

Bluegrass Community and Technical College
Programming Requirements Document
UserAccount Class – More Practice Creating and Using a User-Defined Class

NARRATIVE DESCRIPTION

The first week (Module 7) we studied user-defined classes, you created a Tile class and a driver to use the class was provided. The second week (module 8), we used the Tile class and created a driver (a game) to use the Tile class.

It is **very** important to remember: classes are used by **many** drivers (applications) in the real world. When you are assigned a homework assignment, like this week, which requires you to create a class and a driver, please remember, the class is designed to be used by many drivers. You just happen to be creating one of the drivers.

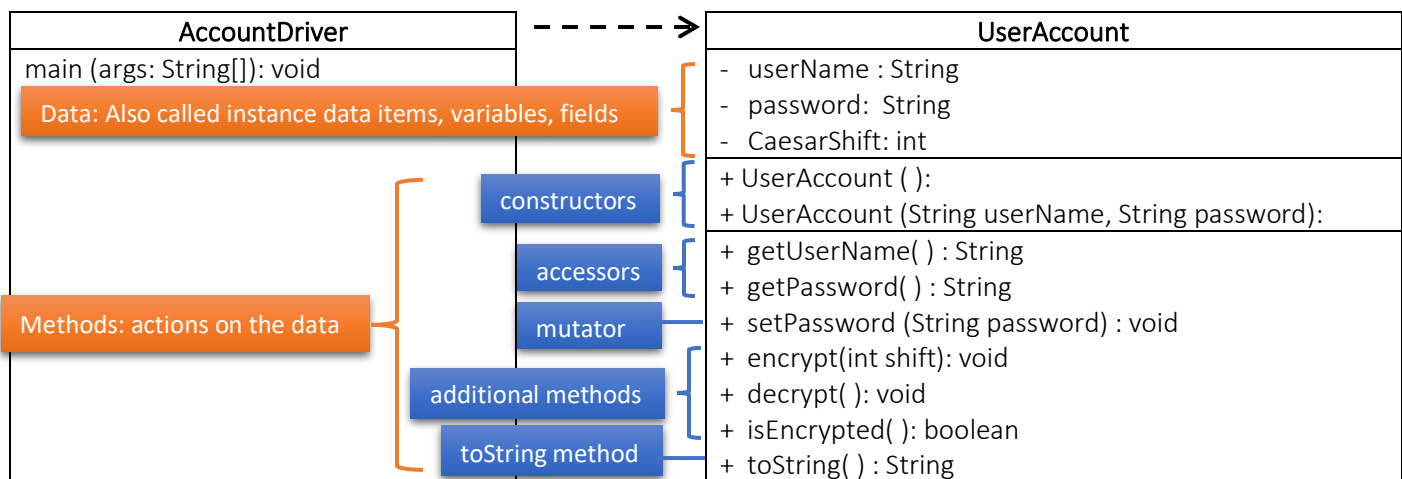
This week's assignment will create a user-defined Java class to maintain data about a user's credentials to log into an application or system. This will include a user's name and password and an additional piece of data to help with encryption. We will visit the Caesar cipher again.

The **Caesar cipher**, also known as a shift cipher, is one of the simplest forms of encryption. It is a substitution cipher where each letter in the original message (called the plaintext) is replaced with a letter corresponding to a certain number of letters up or down in the alphabet or ASCII coding. You can review the Caesar cipher, and other ciphers, at this simple website: <https://www.secplicity.org/2017/05/25/historical-cryptography-ciphers/>

Here is a sample of the ASCII table: <http://bpastudio.csudh.edu/fac/lpress/netapps/hout/sascii.htm>

We will use a Caesar cipher to encrypt the password. Shift between 1-9 are allowed. All other shifts would be invalid.

Here is the following UML diagram for your homework assignment:



(+ represents the public modifier and – represents the private modifier)

Zipped with this homework assignment is a **UserAccount** class. Save the file on your computer. You should note the saved UserAccount class has the encrypt() method already coded for you. Update UserAccount it as shown in the UML diagram above. Details for this are on the next page.

Notes about UserAccount:

- When an object is instantiated, the constructor will set the CaesarShift value to zero. A value of zero means the password is not encrypted.
- The default constructor (no parameters) will set the userName and password to "" and CaesarShift to 0. A CaesarShift value of 0 means the password is not encrypted.
- All parameters should be validated before updating any class data within UserAccount (in the constructor and mutator).
 - The userName must be a minimum of 8 characters and no longer than 16
 - The password must be a minimum of 8 characters and no longer than 20
- Create the constructors, accessors, and password mutator shown in the UML diagram. You will not create a mutator for userName because we typically do not change a userName but rather create a new account for a different name and deactivate the old name. You will not create a mutator for CaesarShift, that will be updated when the encrypt() method is called and when the decrypt() method is called.
- When the password is encrypted, valid shift amounts are 1-19. Remember, a value of zero means the password is not encrypted. The class variable CaesarShift is updated when the password is encrypted. ***The encrypt method has been provided.*** Notice it updates CaesarShift when the password is encrypted. It does not encrypt a password that is already encrypted, and it does not encrypt if the shift is not between 1-19. Notice how the charAt() method is used casting is used (to a char).
- The encrypt method has been provided. Notice it updates CaesarShift when the password is encrypted. It does not encrypt a password that is already encrypted. Notice how the charAt() method is used and the casting to a (char). You can use a similar method for the decrypt method. The encrypt method was provided to ease your coding and for you to learn how it could be done efficiently.
- Use the encrypt() method as a model to create the decrypt method. The decrypt method should not decrypt a password that is not encrypted. HINT: CaesarShift is > 0 if the password is encrypted. When a password is decrypted, set the CaesarShift to zero.
- The isEncrypted method returns true if the password is currently encrypted and returns false if it is not encrypted. HINT: CaesarShift is > 0 if the password is encrypted. When a password is decrypted, set the CaesarShift to zero.
- The toString() method should display the userName and password. The password may be encrypted or not. toString() will not worry about this for this exercise. It simply returns the userName and current version of the password.

To test the UserAccount class, create a simple driver that will test all methods:

- Test the constructor:
 - Display a message to the user that constructors are being tested.
 - Allow a user to enter a userName and password.
 - Create a UserAccount object with this data.
 - Display the object using the toString() method.
 - Create a loop allow the use to test your constructor for 3 people. Test with valid and invalid passwords to see how your class reacts and update your code if it does not act appropriately.
- Test the accessors:
 - Display a message to the user that accessors are being tested.
 - Create a UserAccount for John Smith, with a password of Zab98W55k.
 - Display the data using the accessors for name and password.
 - Debug your code if accessors are not returning correct values.
- Test the mutator:
 - Display a message to the user that mutators are being tested.
 - Ask the user for a new password for John Smith.
 - Update the password using the mutator for password.
 - Display the new password using the accessor for password.
 - Debug your code if mutators are not working correctly.

- Test the encrypt() method
 - Display a message to the user that the encryption method will be tested.
 - Display John Smith's data (name and password) using toString() with a message "Before encryption"
 - Encrypt the current password for John Smith after asking the user for a shift amount (1-19 are valid values).
 - Display the name and password using toString() with a message "After encryption - first time".
 - Try to encrypt the password again.
 - Display the name and password using toString() with a message "After encryption - second time".
 - Repeat these steps a second time so that valid and invalid values for the shift amount can be tested.
- Test the decrypt() method
 - Display a message to the user that the decryption method is being tested.
 - Create a new UserAccount for Sally McIntosh with a password of B1N8hop10.
 - Display the data for Sally using the toString() method with a message "Before encryption"
 - Input a shift value.
 - Encrypt the password.
 - Display the name and password using toString() with a message "After encryption".
 - Call the isEncrypted() password and display the result (it should be true) with a message "Encryption status".
 - Decrypt the password.
 - Display the name and password using toString() with a message "After decryption".
 - Try to decrypt the password again.
 - Display the name and password using toString() with a message "After decryption".
 - Call the isEncrypted() password and display the result (it should be false) with a message "Encryption status".
 - Display a message "Testing ended"

Additional Important Concepts


Your driver is an example of a **unit test** for the UserAccount class. In a business setting, the UserAccount class would be a part of a larger computer system/application. As each class, or portion of a larger computer system is completed, it is tested independently to insure it works properly. **This is referred to as unit testing.** Unit testing is done prior to testing the code within in the context of the larger computer system.

When you perform unit testing, you should try valid data and a variety of invalid data. Your goal is to test the UserAccount class to insure it handles ALL data correctly. Try all types of "odd" or "weird" samples of data. (Your instructor does the same when she tests your code.)

If this were a driver for a real application (not for testing), you would validate userName, password, and shift in the driver before instantiating an object and again in the UserAccount class for method parameters. In other words, the application would only pass valid data (arguments) when it calls a method.

To be precise, it is important to know the difference between an argument and a parameter. A **parameter** is a variable in a method definition. You will see these in the UserAccount class. For example, in the method below, shift is a parameter:


```
public void encrypt(int shift)
{
    // code for encrypt()
}
```



Parameter

When a method is called from a driver, the **arguments** are the data you pass into methods. For example, the following is code in a driver. `shiftAmount` in the last line is an argument which is passed to the `encrypt()` method:

```
UserAccount customer = new UserAccount("manderson5","someCoolPassword");  
int shiftAmount = 8;  
customer.encrypt(shiftAmount);
```



Argument

In a real setting, there are multiple layers of data validation: (1) the class performs data validation, (2) the driver/application performs data validation and (3) there other security mechanisms in place. Remember: it is not unusual for different people to write different portions of a computer system. Each is person is responsible for securing their own code or their portion of the larger system.

Zip your **UserAccount** class and your driver together for grading.

NEW CONCEPTS ASSESSED AND ILLUSTRATED (IN ADDITION TO ANY PREVIOUSLY LEARNED)

- Caesar cipher
- `charAt()`
- Casting to a `char` data type
- Practice creating and using user-defined Java classes.
- Securing your code by performing unit testing and data validation.

SOFTWARE REQUIREMENTS

All standard requirements (documentation, quality, indentation, spacing, etc.) are included, plus:

- R1: Create the `UserAccount` class.
- R2: Validate all parameters in `UserAccount`.
- R3: **Encryption and decryption is handled properly.**
- R4: Per the UML diagram, accessors are created correctly.
- R5: Per the UML diagram, mutators are created correctly.
- R7: The driver is written to properly test the `UserAccount` class as defined.

SECURITY CONSIDERATIONS

The public and private access modifiers have security implications. Use a private modifier for all instance data items. Validate all user input and method parameters.

SPECIAL NOTES

None.