Computer Programming

Unit 2 WS01 Decimal Formatting

We have currently found ways to produce decimal results that meet our expectations. For example, we have techniques which allow us to round money to two places after the decimal point. This may be nice; however, if the last digit of our decimal is zero, it does not get displayed. This is especially important if we are dealing with dollars and cents. Our current ways do have some weaknesses and tend to be cumbersome.

In this course, we will focus upon on using the Decimal Format feature of JAVA to learn how to format our decimal results. The Decimal Format class will allow us to format our results to include commas (for large numbers), dollar signs, and the number of places before and after the decimal point.

First of all, much like public static void main (String [ ] args) being required at the beginning of all of our JAVA classes, in order to use the Decimal Format class our program must include the code shown below.

REQUIRED IMPORT STATEMENT:

```
Package whateverPackageNameYouHaveChosen;
import java.text.*;                    //Notice this is directly after your package name.
                                       //NOT after public static void main …
```

EXAMPLE: Suppose we wish to format a given double a=13.95667 to the nearest hundredth. The code below will do this.

```
double a=13.95667;
DecimalFormat df = new DecimalFormat("##.##");      // df is a variable we choose
System.out.println(df.format(a));                   // df is just the name of the formatter
                                                    // Output is 13.96
```

NOTE: The # symbol is telling JAVA to print a number if it is there!

PROBLEM: The following code will then output 14 only … not 14.00 (read and understand) …

```
double a=13.99667;
DecimalFormat df = new DecimalFormat("##.##");      // df is a variable we choose
System.out.println(df.format(a));                   // df is just the name of the formatter
                                                    // Output is 14
```

If this code was formatting money, the output should be 14.00 to show 14 dollars and 0 cents. There is a fix for this … remember, the # is telling JAVA to print a number if it is there. If the # is changed to a zero, JAVA is forced to show a zero if nothing is there. Notice the following code that will print numbers when they are there, but will show zeros if anything is missing:

```
double a=13.99667;
DecimalFormat df = new DecimalFormat("##.##");      // df is a variable we choose
System.out.println(df.format(a));                   // df is just the name of the formatter
                                                    // Output is 14.00
```

HOW TO ADD SYMBOLS:

When dealing with money, it is usually desired to have the $ show up before the output. In addition, when dealing with very large numbers it is usually desired to use commas (ex: 1,000,000). So, rather than trying to use concatenation and multiple print statements that will break up our output repeatedly, the decimal formatter can automatically add these for us!

EXAMPLE: Notice in the code below that the output is now formatted as dollars and cents and shows it!

```
double a=13.99667;
DecimalFormat df = new DecimalFormat("$##.##");      // df is a variable we choose
System.out.println(df.format(a));                    // df is just the name of the formatter
                                                     // Output is $14.00
```

In fact, DecimalFormat can be forced to output anything. Look at what happens in the following:

```
double a=13.99667;
DecimalFormat df = new DecimalFormat("abc$##.##");   // df is a variable we choose
System.out.println(df.format(a));                    // df is just the name of the formatter
                                                     // Output is abc$14.00
```

So, suppose we wish to multiply the numbers 854.31 by 52.31 and output the result as a dollar amount with commas. The actual product is 44,688.9561. The code below will do this AND will add the comma every 3 decimal only when needed!

```
double a=854.31;                                     //Even though the '#' sign
double b=52.31;                                      //appears 6 times, it will
DecimalFormat df=new DecimalFormat("$###,###.00");   //only use what's necessary
System.out.println(df.format(a*b));                  //Output is $44,688.96
```

ROUNDING ISSUES:

This particular decimal formatting code is very powerful, but does have one significant limitation. It always ***rounds*** to the nearest stated value. There may be instances where you need results always rounded up or always rounded down. A likely fix would be to add or subtract the appropriate decimal amount from each value.

For example, if you wanted to have results go to the nearest hundredth (3 decimal places) <u>but always round up</u>, you would add 0.005 to your variable before you do the final decimal formatting. It works!

If you instead would like to have results go to the nearest hundredth (3 decimal places), <u>but always round down</u>, you would subtract 0.005 from your variable before you do the final decimal formatting. It works!

Exercises

1) Create a class titled ***powerRounding***. Initialize two double variables to 3.45 and 4.56. Use your variables and DecimalFormat to write JAVA code that will find the value of 3.45^4.56 rounded to the nearest tenth.

2) An employer pays you $9.75 per hour for a summer job. Over the summer, you will work 424.25 hours and receive one lump-sum paycheck in the fall. Create a class titled ***summerWages***. Initialize two variables to equal your hourly wage and hours work. Use your variables and DecimalFormat to print a statement summarizing the number of hours worked, hourly wage, and total wages that you will earn. Be sure to format money using $ and showing all cents.

3) In working out a large problem (number problem that is, not personal problem), you need to calculate the answer to 2622.7 * 3681.455. Create a class called ***bigNumbers*** that uses DecimalFormat to multiply these two numbers, round to two decimal places, and show commas in the appropriate positions needed for large numbers (you do not need to store the two numbers as variables) .