CSE 142, Autumn 2018
Programming Assignment #4: Budgeter (20 points)

Due Tuesday, October 23, 2018, 9:00 PM

This interactive program focuses on `if/else` statements, `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.

This program prompts a person for income and expense amounts, then calculates their net monthly income. Below are two example logs of execution from the program. This program's behavior is dependent on the user input (user input is bold and underlined below to make it stand out). Your output should match our examples exactly when 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 asks for your monthly income and
expenses, then tells you your net monthly income.

How many categories of income? 3
    Next income amount? $1000
    Next income amount? $205.25
    Next income amount? $175.50

Enter 1) monthly or 2) daily expenses? 1
How many categories of expense? 4
    Next expense amount? $850
    Next expense amount? $49.95
    Next expense amount? $75
    Next expense amount? $120.67

Total income = $1380.75 ($44.54/day)
Total expenses = $1095.62 ($35.34/day)

You earned $285.13 more than you spent this month.
You're a big saver.
<<your custom message>>
```

```
This program asks for your monthly income and
expenses, then tells you your net monthly income.

How many categories of income? 2
    Next income amount? $800
    Next income amount? $200.25

Enter 1) monthly or 2) daily expenses? 2
How many categories of expense? 1
    Next expense amount? $45.33

Total income = $1000.25 ($32.27/day)
Total expenses = $1405.23 ($45.33/day)

You spent $404.98 more than you earned this month.
You're a big spender.
<<your custom message>>
```

The program begins with an introductory message that briefly explains the program. The program prompts the user for the number of income categories, then reads in that many income amounts.

Next, the program asks whether the user would like to enter monthly or daily expenses. (The user enters 1 for monthly and 2 for daily.) The program will then read in a number of expense categories and an amount for each category, similar to how income was read.

The program should then print out the total amount of income and expenses for the month, as well as the average amount of each per day. You may assume a month has exactly 31 days, though your program should be able to be easily modified to change this assumption (see below). After printing the total income and expenses, the program should print out whether the user spent or earned more money for the given month and by how much. If income and expenses were exactly equal, the user is considered to have earned $0 more than they spent (as opposed to spending $0 more than they earned).

Finally, the program should print out which category the user falls into based on their net income for the month (rounded to two decimal places). The user's net income is the result of subtracting their expenses from their income. The four categories are defined as follows:

More than +$250: "big saver"	**More than -$250 but not more than $0**: "spender"
More than $0 but not more than +$250: "saver"	**-$250 or less**: "big spender"

After printing the user's category, print a custom message of your choice about their spending habits. This message should be different for each range shown above. It should be at least 1 line of any non-offensive text you like.

This program processes user input using a `Scanner`. All monetary inputs will be real numbers; all other input will be integers. You may assume the user always enters **valid input**. When prompted for a value, the user will enter a value of the correct type. The user will enter a number of income and expense categories ≥ 1, and will only ever enter 1 or 2 when asked how to enter expenses. The amount for each category of income and expense will be a non-negative number.

You should define a **class constant** for the number of days in a month. You should refer to this constant throughout your program so that the value can be changed and your program will continue to function correctly.

Development Strategy and Hints:

- Tackle parts of the program one at a time, rather than writing the entire program at once. Write a bit of code, get it to compile, and test what you have so far. If you try to write large amounts of code without attempting to compile it, you may encounter a large list of compiler errors and/or bugs.
- To compute total income and expenses, use a **cumulative sum** as described in lecture and in textbook section 4.2.
- Many students get "cannot find symbol" compiler errors. Common mistakes include forgetting to pass / return a needed value, forgetting to store a returned value into a variable, and referring to a variable by the wrong name.
- All monetary output values are printed with 2 digits after the decimal point. If the second digit after the decimal point (the "hundredths" digit) would be a zero, your program can include this in the output or not. Achieve this with a custom rounding method (as shown in lecture) or with `System.out.printf`. The following code prints variable `x` rounded to the nearest hundredth:

```
double x = 1.2345;
System.out.printf("x is around %.2f in size.\n", x);  // 1.23
```

- Even though your program is rounding numbers to two decimal places when they are displayed, it should not round the numbers that are used to compute results.
- If you are getting averages of 0 regardless of what data the user types, you may have a problem with integer division. See Chapter 2 about types `int` and `double`, type-casting, and how to avoid integer division problems. If you have a value of type `double` but need to convert it into an `int`, use a type-cast such as the following:

```
double d = 5.678;
int i = (int) d;   // 5
```

Style Guidelines:

For this assignment, you are limited to Java features from Chapters 1-4 of the textbook. A major part of this assignment is demonstrating that you understand parameters and return values. Use static methods, parameters, and returns for structure and to eliminate redundancy. For full credit, your program must have and use **at least 4 non-trivial methods** other than `main`. (For reference, our solution is 86 lines long (58 "substantive"), with 7 methods other than `main`, though you do not need to match these numbers.)

Like on previous assignments, you should not have `println` statements in your `main` method. Also, `main` should be a concise summary of the overall program, and should make calls to several other methods that implement the majority of the program's behavior. Your methods will need to make appropriate use of parameters and return values. Each method should perform a coherent task and should not do too large a share of the overall work. Avoid lengthy "chaining" of method calls, where each method calls the next, no values are returned, and control does not come back to `main`. See textbook Chapter 4's case study for a discussion of well-designed versus poorly-designed methods.

When handling numeric data, you are expected to choose appropriately between types `int` and `double`. You will lose points if you use type `double` for variables in cases where type `int` would be more appropriate.

Some of your code will use conditional execution with `if` and `if/else` statements. You are expected to use these statements properly. Review book sections 4.1-4.3 about nested `if/else` statements and factoring them.

Give meaningful names to methods, variables, and parameters. Use proper indentation and whitespace. Follow Java's naming standards as specified in Chapter 1. Limit line lengths to 100 chars. Localize variables when possible; (i.e. declare them in the smallest scope needed). Include meaningful header comments at the top of your program and at the start of each method.