Java Project: maintaining high scores on the Internet

Objectives:

- Manipulating I/O instructions: streams from/to files as well as web pages
- Parsing strings to recover some information in the middle of the line
- Handling Exceptions

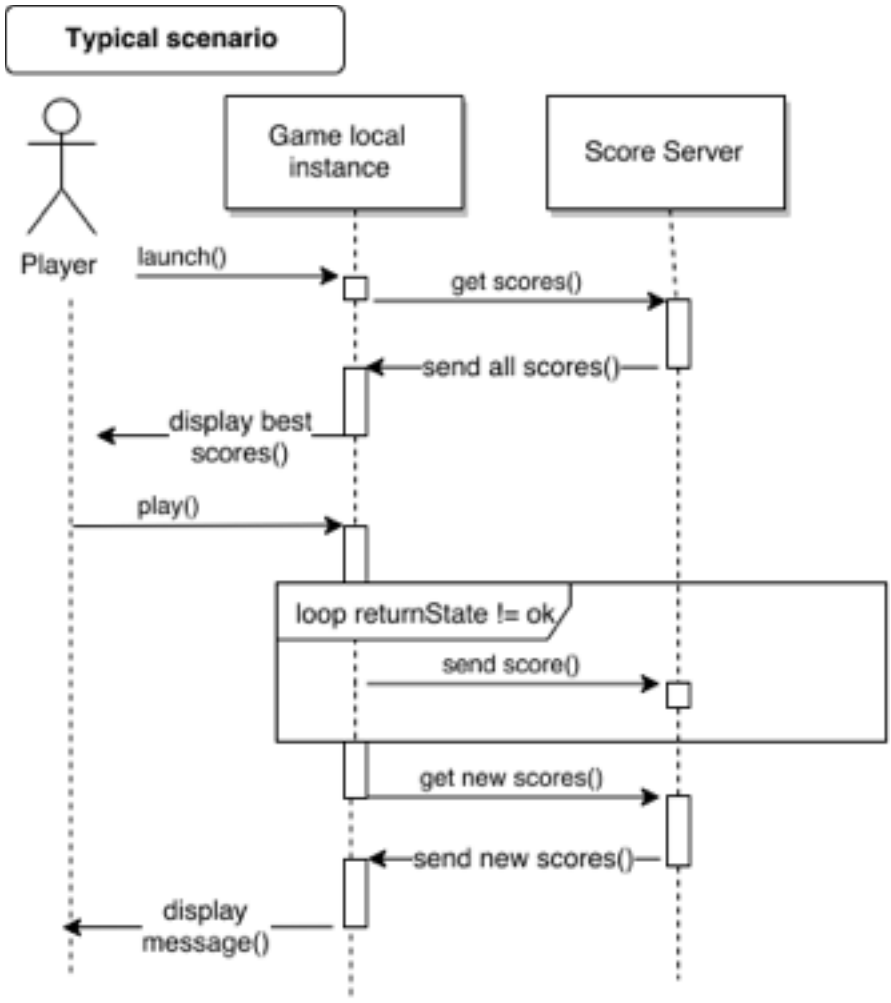
Resources:

- The last chapter of the course (pdf available on Moodle) has some slides on I/O operations and in particular on which Java packages to use to connect to a web page. It also hosts information and examples about the Scanner class to parse strings.
- Several files necessary for the project are available on Moodle.

Constraints :

- The project has to be hosted + often updated on GitHub and documented on GitHub pages
- Warning: there will be a intermediary step to upload on Moodle.

Theme: we aim here at maintaining high scores of a game for a community of players. For this reason, scores have to be stored online on a Score server and fetched by each individual instances of the game somewhere on the planet. The typical scenario that happens when a player launches its copy of the game is displayed in the UML diagram sequence below:



Step 0:

ThingSpeak is an online service dedicated to the storing of values sent by connected objects. We will slightly divert this service to make it act as a score server (don't tell anyone ;) . First thing for your team is to create an account on this website, then create a channel called `HighScores` and have it declare two fields: `score` and `player` (in this precise order). Make your channel public (ticking the correct option for that).@
After its creation, you're on a page where you can control the channel. Go into the *Data Import/Export* panel, then at the top left in the Import part, upload a CSV file named `scoreFile.csv` that you'll find on Moodle.
Then go to the *Public View* panel to ensure you can visualize the latest scores entered (yes, those of the CSV file!). First students to arrive there: please take a snapshot of your screen showing entered values, then post on the Moodle forum dedicated to the project (thanks!). Warning: it could be that the site takes some time generating the correct plot for the data, but if you doubt it, ask me / other students by indicating your channel ID (not its api key).

In the following, you will make different versions of some classes (`HighScore` and `TestHighScore`) that will belong to a package called `scores`. At each step `X` (for X=1,2,3, …), you will propose a class called `HighScoreX` (starting from a copy of the previous version).

Step 1: Simulate a fake game: write a java `TestHighScore` class in the `scores` package, having a main method that:
1. asks the user for a player name (to be read from the standard input, not to be given as argument when launching the program). Hint: use the `Scanner` java class.
2. reads all fake scores from the `scoreSamples.txt` file (available on Moodle)
3. chooses one of these scores at random (not always the same from one run to the other),
4. then outputs the player name and the chosen score on the standard output (that is, in the terminal by default).

(Yes, reading possible scores from a file is imposed).

Implement a method `getScores()` in the `HighScore` class. This method opens an HTTP connection with the ThingSpeak website site and asks for scores. Note that, even if we make the assumption that a new score is published online only if it's better than one of the 10 current best scores, this does not impede very old scores to be on the top 10 list. Hence, your program must read all previous published scores. Look on the ThingSpeak page associated to a channel, in the *Data Import/Export* panel, to know which address should be queried for obtaining published scores. This should look something like:

https://api.thingspeak.com/channels/108025/feeds.csv

if your channel id is 108025. Note that the results are asked here in csv format, which is simpler to parse than XML or JSON formats.
The `getScores()` returns a `String [ ]` containing the different interesting lines of result (we don't care about the « created_at,entry_id,field1,field2 » line, nor about the empty lines).

Then, at the start of the main method in the `TestHighScore` class, call the `getScores()` method after declaring a `HighScore` object and print all lines of scores previously obtained, before starting the game (the game is for the moment a fake one, where just one score is chosen at random in the list loaded from a file).

Step 2:
Create a `BestPlayer` class in the `scores` package having a `String player` and an `int score` as attributes. Also add a compareTo(BestPlayer p) method in this class that returns

- 0 if the score of the implicit player (`this`) is equal to that of the `p` player
- -1 if the score of the implicit player is less than that `p`
- 1 otherwise

Then, add a `tenBestScores(String [] readScores)` method to the `HighScore` class that takes as input the scores read by `getScores()` then parse each line to create a `BestPlayer` object for each line, storing all these objects in a `BestPlayer[] allBest` array. Hint: to parse each line use a `Scanner` object (cf slides of the course +internet). The `tenBestScores` method then looks for the 10 best scores inside `bestArr`, and fills a `BestPlayer [10] top10` array, from the best player to the 10th best player (not need for a sophisticated sort algorithm: just pick the best score, then 2nd best score, …). Use the `CompareTo()` method to compare two players. The `top10` array is returned by the method.

Modify the `TestHighScore` class so that it now only prints the 10 best published scores, rather than the whole list of scores from *ThingSpeak*.

Step 3:
Yes, I'm sure you guess the next step: if the score done by the player of your game is greater or equal to one of the top 10 already published, then your program has to send that score online, together with the name of the player: add a `sendScore(BestPlayer p)` method to the `HighScore` class, that interact with *ThingSpeak* through a GET http connection (see *Update Channel Feed - GET* in the *Data Import/Export* panel).
Change the `TestHighScore` class to report the score+name of the player when it qualifies for being in the top 10. Note: you can raise the random value by 100 or 1000 for instance to do tests where the player for sure enters the top 10. Also in the *Channel Settings* panel of *ThinkSpeak* you can *clear* your channel (delete all published values).

Step 4:
Yes, it could be that the best scores sorted online changed during the game, and we would need some synchronization mechanism, but we'll just forget about that now to make the project simpler.

Add a loop in the main method of the `TestHighScore` class. This loop repeatedly:

1. reads online highscores then print the top 10 highscores.
2. ask the player if he wants to start a new game
3. if yes:
    1. draw a score at random from those read in the file
    2. compares it to the top 10 highscores
    3. Send the player's score and name online if its score is among the top 10

Test your program with two instances of the game running at the same time, the online highscore list should be coherent with results of the players on the two game instances.