Homework 2

Start Assignment

Due No Due Date **Points** 100 **Submitting** a file upload **Available** after Sep 29 at 6pm

In HW 2, we are going to implement a simple *password vault* -- think of LastPass only much simpler.

A *user* is the client for the vault. He/she has a username and a vault password, and uses the vault to store (site / password) pairs for a variety of sites. In a real vault a username might be a long string like an email address and a site might be a long URL, but for our example we will use short usernames like "josie" and passwords like "whocares" and short site names like "amazon", "google", or "netflix."

The `Vault` supplies the following operations:

1. `addNewUser` : Add a new user to the vault (with no site passwords). A username and password is supplied, and the system does no verification or checking except that the username and password must be in correct formats, and the username cannot already be in the vault.
2. `addNewSite` : Add a new password for a site. For example, if a user is creating an account at the site "amazon" for the first time, she calls the vault with the site name, and the site makes up a password for the user/site, returns the (plain text password to the user, and stores the (encrypted) password -- all stored passwords in the vault must be encrypted.
3. `retrieveSitePassword` : Retrieve the password for a site. The user supplies the name of a site, and if she has a stored password for the site, it is returned in plain text.
4. `updateSitePassword` : Request a new password for a site, for example the user wants to change her password on "amazon," and the system generates a new password, stores an encrypted version for the user, and returns the plaintext version.

For security reasons, if a *user* tries three operations with an incorrect password, that user is locked out: prohibited from doing any operation until the user is unlocked.

In addition, the following conditions are mandatory:

- All user names are strings of at least 6 and at most 12 lower-case letters
- All site names are strings of at least 6 and at most 12 lower-case letters
- All passwords -- both vault passwords and site passwords -- must satisfy these conditions
  - Be between 6 and 15 characters long
  - Contain at least one letter (can be upper or lowercase) and one digit (0-9)
  - Contain at least one special character from the set !@#$%^&
- The Vault must store all passwords encrypted
- The Vault does not need to deal with
  - Deleting users or changing vault passwords
  - Deleting site passwords for a user
  - Unlocking a user

Note: You may want to create a regex to validate username(s), site(s), and password(s); however, it's not required to use regex as long as the conditions are met

```
// regex matches when the word has 1 or more lowercase letters (a-z)
String pattern = "[a-z]+";
boolean = word.matches(pattern);

// reference: Regex tutorial (https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285) , Online regex tester (https://regex101.com/r/cO8lqs/10)
// for words, you can use "^The end$" (see anchors)
// for a certain number of characters, you can use {} (see quantifiers)
```

Class Requirements

1. Download the **week 2 starter code** **(https://github.com/stephen-riley/cpsc-5011-fall-2021-projects/tree/main/week_02)** from GitHub - we will work on this during the in-class lab
2. **No changes should be made to the `Encryptor` or `Vault` interfaces.**
3. The `Encryptor` interface is implemented in a class called `CaesarCipher`; the `Vault` interface needs to be implemented in a class called `PasswordVault`.

- Additional documentation on functionality may be found on the provided `Vault` interface.

4. You will only be throwing exceptions in `CaesarCipher`, `PasswordVault` or other classes you create. The driver will be responsible for handling (i.e. catching) exceptions.
5. You may need to throw additional exceptions inside your function implementations. If so, be sure to include the `throws` keyword and the exception name in the function signature.
6. Follow the **(Java) Documentation & Style Standards**.

Some questions to consider:

- How does PasswordVault and CaesarCipher know about each other?
- How are you going to store username/password pairs?
- How are you going to store username with sitename/password?
- How are you going to validate username/sitename and passwords?
- How are you going to generate site passwords?
- How are exceptions handled? Will program continue? If so, how?
- How are you going to test?
- How do you keep track of failed logins?
- How do you keep track of blocked users?

Sample results from calling `PasswordVault` functionality:

```
Added user 'sriley'
Added site 'amazon' for user 'sriley' => generated site password: kc^%78&
Retrieve site password for 'amazon' for user 'sriley' => password: kc^%78&
Update site password for 'amazon' for user 'sriley' => updated password: 363#XqQH2
```

Driver Requirements

Demonstrate all implemented `PasswordVault` functionality in the driver. If you decide to implement the menu-driven console app for extra credit, include sample output in the README that demonstrates all implemented `PasswordVault` functionality.

Sample output:

```
** 1. Testing success case: **

Attempting to add user 'sheila' and password 'mypass123!'
Added user 'sriley'

Attempting to add site 'amazon' for user 'sriley'
Added site 'amazon' for user 'sriley' => generated site password: @^#oH^0&$6^

Attempting to retrieve 'amazon' site password for user 'sheila'
Retrieved site 'amazon' for user 'sriley' => retrieved password: @^#oH^0&$6^

Attempting to update 'amazon' site password for user 'sheila'
Updated site 'amazon' for user 'sriley' => updated password: 03Y20@$!@&


** 2. Testing exceptions for addNewUser: **

Attempting to add user 'bob' and password 'mypass123!'
Error: The username is invalid; enter 6-12 lower-case letters.

Attempting to add user 'bobismyname' and password 'mypass123'
Error: The password is invalid; enter 6-15 characters and at least
one letter, one number, and one special character (!@#$%^&).

Attempting to add user 'sheila' and password 'mypass123!'
Error: The username already exists.


**3. Testing exceptions for addNewSite:**

Attempting to add site 'amazon' for user 'sheila'
Error: The site name already exists for this user.

Attempting to add site 'amazon' for user 'maryismyname'
Error: Username does not exist.

Attempting to add site 'amazon.com' for user 'sheila'
Error: The site name is invalid; enter 6-12 lower-case letters.


** 4. Testing exceptions for retrieveSitePassword: **

Attempting to retrieve 'amazon.com' site password for user 'sheila'
Error: Site name does not exist for this user.


** 5. Testing exceptions for updateSitePassword: **
```

```
Attempting to update 'amazon.com' site password for user 'sheila'
Error: Site name does not exist for this user.

Attempting to update 'amazon' site password for user 'sheila'
Error: Attempted to login with incorrect credentials.

Attempting to update 'amazon' site password for user 'sheila'
Error: Attempted to login with incorrect credentials.

Attempting to update 'amazon' site password for user 'sheila'
Error: Attempted to login with incorrect credentials.

Attempting to update 'amazon' site password for user 'sheila'
Error: Error: Attempted to login with incorrect credentials 3 times
user is locked out of the system.
```

Exceptions

Once you've created all the custom Exception classes in the `Exceptions` package, you can write custom message to clearly notify why the exception was thrown.

```
package exceptions;

public class MyCustomException extends Exception {

    private static final long serialVersionUID = 1L;

    public MyCustomException() {
        super("add custom message here");
    }
    // when throwing exception, call default constructor and custom message
    // does not need to be provided:
    // throw new MyCustomException();

    public MyCustomException(String message) {
        super(message);
    }
    // when throwing exception, call constructor and pass in custom message:
    // throw new MyCustomException("add custom message here");
}
```

Note: You can include one of the constructors or both, depending on your implementation. More information about serialVersionUID may be found here: **https://dzone.com/articles/what-is-serialversionuid (https://dzone.com/articles/what-is-serialversionuid)**

Checked exceptions must honor the **Catch or Specify requirement** and must include a throws clause in the function signature or be handled with a try-catch; otherwise, you'll get a compile error. For example, `IOException` and `FileNotFoundException` are checked exceptions, while `IndexOutOfBoundsException` and `EmptyStackException` are unchecked exceptions.

```
    private void doSomething() throws IOException {
        throw new IOException();
    }

    private void doSomethingAgain() {
        throw new EmptyStackException();
    }
```

A reminder that the driver will be responsible for handling (i.e. catching) all checked exceptions.

Unit Testing

Create a unit test for the `CaesarCipher` class called `CaesarCipherTest`. This unit test can reside in the `encrypt` package. Sufficiently test the constructor, `encrypt`, and `decrypt` functions in the `CaesarCipher` class.

Javadoc

Generate Javadoc documentation for `Encryptor`, `Vault`, `CaesarCipher`, `PasswordVault`, and any additional classes you implement for the HW 2 assignment.

Javadoc documentation is not necessary for the unit tests or the driver. The Javadoc documentation will live in a folder called `docs` under the project, not under `src`.

See **(Java) Documentation & Style Standards**.

Notes:

- Demonstrate all implemented extra credit functionality in the driver.
- You may receive at most 10 extra credit points.

- Add extra credit functionality to your existing HW 2 solution; no separate package(s) necessary.
- Document all extra credit functionality implemented in the README.

Submission

- Use the IntelliJ export feature to create the `[su-username]_hw2.zip` file.
- All Java classes, unit tests, and Javadoc documentation should reside inside the `[su_username]_hw2` package. For example, my username is "sriley", so my package name is `sriley_hw2` (no spaces, no quotation marks). Use your own SU username!
- Click on the "Submit Assignment" button for and submit a single ZIP file. You may submit multiple times. Only your latest submission will be graded.
- Be sure to clean your project before submission.
- Please include a README.md file with informal comments
  - Is your solution fully working or not?
  - Do you believe your unit tests to be complete to test the module's contract? If not, why not?
  - What extra credit problem(s) did you work on (#1, 2, both)? Briefly describe how do you demonstrate the functionality?
  - How much time did you spend on the assignment?
  - Any feedback on the assignment?