Project #1 – User Login
Specifications

Due Thursday, Sept 7
Upload the folder containing the program code and documentation specified in this assignment to your github account with the name, e.g. **`HW1`**
I will NOT fix compile errors or file name conflicts.

Description: Write a program that evaluates User Logins for valid requirements.

Specifications
- To be valid, a User Login must contain at least one uppercase letter, one lowercase letter, one digit and one of the following special characters: **!@#$**. (no, not the period)

In addition, its length must be at least five characters. Space, tab, and return are not allowed.
- Input. The program reads a string from the user. During development, use your own test data, then test your final version against your test data and my test data.
- Output. The program initially outputs results to the screen, in the following format: Login, validity, and errors if not valid. Sample output:

```
Login: Happy#@!888  (valid)
Login: go_t         (invalid)
   -- no uppercase letter
   -- no digit
   -- invalid special character
   -- too short (minimum of 5 characters)
```

Once your program is complete and correct, change output to go to a report file, in the same format, plus a report heading with your name, and the assignment name.

- The program will have one class, named `UserLogin`, and the following methods:
  `main`
  default constructor, others if needed,
  `greetUser` (brief explanation of the program, written to the screen),
  `readUser LoginFromUser`,
  `checkCase`,
  `checkLength`,
  `checkValidty`,
  `printUser LoginValidity` (one UserLogin as above, with UserLogin, validity result, and error messages),
  `addToReport` (concatenates result of checking the current UserLogin to the report variable), and
  `printReport` (to file).

  You may use additional methods as needed.

- *Variables*. With few exceptions, all variables should be at the instance level; do not use static. **Main** will contain an object variable of UserLogin type to call the other methods.

- *Logic*. You may use methods from the String and Character classes as needed. Do not use any built-in methods that do a substantial part of this project.

- *Testing*. You will have your own test data and mine. Testing will be interleaved in the development process. As given below, compile frequently and test the program to that point, using the test data from class. Projects proceed much faster and more smoothly when correctness is checked completely before moving on to the next Step. Be sure to plan enough time to complete final testing, and possibly make corrections.

- *Java knowledg*e. If needed, write small development test programs to investigate or practice any unfamiliar or difficult Java components. For example, you may want to write a small program that simply reads a string from the

user and prints it to an output file, to ensure you can write input and output code correctly. Do not try to incorporate such material into the full program until you have checked your understanding, because debugging is much simpler on the tiny practice program. Get it right before placing it in the full project.

- Compiler dependency. When creating and compiling code, do not simply type in code and then check to see if it's syntactically correct. First consider carefully whether you have followed all Java syntax rules. Over-dependence on the compiler will create time problems on later, larger programs, when you must check on every small piece of code before making progress. As an example, you should be able to write a valid *for* loop without consulting references or getting compiler feedback.

- *Development process*. Follow the development steps listed here. The intent is that the project will develop smoothly by working on a piece at a time, and not moving on until that piece is complete and correct.

Think
Understand the assignment
Design
Do not start coding immediately

and do not work on several Steps simultaneously. When any code is superseded, comment it out ***(do not delete it)***. Update documentation as needed for consistency and completeness. As you complete each step, upload to your github account.

- Step 1: Review these specifications. Bring questions to class or email me.

- Step 2: Create and compile a skeleton program with documentation and stubs for all methods except `main`. `Main` is the top-level method and calls each method, which prints its message and return to `main`. A stub consists of a method header, a body with only an output line showing the method's name, and a fake return value if needed to compile. This skeleton should compile and run, showing each call, and essentially does nothing; it ensures that the program structure and interactions are correct before encoding specific functionality. Do not include the loop for multiple User Logins.

- Step 3: Complete the method to read a UserLogin from the user and the method to print a UserLogin to the screen (note that it can't evaluate the UserLogin yet, so it will print only the UserLogin itself). Compile and run the program with your own test cases. Correct any problems and test again.

- Step 4: One at a time, write the check methods for letters and digits, and test them with your test cases. Each method should print out its individual result (e.g., "length OK"). Correct any problems and run the UserLogins again (regression testing).

- Step 5: Add a loop in `main` that will process as many UserLogins as the user wishes. Terminate the loop by asking the user each time if they have more UserLogins. Run with a few UserLogins to check the loop.

- Step 6: Change the check methods from printing results to setting a variable that remembers whether that characteristic was valid (`lengthOK, uppercaseOK, lowercaseOK, digitOK`). Implement the `checkValidity` method to examine these variables and produce an overall validity result for one UserLogin. Run your test data; verify all results are correct; fix any problems and retest with your data.

- Step 7: Change output to go to a file, including writing the `printReport` method. As the program progresses, it should be accumulating the report in a StringBuffer variable, then write to the file **just once**, at the end. Run all test data and output to the file. If any test cases fail, correct program logic errors and retest all data.

Documentation

- The class as a whole must have a header comment of several lines: program name, program author (you), date (month and year), brief description of the program (one or two sentences).

- Style. Use conventional Java style guidelines: methods start with a verb (doSomething), variables are self-explanatory nouns, code blocks are indented (loop bodies, etc.), blank lines separate methods, and so forth. Also, review your printout for readability before handing it in.

- Development log. Using the included form, keep track of your time while working on this assignment. Do not try to fill it out when the project is completed, because you will not remember details of all the Steps. Include notes on any small, practice programs you wrote along the way, and on any review time.

- Testing feedback. Include clear written corrections if needed on your tests.

- Comment on your level of preparedness for CS 2 based on doing this program. Was the program itself hard? What was unfamiliar to you? How much help did you need from me? How much did you have to look things up? Explain each of your comments.

- Discuss your development process. Is this different from how you've developed programs before? How well did incremental development work for you? What problems did you have with the development process? Any other comments? Refer to your development log notes. This discussion will inform me about your process, as if we had been in the lab together while you were working.

- Discuss testing. Had you done explicit testing before? Whose test plans did you have? How useful was it to have someone else's test cases as well as your own? Did the test cases find errors in your logic? If so, what were they? Did you find any errors in your test cases or the swapped cases?

- Collaboration. This assignment is to be done **on your own, with no help from classmates or anyone else**, with one exception (me). You may, of course, come to office hours with any questions for me, including on the specifications themselves. You may reference your CS 1 textbook, notes, etc. Attempt to do the project without consulting any other references (this includes online searches). The intent is to evaluate your individual, incoming knowledge, and your skill in creating and testing a Java program.

- Plan to finish several days early, to allow for any problems that arise, and to have time for final . There will be additional assigned work (the next project) before this project's due date. Do not try to upload your project at the last minute, to avoid any github difficulties that might interfere with getting the on-time time stamp.

- Late projects. On the original due date, do not submit programs that are incomplete, incorrect, or not working; do not submit incomplete packets. To get on-time credit, projects must satisfy all assignment specifications given here; ask early if any are unclear. Projects will be accepted a week late (beginning of class), with a penalty; late projects must have some Steps completed and working correctly (including related discussions) to receive any credit, and any missing or incorrect sections must be detailed in the cover letter. No projects will be accepted for credit after the late date. If you will be submitting a late project, email me a note to that effect by the beginning of class on the original due date, including how much is completed and tested so far.

- Grading will include program correctness, of course (does it process all UserLogins correctly?), and all the other project components specified here. The Java code is only part of the project; the development is a significant portion of the final grade. Incomplete projects handed in on the on-time due date will be treated as late, and graded accordingly.

- Hand in the following items, completed as specified. The entire packet must be submitted to your github. Submission of your assignment is your statement that you have done the work yourself with few if any references other than CS 1 material; this includes avoiding using Google as much as possible.

- cover letter with all discussions
- source code
- file output from all tests
- development log
- Test plan due date: Thursday, Aug 31. Post your test plan on github.
- Project due date: Thursday, Sept 7, beginning of class. Github copies uploaded (time-stamped) before class.