Homework 6: Manipulating Images with Java, in Parallel

Due Thursday, March 6 at 11:30pm

Turn in your homework via the course web page as an updated version of the `hw6.java` text file that I have provided.

Make sure the file can be successfully compiled with `javac`. There should be *no compilation errors or warnings*; if not you get an automatic 0 for the homework!

Recall the CS131 Academic Honesty Policy! You must list whom you discussed the assignment with at the top of your assignment, and also what other resources you used.

For this assignment we will revisit the image transformations from Homework 4, but this time we will use fork-join parallelism in Java to exploit multicore hardware for better performance.

Java's fork-join framework is available in Java version 7, which is the latest version. It is installed on the SEAS machines in `/usr/local/cs/bin`, and you can download it for your platform here.

As usual, good style matters. **Here are the rules:**

- **Do not modify the names of any types, classes, instance variables, and methods that our code defines in `hw6.java`. We are relying on them for testing purposes; your code will get no credit in cases where the test script can't find what it's looking for.**
- *Never* use type-unsafe features of Java, like casts and the `instanceof` expression. Similarly, never build your own version of `instanceof`, such as a method that returns `true` if an object has a particular class. If you ever need to figure out the class of some object, then your design is not as object-oriented as it should be.
- You may use any number of helper classes and methods that you require.

A few other tips:

- You will find the Java API Documentation useful. There you can find information on the various classes and interfaces in Java's fork-join framework. You may also find the `Math` class to provide useful functionality.

- Write comments where useful to tell the reader what's going on.
- Test your functions on several inputs, including corner cases -- we will be doing the same.

Now on to the assignment! The file `hw6.java` includes four class declarations:

- `ImplementMe` is an exception that is used to mark places in the code that are currently unimplemented and need something from you.
- `RGB` objects each represent an RGB triple. We're using such objects essentially just as tuples (which Java lacks), so the fields are public for easy access.
- `Gaussian` contains a method `gaussianFilter` that produces a Gaussian filter for a given radius and sigma. The method is declared `static`, which indicates that the method does not have a `Gaussian` object as its implicit first argument. Instead the method is invoked as follows: `Gaussian.gaussianFilter(3, 2.0)` In essence, static methods are Java's way of implementing ordinary procedures like you'd write in C.
- `PPMImage` represents a PPM image and plays the role that our dictionaries did in the Python version. This time I've provided the functionality for parsing and unparsing files (methods `fromFile` and `toFile`). Your job is to implement the functions `negate`, `mirrorImage`, and `gaussianBlur`.

Here are the implementation requirements for your three methods:

1. The methods should each use Java's fork-join framework in order to exploit parallelism. The operations are naturally parallelizable since each pixel can be transformed in isolation. You should experiment with different ways of partitioning the problem and different sequential thresholds. For the Gaussian blur in particular, which has non-trivial computation per pixel (especially for large radius values, e.g. 60), you should be able to achieve significant speedups versus a sequential implementation when running on multicore hardware.
2. The methods should be *side-effect-free*, returning a new `PPMImage` object and leaving the old one unchanged.
3. The methods should have the same behavior as their counterparts from Homework 4 with one exception. Since now we will want to run the Gaussian blur with large radius values, our earlier behavior of not blurring pixels on the edges leads to odd-looking pictures. Instead, you should use a *clamping* semantics.

   In particular, every pixel should be blurred, and each blurred pixel should incorporate each element of the Gaussian filter. However, whenever this process requires accessing a pixel whose row (column) is out of bounds, that row (column) should be substituted with the closest row (column) that is in bounds. For example, if the pixel at row -1 and column 1 is required, you'd instead use the pixel at row 0 and column 1, and if the pixel at row -1 and column -2 is required, you'd instead use the pixel at row 0 and column 0. Similarly, if the

pixel at row 100 is required but the image has height 96, then you'd instead use the row value 95.