# EEE 598 Fall 2016 Assignment-2
# Checkpoint-1 Report
# ArtCeleration -
# An Artistic Transformation Library

**Project Members:**
**Omkar Salvi (1209536104)**
**Abhishek Patil (1209417518)**

## Goals

**High Level Objectives:**

- Getting hands on experience of using NDK, NEON in Android application
- Understand and implement library framework and threads
- Implement and communicate with service
- Communication between application, library and service
- Implement different image transform algorithms
- Understand and design non-blocking FIFO queues
- Getting hands on experience of android application development

**Assignment Description:**

ArtCeleration is library framework which is used for converting image into various artistic transforms. The actual transforms will be implemented in a service. The application will send image to library. Library will send the image to service for processing. Library should support multiple requests in non-blocking fashion. That means library should not block the following requests till 1st request is being served completely. Library should allow multiple requests to service. Service may execute these requests in any fashion and send them to library. But library should send the transforms of images back to application in order. I.e. in the same order as the requests were made.

## Design

**Application Components:**

1) **Library :**
    a) **Strategy for queueing requests/responses:**
    We are assigning a unique request id to each request made by application. To implement non-blocking functionality for requests, we have implemented a Queue which will store the request id of each request made by application to the library. When a request is sent to service for processing its request id is pushed into the queue. After service returns the image when processing is finished, we are checking whether the request id associated with this image is at the head of the queue. If it is not at head of queue then it will wait for its turn. When head of the queue is same as waiting processed image's request id, It will be sent back to application. Also element at head of the queue will be popped out. This way we are implementing non-blocking FIFO queue to service multiple requests to application.

    b) **Strategy for accepting requests:**
    In requestTransform function, we are converting bitmap into byte array. This byte array is written to a memory file. Then by using getParcelFileDescriptor method of

MemoryFileUtil we created a parcelable file descriptor for this memory file which has the byte array. Then we created a bundle and put the parcelable file descriptor inside the bundle. Finally we created a Message and put the bundle, bytearray size and request id in it. This message is sent to the service.

c) **Strategy for responding to responses from service:**
The handler class 'ArtLibIncomingHandler' is created inside library, which will handle the result obtained from service. In it's handleMessage method we are processing the message obtained from service. But to provide support for multiple concurrent responses we are implementing the threads. So for each response a thread will be created. Inside this thread we are processing the data obtained from service. First it will wait for its turn as per request id. Then we extract the bundle and get file descriptor from it. From this we read the byte array, convert it into a bitmap and send this bitmap to application via method 'onTransformProcessed' of the 'TransformHandler'.

2) **Service:** Image transforms are performed in an isolated service process. Inside the service to service multiple requests, we are creating a thread for each request. Inside this thread image transform algorithm will be implemented. At this stage we have implemented only 1 class for image transform i.e. 'GaussianBlurTransform.java' . So all the requests to transforms will result in threads executing this class. We are using switch-case structure to select the requested transform functionality. Inside service we created a method onBind which will return the communication channel from the service.

3) **CheckPermission helper in Library** : For API level 23 +, Google has introduced new set of permissions which will ask users to grant permissions to apps while the app is running. With this, user can basically control the applications functionality. These permissions are divided into two categories i.e. normal and dangerous. Normal permissions do not risks user's information and if these permissions are listed in the manifest file then device/system will grant permission automatically. Dangerous permissions can risks user's privacy and these permissions can't be granted without user's invocation.

To take care of this permission check, checkPermission is defined which allow user to have more control over storage permissions. If the API level is greater than application will prompt user to grant permission. If user allows then the media file i.e. processed bitmap is stored in the device storage directory.

**Interfaces between Application Components :**

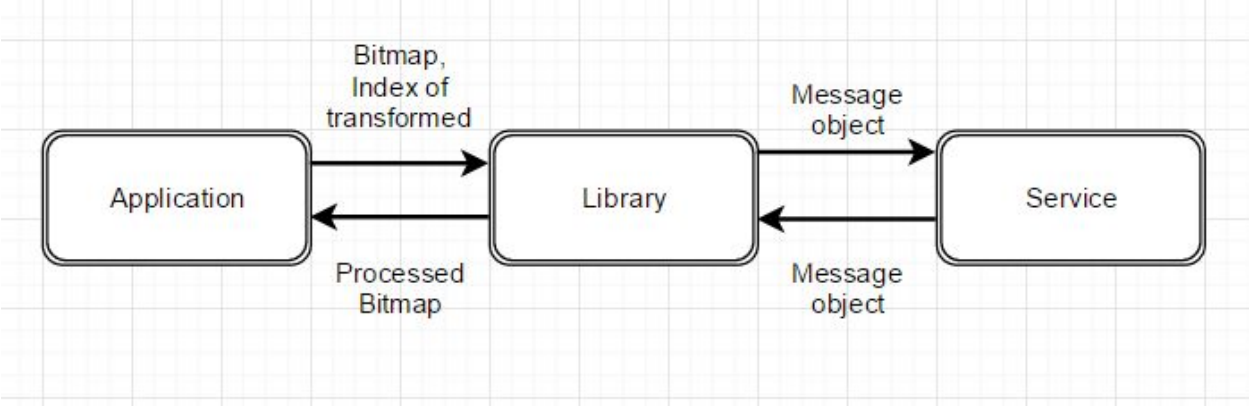


Figure : Block Diagram of complete application designed and Interaction between application components

## Strategy

**Division of labor:**

As the assignment components were completely new to us, we decided to take some time learning about library and service and communication between them. The tutorials given by professor helped a lot in understanding their implementation. Both of us took one major component of the assignment. After this we started working on the internal details of the components. We studied and discussed methods to implement FIFO queue. Then we worked on making the application to handle multiple threads.

**Internal Checkpoints:**

| CheckPoint | Deadline | Team Member |
|---|---|---|
| Library implementation | 10/28/2016 | Omkar |
| Service implementation | 10/28/2016 | Abhishek |

| | | |
|---|---|---|
| FIFO Queue | 11/04/2016 | Omkar |
| Class to implement threads for servicing multiple requests from library to service | 11/04/2016 | Abhishek |
| Class to implement threads for servicing simultaneous responses from service to library | 11/04/2016 | Omkar |
| Comments for all the code | 11/07/2016 | Omkar and Abhishek |
| Documentation | 11/07/2016 | Omkar and Abhishek |

## Challenges Encountered:

1) **Creation of threads both in Library and Service:**
   Thread creation was necessary for handling simultaneous requests and responses between the library and service.

2) **Selection of Data type to implement FIFO queue:**
   There are various ways for implementing first in first out queue in android. Accurate selection of the queue implementation is necessary for non blocking operations.

**How you overcame the challenges:**

1) For creating the threads, we created a class which implements Runnable interface. Then we declared all the parameters required for execution inside the threads, as the member variables of the class. Inside the constructor we set all these parameters. This way we are passing the parameters to thread class. Then in run method of this class, we implemented the functionality expected.

2) First we decided to use timestamp as the deciding factor for FIFO queue. But as there was an unused integer parameter 'arg2' remaining in obtain method of Message class, we decided to use a request id for identifying individual request. This request id was deciding factor for FIFO execution. Thus we finalized to use this integer variable i.e. request id as data structure for queue. We decided to use Concurrentlinkedqueue in android for implementing the queue. A ConcurrentLinkedQueue is an appropriate choice when many threads will share access to a common collection. This implementation employs an efficient non-blocking algorithm.

## Ways to Improve the Assignment:

**Assignment Level:**

Assignment is appropriate level of challenging for the given time period. At the end of checkpoint-1, we found that one of the fundamental aspect of the application is communication between the service and library.

**More breadth-More depth:**

We think this assignment have appropriate breadth and depth, to understand the communication between the client (application user interface), library framework and the service. The second phase of application will be bit challenging as there are number of transforms to be implemented. Synchronization of all threads and the first in first out concurrent linked queue has to be properly implemented. For example, the implementation of location service and modules for the gallery as well as camera activity needed more synchronization with the application.

**Any fundamental flaws system description:**

At the end of phase-1, we feel that the description of goals for phase-1 has given us basic understanding on components of application. However, we feel that little bit broad description of how to synchronize the communication between the service and library could have made it better. This is not actually a flaw but just a secondary improvement in description.

**Enhancements(Design related developer's ideas):**

With the architecture of concurrent linked queue, the communication between the service and library is carried out in proper way. There is no unnecessary load on main classes i.e library framework and implemented service. Hence, errors such as the unfortunately closed application are avoided.

**Privacy/Security/Other Issues:**

- **Some Issues:**

  One of the common security issues with any application on Android is the data processed by application and its storing in the device external storage directory. For bitmaps processed the application uses the media file directory of device. With the internal and external storage permissions, any hacker can access the data stored in the library.

  Also, currently the application uses file descriptor between the library and service. Any open parcel file descriptor will lead to memory leak and the data can be accessed through the memory leak.