Programming Assignment 1

Getting started

This week's project requires the knowledge of only some very basic Java operations on numeric values, but is, nevertheless, not trivial due to its logic. In particular, it focuses on the use of integer division and the remainder (a.k.a. the mod, or %) operator.

When working on a project always make sure you understand the program requirements first, then think about the algorithm that you will use, write a sketch of it in English and test it, and only after you have thought through the details of it, start working on its implementation in Java.

When working on the program it is important to learn to **develop it gradually by implementing one logical step of the algorithm at a time and testing the program after implementing each step.** As one CS professor wrote in his advice to the students: *Never ever use the Freshman Despair Approach:*

1. *Procrastinate;*
2. *Type in all the code;*
3. *Fix all the compile-time errors;*
4. *Fix all the run-time errors;*
5. *Add comments.*

Unless the program is trivial, this approach will always be too time consuming.

Programming Project: GymSchedule **worth 12 points**

Scheduling gym classes.

A neighborhood gym is offering one exercise class multiple times every day. The following rules apply.

1. The first class always starts at 10:30 a.m. and the last one must end not later than 8:30 p. m.; there is a short break after each class, *including the last one.*
2. The price of the class is based on its length. The base price is $3; 20 cents are added to the base price for each full 15 minutes of class time. For example, an hour and 30 minute-long class (i.e. 90-minute long class) would cost 3 + 20 * 6 = 4.20 dollars, since there are 6 consecutive full 15-minute intervals within the 90 minute period. An hour and 29 minute-long class would cost only $4.00, since there are only 5 full 15-minute intervals in an hour and 29 minutes.

Help the owner determine how many classes he can fit in the day's schedule as well as how much to charge for each class by creating a program that works exactly as follows.

The program should get the following information from the user:

1. two parameters denoting the length of the class (in hours and minutes),
2. the length of the after-class break (in minutes),
3. the name of the class.

The program will then compute and output the class information, including:

- the price of the class, based on rule 2 above,
- how many classes would fit in the day's schedule based on rule 1 above,
- until what time would the gym have to stay open (past 8:30 p.m.) in order to allow for one more class and the subsequent break.

Here a sample interaction (user input is shown in **boldface**)

```
This program computes how many gym classes can be offered in a day.
Please enter the name of the class: Aerobics
Please enter the length (in hours and minutes) of the class.
Hours: 1
Minutes: 15
Now enter the length of the after-class break in minutes: 10
**********************************************
Class name: "Aerobics"
Length: 75 min. + 10 min. break
Price: $4.0
Can be offered 7 times a day
To offer one more class, stay open until 21:50,
that's 80 minutes past the regular closing time.
**********************************************
```

Hints:

- The diagram shown in Figure 1 illustrates how
- The easiest way of dealing with time and time intervals is to convert everything to minutes and work with this representation. For example, the start time of 10:30 can be converted to minutes as follows 10 * 60+30 = 630 minutes from the start of the day (midnight), and end of the day is 20*60 + 30=1230. The length of the whole day is 1230 – 630 = 600 minutes.
- The length of the class in the sample interaction above is 1* 60+15 = 75 minutes. With the break, the length of class and break adds up to 85 minutes. To compute how many times the class can be offered, divide the length of the whole day in minutes by the class+break time.
- To convert the minute representation back to hours and minutes, use the integer division and the remainder operators (= and %), e.g. minute value 1310 broken to hours and minutes is 1310/60 = 21 hours and 1310%60 = 50 minutes.

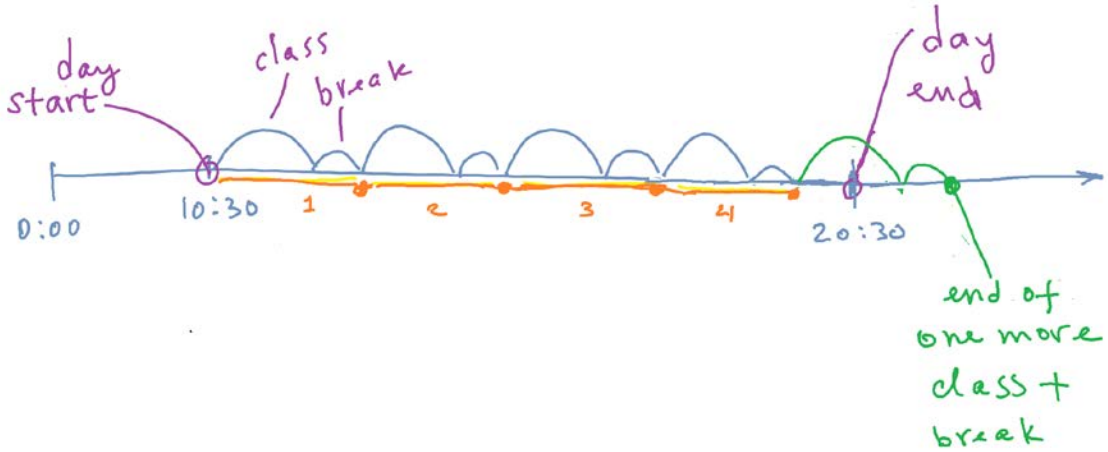

Figure 1. Diagram showing the workday and classes on a timeline. For this scenario, there are four classes per day.

In this project you should not worry about the user entering invalid data (for instance, negative value of hour) and about formatting the dollar amounts and time parameters to be displayed with exactly two digits for cents and minutes. Your program will be tested with valid input only and the formatting of numbers will be ignored in grading.

Grading: The grading schema for this project is roughly as follows:

- Your program should compile without syntax errors to receive any credit. If a part of your program is working, you will receive partial credit, but only if the program compiles without syntax errors.
- 2 points for reading the inputs correctly and outputting them (first two lines after the ****s), in the specified above order and format. Note that in this and all other programs you **must use only one Scanner object**, or points will be subtracted.
- 3 points for correctly computing the price of the class,
- 3 points for the correct computation of the number of times a class can be offered in one day,
- 2 points for correctly computing the extended time of closing and additional minutes gym should be open for one more class,
- 2 points will be awarded for good programming style, as defined below.

Important Note: Your program will be tested by a computer program before I evaluate it. The tester program is not intelligent enough to interpret the output line and deduce which part of it represents which information. **Therefore, the input and output should appear in the exact order that is shown in the sample interaction.**

The **style requirements** for this and future assignments are as follows:

- Include *introductory comments* listing the name of the author and a brief description of the purpose of the program. Also provide comments within the program code for any part of code that may be difficult to follow and would benefit from explanatory text.

- Use variable *names that reflect the purpose of the variable*, and add comments if more information would be helpful (for example, units of measurement or valid values).

- Use *named constants* to store program data where appropriate.

- Indent code within curly braces for improved readability (Eclipse can do most of the indenting automatically, just **use Source-> Correct Indentation and Source-> Format**. Proper indentation makes it easier for you to debug, because then the layout of the code matches the functionality.

Remember that the best way to develop a program is by working on it incrementally and periodically verifying correctness of each developed part. For instance, after developing the code that computes the price, test it on various inputs to make sure that this part works correctly, and after that, proceed with computing the number of classes per day.

You can verify that the intermediate values computed by your program (e.g. the number of full 15-minute intervals in the class) are correct by printing them out. This technique is called *debugging output*. The extra printing statements must later be removed when the final version of the program is produced.

Finally, after testing your program thoroughly, before you submit,

- re-read the requirements again to verify that you have not missed anything,
- close Eclipse, and submit your .java