משימה 3 בקורס תוכנה 1

הוראות הגשה:
2. הגשת התרגילים תעשת ע”י יצירת קובץ zip שימשך את שם המשתמש. לדוגמה, עבור הממשתמש zvainer יقرأ הקובץ zvainer.zip.
3. הקובץ_zip יכיל:
   a. קובץ הפרטים אישים בשמשמicl(details.txt) מהכלי בתCRM ומספר ת.ז. הوزارة ושלכמ.
   b. קבצי-ה-Java של התוכניות שנכתבוalty.sh�מת dáשוכלוב.
   c. קובצי הקסטים עם העתק של כל הקבצי h-Java.

שאלה:
ככל אתרי משולש השאלות הראשים והמסרים כרפוסים למושד מרסים. בנספים ולמויים מתודות המפורטות את הפוסט ומספור ת.ז. המסדרים (קורות הל מספרים עם ערכים שעונים) והמודפסים את התוכן המפורטים페כל קוראים.

1. Consider the leftmost and rightmost appearances of some value in an array. We’ll say that the "span" is the number of elements between those two, inclusive. A single value has a span of 1.
For example:
maxSpan({1, 2, 1, 3}) → 4
maxSpan({1, 4, 2, 1, 4}) → 6
maxSpan({1, 4, 2, 1, 4}) → 6

Implement the method public static int maxSpan(int[] nums). The method returns the maximal span found in a given array. (Efficiency is not a priority).

2. Implement the method public static int[] fix34(int[] nums). The method accepts and integer array as its input and returns a new array containing a permutation of the input array. The method fix34 rearranges the input array such that every 3 is immediately followed by a 4 (e.g. if there is a 3 at position i, there will be a 4 at position i+1). The method keeps the original positions of the 3s but may move any other number.
Assumptions regarding the input:

- The array contains the same number of 3’s and 4’s
- Every 3 is succeeded by a number that is not 3 nor 4
- A 3 appears in the array before any 4

Examples:
fix34({1, 3, 1}) → {1, 3, 1}
fix34({3, 2, 4}) → {3, 4, 2}
3. Implement the method `public static String notReplace(String str)`, which when given a string return a string where every appearance of the lowercase word "is" has been replaced with "is not". The word "is" should not be immediately preceded or followed by a letter -- so for example the "is" in "this" should not be replaced. (Note: Character.isLetter(char) tests if a char is a letter.)

Examples:

- `notReplace("is test") → "is not test"
- `notReplace("is-is") → "is not-is not"
- `notReplace("This is right") → "This is not right"

4. 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 when 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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward x</td>
<td>advance x units forwards</td>
</tr>
<tr>
<td>backwards x</td>
<td>move x units backwards</td>
</tr>
<tr>
<td>left x</td>
<td>turn x degrees counter-clockwise</td>
</tr>
<tr>
<td>right x</td>
<td>turn x degrees clockwise</td>
</tr>
<tr>
<td>tail up</td>
<td>lifts the turtle's tail</td>
</tr>
<tr>
<td>tail down</td>
<td>lowers the turtle's tail</td>
</tr>
</tbody>
</table>
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`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>moveForward(x)</code></td>
<td>advance x units forwards</td>
</tr>
<tr>
<td><code>moveBackward(x)</code></td>
<td>move x units backwards</td>
</tr>
<tr>
<td><code>turnLeft(x)</code></td>
<td>turn x degrees counter-clockwise</td>
</tr>
<tr>
<td><code>turnRight(x)</code></td>
<td>turn x degrees clockwise</td>
</tr>
<tr>
<td><code>tailUp()</code></td>
<td>lifts the turtle's tail</td>
</tr>
<tr>
<td><code>tailDown()</code></td>
<td>lowers the turtle's tail</td>
</tr>
</tbody>
</table>

`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).

```java
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
        // ...  
    }
}
```

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 simple_io.zip 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 finishing 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 simple_io.zip on your project as explained above.
5. In this exercise, you will get the code of the class Circle. Notice that the instance variables of class Circle include the radius, and two double values x0 and y0 which represent the center coordinate of the circle. We wish to change the implementation of class Circle so that instead of these two variables, we will have an instance of class Point as a member instance variable representing the center (download Point.java).

Remember that changing the implementation details should not affect the API of the class! This means that this change should not be noticed by outside users of class Circle. In particular you are not allowed to modify any of the method signatures. In practice you will surely need to change the implementation of some of the methods. Use the Circle.java file given to you and modify it to implement your solution.

To test your modified Circle class, we supply you with a class CircleViewer which you can use to display your circles. CircleTest.java is a simple program that uses CircleViewer. This program creates two circles and a CircleViewer object, and adds both circles to the viewer, using the method addCircle() of the viewer.

The CircleViewer "remembers" the circles it should display (by storing a reference to it), and whenever the update() method is called on it, it uses the Circle interface to retrieve the state of all the circles it should display (i.e., calls getCenter(), getRadius()), and draws them appropriately. To use the CircleViewer class, download shapes.zip and include it in your project as explained above.

Write your own simple program called SimpleCircleTest that tests your circle implementation class. Remember to test all the methods in the class, and use the CircleViewer class.

6. In this section you will implement a utility class for image processing. Your class 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 methods.

**What You Should Do**

Implement the class ImageProcessing according to its API. 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 gray-scaled 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 graphical interface we supplied.
  
  You can do so by creating a main class that will contain the following main method and placing it in the working directory:

  ```java
  public static void main(String[] args) {
      ImageProcessorUI processorUI = new ImageProcessorUI("Image Processor");
  }
  ```

  Executing the main will display the application's screen:

  ![Image Processor Interface](image)

  **Note:** DO NOT submit this main.

- Load an image by clicking 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 and inverting it will yield the image below:
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 `flipX` method should perform a flip relative to the X axis - i.e. vertical flip, and the `flipY` should perform a horizontal flip. The animation below might clear the issue.