MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY

“KHARKOV POLYTECHNICAL INSTITUTE”

LABORATORY WORK № 4

“**Working with Java Arrays and Strings. Creation of Classes** ”

Created by student of 1.КН.201.8г

Chukwu Irele Omike

Checked by

KHARKIV 2019

**1.1 Individual Assignment**

Develop two classes according to individual assignment. The first class should contain reference to an array of second class type. Classes must contain constructors, private fields, access methods (getters and setters), as well as methods needed for implementation of an individual assignment.

Each class should be separately tested. The main() function of the first class should contain creation of necessary object and invocation of methods implementing an individual assignment. Results should be shown on console window.

Particular function is given in the individual task according to your own index in the group students list (index of variant).

Table 1.1 - Individual Assignments

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | First Class | | Second Class | | Main Assignment:  Find and show the following data: |
| Entity | Obligatory Fields | Entity | Obligatory Fields |
| 1 | Weather | Season, comments | Day | Date, temperature, comments | Average temperature, day with the maximum temperature, day with longest comment |
| 2 | Course of studies | Title, presence of an exam | Practical training | Date, topic, count of students | Average count of students, lesson with maximum students present, list of topics with particular word in a title |
| 3 | Tram stop | Name, list of tram lines | Hour | Count of passengers, comments | Total count of passengers, stop with the minimum count of passengers, the longest comment |
| 4 | Course of studies | Title, surname of lecturer | Lecture | Date, topic, count of students | Lecture with maximum students present, list of topics with particular word in a title, the last letter in the lecturer's surname |
| 5 | Weather | Year, comments | Measurement of temperature | Date, temperature, comments | Measurements with the minimum temperature, with the maximum count of words in measurement comments, the last letter in a weather comment |
| 6 | Conference | Title, place | Session | Date, topic, count of members | Average count of members, session with the maximum count of members, length of title |
| 7 | Exhibition | Title, surname of painter | Day | Count of visitors, comments | Total count of visitors, day with the minimum count of visitors, list of comments with the particular word |
| 8 | Subway station | Name, year of opening | Hour | Count of passengers, comments | Total count of passengers, hours with the minimum count of passengers, hours the maximum count of words in comments |
| 9 | Doctor | Surname, specialization | Reception | Day, shift, count of visitors | Total count of visitors, reception with the minimum count of visitors, surname length |
| 10 | Musical group | Name of group, surname of leader | Professional tour | City, year, count of concerts | Professional tour with maximum count of concerts, list of tours to particular city, the last letter in the surname of leader |
| 11 | Workshop | Name, address | Shift | Count of repaired computers | Total count of computers, shift with the maximum count of repaired computers, length of street name |
| 12 | Doctor | Surname, length of service | Reception | Day, count of visitors, comments | Average count of visitors, reception with the minimum count of visitors, reception with the largest comments |
| 13 | Tram line | Number, average traffic interval | Tram stop | Name, count of passengers | Total count of passengers, stops with the minimum count of passengers, stops with the longest name |
| 14 | Twenty-four-hour kiosk | Denomination, address | Hour | Count of purchasers, comments | Total count of buyers, hour the minimum count of buyers, comments with particular words |
| 15 | Musician | Surname, genre | Concert | Date, gate | Common gate, concert with the minimum gate, count of words in genre name |

**1.2 Alignment of a String**

Read string from command line and then add spaces until string length will be equal to a given value. Spaces must be even added at the beginning and at the end of string.

**1.3 Selection Sort (Advanced Task)**

Initialize one-dimensional array of integers by random values. Sort an array using the following algorithm (Selection sort):

* find the index of the minimum value
* exchange this value with the value of the first unsorted item (no exchange is required if the minimum element is already in this position)
* sort the rest of the array, excluding already sorted items.

Display the result on the screen.

**1.4 Finding Fibonacci Numbers**

Implement a function for computing of Fibonacci numbers (up to the 92nd integer) using an auxiliary array (static field). The function parameter should be Fibonacci number's index. The search for Fibonacci numbers is carried out according to the following rule:

*F*(1) = *F*(2) = 1; *F*(*n*) = *F*(*n* - 2) + *F*(*n* - 1)

At the first call, the array is filled until the required number. At subsequent calls, the number either returns from the array, or is calculated using the last two numbers stored in an array. Now an array is filled until a new number. Use the long type to represent numbers.

Perform function testing for different values of numbers entered in an arbitrary order.

**1.5 Abbreviation**

Enter a string of several words from the keyboard. Create a new string consisting of the first letters of words with the converting these letters to uppercase.

**1.6 Quadratic Equation**

Create a class for solving quadratic equation. Provide an analysis of all possible combinations of coefficients and corresponding results (two roots, one root, if there is a linear equation, no solutions, the infinite number of roots). The method of finding the roots (solve ()) should return the number of roots (-1 if there is an infinite number of roots). Getters for roots (x1 and x2) should return values only if the source data were entered and roots were found.

Task1 :

Main.java

**package** creationofclasses ;

**import** java.util.\*;

**import** java.lang.\*;

**public** **class** Main

{

**private** **static** Weather *m\_weather*;

**private** **static** List<Measurements> *m\_measurementList*;

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

{

*m\_measurementList* = **new** ArrayList<Measurements>();

*m\_measurementList*.add(**new** Measurements(**new** String[] { "Today tempature is around 31 celcius", "Tempature 31 celcius."}, "14/5/2018", 31.3));

*m\_measurementList*.add(**new** Measurements(**new** String[] { "Tempature around 35 celcius. Too hot today", "35 celcius tempature"}, "14/5/2018", 35.6));

*m\_weather* = **new** Weather(2018, **new** String[] { "Weather is very hot today" }, *m\_measurementList*.toArray(**new** Measurements[*m\_measurementList*.size()]));

System.***out***.println("Minimum tempature is: " + *getMinimum*().getTempature());

System.***out***.println("Minimum Tempature Date: " + *getMinimum*().getDate());

System.***out***.println("Maximum count of word in comments: " + *getMaximumCountOfWordComment*());

System.***out***.println("Maximum comment length: " + *getMaximumCommentLength*());

System.***out***.println("Last letter of weather comment: " + *getLastLetter*());

}

**private** **static** Measurements getMinimum()

{

**int** index = 0;

**int** length = *m\_measurementList*.size();

**double** tempature = *m\_measurementList*.get(0).getTempature();

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

{

**if** (*m\_measurementList*.get(i).getTempature() < tempature)

{

tempature = *m\_measurementList*.get(i).getTempature();

index = i;

}

}

**return** *m\_measurementList*.get(index);

}

**private** **static** **char** getLastLetter()

{

**return** *m\_weather*.getComments()[0].charAt(*m\_weather*.getComments()[0].length() - 1);

}

**private** **static** String getMaximumCountOfWordComment()

{

**return** *m\_measurementList*.get(*getMaximumCountOfWord*()[0]).getComments()[*getMaximumCountOfWord*()[1]];

}

**private** **static** **int** getMaximumCommentLength()

{

**return** *m\_measurementList*.get(*getMaximumCountOfWord*()[0]).getComments()[*getMaximumCountOfWord*()[1]].length();

}

**private** **static** **int**[] getMaximumCountOfWord()

{

**int** length = *m\_measurementList*.size();

**int** wordCount = *m\_measurementList*.get(0).getComments()[0].length();

**int** measurementIndex = 0;

**int** commentIndex = 0;

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

{

**int** commentCount = *m\_measurementList*.get(i).getComments().length;

**for** (**int** j = 0; j < commentCount; j++)

{

**if** (wordCount < *m\_measurementList*.get(i).getComments()[j].length())

{

wordCount = *m\_measurementList*.get(i).getComments()[j].length();

measurementIndex = i;

commentIndex = j;

}

}

}

**return** **new** **int**[] { measurementIndex, commentIndex };

}

}

Measurements.java

package creationofclasses ;

import java.text.\*;

import java.util.\*;

public class Measurements

{

private String[] m\_comments;

private String m\_date;

private double m\_tempature;

public Measurements()

{

}

public Measurements(String[] comments, String date, double tempature)

{

m\_date = date;

m\_comments = comments;

m\_tempature = tempature;

}

public void setComments(String[] comments)

{

m\_comments = comments;

}

public void setDate(String date)

{

m\_date = date;

}

public void setTempature(double tempature)

{

m\_tempature = tempature;

}

public String[] getComments()

{

return m\_comments;

}

public String getDate()

{

return m\_date;

}

public double getTempature()

{

return m\_tempature;

}

}

Weather.java:

**package** creationofclasses ;

**public** **class** Weather

{

**private** **int** m\_year;

**private** String[] m\_comments;

**private** Measurements[] m\_measurements;

**public** Weather(**int** year, String[] comments, Measurements[] measurements)

{

m\_year = year;

m\_comments = comments;

m\_measurements = measurements;

}

**public** **void** setYear(**int** year)

{

m\_year = year;

}

**public** **int** getYear()

{

**return** m\_year;

}

**public** **void** setComments(String[] comments)

{

m\_comments = comments;

}

**public** String[] getComments()

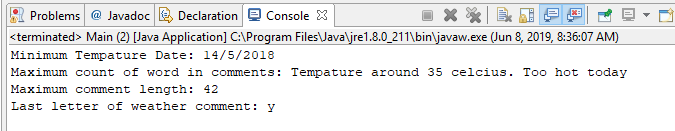
{

**return** m\_comments;

}

}

Execution:



Task2 :

Code:

**package** allignmentofstring;

**import** java.util.\*;

**public** **class** Main

{

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

{

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Type something: ");

String inputText = scanner.next();

String result = " ";

**int** length = inputText.length();

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

{

result += inputText.charAt(i) + " ";

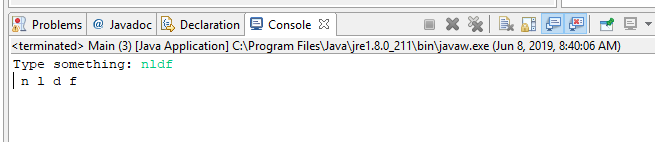
}

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

}

}

Execution:



Task 3

The code :

**package** selectionsort;

**import** java.util.\*;

**public** **class** Main

{

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

{

**int**[] numbers = **new** **int**[10];

System.***out***.println("Original Array");

**for** (**int** i = 0; i < 10; i++)

{

Random rand = **new** Random();

numbers[i] = rand.nextInt(100);

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

}

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

System.***out***.println("Minimum element index in unsorted array: " + *findMinimumIndex*(numbers));

*selectionSort*(numbers);

System.***out***.println("Sorted Array");

**for** (**int** i = 0; i < 10; i++)

{

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

}

}

**private** **static** **int** findMinimumIndex(**int**[] array)

{

**int** index = 0;

**int** minimum = array[0];

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

{

**if** (array[i] < minimum)

{

minimum = array[i];

index = i;

}

}

**return** index;

}

**public** **static** **void** selectionSort(**int**[] array)

{

**for** (**int** i = 0; i < array.length - 1; i++)

{

**int** index = i;

**for** (**int** j = i + 1; j < array.length; j++)

{

**if** (array[j] < array[index])

{

index = j; // minimum index

}

}

**int** smallerNumber = array[index];

array[index] = array[i];

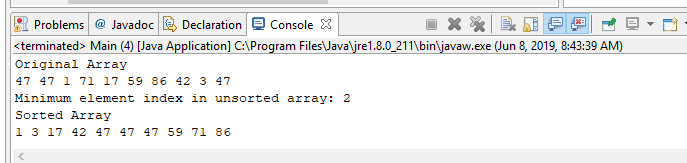
array[i] = smallerNumber;

}

}

}

Execution :



Task 4:

The code :

**public** **class** Main

{

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

{

**int** number = 92;

**for** (**int** i = 1; i <= number; i++)

{

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

}

}

**private** **static** **int** fibonacciRecursion(**int** number)

{

**if** (number == 1 || number == 2)

{

**return** 1;

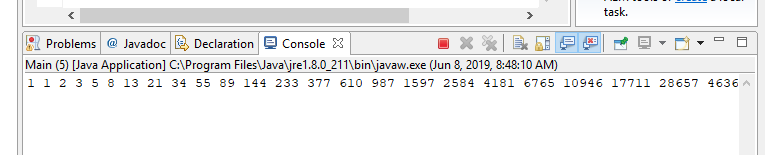
}

**return** *fibonacciRecursion*(number - 1) + *fibonacciRecursion*(number - 2);

}

}

Execution:



Task 5:

The code:

**import** java.util.\*;

**public** **class** Main

{

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

{

**final** **int** length = 6;

String[] words = **new** String[length];

Scanner scanner = **new** Scanner(System.***in***);

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

{

System.***out***.print("Type something: ");

words[i] = scanner.next();

}

String newWord = "";

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

{

newWord += Character.*toString*(words[i].charAt(0)).toUpperCase();

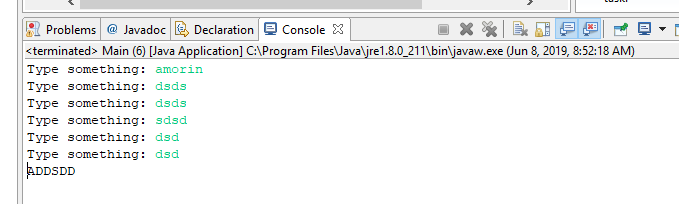
}

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

}

}

Execution



Task6:

The code:

**package** equa;

**import** java.util.\*;

**public** **class** Main

{

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

{

Scanner scanner = **new** Scanner(System.***in***);

**double** firstRoot = 0;

**double** secondRoot = 0;

System.***out***.print("Please enter a: ");

**double** first = scanner.nextDouble();

System.***out***.print("Please enter b: ");

**double** second = scanner.nextDouble();

System.***out***.print("Please enter c: ");

**double** third = scanner.nextDouble();

QuadraticEquation equation = **new** QuadraticEquation(first, second, third);

equation.solve();

}

}

**package** equa;

**public** **class** QuadraticEquation

{

**private** **double** m\_first;

**private** **double** m\_second;

**private** **double** m\_third;

**public** QuadraticEquation(**double** a, **double** b, **double** c)

{

m\_first = a;

m\_second = b;

m\_third = c;

}

**public** **void** solve()

{

**double** firstRoot = 0;

**double** secondRoot = 0;

**double** determinant = (m\_second \* m\_second) - (4 \* m\_first \* m\_third);

**double** sqrt = Math.*sqrt*(determinant);

**if** (determinant > 0)

{

firstRoot = (-m\_second + sqrt) / (2 \* m\_first);

secondRoot = (-m\_second - sqrt) / (2 \* m\_first);

System.***out***.println("Roots are: " + firstRoot + " and " + secondRoot);

}

**else** **if** (determinant == 0)

{

System.***out***.println("Root is: " + (-m\_second + sqrt) / (2 \* m\_first));

}

}

}