THREADING IN ANDROID

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

THE UI THREAD

Every app has its own “main” thread that runs UI objects such as View objects; this thread is often called the UI thread.

The system does *not* create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly.

If everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI.

See example: **ThreadingNoThreading**

THE UI THREAD

Every app has its own “main” thread that runs UI objects such as View objects; this thread is often called the UI thread.

The system does *not* create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly.

If the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the "application not responding" (ANR) dialog.

https://developer.android.com/training/articles/perf-anr.html

THE UI THREAD

The Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread — you must do all manipulation to your user interface from the UI thread.

There are simply two rules to Android's single thread model:

1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

See example: ThreadingSimple

```java
public class SimpleThreadingExample extends Activity {

    private static final String TAG = "SimpleThreadingExample";

    private Bitmap mBitmap;
    private ImageView mIView;
    private int mDelay = 5000;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mIView = (ImageView) findViewById(R.id.imageView);

        final Button loadButton = (Button) findViewById(R.id.loadButton);
        loadButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                loadIcon();
            }
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener((v) -> {
                Toast.makeText(SimpleThreadingExample.this, "I'm Working",
                        Toast.LENGTH_SHORT).show();
        });
    }

    private void loadIcon() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(mDelay);
                } catch (InterruptedException e) {
                    Log.e(TAG, e.toString());
                }
                mBitmap = BitmapFactory.decodeResource(getResources(),
                        R.drawable.painter);

                // This doesn't work in Android
                mIView.setImageBitmap(mBitmap);
            }
        }).start();
    }
}
```

This crashes your app.

Why?

```java
public class SimpleThreadingExample extends Activity {

    private static final String TAG = "SimpleThreadingExample";

    private Bitmap mBitmap;
    private ImageView mIView;
    private int mDelay = 5000;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mIView = (ImageView) findViewById(R.id.imageView);

        final Button loadButton = (Button) findViewById(R.id.loadButton);
        loadButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                loadIcon();
            }
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener((v) -> {
                Toast.makeText(SimpleThreadingExample.this, "I'm Working",
                        Toast.LENGTH_SHORT).show();
        });
    }

    private void loadIcon() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(mDelay);
                } catch (InterruptedException e) {
                    Log.e(TAG, e.toString());
                }
                mBitmap = BitmapFactory.decodeResource(getResources(),
                        R.drawable.painter);

                // This doesn't work in Android
                mIView.setImageBitmap(mBitmap);
            }
        }).start();
    }
}
```

Because tasks that you run on a thread from a thread pool aren't running on your UI thread, they don't have access to UI objects.

RUNNING ON THE UI THREAD

To fix this problem, Android offers several ways to access the UI thread from other threads, including:

Activity.runOnUiThread(Runnable)

View.post(Runnable)

View.postDelayed(Runnable, long)

See example: ThreadingRunningOnUIThread

RUNNING ON THE UI THREAD

To fix this problem, Android offers several ways to access the UI thread from other threads, including:

Activity.runOnUiThread(Runnable)

View.post(Runnable)

View.postDelayed(Runnable, long)

```
    private void loadIcon() {
        new Thread((Runnable) () → {
                try {
                    Thread.sleep(mDelay);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                mBitmap = BitmapFactory.decodeResource(getResources(),
                        R.drawable.painter);
                SimpleThreadingExample.this.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mImageView.setImageBitmap(mBitmap);
                    }
                });
        }).start();
    }
}
```

This implementation is thread-safe: the decoding operation is done from a separate thread while the ImageView is manipulated from the UI thread.

RUNNING ON THE UI THREAD

Thread and Runnable are basic classes that, on their own, have only limited power. 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.

Another option is to use a threading class provided by Android called AsyncTask.

This greatly simplifies the execution of worker thread tasks that need to interact with the UI.

ASYNCTASK

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.

See example: **ThreadingAsyncTask**

AsyncTaskActivity.java

```
public class AsyncTaskActivity extends Activity {

    private final static String TAG = "ThreadingAsyncTask";

    private ImageView mImageView;
    private ProgressBar mProgressBar;
    private int mDelay = 500;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mImageView = (ImageView) findViewById(R.id.imageView);;
        mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

        final Button button = (Button) findViewById(R.id.loadButton);
        button.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                new LoadIconTask().execute(R.drawable.painter);
            }
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener((v) -> {
                Toast.makeText(AsyncTaskActivity.this, "I'm Working",
                        Toast.LENGTH_SHORT).show();
        });

    }

    class LoadIconTask extends AsyncTask<Integer, Integer, Bitmap> {


        @Override
        protected void onPreExecute() { mProgressBar.setVisibility(ProgressBar.VISIBLE); }

        @Override
        protected Bitmap doInBackground(Integer... resId) {...}

        @Override
        protected void onProgressUpdate(Integer... values) { mProgressBar.setProgress(values[0]); }

        @Override
        protected void onPostExecute(Bitmap result) {...}

        private void sleep() {...}
    }
}
```

ASYNCTASK

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

AsyncTasks should ideally be used for short operations (a few seconds at the most.)

An asynchronous task is defined by **3 generic types and 4 executions steps.**

ASYNCTASK GENERIC TYPES

Params - the type of the parameters sent to the task upon execution.

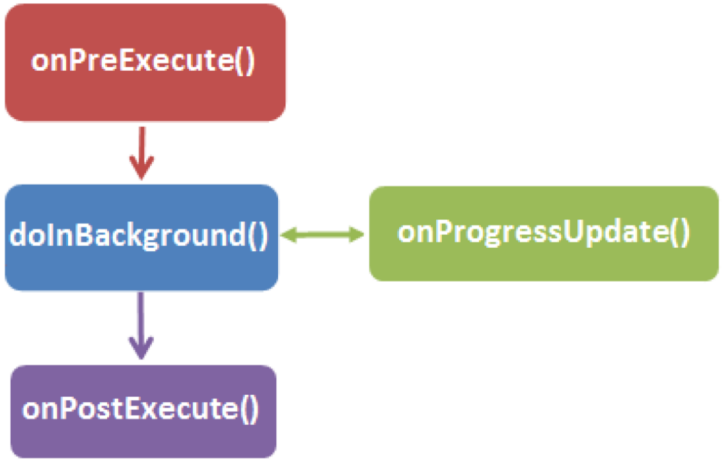
Progress - the type of the progress units published during the background computation.

Result - the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type **Void**:

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

4 STEPS



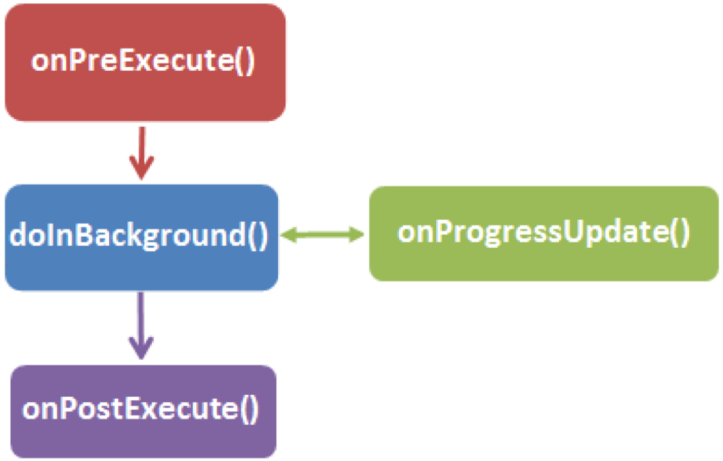
onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the user interface.

doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing.

This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step.

This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.

4 STEPS



onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.

onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

NOTES ON ASYNCTASK

There are a few threading rules that must be followed for this class to work properly:

- The AsyncTask class must be loaded on the UI thread. This is done automatically as of JELLY_BEAN.
- 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.)

HANDLERS

Used to pass messages between two threads.

More flexible than AsyncTask in that it can be used with any two threads, not just the UI Thread and a background thread.

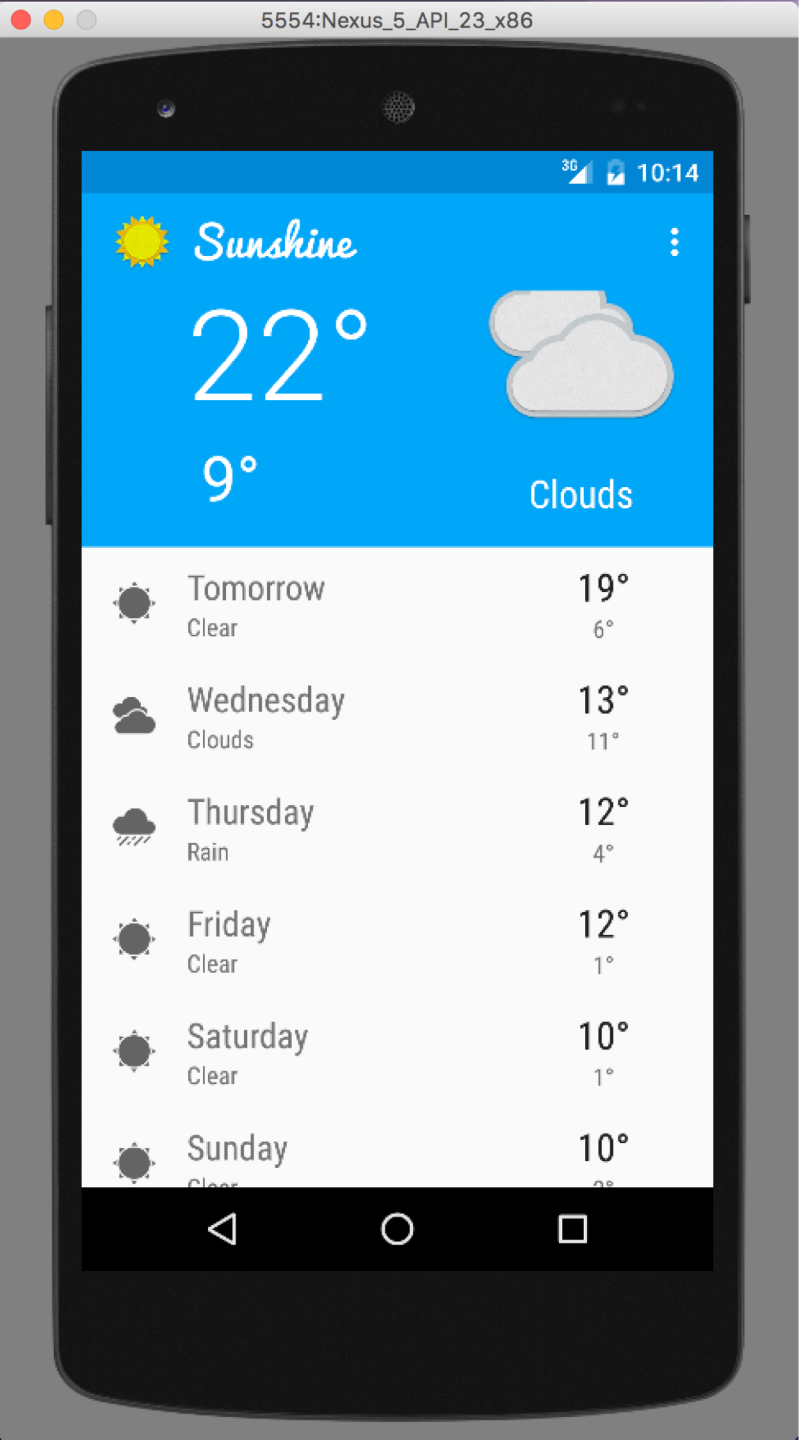
Each Handler is associated with a specific thread.

One Thread can hand work to another Thread by sending Messages or posting Runnables to the Handler associated with that Thread.

See examples:
ThreadingHandlerRunnable **ThreadingHandlerMessages**

```
public class HandlerRunnableActivity extends Activity {

    private ImageView mImageView;
    private ProgressBar mProgressBar;
    private Bitmap mBitmap;
    private int mDelay = 500;
    private final Handler handler = new Handler();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mImageView = (ImageView) findViewById(R.id.imageView);
        mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

        final Button button = (Button) findViewById(R.id.loadButton);
        button.setOnClickListener((v) → {
                new Thread(new LoadIconTask(R.drawable.painter)).start();
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener((v) → {
                Toast.makeText(HandlerRunnableActivity.this, "I'm Working",
                        Toast.LENGTH_SHORT).show();
        });

    }
```

```
private class LoadIconTask implements Runnable {
    int resId;

    LoadIconTask(int resId) { this.resId = resId; }

    public void run() {
        handler.post(() → { mProgressBar.setVisibility(ProgressBar.VISIBLE); });

        mBitmap = BitmapFactory.decodeResource(getResources(), resId);

        // Simulating long-running operation

        for (int i = 1; i < 11; i++) {
            sleep();
            final int step = i;
            handler.post(new Runnable() {
                @Override
                public void run() {
                    mProgressBar.setProgress(step * 10);
                }
            });
        }

        handler.post(new Runnable() {
            @Override
            public void run() {
                mImageView.setImageBitmap(mBitmap);
            }
        });

        handler.post(new Runnable() {
            @Override
            public void run() {
                mProgressBar.setVisibility(ProgressBar.INVISIBLE);
            }
        });
    }
}
```



Touches on the following:

- View and ViewGroup classes
- Layouts
- Threading with AsyncTask

- Networking, e.g. using a web service

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.

Next:

Networking and Connectivity