus 4! = 1*2*3*4 = 24. By definition, 0! = 1. Factorial is not defined for negative numbers.

1. Write a program that asks the user for a non-negative integer and computes and prints the factorial of that integer. You'll need a while loop to do most of the work—this is a lot like computing a sum, but it's a product instead. And you'll need to think about what should happen if the user enters 0.

2. Now modify your program so that it checks to see if the user entered a negative number. If so, the program should print a message saying that a nonnegative number is required and ask the user the enter another number. The program should keep doing this until the user enters a nonnegative number, after which it should compute the factorial of that number. **Hint:** you will need another while loop **before** the loop that computes the factorial. You should not need to change any of the code that computes the factorial!

A Guessing Game

File *Guess.java* contains a skeleton for a program to play a guessing game with the user. The program should randomly generate an integer between 1 and 10, then ask the user to try to guess the number. If the user guesses incorrectly, the program should ask them to try again until the guess is correct; when the guess is correct, the program should print a congratulatory message.

1. Using the comments as a guide, complete the program so that it plays the game as described above.
2. Modify the program so that if the guess is wrong, the program says whether it is too high or too low. You will need an if statement (inside your loop) to do this.
3. Now add code to count how many guesses it takes the user to get the number, and print this number at the end with the congratulatory message.
4. Finally, count how many of the guesses are too high and how many are too low. Print these values, along with the total number of guesses, when the user finally guesses correctly.

```
// ****************************************************************
//   Guess.java
//
//   Play a game where the user guesses a number from 1 to 10
//
// ****************************************************************
import java.util.Scanner;
import java.util.Random;

public class Guess
{
    public static void main(String[] args)
    {
      int numToGuess;       //Number the user tries to guess
      int guess;            //The user's guess

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

      //randomly generate the  number to guess

      //print message asking user to enter a guess

      //read in guess

      while (  )  //keep going as long as the guess is wrong
        {
          //print message saying guess is wrong
          //get another guess from the user
      }

      //print message saying guess is right
    }
}
```

Baseball Statistics

The local Kids' League coach keeps some of the baseball team statistics in a text file organized as follows: each line of the file contains the name of the player followed by a list of symbols indicating what happened on each at bat for the player. The letter h indicates a hit, o an out, w a walk, and s a sacrifice fly. Each item on the line is separated by a comma. There are no blank spaces except in the player name. So, for example the file could look as follows:

Sam Slugger,h,h,o,s,w,w,h,w,o,o,o,h,s
Jill Jenks,o,o,s,h,h,o,o
Will Jones,o,o,w,h,o,o,o,o,w,o,o

The file *BaseballStats.java* contains the skeleton of a program thats reads and processes a file in this format. Study the program and note that three Scanner objects are declared.

- One scanner (*scan*) is used to read in a file name from standard input.
- The file name is then used to create a scanner (*fileScan*) to operate on that file.
- A third scanner (*lineScan*) will be used to parse each line in the file.

Also note that the main method throws an IOException. This is needed in case there is a problem opening the file.

Complete the program as follows:

1. First add a while loop that reads each line in the file and prints out each part (name, then each at bat, without the commas) in a way similar to the URLDissector program in Listing 5.11 of the text. In particular inside the loop you need to
   a. read the next line from the file
   b. create a comma delimited scanner (lineScan) to parse the line
   c. read and print the name of the player, and finally,
   d. have a loop that prints each at bat code.

2. Compile and run the program to be sure it works.

3. Now modify the inner loop that parses a line in the file so that instead of printing each part it counts (separately) the number of hits, outs, walks, and sacrifices. Each of these summary statistics, as well as the batting average, should be printed for each player. Recall that the batting average is the number of hits divided by the total number of hits and outs.

4. Test the program on the file *stats.dat* and *stats2.dat* .

```
// ********************************************************************
//   BaseballStats.java
//
//   Reads baseball data in from a comma delimited file. Each line
//   of the file contains a name followed by a list of symbols
//   indicating the result of each at bat: h for hit, o for out,
//   w for walk, s for sacrifice.  Statistics are computed and
//   printed for each player.
// ********************************************************************

import java.util.Scanner;
import java.io.*;

public class BaseballStats
{
    //-----------------------------------------------
    //  Reads baseball stats from a file and counts
    //  total hits, outs, walks, and sacrifice flies
    //  for each player.
    //-----------------------------------------------
    public static void main (String[] args) throws IOException
```

```
    {
        Scanner fileScan, lineScan;
        String fileName;

        Scanner scan = new Scanner(System.in);

        System.out.print ("Enter the name of the input file: ");
        fileName = scan.nextLine();
        fileScan = new Scanner(new File(fileName));

        // Read and process each line of the file



    }
}
```

stats.dat

```
Willy Wonk,o,o,h,o,o,o,o,h,w,o,o,o,o,s,h,o,h
Shari Jones,h,o,o,s,s,h,o,o,o,h,o,o,o,o
Barry Bands,h,h,w,o,o,o,w,h,o,o,h,h,o,o,w,w,w,h,o,o
Sally Slugger,o,h,h,o,o,h,h,w
Missy Lots,o,o,s,o,o,w,o,o,o
Joe Jones,o,h,o,o,o,o,h,h,o,o,o,o,w,o,o,o,h,o,h,h
Larry Loop,w,s,o,o,o,h,o,o,h,s,o,o,o,h,h
Sarah Swift,o,o,o,o,h,h,w,o,o,o
Bill Bird,h,o,h,o,h,w,o,o,o,h,s,s,h,o,o,o,o,o,o
Don Daring,o,o,h,h,o,o,h,o,h,o,o,o,o,o,o,o,h
Jill Jet,o,s,s,h,o,o,h,h,o,o,o,h,o,h,w,o,o,h,h,o
```

stats2.dat

```
Barry Bands,h,h,w,o,o,o,w,h,o,o,h,h,o,o,w,w,w,h,o,o
```

More Guessing

File *Guess.java* contains the skeleton for a program that uses a while loop to play a guessing game. (This problem is described in the previous lab exercise.) Revise this program so that it uses a *do ... while* loop rather than a while loop. The general outline using a *do... while* loop is as follows:

```
// set up (initializations of the counting variables)
....

do
{
    // read in a guess
    ...

    // check the guess and print appropriate messages

    ...
 }
 while ( condition );
```

A key difference between a *while* and a *do ... while* loop to note when making your changes is that the body of the do ... while loop is executed before the condition is ever tested. In the while loop version of the program, it was necessary to read in the user's first guess before the loop so there would be a value for comparison in the condition. In the do... while this "priming" read is no longer needed. The user's guess can be read in at the beginning of the body of the loop.

Election Day

It's almost election day and the election officials need a program to help tally election results. There are two candidates for office—Polly Tichen and Ernest Orator. The program's job is to take as input the number of votes each candidate received in each voting precinct and find the total number of votes for each. The program should print out the final tally for each candidate—both the total number of votes each received and the percent of votes each received. Clearly a loop is needed. Each iteration of the loop is responsible for reading in the votes from a single precinct and updating the tallies. A skeleton of the program is in the file *Election.java*. Open a copy of the program in your text editor and do the following.

1. Add the code to control the loop. You may use either a while loop or a do...while loop. The loop must be controlled by asking the user whether or not there are more precincts to report (that is, more precincts whose votes need to be added in). The user should answer with the character y or n though your program should also allow uppercase repsonses. The variable *response* (type String) has already been declared.
2. Add the code to read in the votes for each candidate and find the total votes. Note that variables have already been declared for you to use. Print out the totals and the percentages after the loop.
3. Test your program to make sure it is correctly tallying the votes and finding the percentages AND that the loop control is correct (it goes when it should and stops when it should).
4. The election officials want more information. They want to know how many precincts each candidate carried (won). Add code to compute and print this. You need three new variables: one to count the number of precincts won by Polly, one to count the number won by Ernest, and one to count the number of ties. Test your program after adding this code.

```
// ************************************************************
//   Election.java
//
//   This file contains a program that tallies the results of
//   an election.  It reads in the number of votes for each of
//   two candidates in each of several precincts.  It determines
//   the total number of votes received by each candidate, the
//   percent of votes received by each candidate, the number of
//   precincts each candidate carries, and the
//   maximum winning margin in a precinct.
// ************************************************************

import java.util.Scanner;

public class Election
{
    public static void main (String[] args)
    {
      int votesForPolly;  // number of votes for Polly in each precinct
      int votesForErnest; // number of votes for Ernest in each precinct
      int totalPolly;     // running total of votes for Polly
      int totalErnest;    // running total of votes for Ernest
      String response;    // answer (y or n) to the "more precincts" question

      Scanner scan = new Scanner(System.in);

      System.out.println ();
      System.out.println ("Election Day Vote Counting Program");
      System.out.println ();

      // Initializations

      // Loop to "process" the votes in each precinct

      // Print out the results
    }
}
```

Finding Maximum and Minimum Values

A common task that must be done in a loop is to find the maximum and minimum of a sequence of values. The file *Temps.java* contains a program that reads in a sequence of hourly temperature readings over a 24-hour period. You will be adding code to this program to find the maximum and minimum temperatures. Do the following:

1. Save the file to your directory, open it and see what's there. Note that a *for* loop is used since we need a count-controlled loop. Your first task is to add code to find the maximum temperature read in. In general to find the maximum of a sequence of values processed in a loop you need to do two things:

   - You need a variable that will keep track of the maximum of the values processed so far. This variable must be initialized before the loop. There are two standard techniques for initialization: one is to initialize the variable to some value *smaller* than any possible value being processed; another is to initialize the variable to the first value processed. In either case, after the first value is processed the maximum variable should contain the first value. For the temperature program declare a variable *maxTemp* to hold the maximum temperature. Initialize it to -1000 (a value less than any legitimate temperature).
   - The maximum variable must be updated each time through the loop. This is done by comparing the maximum to the current value being processed. If the current value is larger, then the current value is the new maximum. So, in the temperature program, add an if statement inside the loop to compare the current temperature read in to maxTemp. If the current temperature is larger set maxTemp to that temperature. NOTE: If the current temperature is NOT larger, DO NOTHING!

2. Add code to print out the maximum after the loop. Test your program to make sure it is correct. Be sure to test it on at least three scenarios: the first number read in is the maximum, the last number read in is the maximum, and the maximum occurs somewhere in the middle of the list. For testing purposes you may want to change the HOURS_PER_DAY variable to something smaller than 24 so you don't have to type in so many numbers!

3. Often we want to keep track of more than just the maximum. For example, if we are finding the maximum of a sequence of test grades we might want to know the name of the student with the maximum grade. Suppose for the temperatures we want to keep track of the time (hour) the maximum temperature occurred. To do this we need to save the current value of the *hour* variable when we update the *maxTemp* variable. This of course requires a new variable to store the time (hour) that the maximum occurs. Declare *timeOfMax* (type int) to keep track of the time (hour) the maximum temperature occurred. Modify your *if* statment so that in addition to updating *maxTemp* you also save the value of *hour* in the *timeOfMax* variable. (WARNING: you are now doing TWO things when the *if* condition is TRUE.)

4. Add code to print out the time the maximum temperature occurred along with the maximum.

5. Finally, add code to find the minimum temperature and the time that temperature occurs. The idea is the same as for the maximum. NOTE: Use a separate *if* when updating the minimum temperature variable (that is, don't add an *else* clause to the *if* that is already there).

```
// ********************************************************
//   Temps.java
//
//   This program reads in a sequence of hourly temperature
//   readings (beginning with midnight) and prints the maximum
//   temperature (along with the hour, on a 24-hour clock, it
//   occurred) and the minimum temperature (along with the hour
//   it occurred).
// ********************************************************

import java.util.Scanner;

public class Temps
{
    //-----------------------------------------------------
    //  Reads in a sequence of temperatures and finds the
    //  maximum and minimum read in.
    //-----------------------------------------------------
    public static void main (String[] args)
    {
      final int HOURS_PER_DAY = 24;

      int temp;   // a temperature reading

      Scanner scan = new Scanner(System.in);

      // print program heading
      System.out.println ();
      System.out.println ("Temperature Readings for 24 Hour Period");
      System.out.println ();

      for (int hour = 0; hour < HOURS_PER_DAY; hour++)
      {
          System.out.print ("Enter the temperature reading at " + hour +
                            " hours: ");
          temp = scan.nextInt();
      }

      // Print the results
    }
}
```

Counting Characters

The file *Count.java* contains the skeleton of a program to read in a string (a sentence or phrase) and count the number of blank spaces in the string. The program currently has the declarations and initializations and prints the results. All it needs is a loop to go through the string character by character and count (update the *countBlank* variable) the characters that are the blank space. Since we know how many characters there are (the *length* of the string) we use a count controlled loop—*for* loops are especially well-suited for this.

1. Add the *for* loop to the program. Inside the for loop you need to access each individual character—the *charAt* method of the String class lets you do that. The assignment statement

   ```
   ch = phrase.charAt(i);
   ```

   assigns the variable *ch* (type char) the character that is in index i of the String *phrase*. In your for loop you can use an assignment similar to this (replace i with your loop control variable if you use something other than i). NOTE: You could also directly use phrase.charAt(i) in your if (without assigning it to a variable).

2. Test your program on several phrases to make sure it is correct.

3. Now modify the program so that it will count several different characters, not just blank spaces. To keep things relatively simple we'll count the a's, e's, s's, and t's (both upper and lower case) in the string. You need to declare and initialize four additional counting variables (e.g. *countA* and so on). Your current *if* could be modified to cascade but another solution is to use a *switch* statement. Replace the current *if* with a switch that accounts for the 9 cases we want to count (upper and lower case a, e, s, t, and blank spaces). The cases will be based on the value of the *ch* variable. The switch starts as follows—complete it.

   ```
   switch (ch)
   {
      case 'a':
      case 'A':  countA++;
                 break;

      case ....

   }
   ```

   Note that this switch uses the "fall through" feature of switch statements. If *ch* is an 'a' the first case matches and the switch continues execution until it encounters the *break* hence the countA variable would be incremented.

4. Add statements to print out all of the counts.

5. It would be nice to have the program let the user keep entering phrases rather than having to restart it every time. To do this we need another loop surrounding the current code. That is, the current loop will be nested inside the new loop. Add an outer while loop that will continue to execute as long as the user does NOT enter the phrase *quit*. Modify the prompt to tell the user to enter a phrase or *quit* to quit. Note that all of the initializations for the counts should be inside the while loop (that is we want the counts to start over for each new phrase entered by the user). All you need to do is add the while statement (and think about placement of your reads so the loop works correctly). Be sure to go through the program and properly indent after adding code—with nested loops the inner loop should be indented.

```
// ********************************************************
//   Count.java
//
//   This program reads in strings (phrases) and counts the
//   number of blank characters and certain other letters
//   in the phrase.
// ********************************************************

import java.util.Scanner;

public class Count
{
  public static void main (String[] args)
  {
      String phrase;    // a string of characters
      int countBlank;   // the number of blanks (spaces) in the phrase
      int length;       // the length of the phrase
      char ch;          // an individual character in the string

      Scanner scan = new Scanner(System.in);

      // Print a program header
      System.out.println ();
      System.out.println ("Character Counter");
      System.out.println ();

      // Read in a string and find its length
      System.out.print ("Enter a sentence or phrase: ");
      phrase = scan.nextLine();
      length = phrase.length();

      // Initialize counts
      countBlank = 0;

      // a for loop to go through the string character by character
      // and count the blank spaces

      // Print the results
      System.out.println ();
      System.out.println ("Number of blank spaces: " + countBlank);
      System.out.println ();
    }
}
```

Using the Coin Class

The Coin class from Listing 5.4 in the text is in the file *Coin.java*. Copy it to your directory, then write a program to find the length of the longest run of heads in 100 flips of the coin. A skeleton of the program is in the file *Runs.java*. To use the Coin class you need to do the following in the program:

1. Create a coin object.
2. Inside the loop, you should use the *flip* method to flip the coin, the *toString* method (used implicitly) to print the results of the flip, and the *getFace* method to see if the result was HEADS. Keeping track of the current run length (the number of times in a row that the coin was HEADS) and the maximum run length is an exercise in loop techniques!
3. Print the result after the loop.

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

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

    private int face;

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

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

    // -----------------------------------------------------
    //   Returns the current face of the coin as an integer.
    // -----------------------------------------------------
    public int getFace()
    {
      return face;
    }

    // -----------------------------------------------------
    //   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;
    }
}



// ********************************************************************
// Runs.java
//
// Finds the length of the longest run of heads in 100 flips of a coin.
// ********************************************************************

import java.util.Scanner;

public class Runs
{
    public static void main (String[] args)
    {
      final int FLIPS = 100; // number of coin flips

      int currentRun = 0; // length of the current run of HEADS
      int maxRun = 0;     // length of the maximum run so far

      Scanner scan = new Scanner(System.in);

      // Create a coin object

      // Flip the coin FLIPS times
      for (int i = 0; i < FLIPS; i++)
          {
            // Flip the coin & print the result

            // Update the run information

          }

      // Print the results

    }
}
```

A Rainbow Program

Write a program that draws a rainbow. (This is one of the Programming Projects at the end of Chapter 5 in the text.) As suggested in the text, your rainbow will be concentric arcs, each a different color. The basic idea of the program is similar to the program that draws a bull's eye in Listing 5.15 and 5.16 of the text. You should study that program and understand it before starting your rainbow. The major difference in this program (other than drawing arcs rather than circles) is making the different arcs different colors. You can do this in several different ways. For example, you could have a variable for the color code, initialize it to some value then either increment or decrement by some amount each pass through the loop (you'll need to experiment with numbers to see the effect on the color). Or you could try using three integer variables—one for the amount of red, one for the amount of green, and the other for the amount of blue—and modify these (by adding or subtracting some amounts) each time through the loop (each of these must be an integer in the range 0 - 255). For this technique you would need to use the constructor for a Color object that takes three integers representing the amount of red, green, and blue as parameters. Other possibilities are to make each arc a different random color, or have set colors (use at least 4 different colors) and cycle through them (using an idea similar to the way the bull's eye program switches between black and white).

Vote Counter, Revisited

Chapter 4 had a lab exercise that created a GUI with two buttons representing two candidates (Joe and Sam) in an election or popularity contest. The program computed the number of times each button was pressed. In that exercise two different listeners were used, one for each button. This exercise is a slight modification. Only one listener will be used and its ActionPerformed method will determine which button was pressed.

The files *VoteCounter.java* and *VoteCounterPanel.java* contain slight revisions to the skeleton programs used in the Chapter 4 exercise. Save them to your directory and do the following:

1. Add variables for Sam - a vote counter, a button, and a label.

2. Add the button and label for Sam to the panel.

3. Modify the ActionPerformed method of the VoteButtonListener class to determine which button was pressed and update the correct counter. (See the LeftRight example or the Quote example for how to determine the source of an event.)

4. Test your program.

5. Now modify the program to add a message indicating who is winning. To do this you need to instantiate a new label, add it to the panel, and add an if statement in ActionPerformed that determines who is winning (also test for ties) and sets the text of the label with the appropriate message.

```
//********************************************************
// VoteCounter.java
//
// Demonstrates a graphical user interface and event
// listeners to tally votes for two candidates, Joe and Sam.
//********************************************************
import javax.swing.JFrame;

public class VoteCounter
{
    //-----------------------------------------------
    // Creates the main program frame.
    //-----------------------------------------------
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Vote Counter");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(new VoteCounterPanel());

        frame.pack();
        frame.setVisible(true);
    }
}
```

```
//********************************************************
// VoteCounterPanel.java
//
// Panel for the GUI that tallies votes for two candidates,
// Joe and Sam.
//********************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VoteCounterPanel extends JPanel
{
    private int votesForJoe;
    private JButton joe;
    private JLabel labelJoe;

    //-----------------------------------------------
    // Constructor: Sets up the GUI.
    //-----------------------------------------------
    public VoteCounterPanel()
    {
        votesForJoe = 0;

        joe = new JButton("Vote for Joe");
        joe.addActionListener(new VoteButtonListener());

        labelJoe = new JLabel("Votes for Joe: " + votesForJoe);

        add(joe);
        add(labelJoe);

        setPreferredSize(new Dimension(300, 40));
        setBackground(Color.cyan);
    }

    //*************************************************
    // Represents a listener for button push (action) events
    //*************************************************
    private class VoteButtonListener implements ActionListener
    {
        //-----------------------------------------
        // Updates the appropriate vote counter when a
        // button is pushed for one of the candidates.
        //-----------------------------------------
        public void actionPerformed(ActionEvent event)
        {
            votesForJoe++;
            labelJoe.setText("Votes for Joe: " + votesForJoe);
        }
    }
}
```

Modifying *EvenOdd.java*

File *EvenOdd.java* contains the dialog box example in Listing 5.21 the text.

1. Compile and run the program to see how it works.
2. Write a similar class named *SquareRoots* (you may modify EvenOdd) that computes and displays the square root of the integer entered.

```
//********************************************************************
//  EvenOdd.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the JOptionPane class.
//********************************************************************

import javax.swing.JOptionPane;

class EvenOdd
{
   //-----------------------------------------------------------------
   //  Determines if the value input by the user is even or odd.
   //  Uses multiple dialog boxes for user interaction.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String numStr, result;
      int num, again;

      do
      {
         numStr = JOptionPane.showInputDialog ("Enter an integer: ");

         num = Integer.parseInt(numStr);

         result = "That number is " + ((num%2 == 0) ? "even" : "odd");

         JOptionPane.showMessageDialog (null, result);

         again = JOptionPane.showConfirmDialog (null, "Do Another?");
      }
      while (again == JOptionPane.YES_OPTION);
   }
}
```

A Pay Check Program

Write a class *PayCheck* that uses dialog boxes to compute the total gross pay of an hourly wage worker. The program should use input dialog boxes to get the number of hours worked and the hourly pay rate from the user. The program should use a message dialog to display the total gross pay. The pay calculation should assume the worker earns time and a half for overtime (for hours over 40).

Adding Buttons to *StyleOptions.java*

The files *StyleOptions.java* and *StyleOptionsPanel.java* are from Listings 5.22 and 5.23 of the text (with a couple of slight changes—an instance variable *fontSize* is used rather than the literal 36 for font size and the variable *style* is an instance variable rather than local to the itemStateChanged method). The program demonstrates checkboxes and ItemListeners. In this exercise you will add a set of three radio buttons to let the user choose among three font sizes. The method of adding the radio buttons will be very similar to that in the QuoteOptionsPanel class (Listing 5.25 of the text). Before modifying the program compile and run the current version to see how it works and study the QuoteOptionsPanel example.

Do the following to add the radio buttons to the panel:

1. Declare three objects *small*, *medium*, and *large* of type JRadioButton.

2. Instantiate the button objects labeling them "Small Font," "Medium Font," "Large Font." Initialize the large font button to true. Set the background color of the buttons to cyan.

3. Instantiate a button group object and add the buttons to it.

4. Radio buttons produce action events so you need an ActionListener to listen for radio button clicks. We can use the ItemListener we already have and let it check to see if the source of the event was a radio button. The code you need to add to *actionPerformed* will be similar to that in the QuoteListener in Listing 5.25. In this case you need to set the *fontSize* variable (use 12 for small, 24 for medium, and 36 for large) in the if statement, then call the *setFont* method to set the font for the *saying* object. (Note: the code that checks to see which check boxes have been selected should stay the same.)

5. In *StyleOptionsPanel()* add each button to the ItemListener object. Also add each button to the panel.

6. Compile and run the program. Note that as the font size changes the checkboxes and buttons re-arrange themselves in the panel. You will learn how to control layout later in the course.

```
//********************************************************************
//  StyleOptions.java       Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//********************************************************************

import javax.swing.JFrame;

public class StyleOptions
{
   //-----------------------------------------------------------------
   //  Creates and presents the program frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Style Options");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      StyleOptionsPanel panel = new StyleOptionsPanel();
      frame.getContentPane().add (panel);

      styleFrame.pack();
      styleFrame.setVisible(true);
   }
}
```

```
//********************************************************************
//  StyleOptionsPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel
{
    private int fontSize = 36;
    private int style = Font.PLAIN;
    private JLabel saying;
    private JCheckBox bold, italic;

    //-----------------------------------------------------------------
    //  Sets up a panel with a label and some check boxes that
    //  control the style of the label's font.
    //-----------------------------------------------------------------
    public StyleOptionsPanel()
    {
      saying = new JLabel ("Say it with style!");
      saying.setFont (new Font ("Helvetica", style, fontSize));

      bold = new JCheckBox ("Bold");
      bold.setBackground (Color.cyan);
      italic = new JCheckBox ("Italic");
      italic.setBackground (Color.cyan);

      StyleListener listener = new StyleListener();
      bold.addItemListener (listener);
      italic.addItemListener (listener);

      add (saying);
      add (bold);
      add (italic);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 100));
    }

   //*****************************************************************
   //  Represents the listener for both check boxes.
   //*****************************************************************
   private class StyleListener implements ItemListener
   {
      //--------------------------------------------------------------
      //  Updates the style of the label font style.
      //--------------------------------------------------------------
      public void itemStateChanged (ItemEvent event)
      {
        style = Font.PLAIN;

        if (bold.isSelected())
            style = Font.BOLD;

        if (italic.isSelected())
            style += Font.ITALIC;
```

```
        saying.setFont (new Font ("Helvetica", style, fontSize));
      }
   }
}
```