CMSC204

Assignment 1

Passwords

Concepts tested by this program:

ArrayList

static

Read Files

Javadoc

JUnit Tests

Exceptions

Create an application that will check for valid passwords. The following rules must be followed to create a valid password.

1. At least 6 characters long
2. 10 or more characters is a strong password, between 6 and 9 characters is a weak (but acceptable) password.
3. At least 1 numeric character
4. At least 1 uppercase alphabetic character
5. At least 1 lowercase alphabetic character
6. At least 1 special character
7. No more than 2 of the same character in a sequence

Hello@123 – OK

AAAbb@123 – not OK

Aaabb@123 – OK

**Special Characters:**

Special characters are a selection of punctuation characters that are present on standard US keyboard and frequently used in passwords.

| **Character** | **Name** |
| --- | --- |
|  | Space |
| ! | Exclamation |
| ” | Double quote |
| # | Number sign (hash) |
| $ | Dollar sign |
| % | Percent |
| & | Ampersand |
| ’ | Single quote |
| ( | Left parenthesis |
| ) | Right parenthesis |
| \* | Asterisk |
| + | Plus |
| , | Comma |
| - | Minus |
| . | Full stop |
| / | Slash |
| : | Colon |
| ; | Semicolon |
| < | Less than |
| = | Equal sign |
| > | Greater than |
| ? | Question mark |
| @ | At sign |
| [ | Left bracket |
| \ | Backslash |
| ] | Right bracket |
| ^ | Caret |
| \_ | Underscore |
| ` | Grave accent (backtick) |
| { | Left brace |
| | | Vertical bar |
| } | Right brace |
| ~ | Tilde |

**Operation:**

When the application begins, the user will be presented with a screen that states the above instructions for creating a password, two text entry boxes for typing in a password, and three buttons.

If the user wants to check a single password, they will type in the password in both boxes and select the “Check Password” button.

If the user wants to read in and check a list of passwords, they will select the “Check Passwords in File” button, be presented with a file explorer, and select the file to read from. Those passwords that failed the check will be displayed, with their error message.

**Specifications**:

**The Data Element**

String

**The Data Structure**

ArrayList of Strings

**Utility Class**

1. Create a PasswordCheckerUtility class based on the Javadoc given you.
2. The PasswordCheckerUtility class will have at least three public methods:

* isValidPassword:

This method will check the validity of one password and return true if the password is valid and throw one or more exceptions if invalid.

* isWeakPassword:

This method will che throw an exception.

* getInvalidPasswords

This method will check an ArrayList of passwords and return an ArrayList with the status of any invalid passwords (weak passwords are not considered invalid). The ArrayList of invalid passwords will be of the following format:

<password><blank><message of exception thrown>

* For each password requirement, you must have a static private method that validates the password for that requirement and throws an exception if invalid.

1. Create a separate exception classes for each exception listed in PasswordCheckerUtility Javadoc.

**Hints:**

* Always check for the length of the password first, since that is the easiest and fastest check. Once the password fails one rule, you do not need to check the rest of the rules.
* To check for a special symbol, use the “regular expression” construct. Check the whole password, not just an individual character, using the following:

Pattern pattern = Pattern.compile("[a-zA-Z0-9]\*");

Matcher matcher = pattern.matcher(str);

return (!matcher.matches());

* You will need to import Pattern and Matcher.
* Few helpful links on regular expression”
* <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>
* <https://www.youtube.com/watch?v=rhzKDrUiJVk>
* <https://www.youtube.com/watch?v=sCuOJ8uadOg>

**The GUI:**

The GUI has been provided, however you need to:

* Ask the user to enter the password and to re-type the password. If the two are not the same, inform the user.
* Use methods of PasswordCheckerUtility to check validity of one password when user clicks on “Check Password” button.
* Use methods of PasswordCheckerUtility to check validity of a file of passwords when user clicks on “Check Passwords in File” button.
* Use a FileChooser for the user to select the input file.
* Use try/catch structure to catch exceptions thrown by PasswordCheckerUtility methods

**Exceptions**

Provide exception classes for the following:

1. Length of password is less than 6 characters (class LengthException)

Message – The password must be at least 6 characters long

1. Password doesn’t contain an uppercase alpha character (class NoUpperAlphaException)

Message – The password must contain at least one uppercase alphabetic character

1. Password doesn’t contain a lowercase alpha character (class NoLowerAlphaException)

Message – The password must contain at least one lowercase alphabetic character

1. Password doesn’t contain a numeric character (class NoDigitException)

Message – The password must contain at least one digit

1. Password doesn’t contain a special character (class NoSpecialCharacterException)

Message – The password must contain at least one special character

1. Password contains more than 2 of the same character in sequence (class InvalidSequenceException)

Message – The password cannot contain more than two of the same character in sequence.

1. Password contains 6 to 9 characters which are otherwise valid (class WeakPasswordException)

Message – The password is OK but weak - it contains fewer than 10 characters.

1. For GUI – check if Password and re-typed Password are identical (class UnmatchedException)

Message – The passwords do not match

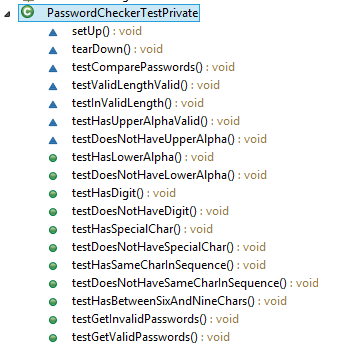
Throw this exception from the GUI, not the utility class.

**Note**: If more than one error is present in a password, use the above order to throw exceptions. For example, if a password is “xxyyzzwwaa$”, it fails rules 2 and 4 above. Throw a NoUpperAlphaException, not a NoDigitException.

**Junit methods**:

* setup
* teardown
* for each password requirement method in the PasswordCheckerUtitily you must have a test case that passes and another one that fails
* success and fail test cases for each public method in the PasswordCheckerUtitily
* The file that you use for getInvalidPasswords method Junit test must contain at least one invalid and one valid of each password requirement case.

This is a sample of Junit test cases that you must have:



A screenshot of a cell phone

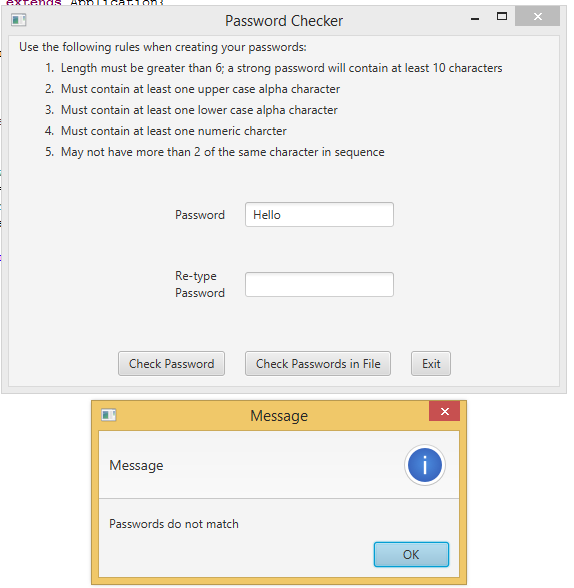
Description automatically generated

If you select the button which reads “Check Passwords in File”, you will be presented with a file chooser. You must navigate to where you stored the file “passwords.txt” and load it. The file will be in the following format (one password per line):

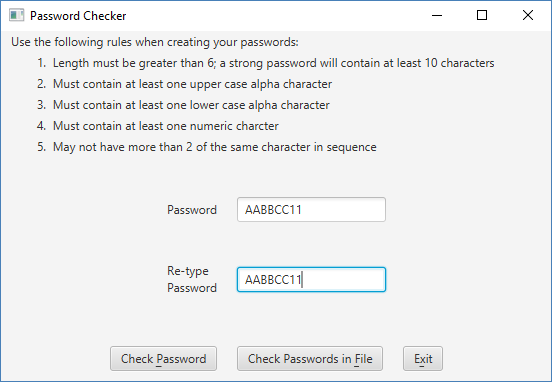
A screenshot of a cell phone

Description automatically generated

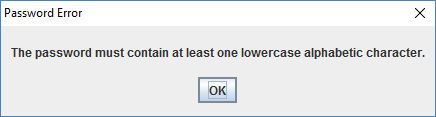
Examples:



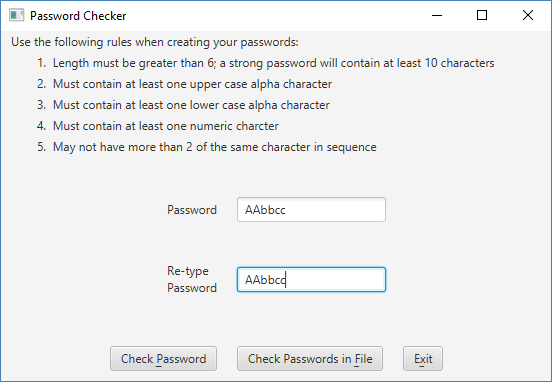
1. No lowercase alphabetic character



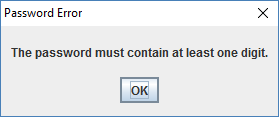
Displayed to user:



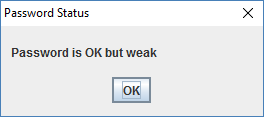
1. No digit



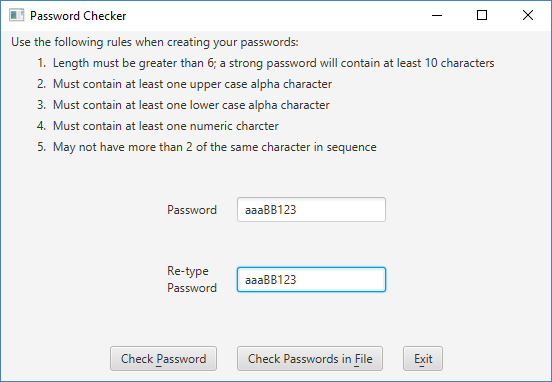
Displayed to user:



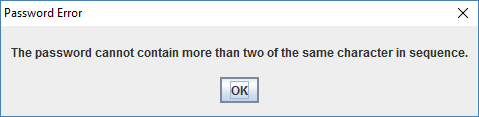
1. If the password is OK, but between 6 and 10 characters, you will see:



1. If password has more than two of the same characters in a row



Displayed to user:

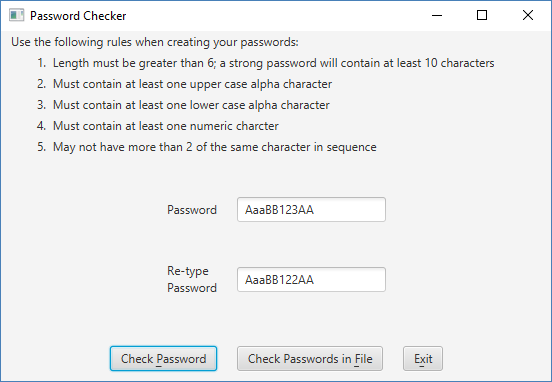


1. If password is valid

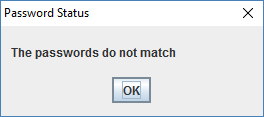
A screenshot of a cell phone

Description automatically generated

1. If the passwords do not match:

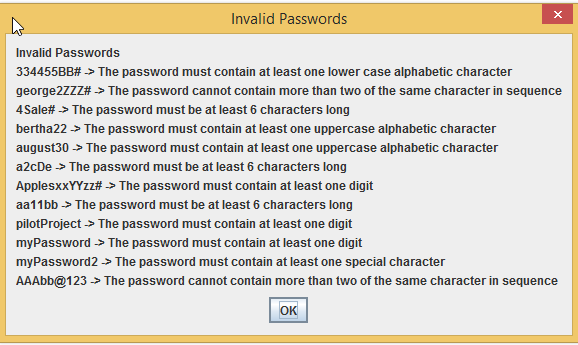


Displayed to user:



6. Based on the file above, when the user selects “Check Passwords in File”:

Displays errors to user when selecting Check Passwords in File. Note that valid passwords (including weak but otherwise valid passwords) are NOT displayed.



Deliverables:

Design:

Initial design document (UML and/or pseudo-code)

Implementation:

Final design document

Java files - The src folder with your driver (javafx application), data manager, exceptions and Junit Test (.java) files

Javadoc files - The entire doc folder with your javadoc for student generated files

Learning Experience document

GitHub screen shot

Deliverable format: The above deliverables will be packaged as follows. Two compressed files in the following formats:

LastNameFirstName\_AssignmentX.zip [compressed file containing following]:

doc [a directory] *include the entire doc folder with the javadoc for*

*student generated files*

file1.html (example)

file2.html (example)

class-use (example directory)

LearningExperience.doc *or other text format*

src [a directory] *contains your driver (javafx application), data manager, exceptions and Junit Test (.java) files*

File1.java (example)

File2.java (example)

File\_Test.java (example)

*GitHub screen shot.*

LastNameFirstName\_AssignmentX\_**Moss**.zip [compressed file containing only]:

*.java file which includes the driver (javafx application), data manager, exceptions and Junit Test (.java) files – NO*

*FOLDERS!!*

File1.java (example)

File2.java (example)

**Grading Rubric**

**CMSC 204 Project #1**

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overview:**

There are two parts to the rubric. First, the project is graded on whether it passes public and private tests. **If it does not compile, a 0 will be given**. These points add up to 100. Second, the score is decremented if various requirements are not met, e.g., missing required methods, missing Junit test cases, no Javadoc, no UML diagram, uses constructs that are not allowed, etc.

**TESTING** (100 pts)

Compiles 35 pts \_\_\_\_\_

Student Junit required student tests 8 pts \_\_\_\_\_

Passes JUnit student tests 7 pts \_\_\_\_\_

Passes public JUnit tests 15 pts \_\_\_\_\_

Passes private instructor tests 15 pts \_\_\_\_\_

Execution: runs without errors (either run-time or logic errors 20 pts \_\_\_\_\_

Possible Sub-total 100 pts \_\_\_\_\_

**REQUIREMENTS** (Subtracts from Testing total)

**Documentation:**

Javadoc for student generated files not submitted (entire doc folder) -5 pts \_\_\_\_\_

Documentation within source code was missing or incorrect -1.5 pts \_\_\_\_\_

Description of what class does was missing

Author’s Name, @author, was missing

Methods not commented properly using Javadoc @param, @return

JUnit STUDENT methods were not implemented -10 pts \_\_\_\_\_

Screen shot of GitHub repo with Assignment1 files was not submitted -5 pts \_\_\_\_\_

MOSS files were not submitted -5 pts \_\_\_\_\_

Learning Experience -2.5 pts \_\_\_\_\_

In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project. What have you learned? What did you struggle with? What would you do differently on your next project?

**Programming Style:**

Incorrect use of indentation, statements, structures -2 pts \_\_\_\_\_

**Design:**

Implementation does not follow final design -4 pts \_\_\_\_\_

Classes do not have the functionality specified, i.e.,

1. PasswordCheckerUtility class -10 pts \_\_\_\_\_

does not have a method to check validity of password

does not have a method to check validity of ArrayList of passwords

does not follow the Javadoc provided

does not have private method for each password requirement

1. GUI does not compile -5 pts \_\_\_\_\_
2. Exceptions classes -10 pts \_\_\_\_\_

does not have exception class for each invalid password rule

does not have exception class for weak password

does not have exception class if password and re-type password don’t match

does not have exception class for each password requirement

Possible decrements: -60 pts \_\_\_\_\_

Possible total grade: 100 pts \_\_\_\_\_