CSE 142, Summer 2013
Programming Assignment #4: Budgeter (20 points)

Due: Tuesday, July 23, 2013, 11:30 PM

This interactive program focuses on `if/else` statements, cumulative sums, `Scanner`, and returning values. Turn in a file named `Budgeter.java`. To use a `Scanner` for console input, you must `import java.util.*;` in your code. The program reads as input two users' incomes and monthly expenses, and reports to them a budget, which user has greater net earnings, and a piece of custom financial advice.

Below is one example log of execution from the program. This program behaves differently depending on the user input; user input is bold and underlined below. Your output should match our examples exactly given the same input. (Be mindful of spacing, such as after input prompts and between output sections.) **Look at the other example logs on the course web site** to get more examples of the program's behavior.

```
This program reads details of the income and
expenses of two individuals and helps
budget their money.

Person #1:
How many sources of income? 1
   Next income name: tutoring
   tutoring monthly amount: 500

How many living expenses? 3
   Next expense name: rent
   rent monthly amount: 450.00
   Next expense name: internet
   internet monthly amount: 45.50
   Next expense name: phone
   phone monthly amount: 60

Total income: 500.0
Total expenses: 555.5
You are 55.5 over budget.

Person #2:
How many sources of income? 2
   Next income name: baristaing
   baristaing monthly amount: 700.0
   Next income name: babysitting
   babysitting monthly amount: 200

How many living expenses? 2
   Next expense name: rent
   rent monthly amount: 600
   Next expense name: insurance
   insurance monthly amount: 100

Total income: 900.0
Total expenses: 700.0
You are 200.0 under budget.
This leaves 6.67 to spend or save each day.

Person 2 has greater net earnings by 255.5

<< your custom financial advice here >>
```

The program begins with an introduction message that briefly explains the program. For each of two people, it then prompts for their number of incomes, then for however many incomes they entered, it prompts for the names and amounts of each income. It then prompts for the number of expenses, and for each expense it prompts for the name and amount of the expense. Each line that prompts for a name or amount begins with three spaces.

After a user has entered all of their expenses, the program prints several details about their budget, including their total income, total expenses, and a message about whether they are "at budget" (their expenses equal their income), "over budget" (their expenses exceed their income), or "under budget" (their income exceeds their expenses) and by how much. If they are under budget, it also reports how much they can spend or save each day of the month to remain at budget (their excess income divided by 30 days). If their income is equal to their expenses, the program prints the message:

```
You are exactly at budget.
```

Once the information has been entered for both people, the program then reports which person had greater net earnings (income minus expenses) and by how much, or if their net earnings are the same, it prints the message:

```
Both people have the same net earnings.
```

Finally, it prints a custom piece of financial advice of your own creation. The advice does not need to be factual or in any way scientifically-based. For example, your advice may be something like the following:

```
Studies show that trimming your diet to
entirely store-brand macaroni and cheese can
decrease your food expenses by 300%.
```

All real-numbered values printed by the program should be rounded to two decimal places. However, you should not try to print exactly two decimal places when there are trailing zeros (e.g. 25.00). To do this requires `printf`, which you can read about in Ch. 4.3 of the textbook, but is not required for this class. You may assume that all user input is valid; that is, all numbers of incomes or expenses are integers, all names of budget items are one-word `Strings`, and all amounts of budget items are real numbers (with any number of decimal places). You may also assume that the user will not enter negative values for any numbers.

Style and Implementation Guidelines:

You should use static methods to eliminate redundant code and to break the problem up into logical subtasks. Your main method should be a concise summary so that a person can easily see the overall structure of the program. It is acceptable to have a few `println`s in main, though you should limit this as much as possible. You should avoid "chaining" long sequences of method calls together without returning to main. **You are to introduce at least five static methods other than main and round2** to break this problem up into smaller subtasks and you should make sure that no single method is doing too much work. Introduce parameters as necessary to eliminate redundancy between your methods and return values so that methods can return computed values to their caller.

In this program, **none of your methods should have more than 15 lines of code** in the body of the method (not counting blank lines or lines with just curly braces on them). The 15-line limitation is a special requirement for this assignment because we want to force you to practice breaking up a program into methods.

Remember that because this program involves both integer data and real data, you need to use appropriate type declarations (type `int` and calls on `nextInt` for integer data, type `double` and calls on `nextDouble` for real-valued data). Finally, you should construct only one `Scanner` object for console input.

Your code will involve conditional execution with if and if/else statements. Part of your grade will come from using these statements appropriately. You may want to review section 4.1 of the textbook about if/else statements and section 4.4 about methods with conditional execution.

For this assignment, you are limited to Java features from Ch. 1-4. Give meaningful names to methods and variables, and use proper indentation and whitespace. Follow Java's naming standards as specified in Chapter 1. Localize variables when possible; declare them in the smallest scope needed. Include meaningful comment headers at the top of your program and at the start of each method. Limit line lengths to 100 chars.

Turn in your program electronically in a file named `Budgeter.java`.

Development Strategy and Hints:

- Look at the program BMI.java from the 07-12 lecture for a good example of eliminating redundant code and breaking a problem up into logical subtasks (see also the discussion of this program in section 4.6 of the textbook).
- As we did in BMI.java, first write the program to just work with one user. Once it successfully works with one user, expand its functionality to work with two users.
- To compute the total incomes and expenses, you will need to cumulatively sum individual incomes and expenses. See textbook section 4.2 about cumulative sums.
- Many students get "cannot find symbol" compiler errors. Common mistakes include forgetting to `import java.util.*;` forgetting to pass or return a needed value, forgetting to store a returned value into a variable, and referring to a variable by the wrong name.
- To round real-numbered values to two decimal places, you should use the round2 method from lecture.
- Use `Math.abs` as necessary to report the difference between two values.
- In the past, we were lenient about partial-line redundancy, or redundancy where lines are partially similar. Now, however, you should do your best to eliminate partial-line redundancy using variables and parameters of type `int`, `double`, and `String`.
- The Output Comparison Tool contains several example executions for you to compare your output to. For full credit, your solution should produce no differences with any of these examples, with the exception of your custom financial advice.