COMP 1006/1406 – Winter 2022

Submit your `MyBlobs.java` file to Brightspace.

This assignment has 10 marks.

Images, Pixels and Blobs

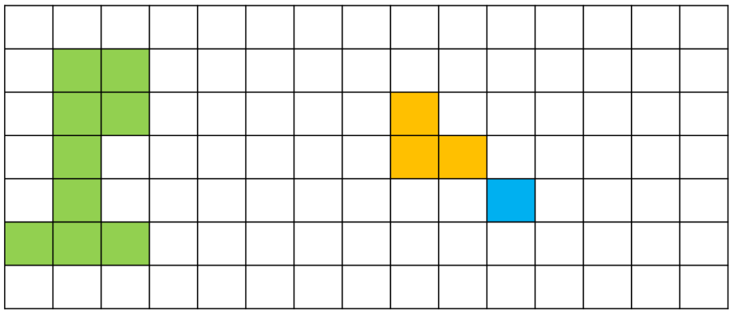
An `Image` is a 2-dimensional grid of `Pixel`s. A pixel's location in the image is specified by its row and column in the image and either has ink (i.e., you can see it) or does not.

A ***blob*** of ink is one more pixels that have ink and are neighbours in the image. Two pixels at positions (r_1, c_1) and (r_2, c_2), respectively, are **immediate neighbours** if one of the following is satisfied:

- $r_1 = r_2$ and $c_1 = c_2 \pm 1$
- $r_1 = r_2 \pm 1$ and $c_1 = c_2$

Here r_i is the row of pixel i and c_i is the column of pixel i. Notice that you can move from one pixel to one of its immediate neighbours by either moving exactly one position up, down, left or right in the image grid. Diagonals are not allowed for immediate neighbours. Two pixels are **neighbours** if there is a sequence of immediate neighbours connecting them.

For example, in the image below, there are THREE (3) blobs. The blob on the left is made up of 9 pixels (coloured green), the middle blob as 3 pixels (coloured yellowish), and the last blob, furthest to the right, consists of a single pixel (coloured blue).



In this assignment, you will implement several algorithms that explore an image to identify all the pixels in a given blob of ink. The needed methods are defined as abstract in the provided `Blobs.java` file. You will create a concrete class called `MyBlobs.java` that overrides these methods with a concrete implementation. A skeleton version of `MyBlobs.java` is provided with the other needed classed (`Blobs.java`, `Image.java`, and `Pixel.java`).

Note: in this assignment, the pixels in a given blob will be stored in a `Deque` (that stores `Pixel`s) instead of a `List`. Recall that the Deque ADT (double-ended queue) is a restricted list that has fast access (adding/removing) from both the front and back of the list. For this assignment, you should NOT need to use any other methods than the `addLast()`, `addFirst()`, `removeFirst()` and `size()` methods of the `ArrayDeque` you will use. The `ArrayDeque` is a concrete class that implements the `Deque` interface. The provided java files have already imported the needed classes/interfaces.

1 Recursive Approach [5 marks]

You will use recursion to find all pixels in a given blob. In particular, you will implement a method called `blobRecursiveHelper()` that uses accumulative recursion.

Recall that in accumulative recursion, you have an extra input parameter that builds up the solution to the problem. When the recursion ends, the final answer is contained in that parameter.

```
blobRecursiveHelper(int row, int col, Deque<Pixel> blobSoFar)
```

In this method, a list of pixels that you have already determined to be in the blob is passed as input (`blobSoFar`). If the current pixel (specified by the input row and column) has not yet been visited and has ink, you add this pixel to `blobSoFar` and then recursively visit all four of this pixel's neighbours. If the pixel has been visited before, then you do nothing (just `return`). You'll need to take care that your code doesn't crash by trying to process pixels that are outside of the image. You'll know if there is a problem if your code crashes with an `ArrayIndexOutOfBoundsException`.

The provided `blobRecursive(int row, int col)` method will call your helper method. It returns a `Deque` of all pixels in the blob that contains the pixel at the initial specified row and column. If the initial pixel does not have any ink then the list should be empty.

As you discover a new pixel that hasn't been visited yet, be sure to add it to the *back* of the list `blobSoFar`. This will ensure that the ordering of the pixels will match their discovery ordering. Marking will **depend** on this ordering being correct.

When making recursive calls to immediate neighbours, always visit them in the following order: up, right, down, left. Note that up means *decreasing* the row value, and left means *decreasing* the column value. As always, row and column values start with value 0.

2 Iterative Approach [10 marks]

A problem with the recursive approach to solving this problem is that you will run out of stack space if the blob is too large (and your program will crash).

Instead of using recursion, we can also discover all pixels in a blob using iteration. Here is pseudocode for the basic algorithm:

```
 1  blobIterative(row,col) -> List of Pixels
 2     blobList <- empty List
 3     workingList <- empty List
 4     add pixel at (row,col) to workingList
 5     while workingList is not empty do
 6        p <- some pixel removed from workingList
 7        if p has ink AND has not been visited then
 8           mark p as visited
 9           add p to back of blobList
10           for each immediate neighbour q of p do
11              add q to workingList
12     output blobList
```

The order of the pixels in the output list will depend on the data structure used for the `workingList` list and implementation details.

If you use a `Stack` (recall the Stack ADT) and the order you select immediate neighbours (in line 10 of the pseudocode) is OPPOSITE to the order in which you make the recursive calls to immediate neighbours, then the output list will be in the same order as your recursive implementation. This is a *depth-first* approach to discovering the pixels.

If you use a `Queue` (recall the Queue ADT), then the order of the output list will start with the initial pixel and then move outwards like the ripple of a wave. This is a *breadth-first* approach to discovering the pixels.

You will override the provided (abstract) `blobIterative()` method. For your implementation, you can use either the Stack or Queue approach (you will still use the `ArrayDeque` to simulate either). Regardless of your choice, you must add immediate neighbours (lines 10-11) in the following order: left, down, right, up. Note that up means *decreasing* the row value, and left means *decreasing* the column value. As always, row and column values start with value 0.

Submission Recap

Submit your `MyBlobs.java` file. Do not submit any other file.