Working with Images in Java

by Julian Robichaux, panagenda
originally published on socialbizug.org, January 2014

Java gives us a lot of options for working with images. In general, you are almost always using the ImageIO class to read or write image files, and working with an Image or a BufferedImage object when dealing with the object in memory. For the basics on how all this works, I will refer you to the Java Tutorials on oracle.com:

http://docs.oracle.com/javase/tutorial/2d/images

Beyond the basics though, there are a number of image-related things that you might want to do. A few examples are listed below in this article.

Basic Image Manipulation

Once you have your BufferedImage in-hand, Java has several native classes available for manipulating the image. These are all implemented using the BufferedImageOp interface (http://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImageOp.html). They are:

- AffineTransformOp: resize, rotate, shear, create a mirror image
- ColorConvertOp: convert to greyscale
- ConvolveOp: sharpen, blur, edge-detect
- LookupOp: invert individual colors, brighten/darken individual colors
- RescaleOp: brighten, darken, tint (does NOT resize, contrary to what the name implies)

Here is an example of converting an image to greyscale:

```
ColorConvertOp op = new ColorConvertOp(
     ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
BufferedImage newBuffImg = op.filter(oldBuffImg, null);
```

There are also some nice explanations of the various operations here: http://www.informit.com/articles/article.aspx?p=1013851

Resizing an Image

While you can use an AffineTransform to resize an image, you will not necessarily get the best quality result when using that method. A good use-case is when you want users to be able to upload images that will be resized very small to be used as thumbnail pictures or avatars. Bad quality resizing — especially going from very large to very small pictures — leads to ugly avatars. No one wants an ugly avatar!

A few years ago there was an excellent article written by Chris Campbell (one of the programmers at Sun who wrote Java graphics processing code at the time) discussing image resizing techniques and their various perils and pitfalls. It's worth reading the whole thing at https://today.java.net/pub/a/today/2007/04/03/perils-of-image-getscaledinstance.html

The gist of the article was that it is best to enlarge or reduce an image by 50% as many times as you can when doing a resize, only using a non-50% scaling operation at the very end. This gave superior quality to resizing an image in a single operation. The resulting code looks something like this:

```
do {
      if (w > targetWidth) {
            w = (int)Math.max(w / 2, targetWidth);
            h = (int)Math.max(h / 2, targetHeight);
      } else {
            w = (int)Math.min(w * 1.5, targetWidth);
            h = (int)Math.min(h * 1.5, targetHeight);
      }

      BufferedImage tmp = new BufferedImage(w, h, type);
      Graphics2D g2 = tmp.createGraphics();
      g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
            RenderingHints.VALUE_INTERPOLATION_BICUBIC);
      g2.drawImage(ret, 0, 0, w, h, null);
      g2.dispose();
      ret = tmp;
} while (w != targetWidth || h != targetHeight);
```

There is also a consideration of which type of resizing algorithm to use (bicubic, nearest neighbor, etc.), but I'm not really qualified to weigh in on that discussion. A good place to start if you're interested is http://en.wikipedia.org/wiki/Image_scaling . Personally the bicubic algorithm has always been good to me, especially with JPG images.

PDF Thumbnails

Speaking of thumbnail images, people sometimes want to generate a thumbnail of the first page of a document. I've used the IcePDF library (http://www.icesoft.org/java/downloads/icepdf-downloads.jsf) to do this before, although there are other libraries like Apache PDFBox (http://pdfbox.apache.org) that also have this functionality.

The basic code for using IcePDF to generate a thumbnail is:

```
      org.icepdf.core.pobjects.Document document =
            new org.icepdf.core.pobjects.Document();
      document.setFile(filePath + fileName);
      BufferedImage image = (BufferedImage)
                  document.getPageImage(0,
                              GraphicsRenderingHints.SCREEN,
                              Page.BOUNDARY_CROPBOX, 0f, 1f);
      File file = new File(filePath + "img_" + fileName + ".png");
      ImageIO.write(image, "png", file);
      image.flush();
      document.dispose();
```

There are several supporting JAR files that come with IcePDF, but I think the only one you need for PDF-to-image conversion is icepdf-core.jar. That's the only one I used anyway. If you are starting with a Word/Excel/Powerpoint type file you can use Apache POI (http://poi.apache.org) to convert the document to a PDF first (please search Google for examples).

Extracting EXIF Data

If you are starting with a JPG file that a user uploaded from a cellphone, it is sometimes useful to extract the EXIF data from the image in order to determine the time/date of the picture, or possibly even the GPS coordinates. A nice library for doing that is metadata-extractor (https://drewnoakes.com/code/exif).

You can read the metadata with code like this:

```
File jpegFile = new File("myImage.jpg");
Metadata metadata = ImageMetadataReader.readMetadata(jpegFile);
for (Directory directory : metadata.getDirectories()) {
    for (Tag tag : directory.getTags()) {
        System.out.println(tag);
    }
}
```

See also: https://code.google.com/p/metadata-extractor/wiki/FAQ

Keep in mind that users can disable GPS data on pictures, and that image manipulation programs often strip EXIF data from pictures too. However, if your users are well-trained and you want to give them an easy way to map locations with photos (like at a construction site or a facility visit), this might be a good answer for you.

Barcodes and QRCodes

Another fun (or at least useful) thing to do is to generate barcodes and QRCodes. A good library to use for this purpose is zxing (https://github.com/zxing/zxing). This library is small, mature, and has support for many different barcode types. Here is an example of creating a Code 128 barcode and a QRCode using zxing:

```
// barcode
int width = 200;
int height = 40;
Code128Writer barcodeWriter = new Code128Writer();
BitMatrix barbits = barcodeWriter.encode("I am a barcode",
     BarcodeFormat.CODE_128, width, height, null);
FileOutputStream barcodeOut = new FileOutputStream(
     new File("c:\\temp\\testBarcode.png"));
MatrixToImageWriter.writeToStream(barbits, "png", barcodeOut);
barcodeOut.close();

// QRCode (make sure you force ISO-8859-1 encoding)
height = width;
String qrstring = "I am a QRCode";
String charset = "ISO-8859-1";
```

```
String qrenc = new String(qrstring.getBytes(charset), charset);
Hashtable hints = new Hashtable();
hints.put(EncodeHintType.CHARACTER_SET, charset);

MultiFormatWriter qrWriter = new MultiFormatWriter();
BitMatrix qbits = qrWriter.encode(qrenc,
     BarcodeFormat.QR_CODE, width, height, hints);
FileOutputStream qrOut = new FileOutputStream(
     new File("c:\\temp\\testQrcode.png"));
MatrixToImageWriter.writeToStream(qbits, "png", qrOut);
qrOut.close();
```

I haven't tried the newer versions of zxing in IBM Notes yet, but if you find that you need an older version, you can still download version 1.7 at http://zxing.googlecode.com/files/ZXing-1.7.zip

Also, zxing does NOT come as a pre-compiled JAR like many other Java libraries do. You'll have to use Maven or (for the older versions) Ant in order to build your own JAR files. It's really not that hard to do and it's a good thing for an aspiring Java programmer to figure out, so I will leave that as an exercise for the reader. However, I will tell you that you only need to build and use the "core" and "javase" JARs to run the example code above.