תוכנה 1

תרגיל מספר 4

הנחיות כלליות:

- קראו בעיון את קובץ נוהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה VirtualTAU בלבד (http://virtual2002.tau.ac.il/).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש (לדוגמא, עבור המשתמש zvainer יקרא הקובץ zvainer.zip) קובץ ה zip יכיל:
  א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז. הזהות שלכם.
  ב. קבצי ה java של התוכניות אותם התבקשתם לממש.
  ג. קובץ טקסט (פורמט txt או doc) עם העתק של כל קבצי ה java
  ד. קובץ טקסט (פורמט txt או doc) בשם answers עם התשובות לשאלות

בשאלות הבאות נעשה שימוש בקבצי jar. כאשר הנכם נדרשים להוסיף קובץ jar (או zip) לפרויקט יש לעשות זאת ב Eclipse ע"י קליק ימני על הפרויקט , בחירה ב Build Path ואח"כ Add External Archives. יש לבחור את הקובץ אותו רוצים להוסיף לפרויקט.

Turtle Graphics

In this question you will write a program that draws a simple picture. You are not required to learn anything about graphics, instead you will use a Turtle class supplied by us that implements turtle graphics as described below.

Introduction - LOGO and Turtle Graphics

LOGO is a simple programming language that is often used to introduce programming concepts as well as planar geometry concepts to children. A LOGO environment consists of a window representing a plane and a turtle that lives in this plane. The turtle has a tail that can be up or down. If the turtle is walking while its tail is down, it leaves behind it a line. When it walks while its tail is up, no line is left behind. The purpose of LOGO is to be able to draw/define various figures by giving instructions to the turtle.
The turtle has a location and a direction. You can give the turtle instructions to change its location and direction causing it to draw some figures along the way. For example, the instruction 'forward 30' tells the turtle to advance 30 units forward in the direction it is looking at. The instruction 'left 45' tells the turtle to turn 45 degrees counter-clockwise (i.e., change its direction by 45 degrees).

Here is a representative list of instructions you can give the turtle of LOGO:

forward x	advance x units forwards
backwards x	move x units backwards
left x	turn x degrees counter-clockwise
right x	turn x degrees clockwise
tail up	lifts the turtle's tail
tail down	lowers the turtle's tail

What You Should Implement

You are not expected to implement the emulation of LOGO by yourself. For this purpose we give you a Java class Turtle. Recall that this class defines all the behaviors of a LOGO turtle. Your program should only create a turtle object and give it instructions by invoking methods on it. The table below lists the methods of a Turtle object. You are encouraged to look at the API documentation of class Turtle.

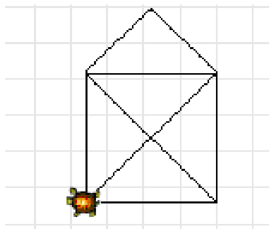
`moveForward(x)`	advance x units forwards
`moveBackward(x)`	move x units backwards
`turnLeft(x)`	turn x degrees counter-clockwise
`turnRight(x)`	turn x degrees clockwise
`tailUp()`	lifts the turtle's tail
`tailDown()`	lowers the turtle's tail

Pentagon.java is an example of a program that uses class Turtle to draw a 'pentagon' figure (Below is the skeleton of the program with some additional comments).

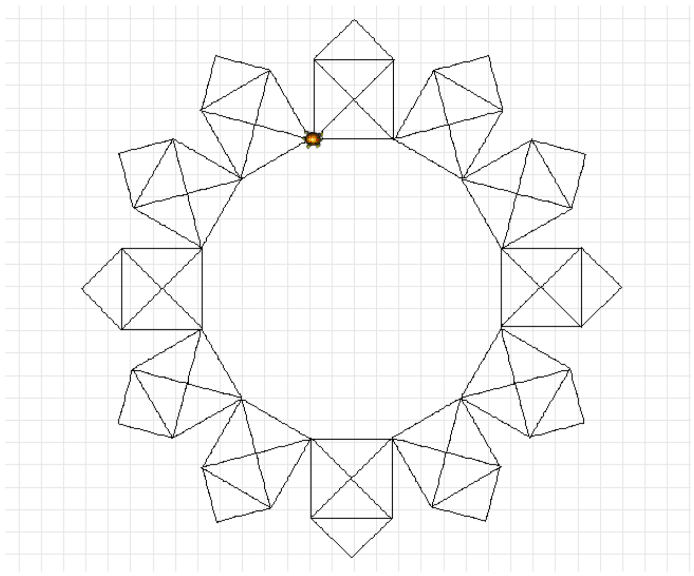
```
class Pentagon {

    public static void main(String[] args) {
        Turtle leonardo = new Turtle(); // Creates the turtle
        leonardo.tailDown();            // Start painting
        leonardo.moveForward(100);      // Advances the turtle
                                        // forward by 100 units
        // ...
    }
 }
```

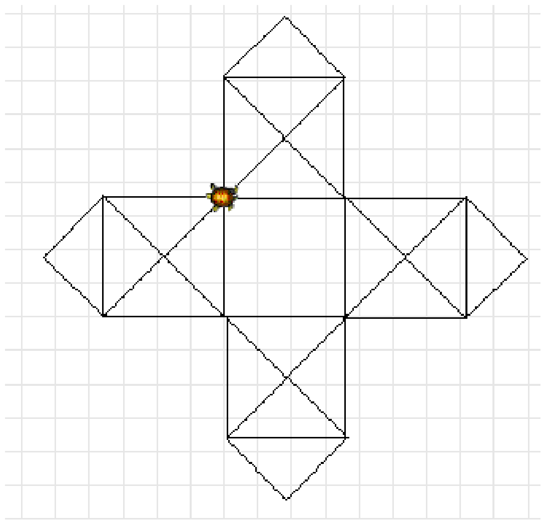
Recall that an Euler drawing is a drawing that can be done without lifting the pencil from the paper. In other words it is a figure that is drawn without going over any line twice. This is an example of an Euler drawing with our turtle.



Write a class called **TurtleDrawing** that draws the above figure *n* times, where *n* is provided by the user. After each figure is drawn the turtle rotates so that at the end it will complete a full cycle. In order to receive input from the user you should use the LineInput class (located in the simpleio.jar file). For example below is the output of TurtleDrawing when it is called to draw 12 figures.



And this is how the output would look for 4 figures:



After finshing drawing, hide the turtle using the method hide().

Technical Details

In order to use our Turtle class and the LineInput class you should include both logo_turtle.jar and simpleio.jar on your project as explained above. Note that Turtle and LineInput are defined in the packages `il.ac.tau.cs.sw1.turtle` and `il.ac.tau.cs.sw1.simpleio` respectively. Make sure to import those classes.

<u>Image Processing</u>

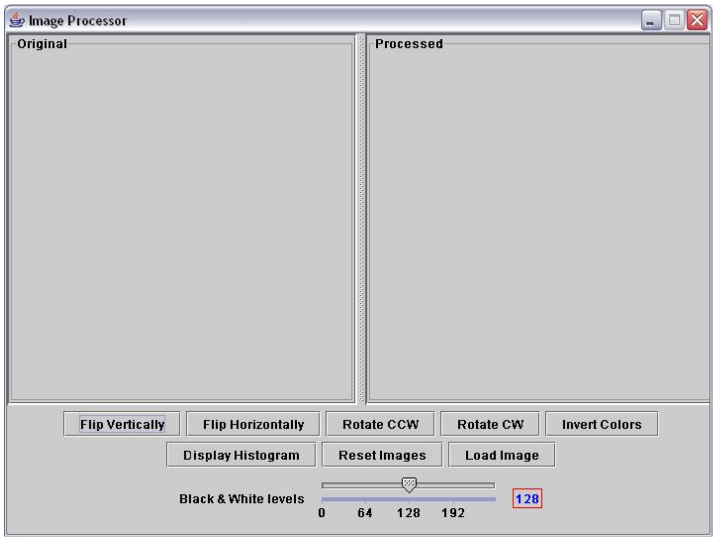
In this section you will implement utilities for image processing. Your functions will be able to rotate, flip and modify gray scales of a displayed image.
Since reading an image file and displaying it is out of the scope of this course, we supply you with a JAR file that loads images and triggers the image processing functions.

What You Should Do

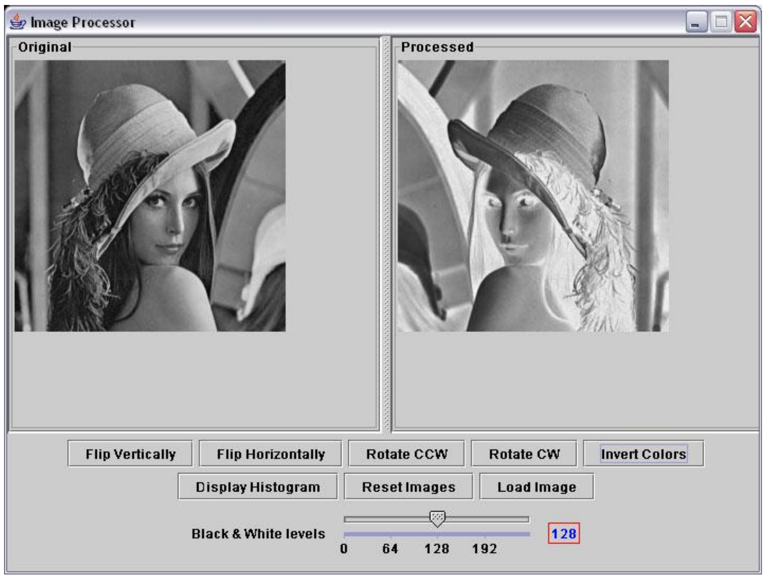
You are given a skeleton for the class `ImageProcessing`. You should implement all the defined methods in the class. The documentation for the class is given here.
Each of the static methods in this class receives as an input a two dimensional array of image data. To simplify things we define that only grayscaled images are supported in our program, thus the image data contains values between 0 and 255, where 0 symbols the **black**, 255 symbols the **white** and anything in between symbols a level of gray.
The graphical interface we supply invokes the **`ImageProcessing`** methods by sending the image data as a two-dimensional int array, receiving the results of the operation (your implementation) and displaying the produced image in the '*Processed*' frame. Every cell in the array is actually the gray level of the pixel (picture element) at its corresponding image position.

Technical Details

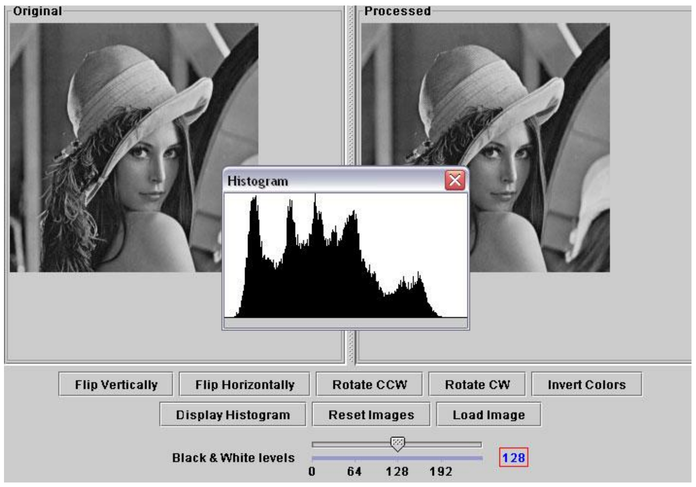
- Download the image_processor.jar, and make sure to include it on your build path as explained above.
  At any stage of your work you can test your methods by executing the graphic interface we supply simply by running the skeleton program. Execution of the main method will display the application's screen (see below).
  Initially none of the functionality will work as clicking on the various buttons invokes your methods. As your implementation progresses you should be able to see the result of it, this will aid you in testing and debugging your implementation.



- To load an image click on the 'Load Image' button and select a gray-scaled image you wish to view. You can find a zip file containing 2 images here that you can play with. For example, loading the 'lena.jpg', flipping it (along the Y axis) and inverting it will yield the image:



- The **histogram** method creates a histogram for the given image data. It returns a 256 cells array that holds the histogram representation. Each cell index in the array is referred as the *column height* for a given gray level that corresponds to the array's index number itself. For example, if the value 30 is in cell 0, that means we have 30 pixels with the value of 0 (black) in the loaded image. More generally: Cell ***i*** holds the number of image pixels with a gray level of ***i.***
Pressing the 'Display Histogram' button will display the loaded image histogram:



- The `invert` method changes the grayscale levels of the image so the 0 becomes 255, 1 becomes 254, 2 become 253 etc.
- The `toBW` method transforms the grayscale image into a black and white one. A pixel's value under a given threshold will become 0 otherwise it will become 255.
- The `flipX` method should perform a flip relative to the X axis - i.e. **vertical** flip, and the `flipY` should perform a **horizontal** flip. See animation for clarification.