Assignment 4: Budgeter (20 points)   *due July 21, 2020, 11:59pm*

This assignment focuses on `if-else` statements, return values, and user input via `Scanner`. Turn in the following Java file using the link on the course website:

- `Budgeter.java` – A program that calculates a user's monthly budget as described below

Program Behavior

This program prompts a person for income and expense amounts, then calculates their net monthly income. Unlike previous assignments, this programs behavior is dependent on input from a user (user input is bold and underlined below). Your output should match our examples exactly when given the same input, but if the input changes, the output will also. Additional execution logs will be posted on the course website, and you can use the Output Comparison Tool to check your output for various inputs.

*Make sure that the format and structure of your output **exactly** match the given logs.*

```
_____ Expected Output #1 _____
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? $250.25
    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 = $1425.75 ($45.99/day)
Total expenses = $1095.62 ($35.34/day)

You earned $330.13 more than you spent this month.
You're a big saver.
<< Your custom message goes here >>
```

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

After reading all the user input, the program should then print out the total amount of income and expenses for the month, as well as the average income and expense per day. You may assume a month has exactly 31 days, though you should use a class constant so that your program can be easily modified to change this assumption (see below). 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 spent $0 more than they earned (as opposed to earning $0 more than they spent).

Finally, the program should print out which category the user falls into based on their net income for the month, where the net income is the result of subtracting the user's expenses from their income. The category should be followed by a custom message of your choice about the user's spending habits. This message must be different for each of the four ranges shown below, and should consist of at least one line of any non-offensive text you choose. The categories are defined below. You may find this graphic helpful in understanding the boundaries of the various categories.

Each category must have a separate, unique message.

Net income range	Category
More than +$250	big saver
More than $0 but not more than than +$250	saver
More than -$250 but not more than $0	spender
-$250 or less	big spender

All monetary values in output should be rounded to two decimal places. This can be achieved either using a rounding method (as shown in lecture) or using `System.out.printf`– either approach is acceptable. If the second digit after the decimal point (the "hundreths" digit) is a zero, your program may or may not print out this zero, as long as it is consistent in all cases. You MUST print out two digits if the second digit is not a zero.

Development Strategy

To be able to use `Scanner` in your code, you will need to include the following line of code at the beginning of your program (before your `public class` declaration):

```
import java.util.*;
```

Hints

This assignment is significantly more complex than previous ones. The following suggestions and hints may help you be more successful:

```
 ┌──────── Expected Output #2 ────────┐
 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 goes here >>
```

- Approach the program one part at a time, rather than trying to write most or all of the program at once. Add additional output to test incomplete portions of your code, and compile and test often.

- You will need to use a cumulative sum to compute the total income and expenses, as described in lecture and section 4.2 of the textbook.

- If you are receiving "Cannot find symbol" errors from the compiler, look carefully at your parameters and return values and ensure they are all included and named correctly.

- Be careful of using integer division when you should be using real number division, and vice versa. Values of type `int` can be used when a `double` is expected, but to use a `double` where you need an `int` you will need to use a cast, such as:

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

Implementation Guidelines

User Input

This program requires you to process user input, which you must do using a `Scanner`. You may assume that all monetary inputs will be real numbers, and that all other input will be integers. You may also assume the user always enters valid input. Specifically, you may assume that:

- the user will always enter a value of the correct type

- the user will always enter a number of income and expense categories ≥ 1

- the user will only ever enter 1 or 2 when asked how to enter expenses

- the user will only enter a non-negative amount for each category of income and expense

Class Constant

As described above, your program should assume there are 31 days in a month by default, but this value should be able to be easily changed. You must introduce a class constant for the number of days in a month, and use this constant throughout your program. See the course website and the Output Comparison Tool for example logs with alternate values for the constant.

Working with Numbers/Rounding

For all numerical values in your program, you should use the type (`int` or `double`) that is more appropriate for the value. In particular, you should NOT simply use `double` for all values. In addition, you must not round values until they are output; all calculations should be performed using the full values.

Permitted Java Features

For this assignment, you are restricted to Java concepts covered in chapters 1 through 4 of the textbook. In particular, you MUST use parameters and return values.

Style Guidelines

You should follow all guidelines in the Style Guide and on the General Style Deductions page of the course website. Pay particular attention to the following elements:

Capturing Structure

Your program must eliminate redundancy to the greatest extent possible. Remember that our definition of redundancy has evolved as we learned new material and you will need to factor code that is similar even if it isnt identical. Your `main` method should be a concise summary of your program's structure. You should not produce any output in your `main` method, though you may have more code in `main` than you have on previous assignments. In addition, each method should perform a single, coherent task. To receive full credit, your program must include at least **four (4)** non-trivial methods other than `main`. (For reference, our solution consists of 86 lines of code, 58 of which are "substantive", and 7 methods besides `main`.)

For this assignment, you should eliminate **all** redundancy.

Using Parameters and Returns

Your program should utilize parameters and return values effectively to produce a well-structured program as described above. Your methods should not accept unncessary or redundant parameters. In particular, your program should include only a single `Scanner` which is passed as a parameter to all required methods. You should NOT declare your `Scanner` as a constant; you must pass it as a parameter. Your program should also utilize return values to avoid "chaining" (where each method calls the next without returning to `main`).

Your program **must** include at least four non-trivial methods, but you are **NOT** required to match our line or method count.

Using Conditionals

Your program will need to utilize conditionals (`if` statements) at various points to achieve the correct behavior. You are expected to use the correct form of these statements (`if`, `if-else`, and/or `if-else if`) for each situation, and to factor common code out of each branch.

The case study in chapter 4 of the textbook and the election program from lecture both demonstrate how to avoid chaining and how to reduce redundancy.

Code Aesthetics

Your code should be properly indented, make good use of blank lines and other whitespace, and include no lines longer than 100 characters. Your class, methods, variables, and constant should all have meaningful and descriptive names and follow the standard Java naming conventions. (e.g. `ClassName`, `methodOrVariableName`, `CONSTANT_NAME`). See the Style Guide for more information.

Commenting

Your code should include a header comment at the start of your program, following the same format described in Assignments 1 and 2. Your code should also include a comment at the beginning of each method that describes that method's behavior, each parameter, and the method's return value. Comments should be written in your own words (i.e. not copied and pasted from this spec) and should not include implementation details (such as describing loops or expressions included in the code). See the Style Guide or lecture code for examples.

Getting Help

If you find you are struggling with this assignment, make use of all the course resources that are available to you, such as:

- Reviewing relevant lecture examples

- Reviewing this week's section handouts

- Reading the textbook

- Visiting the IPL

- Posting a question on the message board

Academic Integrity

Remember that, while you are encouraged to use all resources at your disposal, including your classmates, **all work you submit must be entirely your own**. In particular, you should **NEVER** look at a solution to this assignment from another source (a classmate, a former student, an online repository, etc.). Please review the full policy in the syllabus for more details, and ask the course staff if you are unclear on whether or not a resource is OK to use.