Sahaj software | inspiring brilliance

Guidelines

At Sahaj, we strive to build high quality software that has strong aesthetics (is readable and maintainable), has extensive safety nets to safeguard quality, handles errors gracefully and works as expected, without breaking down, with varying input.

We are looking for people who can write code that has flexibility built in, by adhering to the principles of Object Oriented Development, and have the ability to deal with the real-life constraints / trade-offs while designing a system.

It is important to note that we are not looking for a GUI and we are not assessing you on the capabilities around code required to do the I/O. ***The focus is on the overall design****. So, while building a solution, it would be nicer if input to the code is provided either via unit tests or a file. Using command line (for input) can be tedious and difficult to test, so it is best avoided.*

Following is a list of things to keep in mind, before you submit your code, to ensure that your code focuses on attributes, we are looking for -

- Is behaviour of an object distinguished from its state and is the state encapsulated?
- Have you applied SOLID principles to your code?
- Have you applied principles of YAGNI and KISS (additional info here)?
- Have you unit tested your code or did TDD? If you have not, we recommend you read about it and attempt it with your solution. While we do not penalise for lack of tests, it is a definite plus if you write them.
- Have you looked at basic refactoring to improve design of your code? Here are some guidelines for the same.
- Finally, and foremost, are the principles applied in a pragmatic way. Simplicity is the strongest of the trait of a piece of code. However, easily written code may not necessarily be simple code.

Sahaj software | inspiring brilliance

Problem Statement

(Average time to write a solution 4-8 hrs)

A new game in carrom-board called **Clean Strike** is played by 2 players with multiple *turn*s. A *turn* has a player attempting to strike a coin with the striker. Players alternate in taking turns. The game is described as follows:

- There are 9 black coins, a red coin and a striker on the carrom-board
- Strike - When a player pockets a coin he/she wins a point
- Multi-strike - When a player pockets more than one coin he/she wins 2 points. All, but 2 coins, that were pocketed, get back on to the carrom-board
- Red strike - When a player pockets red coin he/she wins 3 points. If other coins are pocketed along with red coin in the same turn, other coins get back on to the carrom-board
- Striker strike - When a player pockets the striker he/she loses a point
- Defunct coin - When a coin is thrown out of the carrom-board, due to a strike, the player loses 2 points, and the coin goes out of play
- When a player does not pocket a coin for 3 successive turns he/she loses a point
- When a player **fouls** 3 times (a *foul* is a turn where a player loses, at least, 1 point), he/she loses an additional point
- A **game is won** by the first player to have won at least 5 points, in total, and, at least, 3 points more than the opponent
- When the coins are exhausted on the board, if the highest scorer is not leading by, at least, 3 points or does not have a minimum of 5 points, the game is considered a draw

Write a program that takes in the outcome of each turn as input and outputs the result of the game as and when applicable along with necessary statistics that supports the result. Please find sample input and output below:

Sample Input:

```
Player 1: Choose an outcome from the list below
    1. Strike
    2. Multistrike
    3. Red strike
    4. Striker strike
    5. Defunct coin
    6. None
> 1

Player 2: Choose an outcome from the list below
    1. Strike
    2. Multistrike
    3. Red strike
    4. Striker strike
    5. Defunct coin
    6. None
> 6
.
.
.
Player 1 won the game. Final Score: 15-11
```