Ruby Challenge

Description

This project is designed to test your knowledge of back-end web technologies, specifically in Ruby and assess your ability to create back-end products with attention to details, standards, and reusability.

Assignment

The goal of this exercise is to demonstrate your ability to build a greenfield project, specifically a command-line application to score a game of ten-pin bowling (https://en.wikipedia.org/wiki/Ten-pin_bowling#Rules_of_play).

The code should handle the bowling scores rules described in the specs and here: https://www.youtube.com/watch?v=aBe71sD8o8c

Mandatory Features

- The program should run from the command-line and take a text file as input

- The program should read the input text file and parse its content, which should have the results for several players bowling 10 frames each, written according to these guidelines:

    a. Each line represents a player and a chance with the subsequent number of pins knocked down.
    b. An 'F' indicates a foul on that chance and no pins knocked down (identical for scoring to a roll of 0).
    c. The rows are tab-separated.

Example:

```
Jeff 10
John 3
John 7
Jeff 7
Jeff 3
John 6
John 3
Jeff 9
Jeff 0
John 10
Jeff 10
John 8
John 1
Jeff 0
Jeff 8
John 10
Jeff 8
Jeff 2
John 10
Jeff F
Jeff 6
John 9
John 0
Jeff 10
John 7
John 3
Jeff 10
John 4
John 4
Jeff 10
Jeff 8
Jeff 1
John 10
John 9
John 0
```

- The program should handle bad input like more than ten throws (i.e., no chance will produce a negative number of knocked down pins or more than 10, etc), invalid score value or incorrect format

- The program should output the scoring for the associated game according to these guidelines:

  a. For each player, print their name on a separate line before printing that player's pinfalls and score.
  b. All values are tab-separated.
  c. The output should calculate if a player scores a strike ('X'), a spare ('/') and allow for extra chances in the tenth frame.

So for the above game for Jeff, the classic scoring would be written:

Frame	1		2		3		4		5		6		7		8		9		10		
Pinfalls		X	7	/	9	0		X	0	8	8	/	F	6		X		X	X	8	1
Score	20		39		48		66		74		84		90		120		148		167		

Your program should print out a similar score to standard out, in the format:

```
Frame       1       2       3       4       5       6       7       8       9       10
Jeff
Pinfalls        X   7   /   9   0       X   0   8   8   /   F   6       X       X   X   8   1
Score       20      39      48      66      74      84      90      120     148     167
John
Pinfalls    3   /   6   3       X   8   1       X       X   9   0   7   /   4   4   X   9   0
Score       16      25      44      53      82      101     110     124     132     151
```

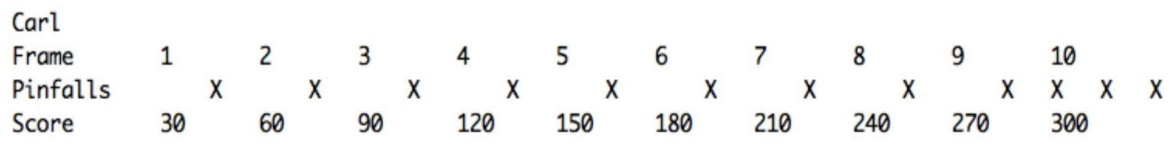
Here is the same output with hidden whitespace revealed:

```
Frame»  »   1»  »   2»  »   3»  »   4»  »   5»  »   6»  »   7»  »   8»  »   9»  »   10¶
Jeff¶
Pinfalls»   »   X»  7»  /»  9»  0»  »   X»  0»  8»  8»  /»  F»  6»  »   X»  »   X»  X»  8»  1¶
Score»  »   20» »   39» »   48» »   66» »   74» »   84» »   90» »   120»»   148»»   167¶
John¶
Pinfalls»   3»  /»  6»  3»  »   X»  8»  1»  »   X»  »   X»  9»  0»  7»  /»  4»  4»  X»  9»  0¶
Score»  »   16» »   25» »   44» »   53» »   82» »   101»»   110»»   124»»   132»»   151¶
```

Your program should be able to handle all possible cases of a game both including a game where all rolls are 0, all rolls are fouls (F) and a perfect game, where all rolls are strikes:

```
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
Carl 10
```

Frame	1		2		3		4		5		6		7		8		9		10		
Pinfalls		X		X		X		X		X		X		X		X		X	X	X	X
Score	30		60		90		120		150		180		210		240		270		300		



```
Carl
Frame       1       2       3       4       5       6       7       8       9       10
Pinfalls        X       X       X       X       X       X       X       X       X   X   X   X
Score       30      60      90      120     150     180     210     240     270     300
```

- Unit test: Tests should cover at least the non-trivial classes and methods

Bonus

- Integration test: At least cover the three main cases: Sample input (2 players), perfect score, zero score

Considerations

- SRP: Single Responsibility Principle (Classes are self contained. They do the task they need to do and nothing else).
- Liskov’s Substitution Principle: Interfaces (OOP, Swap principle... Makes the program able to be extended).
- Dependency Inversion Principle: Code should depend on interfaces, no concrete implementations.
- Coupled code: If code has too few classes we should not accept this candidate, even if the program works perfectly. Code violates SRP, classes demonstrate mixed concerns. Bad OOP in general.
- Abuse of class methods and singleton: Usually this indicates a junior candidate since this makes difficult to use the Substitution Principle.
- Duplicated code: Reused code should be encapsulated.
- Include well-known libraries.
- Packaging complete. Zip should contain a readme file explaining how to compile the project, and contain the test text file to check the output
- Project structure: It should be the standard Ruby project layout, no IDE specific or custom.
- Keep your code versioned with Git locally.

Deliverables

When you finish the assignment, send a zip file (don’t forget to include the .git/ folder.) or upload your project to your Git repo (Github, BitBucket, etc...) and share the repository link with your initial contact via email. Indicate which, if any, of the bonus tasks you completed.

If you didn’t manage to finish everything, please tell us which parts you completed.