Department of Computing Sciences
SUNY Brockport

CSC 205 (Fundamentals of Data Structures)

Partner 1 Name: _____

Partner 2 Name: _____

Lab Exercise 1

Programming Exercise: (Use Eclipse IDE)

Note: Study the following sample program called **`TelephoneListConverter`** (actual code and test plan). In this lab you will be creating similar artifacts and writing software for another, very similar, problem (it belongs to the same family).

GIVEN PROBLEM STATEMENT:

We have a list of telephone numbers some of which contain letters as well as numbers (for example: 395-COLD). We want to convert all of them into pure numbers using a standard transformation from letters to numbers:

ABC	DEF	GHI	JKL	MNO	PQRS	TUV	WXYZ
2	3	4	5	6	7	8	9

Write a program to do this.

Test Plan:

Input	Expected Output	Computed Output	Do they match?
777-HOME	777-4663		
777-home	777-4663		
777-SAVE	777-7283		
777-save	777-7283		
1-800-TALK-LAW	1-800-8255-529		
1-800-talk-law	1-800-8255-529		
123-4567	123-4567		
123#4567	123$4567		
12$$%%&&	12$$$$$$		
ABC DEF GHHH	222$333$4444		
IJK-LMN-OPQR	455-566-6777		

STU-VWX-YZZZ	788-899-9999		
abc def ghhh	222$333$4444		
ijk-lmn-opqr	455-566-6777		
stu-vwx-yzzz	788-899-9999		

Coding style:

- Follow standard java conventions for naming classes, methods, variables, constants, etc.
  - Class names begin with an uppercase letter and use the 'camel' notation (i.e. if there are two or more words, first letter of each word is capitalized).
  - Method names, variable names begin with a lowercase letter and then use camel notation for second, third … words.
  - Constants: the whole word is in all capital letters
- Use javadoc style comments for class headers and method headers

```
/**
* comments
*/
```

- Use a line `//----------` as separation between methods
- Use `//` comments to explain each major segment of code
- See the sample program below

TESTER CLASS

```
package lab01;

/**
 * A Tester class for the TelephoneListConverter
 * @author TMR
 * @version May 2017
 */
public class TelephoneListConverterTester
{
    //-----------------------------------------------------------------
    public static void main(String[] args)
    {
        //Create test data
        String[] mixedNumbers = {
                "777-HOME", "777-home", "777-SAVE",
                "777-save", "1-800-TALK-LAW", "1-800-talk-law",
                "123-4567", "123#4567", "12$$%%&&",
                "ABC DEF GHHH", "IJK-LMN-OPQR", "STU-VWX-YZZZ",
                "abc def ghhh", "ijk-lmn-opqr", "stu-vwx-yzzz"
        };

        String[] expectedAnswers = {
                 "777-4663", "777-4663", "777-7283",
                 "777-7283", "1-800-8255-529", "1-800-8255-529",
                 "123-4567", "123$4567", "12$$$$$$",
                 "222$333$4444", "455-566-6777", "788-899-9999",
                 "222$333$4444", "455-566-6777", "788-899-9999"
        };
        //Create the converter object
        TelephoneListConverter tlc =
            new TelephoneListConverter(mixedNumbers);

        //Get the mapped list
        String[] pureTelNumList = tlc.getPureTelNumList();
```

```
        //Print the table
        System.out.println("Table of Test Results: "+
            "\n-----------------------------------------------" +
            "\nInput Value \tExpected Result \tComputed Result ");
        for (int index = 0; index < pureTelNumList.length; index++)
        {
            System.out.println(mixedNumbers[index] + "\t" +
                        expectedAnswers[index] + "\t" +
                        pureTelNumList[index]);
        }
    }
}

//--------------------COMPLETED CODE-----------------------
package lab01;

/**
 * Converts a list of mixed letter-digit telephone numbers
 * into pure telephone numbers using the standard mapping.
 *
 * @author T.M. Rao
 * @version May 2017
 */
public class TelephoneListConverter
{
    //---------------------------------------------------
    //Input list of mixed numbers
    private String[] mixedTelNumList;

    //---------------------------------------------------
    //Resulting list of pure numbers
    private String[] pureTelNumList;

    //---------------------------------------------------
    public TelephoneListConverter(String[] mixedList)
    {
        //Assign parameter to instance variable
        mixedTelNumList = mixedList;

        //Create another string array of the same size
        pureTelNumList = new String[mixedTelNumList.length];

        //Do the mapping
        for (int index = 0; index < mixedTelNumList.length; index++)
        {
            pureTelNumList[index] = mapString(mixedTelNumList[index]);
        }
    }

    //---------------------------------------------------
    /**
     * @param mixedNumber: A string object
     * @return String: result of the mapping
     */
    private String mapString(String mixedNumber)
    {
        //Start with empty result string
        String result = "";

        //for each character in the mixedNumber
        for (int index = 0; index < mixedNumber.length(); index++)
```

```
        {
            //map it to number and concat it to the result
            result = result + mapChar(mixedNumber.charAt(index));
        }

        //return result string
        return result;
    }


    //-----------------------------------------------------
    /**
     * This maps one character to one number using the standard
     * mapping. Any chars other than numbers, uppercase or lowercase
     * letters, will be flagged as errors and left unchanged
     */
    private char mapChar(char ch)
    {
      //map the ch to upper case
      char thisChar = Character.toUpperCase(ch);

        if (thisChar >= 'A' && thisChar <= 'C')
            return '2';
        if (thisChar >= 'D' && thisChar <= 'F')
            return '3';
        if (thisChar >= 'G' && thisChar <= 'I')
            return '4';
        if (thisChar >= 'J' && thisChar <= 'L')
            return '5';
        if (thisChar >= 'M' && thisChar <= 'O')
            return '6';
        if (thisChar >= 'P' && thisChar <= 'S')
            return '7';
        if (thisChar >= 'T' && thisChar <= 'V')
            return '8';
        if (thisChar >= 'W' && thisChar <= 'Z')
            return '9';
        if (thisChar >= '0' && thisChar <= '9')
            return thisChar;
        if (thisChar == '-')
            return thisChar;

        //Any other character is invalid. We return $
        return '$';
    }

    //-----------------------------------------------------
    String[] getPureTelNumList()
    {
        return pureTelNumList;
    }
}
```

Your Problem: You have been given list of words (call it **`originalList`**) each of which can contain letters, digits or any other characters. The encoding of these words is done as follows:

Characters are mapped as shown in the table below:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b

0	1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	0	1

A maps to **C**, etc. Any character not listed here maps to itself. Thus a '**?**' maps to '**?**'

Write a Java program to perform the encoding and decoding operations: Your program should accept the original list and create an `encodedList` where each word is an encoding of the corresponding word in the `originalList`. It should then create a `decodedList` in which each word is a decoding of the corresponding word in the `encodedList`. Observe that the `decodedList` is identical to the original list.

Write a **class** called **`EncodeDecode`** that has:

The following instance variables:

- `originalList` – This is an array of Strings, it is assigned a value by the constructor
- `encodedList` – Also an array of Strings. For each word of the original list, this will contain the corresponding encoded word
- `decodedList` – Also an array of Strings. For each word in the `encodedList` this will contain its decoded word, which should be the same as the corresponding word in the original list.

The following methods:

- Constructor: The original list is passed to the constructor as a parameter `oL`. The constructor assigns `oL` to `originalList`. It will then call the encode method for each of the words in the `originalList` to populate the `encodedList`. It then calls the decode method for each word in the `encodedList` to populate the `decodedList`.
- `String encode (String originalWord)` – this maps every character in original word to 2 positions forward, with wraparound.
- `String decode (String codedWord)` – this maps every character in coded word to 2 positions back, with wraparound.
- `char forwardMap(char ch)` – supporting method, it maps the given `ch` to 2 positions forward (if the `ch` is not a letter or digit, it maps to itself)
- `char backMap(char ch)` – supporting method, it maps the given `ch` to 2 positions back (if the `ch` is not a letter or digit, it maps to itself)
- `getEncodedList` and `getDecodedList` – just get methods

You should create (in this order) and turn in (submit on Blackboard) the following documents.

1. Test Plan: Make a table showing the input data and the expected output. Ensure that all parts of your code are properly tested.
2. Actual Code: Write the class, instance variables and methods *skeleton*
3. "Flesh out" the skeleton with the method bodies
4. A class called **`EncodeDecodeTester`** that does the testing using the test plan you have created.
5. Terminal Window showing the results.

The above will be graded only for completion.