```java
package edu.asu.msrs.artcelerationlibrary;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.MemoryFile;
import android.os.Message;
import android.os.Messenger;
import android.os.ParcelFileDescriptor;
import android.os.RemoteException;
import android.util.Log;
import android.widget.Toast;

import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.Buffer;
import java.nio.ByteBuffer;
import java.util.LinkedList;

import static java.security.AccessController.getContext;


public class ArtLib {
    private TransformHandler artlistener;
    private Activity mActivity;
    String TAG = "ArtLib";


    public ArtLib(Activity activity) {
        mActivity = activity;
        init();
    }

//    static {
//        System.loadLibrary("my-native-lib");
//    }


    // To test NEON
    //public native String StringFromJNI();
    //public native String stringFromJNI();

    Context mContext;
    private Messenger mMessenger = null;
    private Messenger mService;
    private boolean mBound;
    String str = "Invalid arguments";


    LinkedList<ReqArgs> mList = new LinkedList<ReqArgs>();
    ReqArgs reqContainer = new ReqArgs();

    ServiceConnection mServiceConnection = new ServiceConnection(){

        @Override
        public void onServiceConnected(ComponentName componentName, IBinder service) {
            mMessenger = new Messenger(service);
            mBound = true;
            Log.v("test","Connected");


            Message msg  = Message.obtain(null, ArtTransformService.COLOR_FILTER);
            msg.replyTo = mReceive;


        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            mMessenger = null;
            mBound = false;
        }
    };

    public void init(){
        mActivity.bindService (new Intent(mActivity, ArtTransformService.class), mServiceConnection, Context.BIND_AUTO_CREATE);
    }

    public String[] getTransformsArray(){
        String[] transforms = {"Color Filter", "Motion Blur", "Sobel Edge", "Gaussian Blur", "ASCII Art"};
        return transforms;
    }

    public TransformTest[] getTestsArray(){
        TransformTest[] transforms = new TransformTest[5];
        transforms[0]=new TransformTest(0, new int[]{26, 26, 30, 80, 100, 150, 170, 230,
                1, 68, 30, 10, 150, 150, 200,
                30, 100, 130, 130, 80, 200, 250, 240, 5}, new float[]{0.1f, 0.2f, 0.3f}); // Color Filter
        transforms[1]=new TransformTest(1, new int[]{1,4}, new float[]{0.3f, 0.2f, 0.3f}); // Motion Blur
        transforms[2]=new TransformTest(2, new int[]{0}, new float[]{0.5f}); //Sobel Edge
        transforms[3]=new TransformTest(3, new int[]{6}, new float[]{3f}); // Gaussian Blur
        transforms[4]=new TransformTest(4, new int[]{51,42,33}, new float[]{0.5f, 0.6f, 0.3f}); //ASCII Art

        return transforms;
    }

    //Function: RegisterHandler to artLib
    //Input: transformHandler
    //Output: null

    public void registerHandler(TransformHandler artlistener){
        this.artlistener=artlistener;

    }

    class ProcessedImgHandler extends Handler{
        @Override
        public void handleMessage(Message msg) {
            Log.d(TAG,"Processed img received: "+ msg.what);

            switch (msg.what){
//              case ArtTransformService.MSG_MULT:
//                  int result = msg.what;
//                  Log.d(TAG,"MULT: "+ result);
                case 10:
                    Bundle retBundle = msg.getData();
                    if (msg.getData() == null){
                        Log.d(TAG,"No data bundle");
                        return;
                    }
                    else {
                        ParcelFileDescriptor pfd_ret = (ParcelFileDescriptor) retBundle.get("pfd_ret");
                        FileInputStream fios_ret = new FileInputStream(pfd_ret.getFileDescriptor());

                        Bitmap procImg = toBitmap(readProcessed(fios_ret));
                        artlistener.onTransformProcessed(procImg);
```

```java
                    //    int result_1 = msg.what;
                    //    Log.d("ArtLib","MULT: "+ result_1);
                }
                break;
            default:
                break;
        }
    }
}

final Messenger mReceive = new Messenger(new ProcessedImgHandler());

// Function: requestTransform to the activity
// Input: Bitmap image
//  Output: Boolean result
public boolean requestTransform(Bitmap img, int index, int[] intArgs, float[]
floatArgs) {

    if (args_verfication(index,intArgs,floatArgs) == false)
        return false;

    else{

        ReqArgs reqArgs = new ReqArgs();
        reqArgs.index = index;
        reqArgs.intArgs = intArgs;
        reqArgs.floatArgs = floatArgs;
        reqArgs.img = img;
        reqArgs.img_height = img.getHeight();
        reqArgs.img_width = img.getWidth();
        mList.add(reqArgs);


        //    Log.d(TAG, "The size is + " + String.valueOf(mList.size()));
        reqContainer = mList.pollFirst();
        ByteBuffer buffer = ByteBuffer.allocateDirect(reqContainer.img.
getByteCount());
        reqContainer.img.copyPixelsToBuffer(buffer);

        byte[] bytes = buffer.array();

        try {
            MemoryFile memFile = new MemoryFile("somename", bytes.length);
            memFile.allowPurging(true); //
            memFile.writeBytes(bytes, 0, 0, bytes.length);

            ParcelFileDescriptor pfd = MemoryFileUtil.getParcelFileDescriptor(
memFile);

```

```
            // int what = ArtTransformService.MSG_MULT;

            Bundle dataBundle = new Bundle();
            dataBundle.putParcelable("pfd", pfd);
            dataBundle.putInt("index", reqContainer.index);
            dataBundle.putInt("width", reqContainer.img_width);
            dataBundle.putInt("height", reqContainer.img_height);
            dataBundle.putIntArray("args1", reqContainer.intArgs);
            dataBundle.putFloatArray("args2", reqContainer.floatArgs);



            Log.d(TAG, "The index is + " + String.valueOf(reqContainer.index));
            Log.d(TAG, "The int args is " + String.valueOf(reqContainer.intArgs[0]));
            int what = reqContainer.index;

            Message msg = Message.obtain(null, what);
            msg.replyTo = mReceive;
            msg.setData(dataBundle);
            memFile.close();


            try {
                if (mMessenger == null)
                    Log.v("test", "null");
                mMessenger.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }catch(IOException e){
            e.printStackTrace();
        }
        return true;
    }
}



//Function: Convert input file stream from the service to buffer
//Input: FileInputStream
//Output: Buffer

public Buffer readProcessed(FileInputStream input)
{

    byte[] byteArray = null;
    Buffer buf = null;
    try
    {
            //InputStream inputStream = new FileInputStream(f);
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            byte[] b = new byte[1024*8];
            int bytesRead =0;

            while ((bytesRead = input.read(b)) != -1)
            {
                bos.write(b, 0, bytesRead);
            }

            byteArray = bos.toByteArray();
            buf = ByteBuffer.wrap(byteArray);



        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return buf;
    }

    //Function: Convert buffer into bitmap
    //Input: Buffer
    //Output: Bitmap object

    public Bitmap toBitmap(Buffer buf){

        Bitmap.Config conf = Bitmap.Config.ARGB_8888;
        Bitmap bmp = Bitmap.createBitmap(reqContainer.img_width, reqContainer.
img_height, conf);

        bmp.copyPixelsFromBuffer(buf);
        return bmp;
    }

    public boolean args_verfication (int index, int[] args1, float[] args2 ){

        switch (index){
            case 0: // Color filter
                if ( args1.length != 24  ){
                    return false;
                } else
                    for(int i = 0; i< args1.length; i++){
                        if (args1[i] < 0 || args1[i] > 255) {
                            return false;
                        }
                        else {}
```

```java

                }
            break;

        case 1:  // Motion blur
            if ( args1[0] != 0 && args1[0] !=1 || args1[1] > 255 || args1[1] < 0){
                return false;
            }  else
                break;

        case 2: // Sobel edge
            if ( args1[0] != 0 && args1[0] !=1 && args1[0] !=2){
                return false;
            } else
                break;

        case 3: // Gaussian blur
            if (args1[0] < 0 || args1[0] > 255 || args2[0] < 0){
                return false;
            } else

                break;
        case 4: // ASCII art
            break;
        }

        return true;

    }



}
```