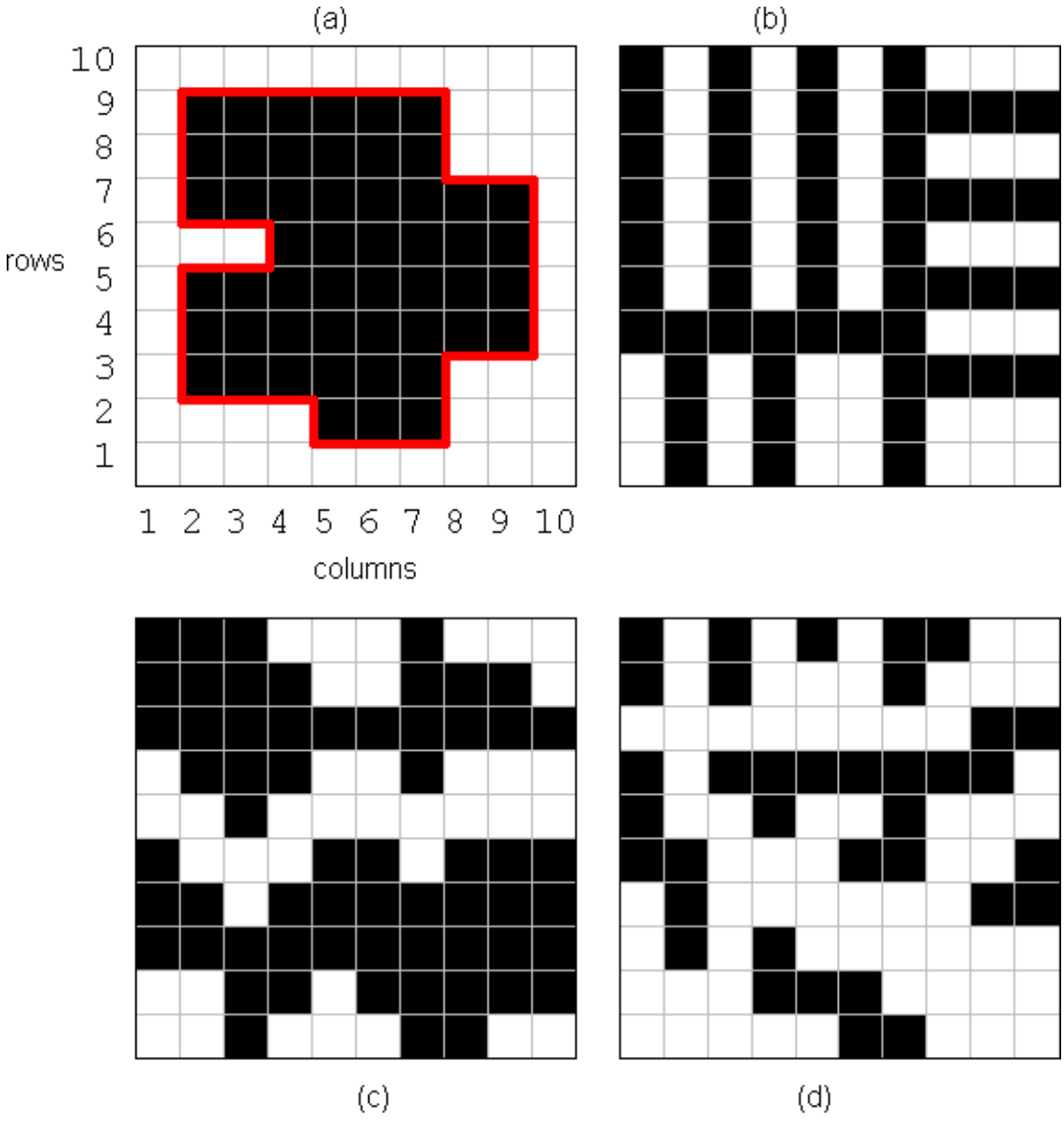
COMP1406 - Assignment #6

(Due: **Fri. Apr 24th @ 11:59pm**)

In this assignment, you will practice using recursion, as well as file operations. Submit a single ZIP file called assignment6.zip containing your IntelliJ project. This assignment has 50 marks – see the marking scheme posted on cuLearn for details.

Assume that we have an assembly line that can take a picture of a machine part which moves along a conveyor belt. The picture (or image) is represented as a 2D grid of pixels which are either black or white. The pixels can be accessed by specifying the row and column of the pixel where rows and columns are specified by an integer value. The machine examines the images and attempts to determine whether or not the parts are *broken*. A **broken** part will appear as a set of black pixels which are not all connected together (i.e., there is a separation between one or more sets of black pixel groups. Here are some examples of four possible images. Note that (c) and (d) represent images of ***broken*** parts. The red border represents the perimeter of a part composed of black pixels.



(1) The **PartImage** class

Download the provided A6-Base-Code.zip file and extract the included IntelliJ project. This project contains a PartImage class, which represents one of these images using a 2-dimensional array of **booleans**.

You must complete the following methods in the PartImage class. Note that all the recursive methods may be implemented in a 'destructive' sense. That is, you can change the state of the object within your recursive implementation and do not need to recover to the original starting state.

- Complete the static **readFromFile(String fileName)** method that will open the specified file, which will contain the data for a single part's image. This method must read the part image data from the file and return a new PartImage object with the correct attributes to represent that part image. The data in the file will be organized like a 2D array, with each line representing a row of the part's image data. Each row will contain some number of comma-separated 0s and 1s, indicating a white or a black space respectively. For example, the data contained in a file representing part (a) from above might look like:

```
0,0,0,0,0,0,0,0,0,0
0,1,1,1,1,1,1,0,0,0
0,1,1,1,1,1,1,0,0,0
0,0,0,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,0,0,0
0,0,0,0,1,1,1,0,0,0
0,0,0,0,0,0,0,0,0,0
```

  Note that you cannot assume that you know how wide each row will be or how many rows are present. You will have to compute this in order to initialize the size of any 2D boolean arrays your class uses. Three possible exceptions could arise within this method:
  1. FileNotFoundException – if the specified file is not found, your method's code should handle the exception gracefully and return a null object.
  2. IOException – if there is another type of input error, your method's code should handle the exception gracefully and return a null object.
  3. InvalidPartImageException – if any line in the file does not contain the same number of values as all other lines (e.g., 9 columns in one line and 10 in another), or one of the values is not 0/1, your method must throw a new InvalidPartImageException with the current filename. This exception will be handled by the provided test code.
- Complete the **print()** method that will print out the image in the console window which may print something like what is shown below for the image in picture (a) above.

```
----------
-******---
-******---
-********-
---******-
-********-
-********-
-******---
----***---
----------
```

- Complete the **findStart()** method that returns the location (row/column) of any black pixel in the image. It uses the **Point2D** as the return value. If no black pixel is found, the method should return **null**.

- Complete the **partSize()** method so that it returns an **int** representing the number of black pixels in the image. This can be done using a double FOR loop.

- Complete the method called **expandFrom(int r, int c)** that recursively examines the image by traversing its pixels, examining (i.e., "visiting") black pixels in all directions that are connected to a starting point black pixel (specified by the parameters). The idea is that this method will determine all black pixels that are connected to the starting pixel. Additionally, the method should set the black pixels to white *(hints: when a white pixel is encountered during the recursion, this is an indication that you don't need to keep recursively checking in that direction)*. This method **MUST be done recursively without loops, otherwise you will receive 0 marks for this part**.

- Complete the method called **perimeterOf(int r, int c)** which should return an integer representing the perimeter of the part which starts at the given (r,c) point. The method **MUST be done recursively without loops, otherwise you will receive 0 marks for this part**. For example, in the image (A) shown above, the perimeter is **36** (which is the red border length, not the number of black cells on the border) and in image (B) it is **106**. As for the other two images, it depends where you started expanding from, since the piece is broken into multiple parts. *(hints: You will need to mark pixels as being visited (perhaps use another 2D array). Make sure not to re-visit these pixels. Also, make sure to handle borders correctly. A black pixel on the top/left border corner, for example, will add 2 to the perimeter count for the top and the left.)*

- Complete the method called **countPieces()** which should return an integer representing the number of separate pieces contained in the part image. For example, part (a) above contains 1 piece, where part (d) contains 8 pieces.

Once you have implemented these, you can use the PartImageTester class to test your solutions.