Section 1 - The *if* Statement

3B.1.1 The Simplest Form of the *if*

To give your program the feel that there is an intelligent being inside of it -- a little person carrying on a conversation with a user -- you need some form of a conditional statement, otherwise known as an ***if statement***.

The form of the statement is:

```
   if ( <condition> )
      <statement> ;

   <some other statement>;
   // etc.
```

If the **<condition>** is true, then **<statement>** is executed, otherwise it is skipped, and control passes to **<some other statement>**, the one that follows the ***if*** construct. Like the ***loops*** we will encounter, *at most one statement can be controlled by an* ***if.*** However, this isn't a restriction since multiple statements can be bundled into a single compound statement, surrounded by braces:

```
   if ( <condition> )
   {
      <statement> ;
      <statement> ;
      <statement> ;
   }
```

The **<*condition*>** is similar to what goes inside a ***while*** or the test of a ***for*** loop (which we will cover soon). It is something that can either be ***true*** or ***false***, like:

```
   if ( x > y )
      x = x + 4;
```

This says that ***if x is greater than y***, then execute the next statement which adds 4 to x. Otherwise skip that next statement and go on to the one following it.

```
import java.util.Scanner;

public class Foothill
{
   public static void main(String[] args)
   {
      String strAge;
      int intAge;

      // declare an object that can be used for console input
      Scanner inputStream = new Scanner(System.in);

      // get age from user
      System.out.print("How old are you? ");
      strAge = inputStream.nextLine();
      intAge = Integer.parseInt(strAge);

      if (intAge >= 21)
         System.out.println("Enjoy your beer." );
      System.out.print("Have a nice day.");
   }
}
```

The output of run 1:

```
How old are you? 21
Enjoy your beer.
Have a nice day.
```

The output of run 2:

```
How old are you? 19
Have a nice day.
```

Section 2 - The *if/else* Statement

3B.2.1 The *if/else* Statement

The **if** statement is a ***one-way*** choice: if the condition is true we do something, if not we don't. We would now like an either/or, or ***two-way*** choice: ***if*** the condition is true, ***do this*, *if not, do that***. Once we can accomplish that, we can respond one way if the user is under age, and another way if the user is 21 or greater.

For this we add an ***else*** statement:

```
import java.util.Scanner;

public class Foothill
{
   public static void main(String[] args)
   {
      String strAge;
      int intAge;

      // declare an object that can be used for console input
      Scanner inputStream = new Scanner(System.in);

      // get age from user
      System.out.print( "How old are you? " );
      strAge = inputStream.nextLine();
      intAge = Integer.parseInt(strAge);

      if (intAge >= 21)
         System.out.println( "Enjoy your beer." );
      else
         System.out.println( "Sorry, I can't serve you." );

      System.out.print( "Have a nice day." );
   }
}
```

Try the program after making this change and see how it behaves.

Here's another example. Remember how to generate a remainder? Use the ***modulo*** operator, **%**. Here is a console application that gets a number from the user and tells the user whether or not the number is even or odd. Make sure you can follow the logic - why does this program give the correct answer? You have to understand the **%** operator to answer that question.

```
   import java.util.Scanner;
   public class Experiment_3
   {
      public static void main (String[] args)
      {
         Scanner input = new Scanner(System.in);
         int userValue;

         // get the user's number
         System.out.print( "Enter an int: " );
         userValue = input.nextInt();

         if ( userValue % 2 == 0 )
            // if we divide by 2 with no remainder, it's even
            System.out.println( userValue + " is even." );
         else
            // if we divide by 2 with a 1 remainder, it's odd
            System.out.println( userValue + " is odd." );
      }
   }

/* ------------------- output -------------------

RUN 1:
Enter an int: 23
23 is odd.

RUN 2:
Enter an int: 54
54 is even.

------------------------------------------------- */
```

3B.2.2 Indentation of *if/else*

If there are multiple statements inside the if (or else) we need braces. Notice the placement of the braces and indentation:

```
   if ( myAge > 65 )
   {
      System.out.println( "Welcome to Retirement." );
      System.out.println( "Would you like to join AARP?" );
   }
   else
   {
      // whatever goes here ...
   }
```

There are a few other acceptable ***if/else*** styles (see your text or the style booklet for this course).

Point Penalty Disclosure

You must be consistent and use an acceptable style or you will lose one point for every instance of an incorrect style.

Here are a few brace styles for if and if/else statements that I won't, i.e., **WILL NOT**, accept:

```
// double indentation. bad:
if ( myAge > 65 )
   {
      System.out.println( " Welcome to Retirement.");
      System.out.println( "Would you like to join AARP?");
   }
```

nor this

```
// after opening brace, nothing should be on rest of line. bad:
if ( myAge > 65 )
{   System.out.println( " Welcome to Retirement.");
    System.out.println( "Would you like to join AARP?");
}
```

nor this

```
// body of if on same line as if:  bad:
if ( myAge > 65 ) System.out.println( " Welcome to Retirement.");
```

That is, don't place **anything** on the same line as an opening brace after you type it. The reason is it either causes a jagged and confused inner block or a hard-to-read outer framework.

Section 3 - The *if/else/if* Block

3B.3.1 *if/else ... if/else ...*

A common control structure in Java is one in which a number of mutually exclusive conditions are expected, each with its own consequences. By inserting ***else if*** between an ***if statement*** and an ***else statement***, we get this effect - as soon as a condition in the **if** or one of the **if/else** lines is met, the statement or block it controls is executed, and the program continues by skipping to the end of the entire if/else chain:

```
   import java.util.Scanner;
   public class Experiment_3
   {
      public static void main (String[] args)
      {
         Scanner input = new Scanner(System.in);
         int depth;

         // get the user's number
         System.out.print("What is your diving depth in feet? ");
         depth = input.nextInt();

         if (depth < 33)
            System.out.println("Use 2 atm for your calculation.");
         else if (depth < 66)
            System.out.println("Use 3 atm for your calculation.");
         else if (depth < 99)
            System.out.println("Use 4 atm for your calculation.");
         else if (depth < 132)
            System.out.println("Use 5 atm for your calculation.");
         else
            System.out.println("You are down too deep!");
      }
   }

/* ------ two sample runs for this program --------

What is your diving depth in feet? 45
Use 3 atm for your calculation.

What is your diving depth in feet? 140
You are down too deep!

----------------------------------------------  */
```

It looks as if they are not really mutually exclusive, since the < 33 case is "contained in" the < 66 case. But since we don't test < 66 unless the < 33 has failed, then the < 66 case is really going to catch numbers between 33 and 66, and not < 33.

As a technical matter, these are really a bunch of nested if/else statements:

```
if (depth < 33)
   System.out.println("Use 2 atm for your calculation.");
else
{
   if (depth < 66)
      System.out.println("Use 3 atm for your calculation.");
   else
   {
      if (depth < 99)
         System.out.println("Use 4 atm for your calculation.");
      else
      {
         if (depth < 132)
            System.out.println("Use 5 atm for your calculation.");
         else
             System.out.println( "You are way too deep!");
      }
   }
}
```

However, showing this nesting does nothing to improve the understanding of the program. In fact, it obfuscates the true nature of our logic, which is to pigeon-hole a value. The above poor style, is called ***progressive indentation***. It should not be used if all the various cases being tested have equal status. There is nothing "better" about the depth being in the 66-99 range than there is about it being in 33-66 range. Progressive indentation is used when there is some true nesting relationships in the logic. That will be evident in the next example, but you can ask me in the forums if you have any doubt.

Don't Lose Points

Do not use **progressive indentation** like the last example, but rather even indentation (AKA linear or stacked indentation) like the first one.   You will lose a point if you violate this rule, and neither you nor I want that!

If there is logic that is truly nested, then we would use progressive indentation, and perhaps braces, even when not needed:

```
if (state == 'y')
{
   if (price < .12)
      tax = 0;
   else if (price < .24)
      tax = .01;
   else if (price < .36)
      tax = .02;
   else
      tax = price * .083;
}
else
   tax = 0;
```

total = price + tax;

Here I have combined a linear if/else if with a nesting (progressive indentation). Once we know whether or not we are in-state, we can determine the tax. These are two separate questions and should be treated with nesting. However, if we are in-state, the questions of being < .24 or < .36 are part of a pigeon-holing process and should be linearly indented or stacked.

Section 4 - Relational Expressions

3B.4.1 Boolean Type

Before we continue, let me introduce a new primitive data type: **boolean**. **boolean** objects can have only one of two values: **true** or **false**. Those two words, or **boolean *literals***, are reserved words and are the constants of the **boolean** type.

```
boolean x;

x = true;
```

This is the **boolean** version of declaring a variable and then setting it to one of the two legal constants. We'll use this in a moment

3B.4.2 The Relational

In a statement like

```
if (price < .12)
```

it is pretty clear what we are asking about ***price: is it less than .12?***. The complete vocabulary for this type of expression is:

is equal to	is less than	is greater than	is less than or equal to	is greater than or equal to	is not equal to
==	<	>	<=	>=	!=

These are questions, not commands. We say

```
x+y <= 100
```

and we are asking ***is x+y less than or equal to 100?***. If the answer is yes, then the entire expression is replaced with the **boolean** value, **true**. If not, the expression is replaced with the **boolean** value **false**.

```
if (a == b)
   a--;
```

means, ***if a equals b, then decrement (subtract 1 from) a***. (**a**++ is shorthand for **a = a+1** and **a--** is shorthand for **a = a-1**. These are called the ***increment*** and ***decrement*** operators. While I'm on the subject, **a += 4** is shorthand for **a = a+4**, **a *= 5** means **a = a*5**, **a /= 2** means **a = a/2**, and so on.)

The full description of the short expression:

```
if (a == b)
   a--;
```

is as follows:

1. **a == b is tested**. If **a** is equal to **b**, then this expression is replaced by the **boolean** constant **true**. If not, it is replaced by **false**.
2. The expression now becomes either:

```
3.    if (true)
4.       a--;
```

   or

```
      if (false)
         a--;
```

   In the former case, the statement **a--** is executed. In the latter, it is not executed.

A warning to C and C++ programmers:

Boolean is not an integer type. You cannot say

if (x)

in the case that **x** is an **int**. You will get a compiler error.

Section 5 - Logical Operators

3B.5.1 Logical Operators

If the user types a capital **Y** instead of a lower-case **y** in response to:

```
   System.out.print( "Are you in-state? ");
```

then the test:

```
  if (state == 'y')
```

would be considered ***false***. We want to allow for either a **Y** or a **y**. This can be done by combining the two relationals:

```
    (state == 'y')
```

and

```
    (state == 'Y')
```

using the logical ***or***, ||, like so:

```
     if ( (state == 'y') || (state == 'Y') )
```

(To type the || you would hit the vertical pipe bar, |, twice.)

Now the entire expression will evaluate to **true** if either state == 'y' OR state == 'Y'. The parentheses are not necessary in this expression, but they are a good idea: Java has lots of operators whose precedence is counterintuitive. Instead of trying to remember which they are, just over-do the parentheses.

3B.5.2 Truth Tables for the Three Principal Logical Operators

The ***or***, || operator, ***and***, &&, operator and ***not,*** !, operator are defined like so:

OR

a	b	a||b
true	true	true
false	true	true
true	false	true
false	false	false

AND

a	b	a&&b
true	true	true
false	true	false
true	false	false
false	false	false

NOT

a	!a
true	false
false	true

With these statements, you can build rather complex logical expressions. You will see and create many, and there are lots of examples in the text. For now the important thing to remember is that whenever a logical expression is called for, it evaluates to either **true** or **false**, and this is interpreted by the larger statement (the ***if*** statement, e.g.) and a certain action is taken as a result.

3B.5.3 An Example

Here is a (silly but illustrative) state tax example.

Pay attention to:

1. Indentation.
2. User Input
3. Conversion of **String** to **char**
4. Use of print() vs. println()
5. Logical expression

```
import java.util.Scanner;

 public class Sample
 {
    public static void main (String[] args)
    {
       Scanner input = new Scanner(System.in);
       char inState;
       double price, tax;
       String tempString;

       // find out if we should charge state tax
       System.out.print( "Are you in-state? ");
       tempString = input.next();

       // copy the first char only
       inState = tempString.charAt(0);

       // Get the amount
       System.out.print( "What is the amount? ");
       price = input.nextDouble();

       if (inState == 'y' || inState == 'Y')
       {
          if (price < .12)
            tax = 0;
          else if (price < .24)
            tax = .01;
          else if (price < .36)
            tax = .02;
          else
            tax = price * .083;
       }
       else
          tax = 0;

       System.out.println( "The amount of tax is " + tax);
    }
 }
```

3B.5.4 An Example with Strings

If we want to test whether the user typed a "yes" or "no" in response to a question, one easy way to do it (as we have already demonstrated) is to pluck the first character from the answer and test it against 'y' or 'n'. This gives the user a little leeway in his spelling skills. If he messes up the rest of the word, we will forgive him. As long as he types the first letter correctly, we'll act on that, alone.

What if we want to make sure the user types some word or phrase, *exactly*? We have to compare the *entire* string. The way to do this in Java is using the equals() method. It is very simple:

```
   if ( userString.equals("Yes") )
      System.out.println("You typed Yes");
```

Stated in other words:

string1.**equals**(String string2) evaluates to **true** if **string1** and **string2** store identical strings . It returns **false**, otherwise.

We can turn this into a complete program that demands that the user type in either the word **"Yes"** or **"yes"**, but nothing else.

```
import java.util.Scanner;

public class Foothill
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);

      // user-provided values
      String userString;

      // grab the string
      System.out.print( "Type in a word: " );
      userString = input.nextLine();

      if ( userString.equals("Yes")  || userString.equals("yes") )
         System.out.print("You typed an acceptable form of \"yes\".");
      else
         System.out.print("You did not type an acceptable form of \"yes\".");
   }
}
/* ------------------- Sample Run #1 --------------------
Type in a word: yes
You typed an acceptable form of "yes".
---------------------- End Sample Run #1 ---------------- */


/* ------------------- Sample Run #2 --------------------
Type in a word: YES
You did not type an acceptable form of "yes".
---------------------- End Sample Run #2 ---------------- */
```

Notice that we have also demonstrated a technique for including a double quote mark, ", inside a string literal by preceding it with a so-called ***escape character***, otherwise known as the ***backslash*, \.**

Warning About Logical Operators

Be careful to use double && and ||, not single & and |. In Java the single operators mean something entirely different than the double operators and will usually result in logical errors (that means bugs).

Section 6 - A Sample Program Analysis

The key to becoming a successful programmer starts with complete comprehension of the written assignment statement. In other words, it is all about using the English language, not Java or some elaborate mathematical formulae. Whenever there is something in the program specification (or ***the spec***, informally) that seems vague, it is the programmer's responsibility to ask for clarification. You can't blame ***the spec*** if you have access to the spec *writer*, because you have the ability to ask a question.

Here is a very easy program that has a lot of "words" in the description. Today we show how to break down these chatty specs into small, manageable pieces, and attack each piece, individually.

3B.6.1 Statement of Problem

Ask the user how many miles per gallon his car gets in the city and "read in" the value as program input. Then, ask the user approximately how far he has to drive to get to Foothill College (or the nearest place where Java is taught) and capture that value.

Echo this information back to the user (by sending output to the screen) in a sentence, such as *"You get 21.2 MPG, city, and you drive 14 miles round trip each time you come to campus."* There are no computations here; simply display the values that the user entered to make sure your program read in those two values correctly.

If either of these values is negative, issue an error message and end the program. Otherwise, continue.

Assume that the face-to-face section of the same course meets two nights a week over a 12 week period (this includes travel for registration). Tell the user (in an *output* statement) how much money he is saving by taking this class completely on-line assuming three different prices per gallon: $2.75, $3.75, $4.75. In other words, give the user three different savings amounts based on these three different fuel prices.

Your instructor teaches an average of 120 students per quarter. Tell the user how much money is saved *per year*, collectively, by all the instructor's students under the assumption that these students' mileage and distance driven are the same as that entered by the user. Figure three academic quarters per year for this computation.

Foothill College serves about 3500 fully on-line students per quarter. Tell the user how much is saved per year by Foothill's on-line students, collectively, using the above assumptions. (Of course, all this assumes that the students would not be on campus for other reasons before or after the Java class.)

3B.6.2 Strategy

We will do this problem in steps. While learning a language, we always break the program into small chunks and add on slowly, compiling and running each time. We will not go on to the next phase until we have completely compiled and run the last phase without any errors.

Read this last paragraph until you understand it. I said that we *compile and run* each partial program before adding more code into the program. This method does not work if you merely throw in sections of source code after source code, without testing and fully debugging the prior section.

3B.6.3 Phase One

Let's take on only the first part *which asks two questions and echoes the information to the screen without doing any computation.* Before you look at the solution, see if you can do this small portion on your own, using the examples in the earlier modules. Allow yourself to find and fix errors, one at a time, until you have done this small portion. Here is the instructor solution to this first step:

```
import java.util.Scanner;

public class Foothill
{
    public static void main(String[] args)
    {
        // user-provided values
        double mpg, roundTripDistance;

        // declare an object that can be used for console input
        Scanner inputStream = new Scanner(System.in);

        // and a string variable for catching all input
        String strUserInput;

        // Ask for user input
        System.out.print("How many miles per gallon does your car get, city? ");
        strUserInput = inputStream.nextLine();
        mpg = Double.parseDouble(strUserInput);

        System.out.print( "How many miles would you travel, round trip, \n  to"
           + " come to the nearest college campus? " );
        strUserInput = inputStream.nextLine();
        roundTripDistance = Double.parseDouble(strUserInput);

        // Echo results to screen
        System.out.println( "\n\nYou said that you get " + mpg
           + " MPG city and have a \n"
           + " round trip of "
           + roundTripDistance + " miles to/from campus.\n" );
     }
}

/* -------------------- Sample Run ---------------------
How many miles per gallon does your car get, city? 23.5
How many miles would you travel, round trip,
  to come to the nearest college campus? 16


You said that you get 23.5 MPG city and have a
 round trip of 16.0 miles to/from campus.
---------------------- End Sample Run ----------------- */
```

Notice the importance of doing this first phase. It does no computations yet allows us to debug a bunch of tiny but deadly errors before going on to the more difficult parts. In particular, we have spent some time and care doing the following things in the first part:

Notes on Phase 1

- We got out all compiler errors. It probably took five or ten compiles to get out all the compiler errors.

- We made the output look good by putting '\n' characters and spaces when and where they were needed. Every time we ran this, we found that there was something more to fix - it probably took four to six runs to get out all the run-time and neatness errors.
- We made sure the indentation was perfect before moving on.
- We can see that the values the user entered have been correctly echoed to the screen. Many problems in programming come because the programmer is not getting the values accurately from the user. Here we have made certain that we have good values stored in our variables before attempting to compute with them.

3B.6.4 Phase Two

For the second part, let's implement the instruction *"If either of these values is negative, issue an error message and end the program. Otherwise, continue."*

```
import java.util.Scanner;

public class Foothill
{
    public static void main(String[] args)
    {
        // user-provided values
        double mpg, roundTripDistance;

        // declare an object that can be used for console input
        Scanner inputStream = new Scanner(System.in);

        // and a string variable for catching all input
        String strUserInput;

        // Ask for user input
        System.out.print("How many miles per gallon does your car get, city? ");
        strUserInput = inputStream.nextLine();
        mpg = Double.parseDouble(strUserInput);

        System.out.print( "How many miles would you travel, round trip, \n  to"
           + " come to the nearest college campus? " );
        strUserInput = inputStream.nextLine();
        roundTripDistance = Double.parseDouble(strUserInput);

        inputStream.close();  // kills a warning, but not needed

        // Echo results to screen
        System.out.println( "\n\nYou said that you get " + mpg
           + " MPG city and have a \n"
           + " round trip of "
           + roundTripDistance + " miles to/from campus.\n" );

        // now test each number to see if either is negative
        if (mpg < 0)
        {
           System.out.println( "Error - negative mpg detected.");
           return;
        }

        if (roundTripDistance < 0)
        {
           System.out.println( "Error - negative distance detected.");
           return;
        }
     }
}
```

```
/* ------------------- Sample Run --------------------
How many miles per gallon does your car get, city? -23
How many miles would you travel, round trip,
  to come to the nearest college campus? 15


You said that you get -23.0 MPG city and have a
 round trip of 15.0 miles to/from campus.

Error - negative mpg detected.
---------------------- End Sample Run ---------------- */
```

Notes on Phase 2

- To be sure we have everything working, we should run this two more times giving different input: correct **mpg** once and an incorrect **distance** once, to see that all contingencies are correctly handled.

3B.6.5 Phase Three

Next, let's add the third part *which computes the savings for the one user only, based on three different price points for gasoline.* Again, see if you can do this next part on your own. Don't worry - you will get lots of errors and make lots of mistakes. That's why you are doing this - to learn how to fight through these mistakes on your own.

Even for this step, I will break it into sub-steps. I will compute only *one* of the three price points to make sure that the logic works. In other words, I will only use the $2.75/Gallon price point. After that is debugged, I will add the other two price points.

```java
import java.util.Scanner;

public class Foothill
{
   public static void main(String[] args)
   {
      // user-provided values
      double mpg, roundTripDistance, result;

      // declare an object that can be used for console input
      Scanner inputStream = new Scanner(System.in);

      // and a string variable for catching all input
      String strUserInput;

      // three price points and the # trips per quarter
      final double LOW_PPG = 2.75;   // final is optional but nice ...
                                     // ... it means "constant"
      final double MED_PPG = 3.75;
      final double HIGH_PPG = 4.75;
      int tripsPerQuarter;   // could be final if you prefer, like above
      double tempVar; // for holding intermediate results

      // Ask for user input
      System.out.print("How many miles per gallon does your car get, city? ");
      strUserInput = inputStream.nextLine();
      mpg = Double.parseDouble(strUserInput);

      System.out.print( "How many miles would you travel, round trip, \n  to"
         + " come to the nearest college campus? " );
      strUserInput = inputStream.nextLine();
      roundTripDistance = Double.parseDouble(strUserInput);

      // Echo results to screen
      System.out.println( "\n\nYou said that you get " + mpg
         + " MPG city and have a \n"
         + " round trip of "
         + roundTripDistance + " miles to/from campus.\n" );

      // now test each number to see if either is negative
      if (mpg < 0)
      {
         System.out.println( "Error - negative mpg detected.");
         return;
      }

      if (roundTripDistance < 0)
      {
         System.out.println( "Error - negative distance detected.");
         return;
      }
      // establish the number of trips per quarter (not needed if we made
      // it final and gave it the value 12 * 2 above in the declaration
      tripsPerQuarter = 12 * 2; // twice a week for 12 weeks

      tempVar = tripsPerQuarter * roundTripDistance; // this is mls/qtr
      tempVar = tempVar / mpg;  // mls/qtr / mls/gal gives #gallons/qtr

      // now compute the #gals * price-per-gal for each of three price points
      // (only do LOW_PPG until debugged)
      result = tempVar * LOW_PPG;
      System.out.println( "At $" + LOW_PPG + " per gallon, you save $" + result
         + " this quarter.\n");

      inputStream.close();  // kills a warning, but not needed

   }
}

/* ------------------- Sample Run --------------------
How many miles per gallon does your car get, city? 28
How many miles would you travel, round trip,
  to come to the nearest college campus? 19


You said that you get 28.0 MPG city and have a
 round trip of 19.0 miles to/from campus.

At $2.75 per gallon, you save $44.785714285714285 this quarter.
---------------------- End Sample Run ---------------- */
```

Notes on Phase 3

- We did the guts of the computations even though we only did one of the requirements of the program: computing the price for only one student for only one price point. Most beginners get stuck because they refuse to take on just a tiny portion of the assignment for complete debugging, thus spending 10 hours on a program that should only take two or three hours.
- We tested the program with simple numbers like **10 mpg** and **10 miles** round trip. This allows us to do the computation in *our head* (24 gallons per quarter times 1.75 = 42) and check the results for validity. It is important to use numbers that are so simple that you can check the results in your head. If the output does not agree with expectations, you have some bugs to fix. After that, you can put in real numbers.
- We see a new key-word, **final**, in some places, such as:
- `      final double MED_PPG = 3.75;`

We are declaring a new "variable" **MED_PPG** and assigning it a value of **3.75**. By adding the word **final** before the type **double**, we are stating that this "variable" will never hold another value other than its initial, **3.75**. *If we try to modify it, we will get a compiler error - which is good.* Many of the so-called *variables* in our program are made **finals** so that we don't accidentally change them. Furthermore, we use a different style convention for **final** variables. Rather than camelCase, we use **ALL_UPPER_CASE** with underscores, _, between words.

More About final

Any time we have a constant that seems to be useful throughout our program (like the minimum value we want to allow some variable to have, say **MIN_ALLOWABLE_PRICE**, **MAX_NAME_LENGTH** or **MED_PPG**, we declare them to be final and use the symbolic name (**MED_PPG**) everywhere in the program after that, never **3.75**. This way, if we want to change that value to, say, **3.50**, we do it in the one final declaration statement and the entire program will use the new value.

Declare Symbolic Constants and Use them Everywhere

For most literals in your program (like **3.75**) use *symbolic constants* by declaring them **final** and use the variable name (like **MED_PPG**) -- the *symbolic constant* -- everywhere. If you use the **literal value** (**3.75**), even once later in the program ***you will lose points***.

We complete this phase by adding the other two price points, which would result in the following at the end of the code:

```
// now compute the #gals * price-per-gal for each of three price points:
result = tempVar * LOW_PPG;
System.out.println( "At $" + LOW_PPG + " per gallon, you save $" + result
   + " this quarter.\n");

result = tempVar * MED_PPG;
System.out.println( "At $" + MED_PPG + " per gallon, you save $" + result
   + " this quarter.\n" );

result = tempVar * HIGH_PPG;
System.out.println( "At $" + HIGH_PPG + " per gallon, you save $" + result
   + " this quarter.\n\n" );
```

I won't show the whole program - you can make this augmentation and run it for yourself.

3B.6.6 Phase Four

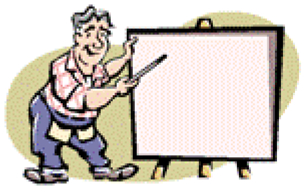
The last parts should be done in two more small steps, I'll do one of them *which computes the savings for your instructor's students, as a group, per year.* You can do the remaining part, which solves the problem for all of Foothill's online students per year. Try the first of these two sub-parts yourself before you look. Read the statement of the last parts carefully because there are a couple ways that you would get the wrong answer if you read too fast or carelessly.

```
import java.util.Scanner;

public class Foothill
{
   public static void main(String[] args)
   {
      // user-provided values
      double mpg, roundTripDistance, result;

      // declare an object that can be used for console input
      Scanner inputStream = new Scanner(System.in);

      // and a string variable for catching all input
      String strUserInput;

      // three price points and the # trips per quarter
      final double LOW_PPG = 2.75;   // final is optional but nice ...
                                     // ... it means "constant"
      final double MED_PPG = 3.75;
      final double HIGH_PPG = 4.75;
      int tripsPerQuarter;   // could be final if you prefer, like above
      double tempVar; // for holding intermediate results

      // Ask for user input
      System.out.print("How many miles per gallon does your car get, city? ");
      strUserInput = inputStream.nextLine();
      mpg = Double.parseDouble(strUserInput);

      System.out.print( "How many miles would you travel, round trip, \n  to"
         + " come to the nearest college campus? " );
      strUserInput = inputStream.nextLine();
      roundTripDistance = Double.parseDouble(strUserInput);

      // Echo results to screen
      System.out.println( "\n\nYou said that you get " + mpg
         + " MPG city and have a \n"
         + " round trip of "
         + roundTripDistance + " miles to/from campus.\n" );

      // now test each number to see if either is negative
      if (mpg < 0)
      {
         System.out.println( "Error - negative mpg detected.");
         return;
      }

      if (roundTripDistance < 0)
      {
         System.out.println( "Error - negative distance detected.");
         return;
      }

      // establish the number of trips per quarter (not needed if we made
      // it "final" and gave it the value 12 * 2 above in the declaration
      tripsPerQuarter = 12 * 2; // twice a week for 12 weeks

      tempVar = tripsPerQuarter * roundTripDistance; // this is mls/qtr
      tempVar = tempVar / mpg;  // mls/qtr / mls/gal gives #gallons/qtr

      // now compute the #gals * price-per-gal for each of three price points:
      result = tempVar * LOW_PPG;
      System.out.println( "At $" + LOW_PPG + " per gallon, you save $" + result
         + " this quarter.\n");

      result = tempVar * MED_PPG;
      System.out.println( "At $" + MED_PPG + " per gallon, you save $" + result
         + " this quarter.\n" );

      result = tempVar * HIGH_PPG;
      System.out.println( "At $" + HIGH_PPG + " per gallon, you save $" + result
         + " this quarter.\n\n" );

      // do the same thing, but this time for loceff's classes as a whole and
      // for the whole year, not just one quarter:
      result = tempVar * LOW_PPG * 120 * 3; // we could do this more
                                            // efficiently by
                                            // introducing another temp
                                            // variable, but this extends the
                                            // above forumla in a way that is
                                            // easy to understand.
      System.out.println( "At $" + LOW_PPG + " per gallon, the instructor's "
         + "students will collectively \nsave $" + result + " per year.\n" );

      inputStream.close();  // kills a warning, but not needed

      // you can throw in the other price points yourself here ... then finish.
   }
}

/* ------------------- Sample Run --------------------
How many miles per gallon does your car get, city? 28
How many miles would you travel, round trip,
  to come to the nearest college campus? 21


You said that you get 28.0 MPG city and have a
 round trip of 21.0 miles to/from campus.

At $2.75 per gallon, you save $49.5 this quarter.

At $3.75 per gallon, you save $67.5 this quarter.

At $4.75 per gallon, you save $85.5 this quarter.


At $2.75 per gallon, the instructor's students will collectively
save $17820.0 per year.
---------------------- End Sample Run ---------------- */
```

Notes on Phase 4

- I stopped without completing the program. You can do this now on your own very easily. Most of it is copy and paste from here on out. Run it and debug it until your program performs the whole enchilada requested above.
- This time we used more realistic numbers. Since we have already proved the earlier computations, it is now trivial to check the newly added computation in our heads. $50 times $360 is $18000 - trivial, right? Still, it is important to check; if we made an error in the code we can catch it now, before moving on.
- There are some cosmetic defects (too many lines skipped in one place, and, elsewhere, we need to add a blank line for clarity). These should be fixed before moving on to the next steps.
- This assignment, although it has many steps, is very simple. If there is any single line -- or even a single word or letter -- that you do not completely understand, you should ask me or your fellow students rather than begin *Assignment #3*.

Prevent Point Loss



- **Do not use ASCII or UNICODE.** Numbers, such as 65 or 57 are meaningless to programmers. Instead use understandable chars like 'A' or '9'. In addition to being more readable and less error-prone, this rule makes your job easier. (1 - 2 point penalty)
- **Write to spec.** Don't change the names or purpose of program variables, methods or output. Write your program so that it does exactly what is asked, not *sort-of* or *an enhanced version of* the assignment. (4 - 12 point penalty)
- **Pull common statements out of multiple if/else blocks.** If you have the same statement inside every one of your if/else blocks in a particular if/else statement, then it should be removed from each block and placed before or after the entire if/else statement. For instance if you have x = x+1; inside the if block and all the else blocks, then you are going to add 1 to x no matter what ... so you may as well add it before or after the block and remove the individual statements inside the blocks. (1 - 2 point penalty)
- **Check for errors before you compute.** If the user can supply an incorrect input to your program, test for this before, not after, you use the input to compute. If it is incorrect, you won't do the computation. In other words, test the input first, and don't do the computation if the test shows invalid input. (2 - 3 point penalty)
- **Indent *if* statements.** Look at examples in the lessons or text - there are many. These are expensive errors. (2-4 point penalty)

Good work this week.