**Sofia University  
Department of Mathematics and Informatics**

**Course : OO Programming with Java**

**Date: October 14, 2018**

**Student Name:**

**Lab No. 3a**

**Problem No. 1**

В редица случаи се налага използване на меню, за представяне на различни възможности за избор при изпълнение на програма.

Да предположим, че имате за задача да позволите да въведе число в плаваща запетая и и той да може да пресметне експонентата, синуса и най- голямото цяло число, което е по- малко или равно на въведеното число.

Нека потребителят прави своя избор, посредством следното меню в текстов формат

* 1. *Въведи число x в плаваща запетая*
  2. *Пресметни и изведи exp(x)*
  3. *Пресметни и изведи sin(x0*
  4. *Пресметни и изведи floor(x)*
  5. *Край*

1. **Напишете** алгоритъма и **UML визуализацията**  **(на хартия**) **за мето**д , който

* изобразява това меню, докато не се избере **опция 5**,
* позволява на потребителя да **въведе номер** на желана опция
* **изпълнява**, избраната опция
* **изчиства** текстовия екран текстовия екран преди ново извеждане на менюто
* **използвайте “*разделяй и владей*” за описване на алгоритъма**

2. Задачата **да се структурира** като програма на **Java**, състояща се от два класа **по аналогия** с **fig 6.3- 6.4**  от **Лекция Седмица 3**

Единият от класовете (**пасивният** клас или т. нар клас за многократно ползване) да реализира табулацията (например, именувайте го ***class*** ***ComputeByMenu***). Нека **този клас да има**

* **клас данна**  ***double x***
* **конструктор за общо ползване**, който инициализира ***x*** при създаване на обект от ***class ComputeByMenu***
* метод ***public void displayMenu() {}***, който да **изчиства** текстовия екран (изведете **50-60** празни реда) и **извежда** менюто
* метод ***public void doSelection(int choice) {},*** който да изпълнява избраната опция с номер ***choice-*** използвайте **клас данната**  ***x*** и методите на ***class Math***
* метод ***public void getUserChoice() {},*** който да извежда менюто и изпълнява функциите му

Другият клас (**активният** клас или т. нар клас за тестване, например, именувайте го ***class*** ***MenuTest*** ) да реализира

* да **създаде обект** от пасивния клас (***class*** ***ComputeByMenu***) като **използва конструктора му за общо ползване** за да **инициализира този обект** със стойност 0.00 за **клас данната**  ***x***
* **изпълнява** метода  ***getUserChoice()*** на така създадения обект за ***class*** ***ComputeByMenu***

**Problem No. 2a**

Write a **Java** application to compute the value of ***exp(x)*** using the representation of the exponent in series.

Input the variable x and the accuracy of computations. Output the computed value, the desired accuracy and the accurate value using the library function *exp()*.Use **JavaFX** components to get the user input and display the output.

**Write** a test application to output the *exp(x)* values in the range *[0,1] f*ormatted with **3 digits after the decimal point**. (use a step for x equal to 0.01)

**Problem No. 2b**

Write a **Console application** that computes the value of by using the formula:



with accuracy  provided as user input. Compare the result with the value returned by the **respective static method** in the **Math** class. Output the approximate and the accurate value, as well as, the given accuracy using formatted numeric output.

**Problem No. 5**

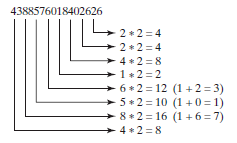
Credit card numbers follow certain patterns.

A credit card number must have between 13 and 16 digits. It must start with:

* 4 for Visa cards
* 5 for Master cards
* 37 for American Express cards
* 6 for Discover cards

In 1954, **Hans Luhn** of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check,* which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.



2. Now add all single-digit numbers from Step 1.

4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

3. Add all digits in the odd places from right to left in the card number.

6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

4. Sum the results from Step 2 and Step 3.

37 + 38 = 75

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a **long** integer. Display whether the number is valid or invalid. Design your program to use the following methods:

/\*\* Return true if the card number is valid \*/

**public static boolean isValid(long number)**

/\*\* Get the result from Step 2 \*/

**public static int sumOfDoubleEvenPlace(long number)**

/\*\* Return this number if it is a single digit, otherwise,

\* return the sum of the two digits \*/

**public static int getDigit(int number)**

/\*\* Return sum of odd-place digits in number \*/

**public static int sumOfOddPlace(long number)**

/\*\* Return true if the digit d is a prefix for number \*/

**public static boolean prefixMatched(long number, int d)**

/\*\* Return the number of digits in d \*/

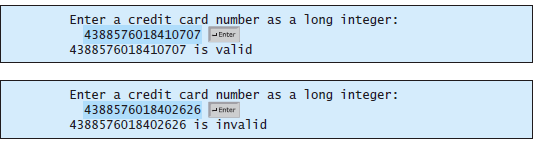
**public static int getSize(long d)**

/\*\* Return the first k number of digits from number. If the

\* number of digits in number is less than k, return number. \*/

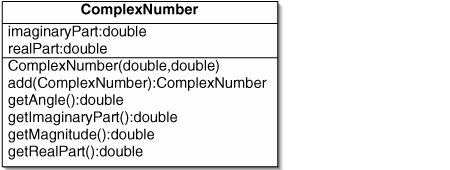
**public static long getPrefix(long number, int k)**

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.)

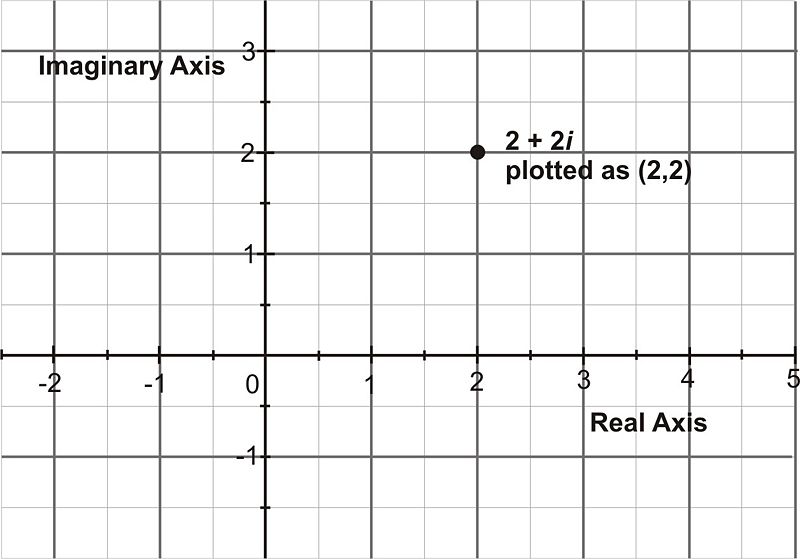


**Problem No. 6**

**Submit both the Java application project for the following class ComplexNumber**



**To compute the angle in the trigonometric representation of a complex number use the static method Math.atan2(double x, double y) explained as** [**http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html#atan2(double,%20double)**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html#atan2(double,%20double))



**If you are given and then use and and the complex number is represented in trigonometric form as , where and**

**Write a ComplexNumberTest class to test the design and code. Collect the user input through dialog boxes (JavaFX)**

**Problem No. 6**

Write a **static** method

public static int sizeof(Class dataType)

that returns the size in bytes of the primitive datatypes in Java.

Sample output:

**Output:**

size of byte in Java is (in bytes) : 1

size of short in Java is (in bytes) :2

size of char in Java is (in bytes) :2

size of int in Java is (in bytes) :4

size of long in Java is (in bytes) :8

size of float in Java is (in bytes) :4

size of double in Java is (in bytes) :8

**Lab No. 3Plus**

**Problem No. 1**

Write a JavaFX application *class* *Geometry* to **draw a circle** in a *Group*. The *Circle* should be in the center of the *Scene* and have a *radius* which **is 1/3 of the minimum** of the height and width of the *Scene*. Draw also **a** *line* **AB orthogonal** to the x- axis starting from a given x coordinate value. **Find the** *intersection* **points** of the line AB with the circle. In case there are no intersection points display an *Alert* dialog window with a warning message and quit the application. **Draw small circles** with centers at the intersection points **and display the text of the coordinates on the intersection**, appropriately formatted with 2 digits after the decimal point. Draw the circle and the line *AB* in different colors (red and blue). The *x***- and** *y***-coordinates of the line** *AB*should be **read before drawing the line with** *TextInputDialog*and **be validated to be in the interval** *[0, w***], where** *w* **is the width of the** *Scene*.

**Problem No. 2**

Write a **JavaFX application with** *class* *Ellipse* that displays an **ellipse** centered in the middle of a *Scene* with given **large** and **small** axis, respectively *a* and *b* **using the polar vector representation** of the ellipse

+ where

and

are the coordinates middle of the *scene*

The **large** and **small** axis, respectively *a* and *b* should be **read at test time in** *class* *EllipseTest* **from standard input**

**Hint**: Apply discretization of the interval and connect adjacent points with *x***- and** *y***-coordinates** through line segments .

**Problem No. 3**

Write a **Java** *class* *Sine* that displays the sine function inside a *Group* panel inside a *Scene*. Consider an appropriate segmentation for the argument of x of *y= sin(x)* in the interval for *x [0, 2\*PI]* and connect the neighboring segments with *Line* segments to obtain a smooth presentation of the *sine* curve. Read the number of discrete points in *[0, 2\*PI]* **with a** *TextInputDialog.*

Consider scalling of the sine values so that the sine function is displayed along the middle line across the *Scene* and has a 14 px padding from the borders of the Scene . Draw the *x*- axis of the sine plot.