Assignment 07 – Decomposed Tennis

Due: Friday, November 24, 2017 @ 11:59pm

In the last assignment, we created several methods. In this assignment, you will be doing this again, but you will also have to *design* the methods.

The process of taking a large program and breaking it up is called functional decomposition. That will be the major portion of the work for this assignment. Technically, this entire assignment could be done in one function, but I wouldn't want to see what that looks like. 😱

Tennis Scoring Challenge

Sometimes you want your program to make a racquet, sometimes you don't...

Scoring a tennis match is easy. The matches are best of three, so winning two sets wins the match. To win a set, you just have to win 6 games. Of course, you have to win by at least two games, so if you've won 6 games and your opponent won 5 games, you aren't done. Plus if you both have exactly 6 wins, that will trigger a tie-breaker game which has a different set of rules then a standard game… How does that go again? Hmm, perhaps you should refer to an official source?

That page lists many alternate forms of scoring. We will be using advantage scoring for the games, so a player must win a game by at least two points to win a game. For the sets, we will use tie-break sets. That means that if the set reaches 6-6, a special tie-break game will be played to 7 points (with the same rule about winning by at least two points).

MAKE THIS HAPPEN:

1. Ask the user whether they would like to score a game, tie-break game, set or match.
2. The input will be a sequence of 1s and 2s. All other input should be ignored with an error printed for the user.
3. As the games, sets and match proceed, you should print out when a game is won, a set is won or a match is won. In addition, any time a score is changed, it should be printed.

EXAMPLE OUTPUT:

Here is a sample run (user input is in bold):

```
Would you like to score a (g)ame, (o)vertime game, (s)et, or a (m)atch: g
1
Game score: 1-0
1
Game score: 2-0
1
Game score: 3-0
1
Game score: 4-0

Game over: Player 1 won
```

Note that you can enter multiple inputs on one line. Here is a (shortened) run that scores an entire match:

```
Would you like to score a (g)ame, (o)vertime game, (s)et, or a (m)atch: m
1 2 1 2 1 1
Game score: 1-0
Game score: 1-1
Game score: 2-1
Game score: 2-2
Game score: 3-2
Game score: 4-2

Game over: Player 1 won
Set score: 1-0
2 2 2 2
Game score: 0-1
Game score: 0-2
Game score: 0-3
Game score: 0-4

Game over: Player 2 won
Set score: 1-1

...

Set over: Player 1 won
Match score: 1-0
2 1 2 1 2 1 2 1 1
Game score: 0-1
Game score: 1-1
Game score: 1-2
Game score: 2-2
Game score: 2-3
Game score: 3-3
Game score: 3-4
Game score: 4-4
Game score: 5-4
Game score: 6-4

Game over: Player 1 won
Set score: 1-0

...

Set over: Player 1 won
Match score: 2-0

Match over: Player 1 won
```

Here is an example run with some errors:

```
Would you like to score a (g)ame, (o)vertime game, (s)et, or a (m)atch: g
One
Invalid player # entered: One
1
Game score: 1-0
9
Invalid player # entered: 9
1
Game score: 2-0
-1
Invalid player # entered: -1
0.1
Invalid player # entered: 0.1
1 1
Game score: 3-0
Game score: 4-0

Game over: Player 1 won
```

NOTES:

- It is your responsibility to break the program down into functions. You are likely to have at least 8 functions, including your `run` function, but you may have several more depending on how you decide to proceed.
- Nothing in your program should be `static`.
- All methods other than your `run` method should be `private`.
- You should create one `Scanner` and share it among all of your functions. This is done for you in the starter code.
- To help you avoid errors when the user enters a non-number, check out the `hasNextInt` function in the `Scanner` API documentation.
- If the user enters an invalid choice for what to score, your program should print an error and do nothing else.
- All of the methods in your class must have a comment with at least a one sentence description. In addition to this, your `run` method and at least 4 other methods must be documented with a full JavaDoc as described by your instructor.
- You do **not** need to document the class, as this has been done for you in the starting code, but you **should** add your name.

Submission Instructions

Create a folder named `Asg07_`*`lastname_firstname`* and place your BlueJ project (the contents of the `solution` folder) in the folder. Submit your source code to the submit folder (`I:\Labs\CompSci\Submit\1501\00x`). If you are submitting your files via the web interface from off campus, you will need to compress your files into one .zip file.