Andorid Programming

Week 8

Mina Jung

EECS, Syracuse University

Spring 2017

Part I

Multhi-threading in Android Apps

Outline I

- Android system starts a new Linux process for the application with a single thread of execution
  - By default, all components of the same application run in the same process and thread (called the "main" thread)
  - If an application component starts and there already exists a process for that application, then the component is started within that process and uses the same thread of execution
  - You can arrange for different components in your application to run in separate processes, and you can create additional threads for any process

- manifest entry for each type of component element – <activity>, <service>, <receiver>, and <provider> – supports an android:process attribute

- <application> element also supports an android:process attribute, to set a default value

- main(UI) thread is in charge of dispatching events to the appropriate user interface
  - all components that run in the same process are instantiated in the UI thread
  - when app performs intensive work in response to user interaction, this single thread model can yield poor performance
  - every long-running operation such as database initialization or heavy calculation blocks the UI thread
  - causes the app to seem slow and the response time for input events delayed
    - app results in having serious performance problems
  - Application Not Responding (ANR)

- two rules to Android's single thread model:
  - Do not block the UI thread
  - Do not access the Android UI toolkit from outside the UI thread

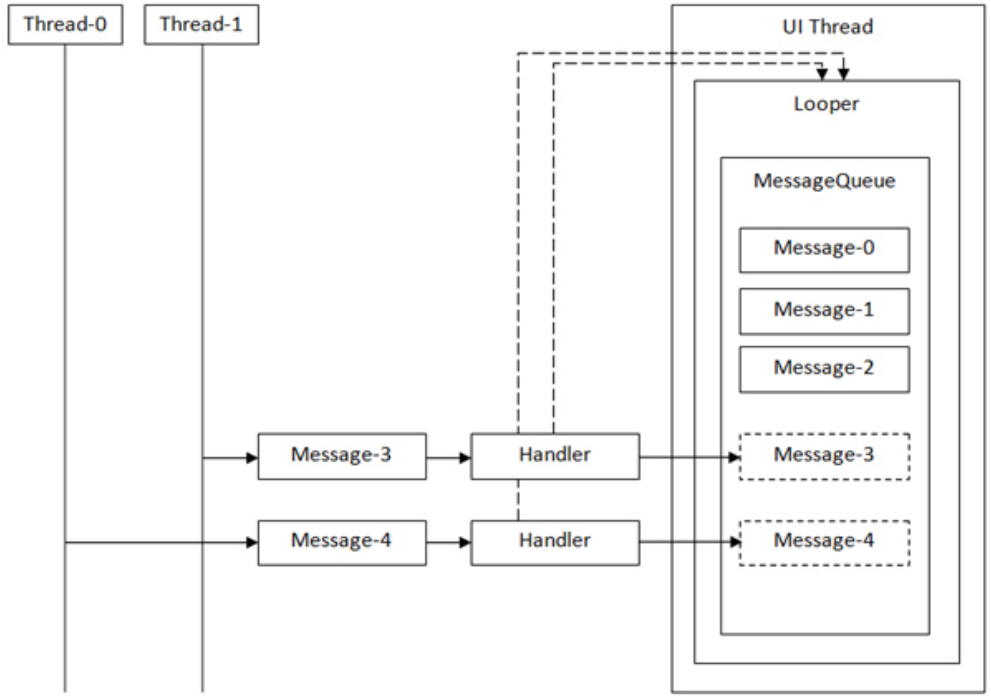
- worker(background) threads
  - cannot update the UI from any thread other than the UI thread or the "main" thread
  - create different kinds of the background workers and perform operations without blocking the main thread
  - operations should be carried out on separate threads
    - Networking
    - Database operations
    - Heavy calculations
    - Object's long initialization

- Android offers several ways to access the UI thread from other threads
  - Activity.runOnUiThread(Runnable)
  - View.post(Runnable)
  - View.postDelayed(Runnable, long)

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // a potentially  time consuming task
            final Bitmap bitmap =
                    processBitMap("image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

- thread-safe : background operation is done from a separate thread while UI thread manipulates Views

- However, as the complexity of the operation grows, this kind of code can get complicated and difficult to maintain. To handle more complex interactions with a worker thread, you might consider using a Handler in your worker thread, to process messages delivered from the UI thread.



- Using AsyncTask
  - perform asynchronous work on user interface
  - perform the blocking operations in a worker thread and then publish the results on the UI thread, without requiring developers to handle threads and/or handlers
  - subclass AsyncTask
    - implement the doInBackground() callback method, which runs in a pool of background threads. To update
    - implement onPostExecute() to update UI, which delivers the result from doInBackground() and runs in the UI thread
    - run the task by calling execute() from the UI thread
- Thread-safe methods

- To provide a good user experience all long running operations in an Android application should run asynchronously
- AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers
- AsyncTask accepts three generic types : AsyncTask<Params, Progress, Result>
  1. Params - the type that is passed into the execute() method
  2. Progress - the type that is used within the task to track progress
  3. Result - the type that is returned by doInBackground()

- For example AsyncTask<String, Void, Bitmap>
  - requires a string input to execute
  - does not record progress
  - returns a Bitmap after the task is complete

- Need to implement methods
  1. onPreExecute - executed in the main thread to do things like create the initial progress bar view
  2. doInBackground - executed in the background thread to do things like network downloads
  3. onProgressUpdate - executed in the main thread when publishProgress is called from doInBackground
  4. onPostExecute - executed in the main thread to do things like set image views

```
// The types specified here are the input data type, the progress
    type, and the result type
private class MyAsyncTask extends AsyncTask<String, Void, Bitmap> {
    protected void onPreExecute() {
        // Runs on the UI thread before doInBackground
        // Good for toggling visibility of a progress indicator
        progressBar.setVisibility(ProgressBar.VISIBLE);
    }

    protected Bitmap doInBackground(String... strings) {
        // Some long-running task like downloading an image.
        Bitmap = downloadImageFromUrl(strings[0]);
        return someBitmap;
    }

    protected void onProgressUpdate(Progress... values) {
       // Executes whenever publishProgress is called from
            doInBackground
       // Used to update the progress indicator
       progressBar.setProgress(values[0]);
    }

    protected void onPostExecute(Bitmap result) {
        // This method is executed in the UIThread
        // with access to the result of the long running task
        imageView.setImageBitmap(result);
```

```
            // Hide the progress bar
            progressBar.setVisibility(ProgressBar.INVISIBLE);
        }
}
```

```
public void onCreate(Bundle b) {
    // ...
    // Initiate the background task
    downloadImageAsync();
}

private void downloadImageAsync() {
     // Now we can execute the long-running task at any time.
     new MyAsyncTask().execute("http://www.example.com/image.jpg");
}
```

- A task can be cancelled at any time by invoking cancel(boolean)
  - isCancelled() to return true
- onCancelled(Object) will be invoked after doInBackground(Object[]) returns
- To ensure that a task is cancelled as quickly as possible, check the return value of isCancelled() periodically from doInBackground(Object[])

- AsyncTask class must be loaded on the UI thread
- The task instance must be created on the UI thread
- execute(Params...) must be invoked on the UI thread
- Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually
- The task can be executed only once (an exception will be thrown if a second execution is attempted)

- Problematic Example

```
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    exampleOne();
  }

  private void exampleOne() {
    new Thread() {
      @Override
      public void run() {
        while (true) {
          SystemClock.sleep(1000);
        }
      }
    }.start();
  }
}
```

- When a configuration change occurs, the entire Activity will be destroyed and re-created

- easy to assume that Android will clean up and reclaim the memory associated with the Activity and its running thread
- never to be reclaimed
- will be a significant reduction in performance

- After each configuration change, the Android system creates a new Activity and leaves the old one behind to be garbage collected
  - the thread holds an implicit reference to the old Activity and prevents it from ever being reclaimed
  - each new Activity is leaked and all resources associated with them are never able to be reclaimed

- Solution 1 : declare the thread as a private static inner class

```
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    exampleTwo();
  }

  private void exampleTwo() {
    new MyThread().start();
  }

  private static class MyThread extends Thread {
    @Override
    public void run() {
      while (true) {
        SystemClock.sleep(1000);
      }
    }
  }
}
```

- new thread no longer holds an implicit reference to the Activity, and the Activity will be eligible for garbage collection after the configuration change

- thread is leaked and never able to be reclaimed
  - Dalvik Virtual Machine (DVM) keeps hard references to all active threads in the runtime system
  - threads that are left running will never be eligible for garbage collection
- MUST implement cancellation policies for background threads!!

- Solution

```
public class MainActivity extends Activity {
  private MyThread mThread;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    exampleThree();
  }

  private void exampleThree() {
    mThread = new MyThread();
    mThread.start();
  }

  /**
   * Static inner classes don't hold implicit references to their
   * enclosing class, so the Activity instance won't be leaked
       across
   * configuration changes.
   */
  private static class MyThread extends Thread {
    private boolean mRunning = false;

    @Override
    public void run() {
      mRunning = true;
```

```
      while (mRunning) {
        SystemClock.sleep(1000);
      }
    }

    public void close() {
      mRunning = false;
    }
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    mThread.close();
  }
}
```

- onDestroy() ensures that you never accidentally leak the thread

- for the same thread across configuration changes (as opposed to closing and re-creating a new thread each time), consider using a retained, UI-less worker fragment to perform the long-running task

- If a static inner class requires a reference to the underlying Activity in order to function properly, wrap the object in a WeakReference to ensure not to accidentally leak the Activity.

- Weak reference objects, which do not prevent their referents from being made finalizable, finalized, and then reclaimed
- implement canonicalizing mappings
- When the garbage collector determines that an object is weakly reachable
  - atomically clear all weak references to that object and all weak references to any other weakly-reachable objects from which that object is reachable through a chain of strong and soft references
  - declare all of the formerly weakly-reachable objects to be finalizable
  - enqueue those newly-cleared weak references that are registered with reference queues
- https://developer.android.com/reference/java/lang/ref/WeakReference.html

- Volley is an HTTP library that makes networking for Android apps easier and faster

- benefits:
  - Automatic scheduling of network requests
  - Multiple concurrent network connections
  - Transparent disk and memory response caching with standard HTTP cache coherence
  - Support for request prioritization
  - Cancellation request API
    - cancel a single request, or set blocks or scopes of requests to cancel
  - Ease of customization, for example, for retry and backoff

  - Strong ordering that makes it easy to correctly populate UI with data fetched asynchronously from the network
  - Debugging and tracing tools

- integrate easily with any protocol and comes out of the box with support for raw strings, images, and JSON
- not suitable for large download or streaming operations, since Volley holds all responses in memory during parsing
- add the following dependency to your app's build.gradle file:

```
dependencies {
    ...
    compile 'com.android.volley:volley:1.0.0'
}
```

- https://developer.android.com/training/volley/index.html

Part II

Examples

Outline I

- AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ... >


    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
         ...>
         ...
        </activity>
    </application>

    <!-- Permission: Allow Connect to Internet -->
    <uses-permission android:name="android.permission.INTERNET" />
/>

</manifest>
```

```
public class DownloadImageProgressDialogActivity extends
     AppCompatActivity {
    // button to show progress dialog
    Button btn;

    // Progress Dialog
    private ProgressDialog pDialog;
    ImageView mImage;
    // Progress dialog type (0 - for Horizontal progress bar)
    public static final int progress_bar_type = 0;

    // File url to download
    private static String file_url =
         "https://news.syr.edu/wp-content/uploads/2016/12/Hendricks-snow1-60

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

        // show progress bar button
        btn = (Button) findViewById(R.id.downloadButton);
        // Image view to show image after downloading
        mImage = (ImageView) findViewById(R.id.downloadedImage);
        /**
         * Show Progress bar click event
```

```
         * */
        btn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                // clear image source
                mImage.setImageDrawable(null);
                // starting new Async Task
                new DownloadFileFromURL(mImage).execute(file_url);
            }
        });
    }

    /**
     * Showing Dialog
     * */
    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case progress_bar_type: // we set this to 0
                pDialog = new ProgressDialog(this);
                pDialog.setMessage("Downloading file. Please
                    wait...");
                pDialog.setIndeterminate(false);
                pDialog.setMax(100);
                pDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```

```
                pDialog.setCancelable(true);
                pDialog.show();
                return pDialog;
            default:
                return null;
        }
    }


    /**
     * Background Async Task to download file
     * */
    final class DownloadFileFromURL extends AsyncTask<String,
         String, Bitmap> {

        private final WeakReference<ImageView>
             imageViewWeakReference;

        public DownloadFileFromURL(final ImageView imageView ){
            imageViewWeakReference = new
                 WeakReference<ImageView>(imageView);
        }
        /**
         * Before starting background thread
         * Show Progress Bar Dialog
         */
```

```
@Override
protected void onPreExecute() {
    super.onPreExecute();
    showDialog(progress_bar_type);
}

/**
 * Downloading file in background thread
 */
@Override
protected Bitmap doInBackground(String... urls) {
    int count;
    for(count = 0; count <= 100; count+=10)
    {
        SystemClock.sleep(1000);
        publishProgress(""+count);
    }

    Bitmap result =
         MovieUtility.downloadImageusingHTTPGetRequest(urls[0]);

    return result;

}
```

```
/**
 * Updating progress bar
 */
@Override
protected void onProgressUpdate(String... progress) {
    // setting progress percentage
    pDialog.setProgress(Integer.parseInt(progress[0]));
}

/**
 * After completing background task
 * Dismiss the progress dialog
 **/
@Override
protected void onPostExecute(final Bitmap result) {
    // dismiss the dialog after the file was downloaded
    dismissDialog(progress_bar_type);

    if(result != null){

        final ImageView imageView =
             imageViewWeakReference.get();

        if(imageView != null){
            imageView.setImageBitmap(result);
        }
```

```
                }
            }
        }
    }
```

```
public class DownloadImageProgressBarActivity extends
     AppCompatActivity {
    Button btn, btn2;

    ImageView mImage;
    ProgressBar pBar;
    TextView pText;


    // File url to download
    private static String file_url =
         "https://news.syr.edu/wp-content/uploads/2016/12/Hendricks-snow1-60

    private static DownloadFileFromURL myDown;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_download_image_progress_bar);

        // show progress bar button
        btn = (Button) findViewById(R.id.downloadButton2);
        btn2 = (Button) findViewById(R.id.cancelButton);

        // Image view to show image after downloading
        mImage = (ImageView) findViewById(R.id.downloadedImage2);
        pBar = (ProgressBar) findViewById(R.id.progressBar);
```

```
pText = (TextView) findViewById(R.id.textProgress);

pText.setText("Ready to Download ....");
/**
 * Show Progress bar click event
 * */
btn.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // clear image source
        mImage.setImageDrawable(null);
        // starting new Async Task
        myDown = new DownloadFileFromURL(mImage, pText,
            pBar);
        myDown.execute(file_url);
    }
});

/**
 * Cancel download
 * */
btn2.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
```

```
                if(myDown != null)
                    myDown.cancel(true);
            }
        });
    }



    /**
     * Background Async Task to download file
     * */
    final static class DownloadFileFromURL extends
        AsyncTask<String, String, Bitmap> {

        private final WeakReference<ImageView>
            imageViewWeakReference;
        private final WeakReference<TextView> textViewWeakReference;
        private final WeakReference<ProgressBar>
            progressBarWeakReference;

        public DownloadFileFromURL(final ImageView imageView, final
            TextView textView, final ProgressBar progressBar ){
            imageViewWeakReference = new
                WeakReference<ImageView>(imageView);
```

```
        textViewWeakReference = new
                WeakReference<TextView>(textView);
        progressBarWeakReference = new
                WeakReference<ProgressBar>(progressBar);
    }
    /**
     * Before starting background thread
     * Show Progress Bar Dialog
     */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    /**
     * Downloading file in background thread
     */
    @Override
    protected Bitmap doInBackground(String... urls) {
        int count;
        for(count = 0; count <= 100; count+=10)
        {
            if(isCancelled()){
                return null;
            }
            SystemClock.sleep(1000);
```

```
            publishProgress(""+count);
        }

        Bitmap result =
            MovieUtility.downloadImageusingHTTPGetRequest(urls[0]);

        return result;
    }


    /**
     * Updating progress bar
     */
    @Override
    protected void onProgressUpdate(String... progress) {
        // setting progress
        final TextView textView = textViewWeakReference.get();
        if(textView != null){
            textView.setText("Progress: " + progress[0]);
        }
        final ProgressBar progressBar =
            progressBarWeakReference.get();
        if(progressBar != null){
            progressBar.setProgress(Integer.parseInt(progress[0]));
        }
    }
```

```
/**
 * After completing background task
 * Dismiss the progress dialog
 **/
@Override
protected void onPostExecute(final Bitmap result) {

    if(result != null){
        final ImageView imageView =
             imageViewWeakReference.get();
        final TextView textView =
             textViewWeakReference.get();
        if(imageView != null){
            imageView.setImageBitmap(result);
        }
        if(textView != null){
            textView.setText("Enjoy the Picture!!");
        }
    }
}

/* Canceled Task */
@Override
protected void onCancelled(final Bitmap result){
    final TextView textView = textViewWeakReference.get();
```

```
                if(textView != null){
                    textView.setText("Task Cancelled!!");
                }
            }
        }
    }
```

```
public class MovieUtility {
/*
 * HTTP GET Request, JSON data returned
 */
    public static String downloadJSONusingHTTPGetRequest(String
        urlString){
        String jsonString = null;

        try{
            URL url = new URL(urlString);

            HttpURLConnection httpConnection = (HttpURLConnection)
                url.openConnection();

            if(httpConnection.getResponseCode() ==
                HttpURLConnection.HTTP_OK){
                InputStream stream =
                    httpConnection.getInputStream();
                jsonString = getStringfromStream(stream);
            }
            httpConnection.disconnect();
        }
        catch(UnknownHostException e1){
            Log.d("DebugMsg", "UnknownHostException in
                downloadJSONusingHTTPGetRequest");
            e1.printStackTrace();
```

```
            }
            catch (Exception ex){
                Log.d("DebugMsg", "Exception in
                    downloadJSONusingHTTPGetRequest");
                ex.printStackTrace();
            }

            return jsonString;
        }

    /*
     * HTTP GET Request, Image(Bitmap) data returned
     */
        private static String getStringfromStream(InputStream stream){
            String line, jsonString = null;

            if(stream != null){
                BufferedReader reader = new BufferedReader(new
                    InputStreamReader(stream));
                StringBuilder out = new StringBuilder();

                try{
                    while( (line = reader.readLine()) != null){
                        out.append(line);
                    }
                    reader.close();
```

```
                jsonString = out.toString();
            }
            catch (IOException ex){
                Log.d("DebugMsg", "IOException in downloadJSON");
                ex.printStackTrace();
            }
        }
        return jsonString;
    }


    public static Bitmap downloadImageusingHTTPGetRequest(String
         urlString){
        Bitmap image = null;

        try{
            URL url = new URL(urlString);

            HttpURLConnection httpConnection = (HttpURLConnection)
                 url.openConnection();

            if(httpConnection.getResponseCode() ==
                 HttpURLConnection.HTTP_OK){
                InputStream stream =
                     httpConnection.getInputStream();
```

```
                image = getImagefromStream(stream);
            }
            httpConnection.disconnect();
        }
        catch(UnknownHostException e1){
            Log.d("DebugMsg", "UnknownHostException in
                downloadImageusingHTTPGetRequest");
            e1.printStackTrace();
        }
        catch (Exception ex){
            Log.d("DebugMsg", "Exception in
                downloadImageusingHTTPGetRequest");
            ex.printStackTrace();
        }

        return image;
    }

    private static Bitmap getImagefromStream(InputStream stream){
        Bitmap image = null;

        if(stream != null){
            image = BitmapFactory.decodeStream(stream);

            try{
                stream.close();
```

```
                }
                catch (IOException ex){
                    Log.d("DebugMsg", "IOException in
                        getImagefromStream");
                    ex.printStackTrace();
                }
            }
            return image;
        }

    /*
     * HTTP POST Request, send JSON data
     */
        public static void sendHttpPostRequest(String urlString,
            JSONObject json){
            HttpURLConnection httpConnection = null;

            try{
                URL url = new URL(urlString);

                httpConnection = (HttpURLConnection)
                    url.openConnection();

                httpConnection.setDoOutput(true);
                httpConnection.setChunkedStreamingMode(0);
```

```
        OutputStreamWriter out = new
            OutputStreamWriter(httpConnection.getOutputStream());
        out.write(json.toString());
        out.close();

        if(httpConnection.getResponseCode() ==
            HttpURLConnection.HTTP_OK){
            InputStream stream =
                httpConnection.getInputStream();
            BufferedReader reader = new BufferedReader( new
                InputStreamReader(stream) );
            String line;

            while( (line=reader.readLine()) != null ){
                Log.d("DebugMsg: PostRequest", line);
            }

            reader.close();
            Log.d("DebugMsg: PostRequest", "POST request
                returns OK");
        }
        else
            Log.d("DebugMsg: PostRequest", "POST request
                returns error");
    }
    catch(Exception ex){
```

```
                Log.d("DebugMsg", "Exception in sendHttpPostRequest");
                ex.printStackTrace();
            }

            if(httpConnection != null)
                httpConnection.disconnect();
        }
    }
```

Part III

Code for RecyclerView of Movie List Using AsyncTask

Outline I

- No More MovieData Class (No hard-coded info and No images under drawable resource)
- Need to Download info and images
- Need new MovieData Class
  - can get JSON type data from DB server using REST api
  - operation in a background thread of AsyncTask

I. create new movieData in onCreate of Fragment

```
@Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

    setHasOptionsMenu(true);

    setRetainInstance(true);

    movieData = new MovieDataJson();
 }
```

II. invoke AsyncTask from onCreateView of Fragment

```
DownloadJsonFromURL myDown = new
     DownloadJsonFromURL( myAdapter );
myDown.execute(MOVIE_SERVER); // ``~/movies/''
```

III. Implement AsyncTask as inner class of Fragment

- download json data in a background thread
- you need a method to convert json object to movie info (HashMap)
- inform RecyclerView (Through WeakReference of Adapter)

```
final static class DownloadJsonFromURL extends
        AsyncTask<String, Void, MovieDataJson> {

    private final WeakReference<MyRecyclerAdapter> adapterReference;


    public DownloadJsonFromURL(MyRecyclerAdapter adapter){
        adapterReference = new WeakReference<MyRecyclerAdapter>(adapter);
    }

    /**
     * Downloading json data in background thread
     */
    @Override
    protected MovieDataJson doInBackground(String... urls) {
        MovieDataJson tData = new MovieDataJson();
        tData.downloadMovieDataJson(urls[0]);
        return tData;
```

```
        }


        /**
         * After completing background task
         **/
        @Override
        protected void onPostExecute(MovieDataJson tData) {
            movieData.moviesList.clear();

            for(int i = 0; i < tData.getSize(); i++){
                movieData.moviesList.add(tData.moviesList.get(i));
            }

            final MyRecyclerAdapter adapter = adapterReference.get();
            if(adapter != null){
                adapter.notifyDataSetChanged();
            }
        }
}
```

- Which class does set ImageView of each movie item?
- RecyclerView Adapter's onBindViewHolder()
  - ImageView's setImageResource of ViewHolder Class should be replaced
- Caching the Images – https://developer.android.com/topic/performance/graphics/cache-bitmap.html
  - Use a Memory Cache!!!

I. create LruCache(LruCache<String, Bitmap> mImgMemoryCache) in onCreate

```
if( mImgMemoryCache == null){
    final int maxMemory = (int)
        (Runtime.getRuntime().maxMemory() / 1024);

    final int cacheSize = maxMemory/8;

    mImgMemoryCache = new LruCache<String,
        Bitmap>(cacheSize){
        @Override
        protected int sizeOf(String key, Bitmap bitmap){
            return bitmap.getByteCount() / 1024;
        }
    };
}
```

II. Check cache first! If not, download an image in a background thread

```
final Bitmap bitmap =
     mImgMemoryCache.get((String)movie.get("url"));

if(bitmap != null){
     movieImage.setImageBitmap(bitmap);
}
else{
     DownloadMovieImage task = new
          DownloadMovieImage(movieImage);
     task.execute((String) movie.get("url"));
}
```

III. After downloading the image, save it to the cache!

```
    @Override
    protected Bitmap doInBackground(String... urls) {

        Bitmap result =
             MovieUtility.downloadImageusingHTTPGetRequest(urls[0])

        if(result != null){
            mImgMemoryCache.put(urls[0], result);
        }
        return result;
    }

@Override
    protected void onPostExecute(final Bitmap result) {

        if(result != null){
            final ImageView imageView =
                 imageViewWeakReference.get();

            if(imageView != null){
                imageView.setImageBitmap(result);
            }
        }
    }
```

1. Invoke AsyncTask when an item is clicked

```
public void onItemClick(View view, int position) {
    ....
    HashMap<String, ?> movie = (HashMap<String,?>)
        movieData.getItem(position);

    String id = (String) movie.get("id");
    String url;

    url = MOVIE_SERVER + "id/" + id;

    DownloadMovieDetail detailTask = new
        DownloadMovieDetail(mListener);
    detailTask.execute(url);

    // mListener.onItemClicked(position);
}
```

2. Implement AsyncTask (Your Job!)

```
final static class DownloadMovieDetail extends
     AsyncTask<String, Void, HashMap> {

    private final WeakReference<OnFragmentInteractionListener>
         listenerReference;

    public DownloadMovieDetail(OnFragmentInteractionListener
         listener){
        listenerReference = new
             WeakReference<OnFragmentInteractionListener>(listener)
    }

    @Override
    protected HashMap doInBackground(String... urls) {
      ...
    }

    @Override
    protected void onPostExecute(HashMap tData) {
      ...
    }
}
```