# Android Image Processing Made Easy

Handling images in Android can be tricky—bitmaps eat up memory, rotations mess with orientation, and cropping requires precision. That's where utility classes like `BitmapUtils` and `Crop` come in handy. In this post, I'll walk you through a powerful pair of classes from the `com.lzw.flower.utils` package. We'll explore their code, break down what each method does, and show how to use them in your projects. Let's dive in!

---

**BitmapUtils: Your Go-To Image Manipulation Toolkit** The `BitmapUtils` class is a collection of static methods for manipulating `Bitmap` objects. Here's the full code, followed by a breakdown:

```
package com.lzw.flower.utils;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.net.Uri;
import android.provider.MediaStore;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class BitmapUtils {
  public static Bitmap convertGreyImg(Bitmap img) {
    int width = img.getWidth();
    int height = img.getHeight();
    int[] pixels = new int[width * height];
    img.getPixels(pixels, 0, width, 0, 0, width, height);
    int alpha = 0xFF << 24;
    for (int i = 0; i < height; i++) {
      for (int j = 0; j < width; j++) {
        int grey = pixels[width * i + j];
        int red = ((grey & 0x00FF0000) >> 16);
        int green = ((grey & 0x0000FF00) >> 8);
        int blue = (grey & 0x000000FF);
        grey = (int) ((float) red * 0.3 + (float) green * 0.59 + (float) blue * 0.11);
        grey = alpha | (grey << 16) | (grey << 8) | grey;
```

```
      pixels[width * i + j] = grey;
    }
  }
  Bitmap result = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
  result.setPixels(pixels, 0, width, 0, 0, width, height);
  return result;
}

public static Bitmap toGreyImg(Bitmap bitmapOrg) {
  Bitmap bitmapNew = bitmapOrg.copy(Bitmap.Config.ARGB_8888, true);
  if (bitmapNew == null) {
    return null;
  }
  for (int i = 0; i < bitmapNew.getWidth(); i++) {
    for (int j = 0; j < bitmapNew.getHeight(); j++) {
      int col = bitmapNew.getPixel(i, j);
      int alpha = col & 0xFF000000;
      int red = (col & 0x00FF0000) >> 16;
      int green = (col & 0x0000FF00) >> 8;
      int blue = (col & 0x000000FF);
      int gray = (int) ((float) red * 0.3 + (float) green * 0.59 + (float) blue * 0.11);
      int newColor = alpha | (gray << 16) | (gray << 8) | gray;
      bitmapNew.setPixel(i, j, newColor);
    }
  }
  return bitmapNew;
}

public static void saveBitmapToPath(Bitmap bitmap, String imagePath) {
  FileOutputStream out = null;
  File file = new File(imagePath);
  if (file.getParentFile().exists() == false) {
    file.getParentFile().mkdirs();
  }
  try {
    out = new FileOutputStream(imagePath);
    bitmap.compress(Bitmap.CompressFormat.PNG, 100, out);
    out.flush();
  } catch (Exception e) {
    e.printStackTrace();
```

```
    } finally {
      try {
        if (out != null) out.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
}

public static Bitmap rotateBitmap(Bitmap source, float angle) {
    Matrix matrix = new Matrix();
    matrix.postRotate(angle);
    return Bitmap.createBitmap(source, 0, 0, source.getWidth(), source.getHeight(), matrix, true);
}

public static Uri getResourceUri(int resId) {
    return Uri.parse("android.resource://com.lzw.flower/" + resId);
}

public static Bitmap getBitmapByUri(Context cxt, Uri uri) throws IOException {
    return MediaStore.Images.Media.getBitmap(cxt.getContentResolver(), uri);
}

public static int calInSampleSize(BitmapFactory.Options options, int reqWidth) {
    int w = options.outWidth;
    int h = options.outHeight;
    int inSampleSize = 1;
    if (w > reqWidth && reqWidth > 0) {
      inSampleSize = Math.round(w / reqWidth);
    }
    return inSampleSize;
}

public static Bitmap decodeSampledBitmapFromPath(String path, int reqWidth) {
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);
    int inSampleSize = calInSampleSize(options, reqWidth);
    options.inJustDecodeBounds = false;
    options.inSampleSize = inSampleSize;
```

```
    return BitmapFactory.decodeFile(path, options);
  }

  public static Bitmap decodeFileByHeight(String path, int reqH) {
    BitmapFactory.Options opt = new BitmapFactory.Options();
    opt.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, opt);
    int scale = calInSampleSizeByHeight(opt, reqH);
    opt.inSampleSize = scale;
    opt.inJustDecodeBounds = false;
    Bitmap bm = BitmapFactory.decodeFile(path, opt);
    return bm;
  }

  public static int calInSampleSizeByHeight(BitmapFactory.Options options, int reqHeight) {
    int h = options.outHeight;
    int inSampleSize = 1;
    if (h > reqHeight) {
      inSampleSize = Math.round(h * 1.0f / reqHeight);
    }
    return inSampleSize;
  }
}
```

**What's Inside?**

- **Grayscale Conversion**:
  - `convertGreyImg`: Uses a pixel array to batch-process the bitmap into grayscale.
  - `toGreyImg`: Works pixel-by-pixel on a mutable copy, offering an alternative approach. Both use the luminosity formula (`0.3R + 0.59G + 0.11B`) for natural grayscale.
- **File Operations**:
  - `saveBitmapToPath`: Saves a bitmap as a PNG, creating directories as needed.
- **Transformations**:
  - `rotateBitmap`: Rotates an image using a `Matrix`—simple but effective.
- **Loading and Sampling**:
  - `getBitmapByUri` and `getResourceUri`: Load images from URIs or resources.
  - `decodeSampledBitmapFromPath` and `decodeFileByHeight`: Efficiently scale large images by width or height, avoiding memory issues.

**Crop: Precision Cropping with Android's Native Tools** The `Crop` class leverages Android's built-in cropping intent. Here's the code:

```
package com.lzw.flower.utils;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.provider.MediaStore;
import com.lzw.flower.base.App;

import java.io.File;

public class Crop {
  public static void startPhotoCrop(Activity cxt, Uri uri, String outputPath, int resultCode) {
    Intent intent = new Intent("com.android.camera.action.CROP");
    intent.setDataAndType(uri, "image/*");
    int w = App.drawWidth;
    int h = App.drawHeight;
    int factor = gcd(w, h);
    int w1 = w / factor;
    int h1 = h / factor;
    intent.putExtra("crop", "true")
        .putExtra("aspectX", w1)
        .putExtra("aspectY", h1)
        .putExtra("scale", true)
        .putExtra("outputX", w)
        .putExtra("outputY", h)
        .putExtra("outputFormat", Bitmap.CompressFormat.PNG.toString());
    intent.putExtra("noFaceDetection", true);
    intent.putExtra("return-data", false);
    Uri uri1 = Uri.fromFile(new File(outputPath));
    intent.putExtra(MediaStore.EXTRA_OUTPUT, uri1);
    cxt.startActivityForResult(intent, resultCode);
  }

  static int gcd(int a, int b) {
```

```
    if (b == 0) {
      return a;
    } else {
      return gcd(b, a % b);
    }
  }
}
```

**What's Happening Here?**

- `startPhotoCrop`: Launches the system crop activity with a specified `Uri`, aspect ratio (simplified using GCD), and output path. It assumes `App.drawWidth` and `App.drawHeight` are defined elsewhere (e.g., in a base `App` class).
- `gcd`: A recursive method to compute the greatest common divisor, ensuring the aspect ratio is in its simplest form.

---

**Putting It All Together: Usage Examples** Here's how you might use these utilities in an Android app:

```
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import com.lzw.flower.utils.BitmapUtils;
import com.lzw.flower.utils.Crop;

public class MainActivity extends AppCompatActivity {
    private static final int REQUEST_CROP = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Convert an image to grayscale and save it
        Bitmap original = BitmapFactory.decodeResource(getResources(), R.drawable.sample);
        Bitmap grey = BitmapUtils.convertGreyImg(original);
        BitmapUtils.saveBitmapToPath(grey, "/sdcard/DCIM/grey_image.png");
```

```
        // Load and scale an image efficiently
        Bitmap scaled = BitmapUtils.decodeSampledBitmapFromPath("/sdcard/DCIM/photo.jpg", 200);

        // Launch cropping
        Uri imageUri = Uri.fromFile(new File("/sdcard/DCIM/photo.jpg"));
        Crop.startPhotoCrop(this, imageUri, "/sdcard/DCIM/cropped.png", REQUEST_CROP);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_CROP && resultCode == RESULT_OK) {
            // Cropped image is saved at "/sdcard/DCIM/cropped.png"
        }
    }
}
```

**Notes**: - Ensure proper storage permissions in your `AndroidManifest.xml` and runtime checks for file operations. - The `App` class (referenced in `Crop`) should define `drawWidth` and `drawHeight`.

---

### Why These Utilities Rock

1. **Efficiency**: Sampling methods prevent `OutOfMemoryError` when dealing with large images.
2. **Flexibility**: Grayscale, rotation, and cropping cover a wide range of use cases.
3. **Simplicity**: Static methods make integration a breeze—no instantiation required.

---

**Final Thoughts** The `BitmapUtils` and `Crop` classes are a fantastic starting point for any Android app needing image manipulation. Whether you're building a photo editor, optimizing gallery thumbnails, or adding user-driven cropping, this code has you covered. Try it out, tweak it to your needs, and let me know how it works for you!

What image processing challenges have you faced in Android? Share your thoughts below!