**package** mypackage;-

**import** java.util.Arrays;

**import** java.util.HashMap;

**import** java.util.Scanner;

/\*\*

\* Created by Rose on 1/27/17.

\* Assignment for your lecture 3. Please finish all the questions under

\* 'Assignment'. Please try to think the extra credit questions. The deadline

\* of this assignment is 02/04/2017 23:59 PST. Please feel free to contact me

\* for any questions. Please write your comments about this assignment in the

\* end.

\*/

**public** **class** Assignment3 {

/\*\*

\* Given an array, reverse the elements within this array and print the result

\* eg, given{1,2,3,4}, print{4,3,2,1}

\*/

**public** **static** **void** reverseArray(**int**[] inArray) {

System.***out***.println("Program 1 - Reverse Arrays output is:");

System.***out***.println("Array Before Reverse : "+Arrays.*toString*(inArray));

**int** temp;

**for** (**int** i = 0; i < inArray.length/2; i++)

{

temp = inArray[i];

inArray[i] = inArray[inArray.length-1-i];

inArray[inArray.length-1-i] = temp;

}

System.***out***.println("Array After Reverse :"+Arrays.*toString*(inArray));

}

/\*\*

\* Given a non-negative integer represented as a non-empty array of digits, plus one to the integer.

\* Assume the integer do not contain any leading zero, except the number 0 itself.

\* The digits are stored such that the most significant digit is at the head of the array.

\* eg, given {1,2,9}, reutrn{1,3,0}.

\* **@return**

\*/

**public** **static** **int**[] plusOne(**int**[] digits) {

**int** carry = 1;

**int**[] result = **new** **int**[digits.length];

**for** (**int** i = digits.length - 1; i >= 0; i--)

{

**int** val = digits[i] + carry;

result[i] = val % 10;

carry = val / 10;

}

**if** (carry == 1)

{

result = **new** **int**[digits.length + 1];

result[0] = 1;

}

**return** result;

}

/\*\*

\* Write a program that takes an integer as input and returns all the primes between 1 and that integer(inclusive).

\* eg, input is 18, you should return{2,3,5,7,11,13,17}

\*/

**public** **static** **void** generatePrimes(**int** n) {

System.***out***.println("Generate Prime numbers between 1 and " + n);

**for** (**int** i = 1; i<n; i++) {

**boolean** PrimeNumber = **true**;

**for** (**int** j = 2; j < i; j++) {

**if** (i % j == 0) {

PrimeNumber = **false**;

**break**;

}

}

**if** (PrimeNumber) {

System.***out***.print(i + " ");

}

}

}

/\*\*

\* Assume you have a method isSubstring which checks if one word is a substring of another.

\* Given two strings, s1 and s2, write a program to check if s2 is a rotation of s1, using only one call

\* to isSubstring

\* eg, "pineapple" is a rotation of "neapplepi"

\*/

**public** **void** isRotation(String s1, String s2) {

}

/\*\*

\* Given two strings, write a method to decide if one is a permutation of the other

\* hint: the comparison is case sensitive and whitespace is significant

\* **@return**

\*/

**public** **static** **boolean** isPermutation(String s1, String s2) {

HashMap<Character, Integer> map = **new** HashMap<Character, Integer>();

**int** count =1;

**if**(s1.length()!=s2.length())

{

**return** **false**;

}

**for**(Character c: s1.toCharArray())

{

**if**(!map.containsKey(c))

map.put(c, count);

**else**

map.put(c, count+1);

}

**for**(Character c: s2.toCharArray())

{

**if**(!map.containsKey(c))

**return** **false**;

**else**

map.put(c, count-1);

}

**for**(Character c: map.keySet())

{

**if**(map.get(c)!=0)

**return** **false**;

}

**return** **true**;

}

/\*\*

\* Write a program to implement encoding and decoding string. The rule is simple: encode successive

\* repeated characters by the repetition count and the character. For example, the input of encoding()

\* is "aaaabcccaa", you should return "4a1b3c2a". The decoding of "3e4f2e" returns "eeeffffee". Assume

\* the string to be encoded consists of letters of the alphabet, with no digits, and the string to be

\* decoded is a valid encoding.

\* **@return**

\*/

**public** **static** String encoding(String str) {

**int** count = 1;

StringBuilder builder = **new** StringBuilder();

**for**(**int** i = 1; i<str.length(); i++){

**if**(str.charAt(i) == str.charAt(i-1) && i < str.length()-1){

count++;

}

**else** **if**(i == str.length()-1 && str.charAt(i) == str.charAt(i-1)){

count++;

builder.append(str.charAt(i));

builder.append(count);

}

**else** **if**(i == str.length()-1 && str.charAt(i) != str.charAt(i-1)){

builder.append(str.charAt(i-1));

builder.append(count);

count = 1;

builder.append(str.charAt(i));

builder.append(count);

}

**else**{

builder.append(str.charAt(i-1));

builder.append(count);

count = 1;

}

}

str = builder.toString();

System.***out***.println(str);

**return** str;

}

**public** **static** **void** decoding(String str) {

StringBuilder sb = **new** StringBuilder();

**int** repeat = 0;

**for** (**char** c : str.toCharArray()) {

**if** (Character.*isDigit*(c)) {

repeat = repeat \* 10 + Character.*getNumericValue*(c);

sb.append(str.charAt(c));

} **else** {

**while** (repeat > 0) {

sb.append(c);

repeat--;

}

sb.append(c);

}

}

System.***out***.println(sb.toString());

}

//Extra Credit

/\*\*

\*Given an m x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

\* For exmaple, given 1 2 3 , return 7 4 1

\* 4 5,6 8 5 2

\* 7,8,9 9 6 3

\*tip: image could be a square or a rectangle.

\*/

**public** **static** **void** rotate(**int**[][] matrix) {

{

*transpose*(matrix);

**for** (**int** j = 0; j < matrix[0].length/2; j++) {

**for** (**int** i = 0; i < matrix.length; i++) {

**int** x = matrix[i][j];

matrix[i][j] = matrix[i][matrix[0].length -1 -j];

matrix[i][matrix[0].length -1 -j] = x;

}

}

}

}

**private** **static** **void** transpose(**int**[][] matrix) {

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = i; j < matrix[0].length; j++) {

**int** x = matrix[i][j];

matrix[i][j] = matrix[j][i];

matrix[j][i] = x;

}

}

}

/\*\*

\*Given a string containing just the characters '(', ')', return the count of valid parentheses. If the

\* intput is not valid, return -1. A valid parentheses is "()". For example, given "(())", return 2;

\* given "(()))", return -1.

\*/

**public** **void** countValidParentheses(String s) {

//write your code here

}

/\*\*

\* Write anything you think about this assignment here. Easy? Difficult? Too many questions? Less fun?

\* You could write any comments here

\*/

**public** **static** **void** main(String[] args){

System.***out***.println("Program - Permutation");

String s1= "abc";

String s2 ="cba";

System.***out***.println(*isPermutation*(s1,s2));

System.***out***.println("\n---------------------------------");

System.***out***.println("Program - Generate Primes");

*generatePrimes*(35);

System.***out***.println("\n---------------------------------");

System.***out***.println("Program - Encoding");

*encoding*("aabcccccaaa");

System.***out***.println("---------------------------------");

System.***out***.println("Program - Decoding");

*decoding*("a2b1c5a3");

System.***out***.println("---------------------------------");

System.***out***.println("Program - Rotate Array");

**int**[][] matrix = {{1, 3},{2, 4}};

*rotate*(matrix );

**for** (**int** i = 0; i < matrix .length; i++) {

**for** (**int** j = 0; j < matrix [0].length; j++) {

System.***out***.print(matrix [i][j] + " ");

}

System.***out***.println();

}

System.***out***.println("---------------------------------");

*reverseArray*(**new** **int**[]{1, 2, 3, 4});

System.***out***.println("Program 2 - Add One");

System.***out***.println(Arrays.*toString*(*plusOne*(**new** **int**[]{0})));

System.***out***.println(Arrays.*toString*(*plusOne*(**new** **int**[]{2})));

System.***out***.println(Arrays.*toString*(*plusOne*(**new** **int**[]{9,7,6,5})));

}

}