Problem 6: Token Server – A CLASS-BASED PROBLEM

Overview

Two-factor authentication is a common security enhancing technique. In one simple implementation, a user will log in to the system and be granted provisional access. Then the user will be sent a token (to a phone or pager); this token is then used to complete the login process. In this manner, the user proves the existence of “something known” (the initial password) and “something possessed” (the phone or pager). Typically, the token is valid only for a particular (and short) amount of time.

Problem

In this problem, you will simulate the back end of a login process. When a user logs in, you will issue that user a token from the server. When a user returns (via the verify command), you will examine the token that the user presents. There are five possible outcomes that may result: 1) the user may be presenting the token that was most recently issued to this user AND the token has not expired, i.e. a successful login, 2) the user may be presenting the token that was most recently issued to this user, but that token has expired, i.e. an out-of-date token, 3) the user may be presenting a token that was issued to this user, but it is not the most recent token issued to this user, i.e. an obsolete token, 4) one or more tokens may have been issued to this user, but the token presented is not one of them, i.e. bogus token, or 5) the user may never have been issued any tokens at all, i.e. unknown user. You will issue an appropriate status message in each case. You will also be responsible for updating the server about this and any other changes, such as the passing of time.

-over-
Input

The first line of input contains two positive integers indicating the serial number for the TokenGenerator and the length of time for which a token is valid. (The former will be less than one million; the latter will be less than 32.) Each of the remaining lines will consist of a single command to the server. The four valid commands are described below (the # and what follows are the explanation of the command and not part of the actual input:

```plaintext
login <username>
    # logs the user in (and sends token to the pager); advances server clock by one
verify <username> <token>
    # attempts to complete the login; advances the server clock by one
clock
    # advances the server clock by one (essentially an idle command)
shutdown
    # ends the program
```

User names and tokens will consist of at most ten letters and digits. Both are case sensitive. [Note: if the time limit is “1”, then every login command must be followed by a verify command for the same user or access will not be granted!]

Output

One line of output will be created for each verify command. There are four possible outcomes for a verify command: 1) the login may be successful, 2) no token has ever been issued for the specified user, 3) the token specified does not match the most recent token issued to that user, or 4) the token specified for that user does match the most recent token issued to that user, but has expired, i.e. the clock has advanced more than the amount of time allowed for a given token (as was specified in the first line of the input). Examples of each message are shown in the sample output; your output must match the format shown there although the user names will be different. [Note: Repeated logins for the same user are permitted as long as the token used is still valid.]

About the class

You must use the TokenGenerator class in your solution to this problem. Otherwise, you will not be able to know the values of the token. All of the necessary methods are provided for you (although you may not need to use everything that is provided). Be sure to pay close attention to documentation of the class, including the pre- and post-conditions for all of the methods.

-continued on next page-
Example (Tokens served as a result of login commands are shown in italics at the right and are not part of the command)

Input

2018 7
login turing I6bOOe
login BernersLee oeCTsv
login turing 3HRizz
login ada94 1s10i8
verify ada94 1s10i8
verify boole 938asd
clock
verify turing I6bOOe
clock
login 24babbage zbJQsC
verify turing 3HRizz
verify BernersLee 3KJABC
shutdown

Output

Successful login by ada94
Unknown user: boole
Obsolete token submitted by turing
Expired token submitted by turing
Bogus token submitted by BernersLee

Final Notes

Since the login command does not ask for a password, we are not really performing a “what you know authentication”; and since we give the token immediately to the user (rather than sending it to a pre-determined device), we are not really performing a “what you have authentication.” Thus, we are really performing zero-factor authentication in this simulation, not two-factor authentication. Oh well, it’s only a programming contest problem; just don’t do this “in real life.”

Secondly, it is generally a poor idea to give detailed error messages upon failure as this can lead to cryptographic flaws in protocols (look up “padded oracle attack” to learn more). Once again, this is only a programming contest problem; just don’t do this “in real life.”