Com S 227
Fall 2014
Miniassignment 3
40 points

Due Date: Friday, April 17, 11:59 pm (midnight)

"Late" deadline (25% penalty): Monday, April 20, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Blackboard. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit. See the section "What to turn in" at the end of this document

Please start the assignment as soon as possible and get your questions answered right away.

Overview

This is a short assignment designed to give you some practice working with 2d arrays and interfaces. It is also likely to challenge your debugging skills.

Your tasks are to write one method in the class `GridUtil` (which is partially implemented) and to create two new classes that implement a given interface called `ITransform`.

These two classes are called **`SmoothingTransform`** and **`ConwayTransform`**. The full specification consists of:

- this pdf,
- any "official" clarifications posted on Piazza, and
- the online javadoc: http://www.cs.iastate.edu/~cs227/homework/mini3/doc/

The javadoc describes the required methods and their behavior.

What it's about

Many applications using 2d arrays involve a transformation in which data in one cell is combined with data in some of the neighboring cells to compute a new value. All these new values together form a new 2d array that is the same size as the original (and in some cases the new values are copied back into to the original array).

One common type of example is a transformation on an image, which is just a 2d array of integers. For instance, a simple such transformation is "softening" an image to reduce pixelation and make the edges less sharp. The basic idea is that values at each pixel are averaged with a group of neighboring pixels. Here's an illustration:

The one on the left is the original 256x256 pixel image, and the one on the right is the result of taking each pixel and averaging it with its neighbors in a 5x5 area surrounding it.

Another example is *Conway's Game of Life*. Here the array consists only of values 0 or 1 and is initialized with some pattern. At each step, a new value for each cell is calculated from a 3x3

neighborhood according to the following rules, where *S* is the sum of the center cell's neighbors (not counting itself).

- if $S < 2$, the result is 0
- if the center cell is 1 and *S* is 2 or 3, the result is 1
- if $S > 3$ the result is 0
- if the center cell is 0 and *S* is exactly 3, the result is 1

Normally the contents of the array are animated over time to see how the system evolves. There are some examples of such animations on the Wikipedia page,

http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

In both the image example and the Game of Life example, the basic mechanism is the same, in pseudocode:

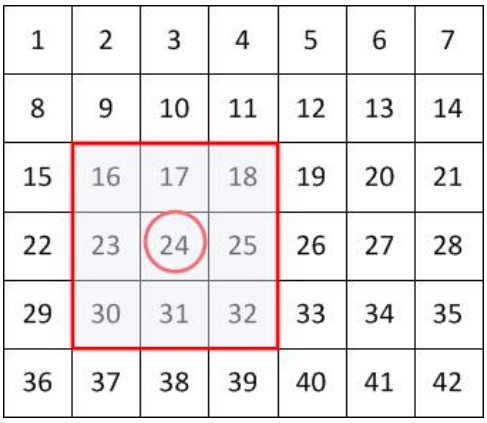
for each cell of the given array
  get a subarray consisting of a neighborhood of appropriate size
    calculate the result of the transformation from the subarray

The loop that carries out the pseudocode above is already written as the method GridUtil.applyAll:

```
public static int[][] applyAll(int[][] arr, ITransform transform)
{
  int numRows = arr.length;
  int numCols = arr[0].length;
  int[][] result = new int[numRows][numCols];
  for (int row = 0; row < numRows; row += 1)
  {
    for (int col = 0; col < numCols; col += 1)
    {
      int radius = transform.getRadius();
      boolean wrapped = transform.isWrapped();
      int[][] subArray = getSubArray(arr, row, col, radius, wrapped);
      int newValue = transform.apply(subArray);
      result[row][col] = newValue;
    }
  }
  return result;
}
```

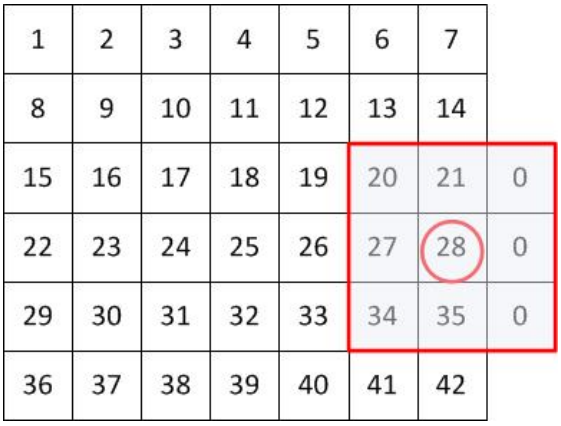
There are two things to notice: First, you will need to implement the method **`getSubArray`**, which finds the neighborhood of the center cell at **`row`**, **`col`** that will be used by the transformation. Second, **`ITransform`** is an interface, and part of your task is to create two sample classes that implement this interface, **`SmoothingTransform`** and **`ConwayTransform`**, corresponding to the two examples described in this section.

Example illustrating getSubArray and "wrapping" behavior



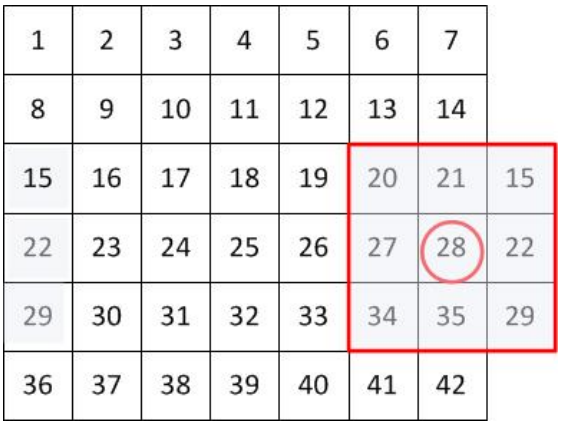
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

Figure 1 - a 3x3 neighborhood (radius 1) at row 3, column 2



1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	0
22	23	24	25	26	27	28	0
29	30	31	32	33	34	35	0
36	37	38	39	40	41	42	

Figure 2 - a 3x3 neighborhood (radius 1) at row 3, column 6 WITHOUT wrapping (cells for out-of-bounds indices are filled with zeros)



1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	15
22	23	24	25	26	27	28	22
29	30	31	32	33	34	35	29
36	37	38	39	40	41	42	

Figure 3 - a 3x3 neighborhood (radius 1) at row 3, column 6 WITH wrapping

Sample code to try

In the **`examples`** package you can find some sample applications you can run to see how your code might be used. (Naturally, you should not rely on such applications for testing purposes, but they might be fun to try.)

- ImageTest - reads an image file and uses the **`GridUtil`** to apply a **`SmoothingTransformation`** to the image and saves the resulting image as a file
- AnimatorMain - starts up a graphical animation of Conway's Game of Life using the **`GridUtil`** and a **`ConwayTransformation`**
- LifeTest - same as above but uses a simple console-based animation.

Getting started

Remember to work incrementally and test as you go. Do a bunch of examples using pencil and paper before you write any code. It can be tricky to get the indices right and you will waste a lot of time if you try to write the code by trial and error. The debugger is your friend. You can also print out a 2d array easily using a method such as **`printArray`** *in the* **`LifeTest`** *example, if you just want to examine the values.*

The SpecChecker

A SpecChecker will be available by April 14.

Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the console output. There are many test cases so there may be an overwhelming number of error messages. *Always start reading the errors at the top and make incremental corrections in the code to fix them.*

When you are happy with your results, click "Yes" at the dialog to create the zip file.

See the document “SpecChecker HOWTO”, which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links” if you are not sure what to do.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, *documentation is not required*. However, we recommend that you

document each method as you create it. It is usually easier to write a method correctly after you have written down what it is supposed to do!

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`miniassignment3`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`miniassignment3`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_mini3.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini3`**, which in turn contains the four files, **`GridUtil`**, **`ITransform`**, **`SmoothingTransform`** and **`ConwayTransform.`** Note that **`ITransform`** should not be modified.

Submit the zip file to Blackboard using the Miniassignment 3 submission link and **verify that your submission was successful by checking your submission history page**. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links."

*Please LOOK at the file you upload and make sure it is right!. If you mess something up and we have to run your code manually, you will receive **at most half the points**.*

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini3**, which in turn should contain the four required files. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip.