**COEN 275 OO Analysis, Design and Programming Spring 2016**

**Assignment 1 (Due 15th April) Total (100 pts)**

In this first assignment, you will learn to use a Java IDE (of your choice, Eclipse, NetBeans etc)

* To build and execute a Java program with **classes**, **objects**, **interface** and **composition**.
* To define classes in a package
* To test your classes.

**Note**: **Define the classes required in question 1 in a package with name, *yourlastname*.assign1**

**Define the classes required in questions 2 and 3 in a package with name, *yourlastname*.docsafe**

**Description**

In this Java program called *DocSafe*, *Document* objects (with a *document id* and the *contents*) can be stored in an object of type *DocStore*. When a document is stored in a *DocStore*, the document’s contents are encrypted, using an *Encryptor* object (with a specific key) associated with DocStore. A document can be retrieved from a DocStore using a document Id and optionally a key. When the document is retrieved using the correct key, document contents are decrypted and returned. When the document is retrieved without a key, document contents in the encrypted form are returned.

**Question 1**

Design and implement an interface **Encryptor**, classes, **EncryptorDecryptor**, **Document** and **DocStore**. See the description of each of them given below:

**interface Enryptor:**

**Methods**

* + String encrypt(String plainText)
  + String decrypt(String plainText)

**Class** **EncryptorDecryptor:**  implements Encryptor interface. This class will support methods to encrypt and decrypt a string of characters using a *shift cipher* algorithm. The string before encryption will be referred to as “*plain text*” and the encrypted string as “*encrypted text*”.

**Shift cipher encryption** is a substitution algorithm and one of the most basic forms of encryption where an offset (key) is used to substitute a different character for every character allowed in the plain text.

For example, if an offset of 1 is used, then ‘A’ is encrypted as ‘B’, ‘B’ is encrypted as ‘C’, and so on. When we reached the end of the character set and we wrap around to the beginning of the set using modulo arithmetic.

**Example:**

Assuming the allowable characters for encryption are

“ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789” and with a shift of 3, the plain text and its encrypted form are shown below.

**Plain text: ABcd012 Encrypted: DEfg345**

Note: The encrypted text depends on the set of characters which are allowed for encryption.

**Data Members:**

**allowableChars** (String): Initialized to the allowable characters in the plaintext, which are **upper and lowercase letters, digits, - (hyphen), . (dot), ! (exclamation) and a comma**.

**offset** (an integer) : is the key used to shift the letters in the original string. The offset should be greater than 0 and less than the total number of allowable characters. It should be initialized in the constructor..

**EncryptorDecryptor (int offset):** is the constructor that initializes the data member offset with the parameter, offset.

**string encrypt (String plaintext):**

This method should encrypt the plaintext using a shift cipher algorithm (see details below) and return the encrypted string. It should use the offset stored as a data member. Each character in the plaintext is encoded into another character using the formula below.

The following formula can be used to shift each character in the plaintext by the given offset.

encryptedCharacterIndex = (positional\_value\_of\_character\_in\_plaintext + offset) mod total\_no\_of\_allowableCharacters.

Characters that are not allowed (not in allowableCharacter set), should be copied as such without encryption to the encoded text.

For example, let us say that the allowable characters are stored in a string variable as shown below:

String allowableChars = “-,.!0123456789ABCDEFGHIJKLMNOPQRSUVWXYZabcdefghijklmnopqrstuvwxyz";

You can get the positional value of a character (for example,A) in the above string using,

int positionalValue = allowableChars.indexOf(‘A’);

You should now use the *positionalValue* and calculate the *encryptedCharacterIndex (a number)* using the formula given. Use *encryptedCharacterIndex* as the index into the string, *allowableChars*, to get the corresponding character at that index (think of *String.charAt()* method to get a character at a particular index).

The encrypted character should now be added to the result string (encrypted text). Please note that String objects are immutable, so you may have to use a StringBuilder object to hold your resultString (encoded text) as you keep adding encoded characters to it. Finally, you will convert the StringBuilder object into a String object and return it from the method. Or, you can use a character array and finally convert the character array into a String object and return it.

**string decrypt (String encryptedText):**

This method should decrypt the encryptedText using the offset stored as a data member.

Use following formula to decrypt each letter in the encryptedText,

decryptedCharacterIndex = (positional\_value\_of\_character\_in\_plaintext + (total\_no\_of\_allowableCharacters - offset) mod total\_no\_of\_allowableCharacters.

Use **decryptedCharacterIndex** as the index into the string, **allowableChars**, to get the corresponding character at that index.

The decrypted character should now be added to the result string (decrypted text). Please note that String objects are immutable, so you may have to use a StringBuilder object to hold your resultString (decoded text) as you keep adding encoded characters to it. Finally, you will convert the StringBuilder object into a String object and return it from the method.

**Some points to note:**

• Offset should be between 0 and total\_no\_of\_allowableCharacters (this set included upper and lowercase letters, digits 0-9 and the 4 punctuation characters as given above), both numbers not inclusive.

• If the plaintext contains any characters that are not allowed (not in allowableCharacter set), they should be copied as such without encryption to the encoded text.

class **Document**:

**Data Members:**

documentId (String)

content (String)

**Methods:**

Necessary constructors

Accessor methods

class **DocStore**

**Data Members:**

store (Document []): An array of Document objects

capacity (an integer): A constant number to indicate the capacity of store.

encryptor (EncryptorDecryptor) : An object of type EncryptorDecryptor, initialized in the constructor.

**Methods:**

**DocStore**(EncryotorDecryptor enc): The data member encryptor is initialized with the parameter, enc.

**saveDoc** (Document doc): A new Document instance is created with the parameter, doc. The contents of this instance are encrypted using the data member, encryptor. The Document instance is now stored in the array, store.

**Document getDoc(int id):** searches for a Document object with the parameter id, and returns it. The contents of this Document object are still in encrypted form.

**getDoc(id,key):** searches for a Document object with the parameter id, compares tge parameter, key with the decrypts the content of the found document and returns it.

**Note:** In addition to the methods and data members specified, you are free to define additional methods or data members.

The following usage scenario is given to illustrate how the application is used.

**Question 2**

Define a class called DocSafeTester with one static method called test().

In test(), create objects of each of the classes you have defined above and call each of their methods with parameters of your choice to test the correctness of the functionality.

**Question 3**

Write a class called DocSafe with main() method.

In the main() do the following:

1. Call the test() of DocSafeTester to show the test results of your classes.
2. Run a usage scenario (use the example shown below) to show that your classes and methods are working per specification.

An example Usage Scenario:

// Create an instance of an Encryptor with an offset

Encryptor enc = new EncryptorDecryptor(5);

// Create an instance of a DocStore

DocStore store = new DocStore (enc);

// Create instances of a Document

Document doc1 = new Document(‘D10’,"This is a test");

Document doc2 = new Document(‘DA9’,"London Eye");

// Store them in DocStore instance

store.saveDoc (doc1);

store.saveDoc (doc2);

// Retrieve the document with id and display the

// contents. The contents should be in the encrypted form

System.out.println(store.getDoc(‘D10’).getContent());

// Retrieve the document with id and the correct offset and

// display the contents.

// The contents should be in the decrypted form

System.out.println(store.getDoc(‘D10’,5).getContent());

// Retrieve the document with id and an incorrect offset and

// display the contents.

// The contents should be in the encrypted form

System.out.println(store.getDoc(‘D10’,2).getContent());

// Try to retrieve the document with an id that does

// not exist in the store. Should display an error message

System.out.println(store.getDoc(‘B67’);

**You must capture the output of your test cases and usage scenario.**

**Point distribution**

Correctness: 75% Naming Conventions: 5% Testing: 10%

Output: 10%

In naming classes, packages, data members, methods and so on, you should follow the recommended Java conventions. See <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

**What to submit:**

You should create a zip file with a readme, source code and output. Name your zip file as assign1\_yourstudentid. The readme file and the source code should include your student id, optionally your name, course number, assignment number and date of submission. The readme file should identify the Java environment that you have used and any limitations of your program.

Late submission will incur a 10% penalty per day.