Software Development for Mobile Devices

Submission for Assignment A10.2C

Introduction

There are many issues in the current application after rigorously analysing. Firstly, the application displays a white screen for a few seconds while loading the file. Secondly, the scrolling is quite slow and more importantly, the application keeps crashing when the list is scrolling quickly. These problems have been identified as related to the thread because the current source code does not handle the thread in Android sufficiently.

This paper aims to optimise the current source code and illustrate the proper way to handle thread in Android application. The paper will commence with the performance optimization. Thereafter, the paper will propose the proper way to improve the usability for this application. Last but not least, the paper will provision insight into the thread in Android for the developer.

Performance Optimization

It is obvious that the performance of the existing application is quite terrible. While scrolling the list view, the memory and CPU utilization keep increasing dramatically has led to the crashing of the application. Based on the data of Android Profiler, the memory rises up to 165MB while the CPU utilization higher than 30%.

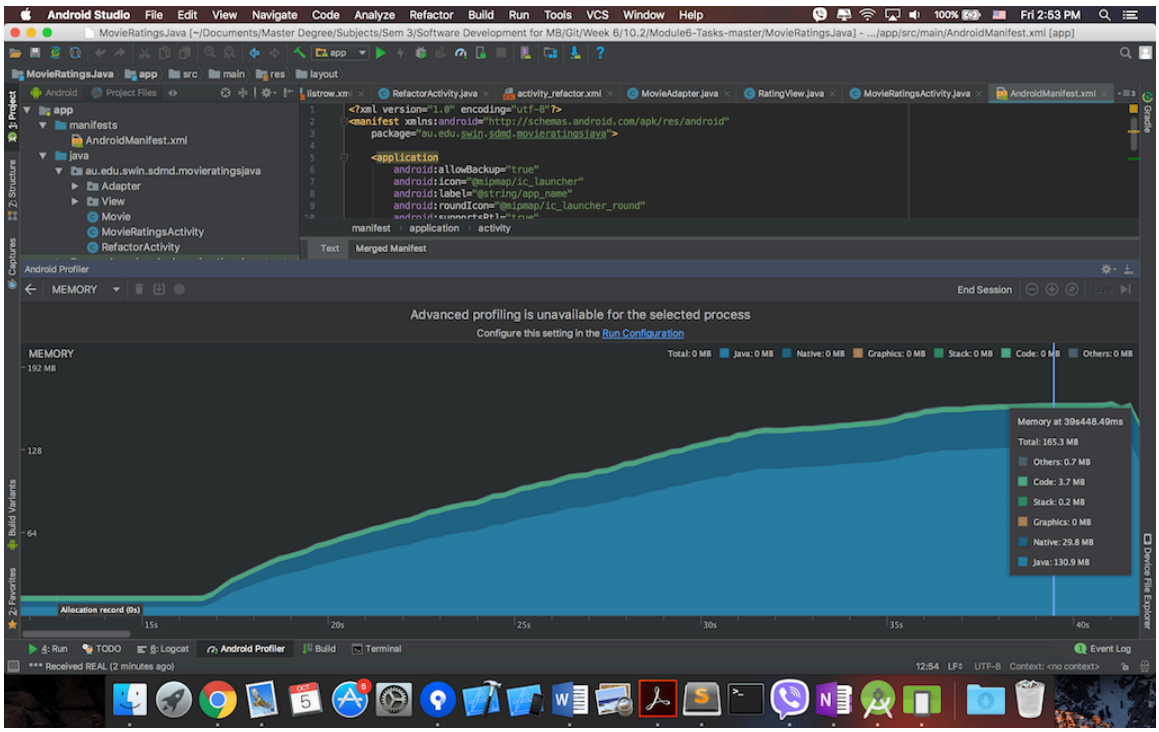

Figure 1: Memory leaked in existing app

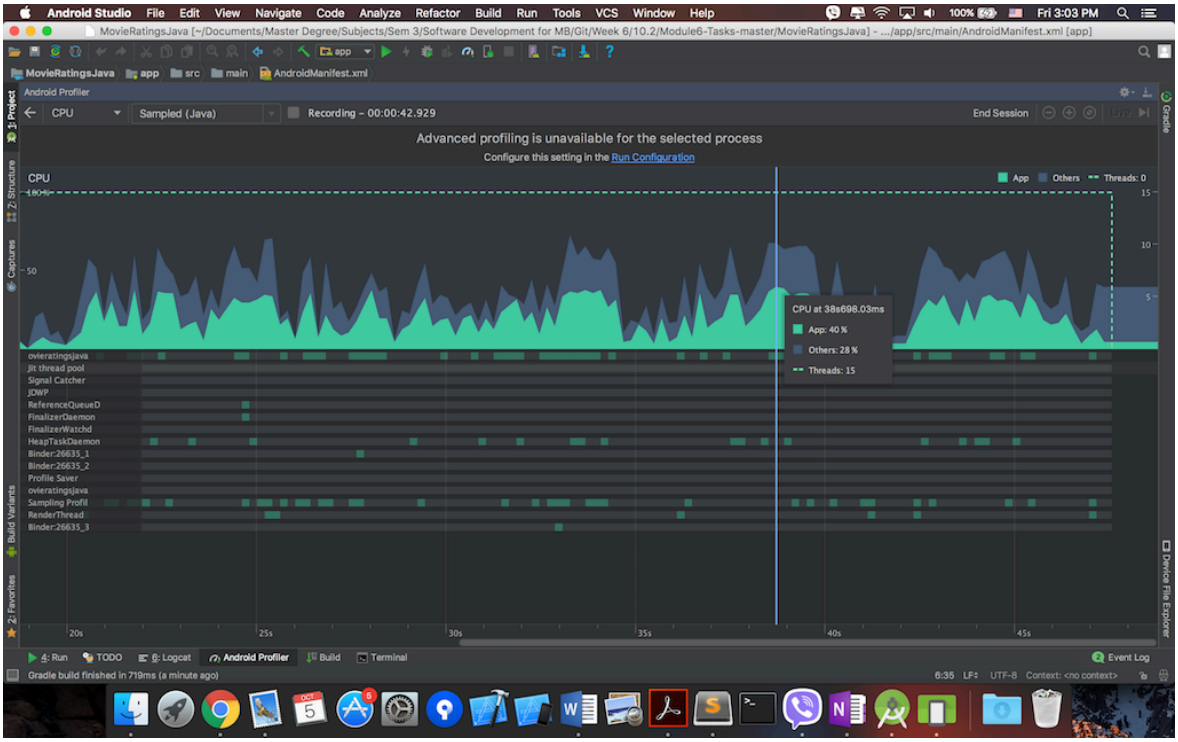

Figure 2: CPU Utilization in current app

Problems

After investigating the application, there are 3 issues are considered to be the main problems.

1. Loading the file in the main thread

The large text file is loaded in the main thread and which causes the delay for the app when running.

```
InputStream inputStream = getResources().openRawResource(R.raw.ratings);
movies = Movie.loadFromFile(inputStream);
```

2. Using list view without view holder

The current application uses the ListActivity to create a list of movie. The adapter, however, does not apply view holder pattern to reuse the row without creating the new one, and thus the row is increasing when the user scrolls the list view as we can see apparently in the log.

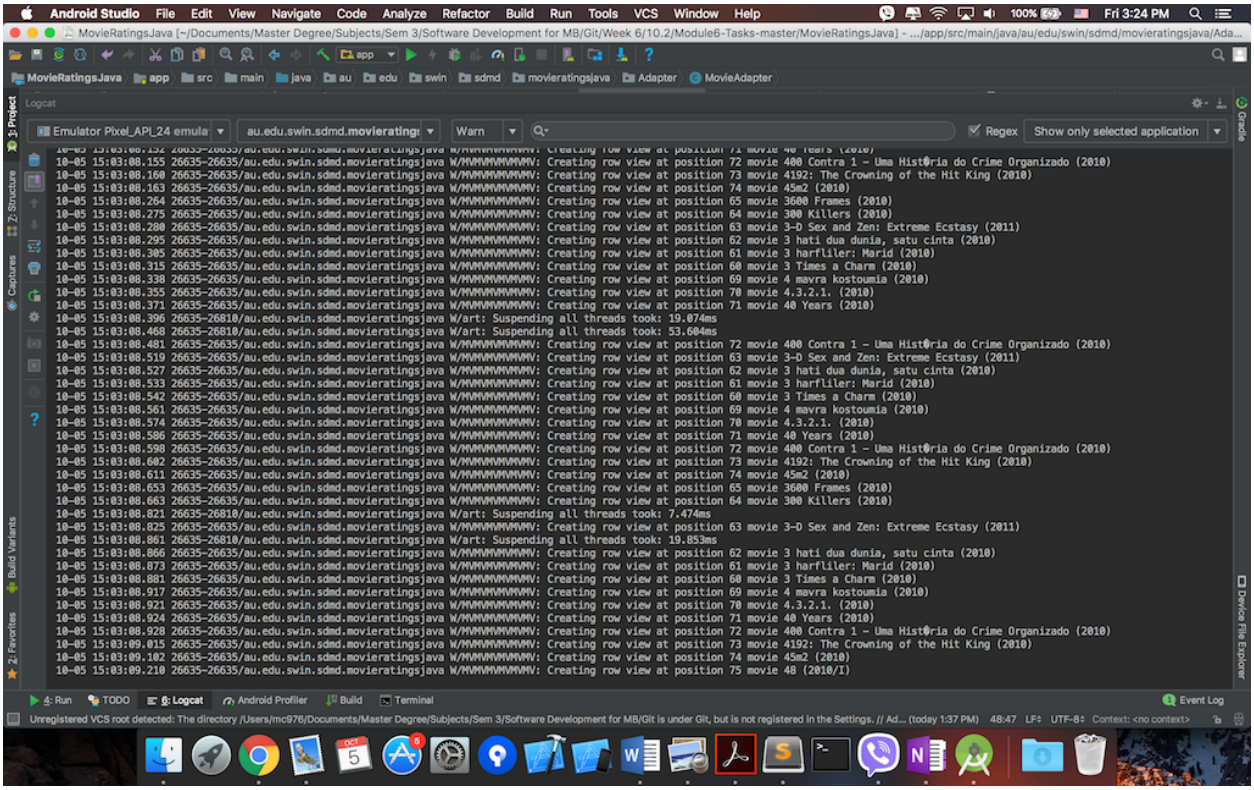

Figure 3: List View without view holder pattern

3. Creating bitmap in the main thread

In addition, the bitmap is drawn based on the movie name and movie rating on the main thread, and hence the CPU dramatically grows.

```
Bitmap movieIcon = getMovieIcon(currMovie.getName(), currMovie.getRating());
icon.setImageBitmap(movieIcon);
```

```
/** Creates a unique movie icon based on name and rating */
private Bitmap getMovieIcon(String movieName, String movieRating) {
    int bgColor = getColor(movieName);
    Bitmap b = Bitmap.createBitmap(192, 192, Bitmap.Config.ARGB_8888);
    b.eraseColor(bgColor); // fill bitmap with the color
    Canvas c = new Canvas(b);
    Paint p = new Paint();
    p.setAntiAlias(true);
    p.setColor(getTextColor(bgColor));
    p.setTextSize(96.0f);
    c.drawText(movieRating, 32, 96, p);
    return b;
}
```

Countermeasure

Problems	Mitigate Solutions
Loading the file in the main thread	Reading the file in background thread and response to update when the loading is successful
Using the list view without view holder pattern	Apply Recycler View instead of List View to reduce the number of creating the row
Creating bitmap in the main thread	Drawing the bitmap in background thread and handle concurrency threads so that the bitmap that has been drawn does not need to redraw a gain

1. Reading the file in background thread by using AsyncTask

The purpose of this approach is to allow the main thread of the Activity focus on loading the UI only. The background thread will load the large file asynchronously without blocking the main thread, and when the data is completely loaded, the callback method is invoked to bind the data into the adapter. The full code is in the Appendix.

2. Apply Recycler View instead of List View to reduce the number of creating the row.

Recycler View is apparently more effective than List View with regards to the performance. It is because the recycler view contains the View Holder pattern, which could help reuse the row for the list, and hence the memory is not increasing as in the list view.

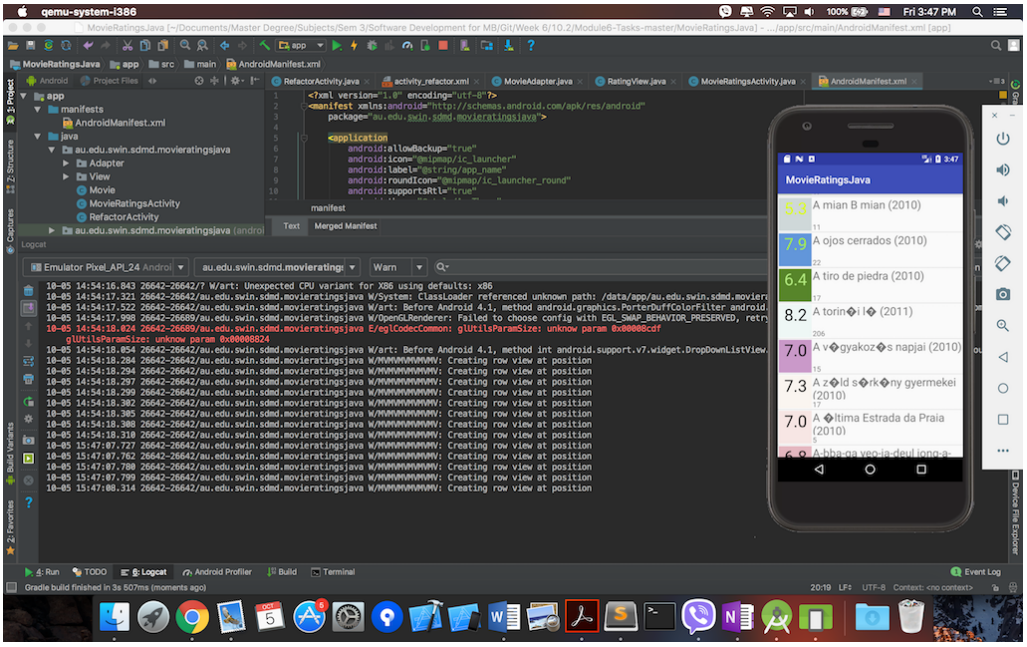

Figure 4: Creating row in recycler view log

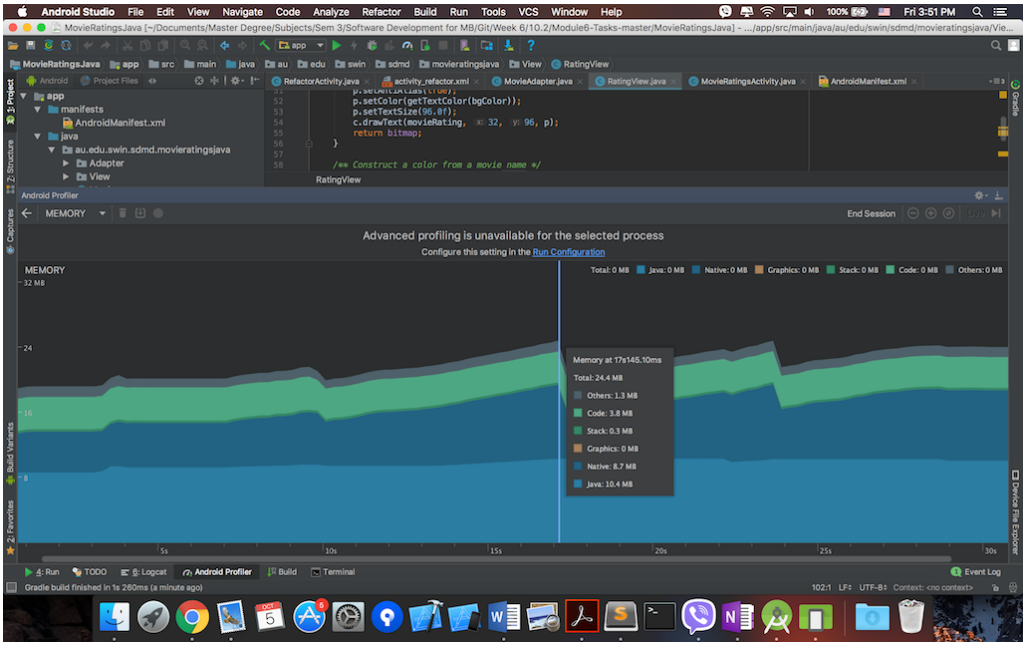

Figure 5: Memory after optimization

The source code for implement the recycler view is in the Appendix.

3. Drawing the bitmap in background thread and handle concurrency threads.

The purpose of drawing bitmap in background thread is to optimise the performance of the application since it helps reduce the CPU utilization. In addition, the custom image view is created in order to make the code looks more elegant. The custom image view will handle drawing bitmap in background thread while the adapter only need to update the UI of row. Furthermore, the AsyncDrawable is implemented in order to help handle the concurrency thread while drawing the bitmap. The detail of the code is in Appendix.

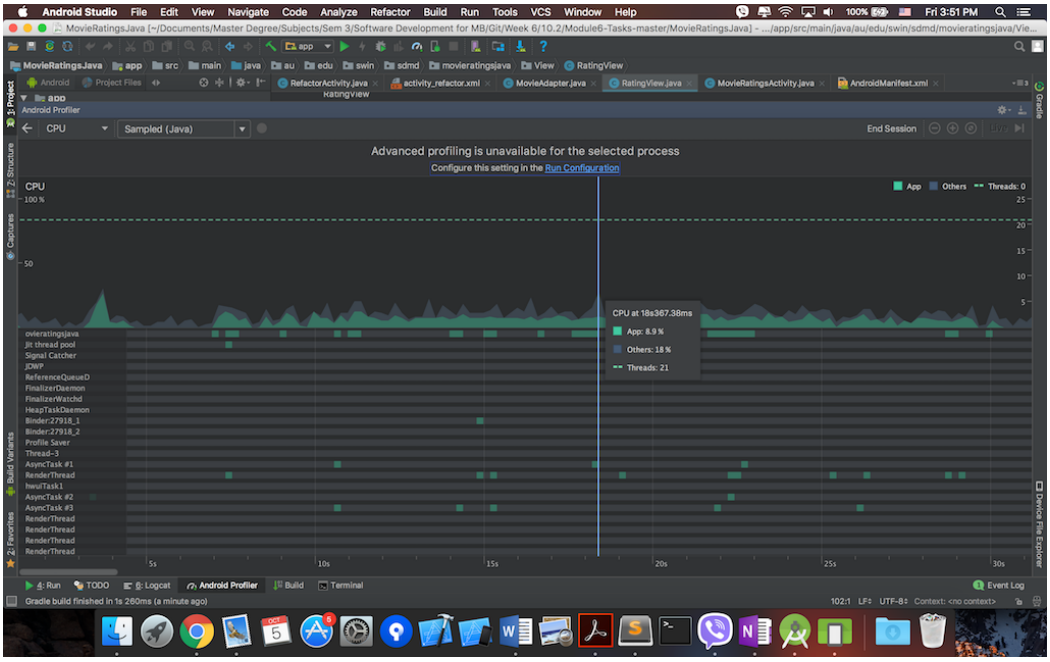

Figure 6: CPU Optimization

Usability Improvement

With the purpose of improving the usability for the user, the process bar has been implemented to indicate the loading process and also the popup message will display to inform to the user that the movie list is loading successfully.

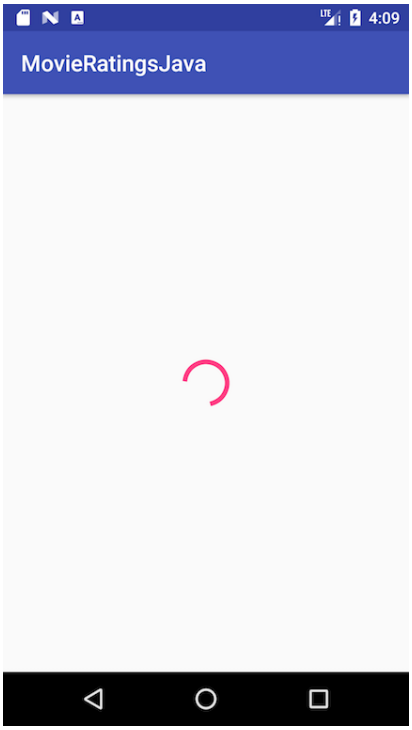

Figure 7: Loading ProcessBar

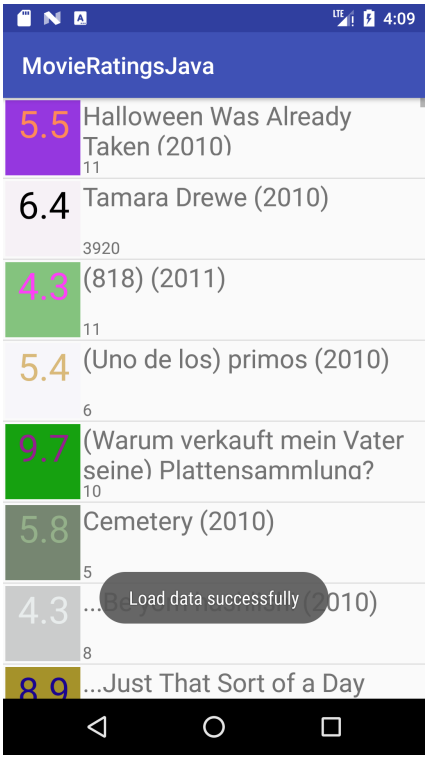

Figure 8: Popup message

References

Multithreading for Performance, viewed 5 October 2018, <https://android-developers.googleblog.com/2010/07/multithreading-for-performance.html>.

Processing Bitmaps Off the UI Thread | Android Developers, viewed 5 October 2018, <https://stuff.mit.edu/afs/sipb/project/android/docs/training/displaying-bitmaps/process-bitmap.html>.

Appendix

Reading the file in AsyncTask code snippet

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_refactor);
    RecyclerView recyclerView = findViewById(R.id.recyclerView);
    progressBar = findViewById(R.id.progressBar_cyclic);
    mAdapter = new MovieAdapter(movieList, this);
    RecyclerView.LayoutManager mLayoutManager = new
LinearLayoutManager(getApplicationContext());
    recyclerView.setLayoutManager(mLayoutManager);
    recyclerView.setAdapter(mAdapter);
    recyclerView.addItemDecoration(new DividerItemDecoration(this,
LinearLayoutManager.VERTICAL));
    GetFileTask task = new GetFileTask();
    task.execute();
}
```

```
class GetFileTask extends AsyncTask<Void, Void, List<Movie>> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressBar.setVisibility(View.VISIBLE);
    }

    @Override
    protected List<Movie> doInBackground(Void... voids) {
        InputStream inputStream = getResources().openRawResource(R.raw.ratings);
        return Movie.loadFromFile(inputStream);
    }

    @Override
    protected void onPostExecute(List<Movie> movies) {
        super.onPostExecute(movies);
        reloadData(movies);
        progressBar.setVisibility(View.GONE);
        Toast.makeText(RefactorActivity.this, "Load data successfully",
Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onProgressUpdate(Void... values) {
        super.onProgressUpdate(values);
    }
}
```

Recycler View optimisation

RefactorActivity

```
public class RefactorActivity extends AppCompatActivity {

    private List<Movie> movieList = new ArrayList<>();
    private MovieAdapter mAdapter;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_refactor);
        RecyclerView recyclerView = findViewById(R.id.recyclerView);
        progressBar = findViewById(R.id.progressBar_cyclic);
        mAdapter = new MovieAdapter(movieList, this);
        RecyclerView.LayoutManager mLayoutManager = new
LinearLayoutManager(getApplicationContext());
        recyclerView.setLayoutManager(mLayoutManager);
        recyclerView.setAdapter(mAdapter);
        recyclerView.addItemDecoration(new DividerItemDecoration(this,
LinearLayoutManager.VERTICAL));
        GetFileTask task = new GetFileTask();
        task.execute();
    }

    private void reloadData(List<Movie> movies) {
        movieList.clear();
        movieList.addAll(movies);
        mAdapter.notifyDataSetChanged();
    }

    class GetFileTask extends AsyncTask<Void, Void, List<Movie>> {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            progressBar.setVisibility(View.VISIBLE);
        }

        @Override
        protected List<Movie> doInBackground(Void... voids) {
            InputStream inputStream = getResources().openRawResource(R.raw.ratings);
            return Movie.loadFromFile(inputStream);
        }

        @Override
        protected void onPostExecute(List<Movie> movies) {
            super.onPostExecute(movies);
            reloadData(movies);
            progressBar.setVisibility(View.GONE);
            Toast.makeText(RefactorActivity.this, "Load data successfully",
Toast.LENGTH_SHORT).show();
        }

        @Override
        protected void onProgressUpdate(Void... values) {
            super.onProgressUpdate(values);
        }
```

```
    }

}
```

MovieAdapter

```
public class MovieAdapter extends RecyclerView.Adapter<MovieAdapter.MovieViewHolder> {
    private List<Movie> movies;
    private Context context;

    public class MovieViewHolder extends RecyclerView.ViewHolder {
        public RatingView icon;
        public TextView title;
        public TextView description;
        public MovieViewHolder(View view) {
            super(view);
            icon = view.findViewById(R.id.row_icon);
            title = view.findViewById(R.id.row_label);
            description = view.findViewById(R.id.row_subtext);
        }
    }

    public MovieAdapter(List<Movie> movies, Context context) {
        this.movies = movies;
        this.context= context;
    }

    @NonNull
    @Override
    public MovieViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
        View itemView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.listrow, parent,false);
        Log.w("MVMVMVMVMVMV", "Creating row view at position ");
        return new MovieViewHolder(itemView) ;
    }

    @Override
    public void onBindViewHolder(@NonNull MovieViewHolder holder, int position) {
        Movie movie = this.movies.get(position);
        holder.title.setText(movie.getName());
        holder.icon.setImageByName(movie.getName(), movie.getRating());
        holder.description.setText(movie.getVotes());
    }

    @Override
    public int getItemCount() {
        return this.movies.size();
    }

}
```

Drawing Bitmap in background

RatingView

```
public class RatingView extends android.support.v7.widget.AppCompatImageView {

    private Bitmap bitmap;
    private Context mContext;

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
    }

    public RatingView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        bitmap = Bitmap.createBitmap(192, 192, Bitmap.Config.ARGB_8888);
    }

    public void setImageByName(String movieName, String movieRating) {
        if (AsyncDrawable.cancelPotentialWork(movieName, this)) {
            final BitmapWorkerTask task = new BitmapWorkerTask(this);
            final AsyncDrawable asyncDrawable =
                    new AsyncDrawable(mContext.getResources(), bitmap, task);
            this.setImageDrawable(asyncDrawable);
            task.execute(movieName, movieRating);
        }
    }

    /** Creates a unique movie icon based on name and rating */
    private Bitmap getMovieIcon(String movieName, String movieRating) {
        int bgColor = getColor(movieName);
        bitmap.eraseColor(bgColor); // fill bitmap with the color
        Canvas c = new Canvas(bitmap);
        Paint p = new Paint();
        p.setAntiAlias(true);
        p.setColor(getTextColor(bgColor));
        p.setTextSize(96.0f);
        c.drawText(movieRating, 32, 96, p);
        return bitmap;
    }

    /** Construct a color from a movie name */
    private int getColor(String name) {
        String hex = toHexString(name);
        String red = "#"+hex.substring(0,2);
        String green = "#"+hex.substring(2,4);
        String blue = "#"+hex.substring(4,6);
        String alpha = "#"+hex.substring(6,8);
        int color = Color.argb(Integer.decode(alpha), Integer.decode(red),
                Integer.decode(green), Integer.decode(blue));
        return color;
    }

    /** Given a movie name -- generate a hex value from its hashcode */
    private String toHexString(String name) {
        int hc = name.hashCode();
        String hex = Integer.toHexString(hc);
        if (hex.length() < 8)
        {
```

```
            hex = hex+hex+hex;
            hex = hex.substring(0,8); // use default color value
        }
        return hex;
    }

    /** Crude optimization to obtain a contrasting color -- does not work well yet */
    private int getTextColor(int bg) {

        int r = Color.red(bg);
        int g = Color.green(bg);
        int b = Color.blue(bg);
        String hex = Integer.toHexString(r)+Integer.toHexString(g);
        hex += Integer.toHexString(b);

        int cDec = Integer.decode("#"+hex);
        if (cDec > 0xFFFFFF/2)  // go dark for lighter shades
            return Color.rgb(0, 0, 0);
        else
        {
            r = (r+128)%256;
            g = (g+128)%256;
            b = (b+128)%256;
            return Color.rgb(r,g,b);
        }
    }


    class BitmapWorkerTask extends AsyncTask<String, Void, Bitmap> {
        private final WeakReference<ImageView> imageViewReference;
        private String movieName;
        private String movieRating;

        public BitmapWorkerTask(ImageView imageView) {
            // Use a WeakReference to ensure the ImageView can be garbage collected
            imageViewReference = new WeakReference<ImageView>(imageView);
        }

        // Decode image in background.
        @Override
        protected Bitmap doInBackground(String... params) {
            movieName = params[0];
            movieRating = params[1];
            return getMovieIcon(params[0], params[1]);
        }

        // Once complete, see if ImageView is still around and set bitmap.
        @Override
        protected void onPostExecute(Bitmap bitmap) {

            if (isCancelled()) {
                bitmap = null;
            }

            if (bitmap != null) {
                final ImageView imageView;
                imageView = imageViewReference.get();
                final BitmapWorkerTask bitmapWorkerTask =
                        AsyncDrawable.getBitmapWorkerTask(imageView);
                if (this == bitmapWorkerTask) {
                    imageView.setImageBitmap(bitmap);
```

```
                }
            }
        }

    }

    static class AsyncDrawable extends BitmapDrawable {
        private final WeakReference<BitmapWorkerTask> bitmapWorkerTaskReference;

        private AsyncDrawable(Resources res, Bitmap bitmap,
                              BitmapWorkerTask bitmapWorkerTask) {
            super(res, bitmap);
            bitmapWorkerTaskReference =
                    new WeakReference<BitmapWorkerTask>(bitmapWorkerTask);
        }

        public BitmapWorkerTask getBitmapWorkerTask() {
            return bitmapWorkerTaskReference.get();
        }

        public static boolean cancelPotentialWork(String movieName, ImageView
imageView) {
            final BitmapWorkerTask bitmapWorkerTask = getBitmapWorkerTask(imageView);

            if (bitmapWorkerTask != null) {
                final String bitmapData = bitmapWorkerTask.movieName;
                if (!bitmapData.equals(movieName) ) {
                    // Cancel previous task
                    bitmapWorkerTask.cancel(true);
                } else {
                    // The same work is already in progress
                    return false;
                }
            }
            // No task associated with the ImageView, or an existing task was
cancelled
            return true;
        }

        public static BitmapWorkerTask getBitmapWorkerTask(ImageView imageView) {
            if (imageView != null) {
                final Drawable drawable = imageView.getDrawable();
                if (drawable instanceof AsyncDrawable) {
                    final AsyncDrawable asyncDrawable = (AsyncDrawable) drawable;
                    return asyncDrawable.getBitmapWorkerTask();
                }
            }
            return null;
        }
    }

}
```