**Name**

**Advanced Programming in Java**

**Lab Exercise 10/16/2024**

**Formatting – Lesson 27**

1. Using the *NumberFormat* class, write code that will create a *double* called *cv*, assign it a value of 18.7713, and then convert it to a *String* rounded off to the nearest hundredth.

Assure that at least 1 decimal place is printed. Print the *String*.

2. What type variable is returned by the *format( )* method of the *NumberFormat* class?

3. Using the *NumberFormat* class, write code that will create a *double* called *dv*, assign it a value of 184.767123, and then convert it to a *String* rounded off to the nearest

thousandth. Assure that at least 2 decimal places are printed.

4. Using the *NumberFormat* class, write code that will input a *double* called *db* from the

keyboard and then convert it to a *String* in which there are at least 3 decimal places… and

at most 4.

5. Assume you already have a floating type variable *mn* that you want to display as dollars and cents (example, $127.15). Using the *NumberFormat* class, write code that will produce such a printout (including the dollar sign).

6. What import does the *NumberFormat* class require?

7. What is the output of the following code?

NumberFormat nf = NumberFormat.getCurrencyInstance( );

System.out.println(nf.format(487.0871));

8. What is the output of the following code?

NumberFormat nf = NumberFormat.getPercentInstance( );

nf.setMinimumFractionDigits(3);

nf.setMaximumFractionDigits(3);

String str = nf.format(4.708832);

System.out.println(str);

9. What class lets you specify patterns like “0,000,000.##” in determining the formatting of a number?

10. In calling the *getNumberInstance( )* method of the *NumberFormat* class, why do we have to preface it with *NumberFormat.* as in *NumberFormat.getNumberInstance( )*?

11. What is printed by the following code? Indicate each space (if any) with a tilde(~).

Formatter fmt = new Formatter( );

fmt.format(“%s--->%-,10.3f--->%08.1e”, “Formatting example”, 189.11081, .07642);

System.out.println(fmt);

12. Suppose you have a *Formatter* object called *f*. Write code that will use *f* to produce

*String s* having *double d* left justified in a field 12 characters wide and rounded to 4

decimal places.

13. Suppose you have a *Formatter* object called *f*. Write code that will use *f* to produce

*String s* having *int i* left justified in a field 11 characters wide. Use comma separators.

14. What is output by the following code? Indicate each space (if any) with a tilde(~).

System.out.printf(“--->|%3.6s|<---”, “x”);

15. Suppose you need to produce a table that looks like the following:

57012 $1,200,586.22

00026 $ 187.91

00729 $ 571,267.03

Here is a code fragment that produces this table:

for(int j = 0; j < 3; j++)

System.out.printf(“???????”, num[j], money[j]);

Supply the correct syntax for “???????” so that there are 13 spaces between the end of the

first column and the start of the visible part of the second column. The *num* array is an *int*

array and *money* is a *double* array.

16. What does the following print?

Formatter fmt = new Formatter( );

int i = 4893;

fmt.format(“start%012dend”,i);

String s = fmt.toString( );

System.out.println(s);

**Project… BaseClass (Shell)**

We will create a generic class (sometimes called a shell) for reading/processing files that also includes a *NumberFormat* object as well as *StringTokenizer* and *Scanner* objects. We will call this class *BaseClass* and every time we have a programming project that requires file input, we will begin by simply pasting in this code. We would then change the name of the class from *BaseClass* to whatever the new class name is to be and finally, lay in the additional code to accomplish the task at hand.

import java.io.\*; //necessary for File and IOException

import java.util.\*; //necessary for StringTokenizer and Scanner

import java.text.\*; //necessary for NumberFormat

public class BaseClass

{

public static void main( String args[] ) throws IOException

{

NumberFormat formatter = NumberFormat.getNumberInstance( );

formatter.setMinimumFractionDigits(3); //may need to change value

formatter.setMaximumFractionDigits(3); //may need to change value

Scanner sf = new Scanner(new File("c:\\temp\_Name\\FileName.in"));

int maxIndx = -1; //-1 so when we increment below, the first index is 0

String text[] = new String[100]; //To be safe, declare more than we need

while(sf.hasNext( ))

{

maxIndx++;

text[maxIndx] = sf.nextLine( );

}

//maxIndx is now the highest index of text[]. Equals -1 if no text lines

sf.close( ); //We opened a file above, so close it when finished.

for (int j = 0; j <= maxIndx; j++)

{

//Typically, only one of the following two will be used.

//StringTokenizer st = new StringTokenizer( text[j] );

//Scanner sc = new Scanner(text[j]);

//…code specific to the task…

}

}

}

**Project… Gymnastics**

Use your new *BaseClass* class to implement the following project. Call the new class *Gym*:

Ten people are judging an international gymnastics competition. Each judge gives a contestant a performance score between 0.0 and 10.0, inclusive, with the score given to one decimal place.

Since some judges favor their own country’s competitors and/or give lower scores than deserved to their country’s rivals, the highest and lowest scores are discarded before averaging the eight other scores. Write a program that will read in the judges’ ten scores, discard the highest and lowest score, and compute the average of the eight other scores to **four** decimal places.

**Input**

Read in one or more data sets (assume you don’t know ahead of time how many) of 10 scores from the file *DataGym.in*. Each data set will use exactly one line of the input text file. There will be ten floating point numbers (each separated from the others by spaces) between 0.0 and 10.0, inclusive (to one decimal place) on each line of the file.

Input file

8.7 6.5 0.1 3.2 5.7 9.9 8.3 6.5 6.5 1.5

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0

**Output**

Print, for each data set that is input, the average to **four** decimal places. This average should be preceded each time by “For Competitor #X, the average score is ”, where X denotes the competitor’s position (starting with 1) in the input file.

Output to screen for above input file

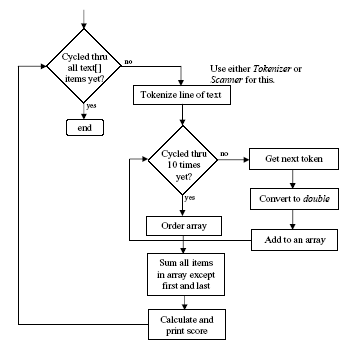
For Competitor #1, the average is 5.8625

For Competitor #2, the average is 0.0000

For Competitor #3, the average is 1.0000

On the next page is a flowchart that shows the flow of the logic involved in solving this problem. Notice two nested loops whose starting points are depicted with diamonds. Typically, diamonds are used for decisions (*if* statements); however, they are used here to depict the decisions of whether to stay in the loops or not.

On highly complex problems it is common practice to begin program design with a flow chart. The gymnastics program is not a very complicated program; however, most students find the flowchart a significant aid on this problem.



**Project… MyPower**

Create a MyPow application that uses the formula to calculate XY. The MyPow application should prompt the user for 2 numbers and then display the result from the formula and, for comparison, show the same result using the Math pow() method. The application should display output similar to:

Enter a value for X: 7

Enter a value for Y: 5

The result from using the formula is: 16806.99999999998

The result from using the Math pow() method is: 16807.0

**Project… Growth**

The formula  can be used for estimating growth where:

y is the final amount

n is the initial amount

k is a constant

t is the time

For example, this formula could be used for estimating population growth in a region or for estimating cell growth in a lab experiment. Create a BacteriaGrowth application that calculates how many bacteria will be present based on this formula. The application should prompt the user for the initial bacteria, the constant k, and the time. Application output should look similar to:

Enter the initial bacteria amount: 5

Enter a constant value for k: 0.8

Enter growth time period in hours: 8

3009.2 bacteria will be present after 8.0 hours.