CSE 142, Autumn 2008
Programming Assignment #7: Personality Test (20 points)
Due: Tuesday, November 18, 2008, 11:30 PM

This assignment focuses on arrays and file processing. Turn in a file named `PersonalityTest.java`. You will also need input file `personality.txt` from the course web site. Save this file in the same folder as your program.

The assignment involves processing data from a personality test. There is a link on the course web site where you can take the personality test yourself. Student answers will be included in a data file distributed to the whole class.

Background Information:

The Keirsey Temperament Sorter (http://www.keirsey.com/) is a personality test that involves answering 70 questions. Each question has two answer choices, which we will refer to as the "A" and "B" answer. The person taking the test is allowed to leave a question blank, in which case the answer will be recorded with a dash (`'-'`).

The Keirsey test measures four independent dimensions of personality:

1. *Extrovert versus Introvert* (E vs. I): what energizes you
2. *Sensation versus iNtuition* (S vs. N): what you focus on
3. *Thinking versus Feeling* (T vs. F): how you interpret what you focus on
4. *Judging versus Perceiving* (J vs. P): how you approach life

Individuals are categorized as being on one side or the other for each dimension. The corresponding letters are put together to form a personality type. For example, if you are an Extrovert, iNtuitive, Thinking, Perceiving person then you are referred to as an ENTP. The "A" answers correspond to E, S, T, and J (the left-hand choices above). The "B" answers correspond to I, N, F, and P (the right-hand choices above). For each dimension, we determine a percentage of B answers the user gave for that dimension between 0 and 100, to indicate whether the person is closer to the "A" or "B" side.

Suppose that someone's answers are as follows (These are the answers given by "Betty Boop" later in this document).

Dimension	**# of "A" answers**	**# of "B" answers**	**% of "B" answers**	**Result**
Extrovert/Introvert	1	9	90%	I
Sensing/iNtuition	17	3	15%	S
Thinking/Feeling	18	2	10%	T
Judging/Perceiving	18	2	10%	J

We add up how many of each type of answer we got for each dimension. Then we compute the percentage of B answers for each dimension. Then we assign letters based on which side the person ends up on for each dimension. In the Extrovert/Introvert dimension, for example, Betty gave 9 "B" answers out of 10 total (90%), which means she is on the B side, which is "Introvert" or I. The overall percentages are (90, 15, 10, 10) which works out to a personality type of ISTJ.

Mechanics of the Personality Test:

Suppose that "Betty Boop" gave the following answers for the 70 questions, in order from 1 to 70:

```
BABAAAABAAAAAAABAAAABBAAAAAABAAAABABAABAAABABABAABAAAAAABAAAAAABAAAAAA
```

The questions are organized into 10 groups of 7 questions, with the following repeating pattern in each group:

1. The first one question in each group is an Introvert/Extrovert question (questions 1, 8, 15, 22, etc).
2. The next two questions are for Sensing/iNtuition (questions 2 and 3, 9 and 10, 16 and 17, 23 and 24, etc).
3. The next two questions are for Thinking/Feeling (questions 4 and 5, 11 and 12, 18 and 19, 25 and 26, etc).
4. The next two questions are for Judging/Perceiving (questions 6 and 7, 13 and 14, 20 and 21, 27 and 28, etc).

In other words, if we consider the I/E to be dimension 1, the S/N to be dimension 2, the T/F to be dimension 3, and the J/P to be dimension 4, the map of questions to their respective dimensions would look like this *(spaces added for emphasis)*:

```
1223344 1223344 1223344 1223344 1223344 1223344 1223344 1223344 1223344 1223344
BABAAAA BAAAAAA ABAAAAB BAAAAAA BAAAABA BAABAAA BABABAA BAAAAAA BAAAAAA BAAAAAA
```

Notice that there are half as many Introvert/Extrovert questions as there are for the other three dimensions.

Input Data:

The personality data consists of line pairs, one per person. The first line has the person's name, and the second has the person's 70 answers (`'A'`, `'B'` or `'-'`). Notice that the A or B can be either upper or lowercase. A dash represents a question that was skipped by that person, as seen in Han Solo's data below.

Input file `personality.txt` (partial):

```
Betty Boop
BABAAAABAAAAAAABAAAABBAAAAAABAAAABABAABAAABABABAABAAAAAABAAAAAABAAAAAA
Bugs Bunny
aabaabbabbbaaaabaaaabaaaaababbbaabaaaabaabbbbabaaaabaabaaaaaabbaaaaabb
Han Solo
BA-ABABBB-bbbaababaaaabbaaabbaaabbabABBAAABABBAAABABAAAABBABAAABBABAAB
```

Program Behavior:

Your program's output begins with an introduction of your own creation, such as an explanation of the personality test and/or a fact about your own type. Next your program asks for the input file name to process. You may assume that the user will type a name of a file that exists and can be read. Next the program prompts the user for an output file name; if this file already exists, your program will overwrite its contents. (This is the default `PrintStream` behavior.)

Log of execution (user input underlined):

```
<< your introduction message here >>

Input file name: personality.txt
Output file name: output.txt
```

Please do not hard-code file names in your code. The input file name used in the above log is `personality.txt`, but the input file could have a different name. You should not need to place the string `"personality.txt"` anywhere in your program's code. Similarly, the output file name happens to be named `output.txt` in the log shown on this page, but this will not always be the case, and your code should not contain the string `"output.txt"`.

Each pair of lines from the input file is turned into a group of lines in the output file with the name, count of As and Bs for each dimension, % Bs for each dimension (rounded to the nearest percent), and personality type. Questions left blank (indicated by a dash, `'-'`) do not contribute to the count of As/Bs, nor to the percent Bs for that dimension. If the person has the same number of As and Bs for a dimension, give them an "X", as with Han Solo below. Assume that the input file is valid, has no errors or illegal data, and that nobody has skipped all questions for a dimension (it would be impossible to determine a percentage in such a case). Reproduce the following output format:

Output file `output.txt` (partial):

```
Betty Boop
Answers: [1A-9B, 17A-3B, 18A-2B, 18A-2B]
Percent B: [90, 15, 10, 10]
Type: ISTJ

Bugs Bunny
Answers: [8A-2B, 11A-9B, 17A-3B, 9A-11B]
Percent B: [20, 45, 15, 55]
Type: ESTP

Han Solo
Answers: [2A-8B, 9A-9B, 11A-9B, 15A-5B]
Percent B: [80, 50, 45, 25]
Type: IXTJ
```

Implementation Guidelines and Hints:

A major purpose of this assignment is to test your understanding of arrays. Therefore you should use arrays to store the various data for each of the four dimensions of the personality test, and to transform data from one form to another. There are three major transformations you should perform: from the original 70 A/B answers to counts of As and Bs; from counts to percentages of Bs; and from percentages to a four-letter personality type string. These transformations are summarized by the following diagram using Han Solo's data:

```
 Answers:   "BA-ABABBB-bbbaababaaaabbaaabbaaabbabABBAAABABBAAABABAAAABBABAAABBABAAB"
    ↓       What is computed                  Resulting output
 A count:   {2, 9, 11, 15}
 B count:   {8, 9, 9, 5}                      Answers: [2A-8B, 9A-9B, 11A-9B, 15A-5B]
    ↓
B percent:  {80, 50, 45, 25}                  Percent B: [80, 50, 45, 25]
    ↓
  Type:     "IXTJ"                            Type: IXTJ
```

To count As and Bs, read data from the input file one line at a time using `Scanner`'s `nextLine` method. Analyze each character within the overall line `String` as a `char` value using the line string's `charAt` method.

The process of converting the 70 answers into counts of As and Bs is one of the most challenging aspects of this program. Pay extra attention to this code to minimize redundancy. If you are having trouble counting the As and Bs, it may help you to print the contents of your arrays. Recall that the method `Arrays.toString` converts an array to a printable `String`. For example, to print the contents of an array named `counts`, you could write the following code:

```
System.out.println("My counts are " + Arrays.toString(counts));
```

You must display percentages of Bs as integers. Use `Math.round` or `printf` to round numbers to the nearest integer. If you use `Math.round`, you will have to cast its result to type int if you want to store the result in an `int` variable or array.

Your program is supposed to produce file output, but it may be easier for you to develop it initially by printing all output to the console. Initially, instead of printing output to a file, consider printing all output to the console, `System.out`. Then once you have the program logic working, add the file output to your program. See textbook section 6.4 for more information about using a `PrintStream` object to produce file output.

Stylistic Guidelines:

Use a **class constant** in your program representing the number of dimensions in the test, as an `int` (with a value of 4). It will not be possible to change this constant and have the program adjust, but it makes the code more readable.

We will grade your method structure very strictly on this assignment. Use at least **four nontrivial methods** besides `main`. These methods should use parameters and returns, including passing and returning arrays, as appropriate. The methods should be well-structured and avoid redundancy. Chapter 7's Case Study program in textbook section 7.5 is a good example of how to write a larger program with several methods that use arrays as parameters.

Your `main` method should be a concise summary of the overall program. It is okay for `main` to contain some code such as `println` statements, or for `main` to open files and perform some file I/O. But the `main` method should not perform too large a share of the overall work itself, such as examining each of the 70 characters of a person's answers to the personality test. Also avoid "chaining," which is when many methods call each other without ever returning to `main`. For reference, our solution is around 100-120 lines long and has 5 methods besides `main`, though you don't need to match this.

We will also check strictly for redundancy on this assignment. If you have a very similar piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a method, by using `for` loops over the elements of arrays, and/or by factoring `if/else` code as described in section 4.3 of the textbook.

You are limited to features in Chapters 1 through 7. Follow past style guidelines such as indentation, names, variables, types, line lengths, and comments (at the beginning of your program, on each method, and on complex sections of code).

Extra Credit:

On this assignment we offer you two opportunities to earn extra credit by adding additional features to your program. You may choose to implement one, both, or neither of these features. Each extra feature is worth **+1 point of extra credit** if implemented properly. A perfect program that implements both features would earn 22 / 20 points. To get the point for a given feature, it must work correctly and also be implemented without poor style or excessive redundancy.

Since attempting to add these features could break other parts of your program, we suggest you save a backup of your program or turn it in once before attempting these features, and then turning it in again if you complete them successfully.

Extra Feature #1: Re-prompting for Input File Name

The first extra feature, worth +1 points, is for your program to re-prompt the user for the input file name if the file typed is not found. After the user types the name of the input file to use, your program should check whether this file exists in the current folder. If so, the program continues as normal. But if not, your program should re-prompt the user repeatedly until the user types a file name that does exist. The following log shows an example of this:

Log of execution (user input underlined):

```
<< your introduction message here >>

Input file name: notfound.doc
Input file name: error.in
Input file name: oops.txt
Input file name: personality.txt
Output file name: results.dat
```

Note: You are not looking for the specific file name `personality.txt`. You are prompting until they type any file name that exists. If the data were stored in `input.dat`, your program would stop prompting when the user types that name. Assume that an existing file contains valid data; you do not have to worry about the case where the user types the name of a file that exists but does not contain personality data. See the lecture slides or book section 6.4 for help with this feature.

Extra Feature #2: Default Output to `System.out`

The second extra feature, also worth +1 points, is for your program to print its output to the console (`System.out`) rather than to the output file if the user types a blank output file name. To do this, you should use `nextLine` to read the output file name and then test whether it is an empty string; if you used `next`, the blank line wouldn't be accepted because it contains no tokens. To implement this extra feature in a non-redundant way, take advantage of the fact that `System.out` is an object of type `PrintStream` and can be passed as a parameter and used just like any other `PrintStream` object.

Log of execution (user input underlined):

```
<< your introduction message here >>

Input file name: personality.txt
Output file name:
Betty Boop
Answers: [1A-9B, 17A-3B, 18A-2B, 18A-2B]
Percent B: [90, 15, 10, 10]
Type: ISTJ

Bugs Bunny
Answers: [8A-2B, 11A-9B, 17A-3B, 9A-11B]
Percent B: [20, 45, 15, 55]
Type: ESTP

Han Solo
Answers: [2A-8B, 9A-9B, 11A-9B, 15A-5B]
Percent B: [80, 50, 45, 25]
Type: IXTJ
```