IMDb TV shows – Add to Watchlist by Rating Test

1. Initialization:
   The web driver is initialized (in this case, chrome).
   It is used to open a new Chrome window and access the IMDb main page.

2. Handeling the .properties File:
   As I was not 100% sure what to expect in certain cases, I used some assumptions:
   - The file is called 'config.properties' and it is located under the same directory as the code. I have placed a template file here as an example.
     Another option would be, for example, to get the path as a string from the user and use it accordingly.
   - The file only has one rating value to compare to.

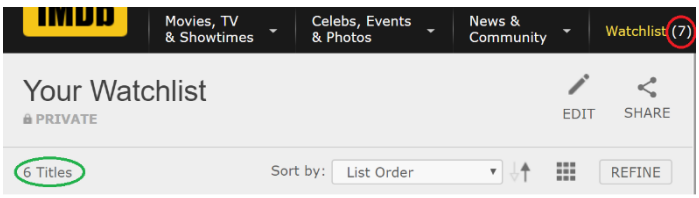
   The .properties file was parsed using Java Properties Class. The tv shows were parsed into a string array and the rating value into a float.
   - If the rating value found is not a valid float or not in range 1-10, an indicative message is printed to the console and the program is terminated (but the test does not fail). Another solution would be to fail the test.

3. Registration:
   In order to use the IMDb watchlist feature one must be signed in.
   The test first checks whether the user is already signed in.
   If not, it signs in with my IMDb account.

4. Adding Shows to Watchlist:
   For every show in the array created in the previews stage, the string is searched in the IMDb search bar.
   We then go over the results.
   There might be results with similar text that are movies/actors/etc… - My solution was to extract the text in brackets and look for "TV Series" or "TV Mini-Series".
   - If a certain TV series could not be found, an indicative message is printed to the console. (This can happen if the .properties file contains strings that are not TV Series in the IMDb DB).

   We click on the first result that meets the conditions and then add it to the watchlist if it is not already in it.

5. Verifying Watchlist:
   After accessing the watchlist, we go over the TV Series array and for each TV Series string we search for it in the watchlist.
   - If a TV Series is not found, an indicative message is printed to the console and the test fails.
   - We sort the list be 'Date Added' for efficiency.
   - Every 60 entries there is a need to click 'Load More' – this is handled with an array of Booleans.
   - During this part of the assignment I noticed that the part marked in red is not always up-to-date and I couldn't rely on it.
     I had to use the green part that was always up-to-date.



6. Quitting the Browser.

JUnit Usage:
I created a JUnit Test Suite.
The part that I chose to be executed before all the tests (@BeforeAll) is the initialization (1) mentioned above. Parts 2,3,4,5 mentioned above are tests in the JUnit Test Suite that are executed in the following order: Handeling the .properties File -> Registration -> Adding Shows to Watchlist -> Verifying Watchlist. This is ensured with the @Order(i) annotation.
The part that I chose to be executed after all the tests (@AfterAll) is Quitting the Browser.

The reason I chose to do this in a test suite is that some of the tests can maybe be used for other purposes (for example the registration test).
In addition, the initialization part should always be done first to set up the web driver and quitting the driver should always be done after all the tests are completed.
The reason I force order between the tests is that in this assignment, one action is a precondition of the other – unless the user registers, we cannot use the watchlist feature, and unless we handle the .properties file and get the tv series array we have no point in doing the 'add to watchlist' test (what shows will we be adding to our watchlist?), etc...
An alternative would be to include everything in one JUnit test.