5COSC005W MOBILE APPLICATION DEVELOPMENT
Lecture 6–7: Background Tasks - Network Connectivity

Dr Dimitris C. Dracopoulos

Module Web page:

http://users.wmin.ac.uk/~dracopd/DOCUM/courses/5cosc005w/5cosc005w.html

Background Tasks

AsyncTask and AsyncTaskLoader

Threads

The main thread

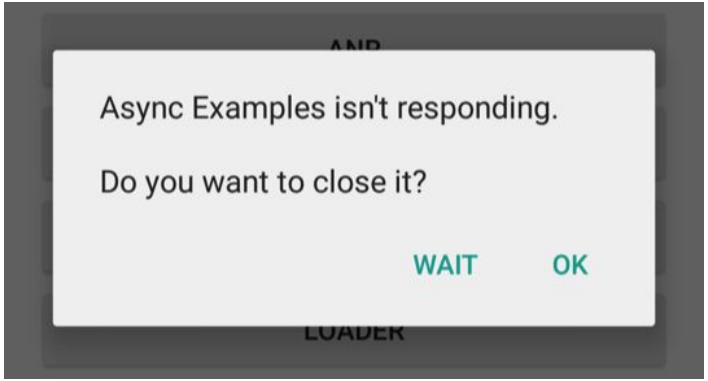
- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

The Main thread must be fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs

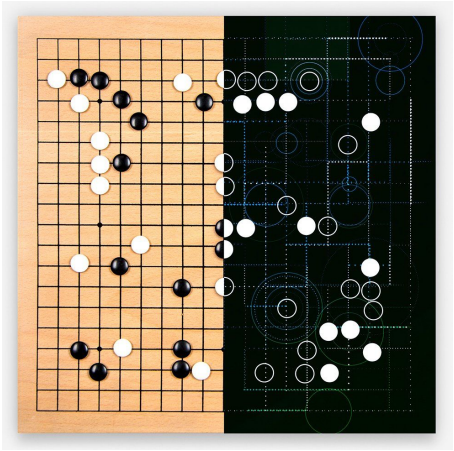
Users uninstall unresponsive apps

- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog



What is a long running task?

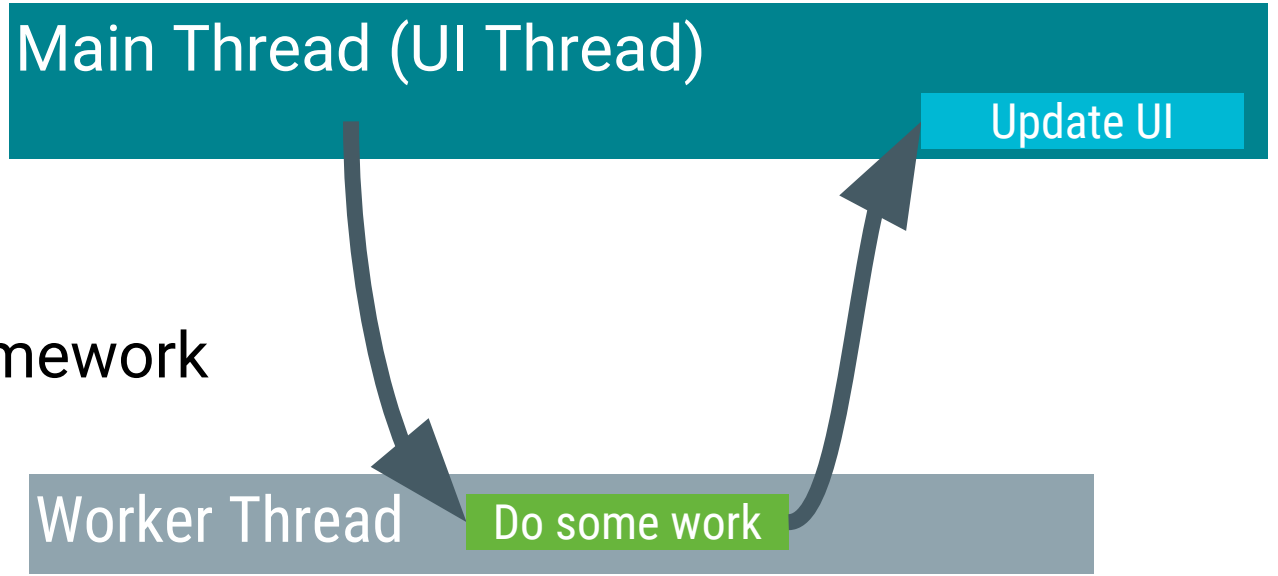
- Network operations
- Long calculations
- Downloading/uploading files
- Processing images
- Loading data



Background threads

Execute long running tasks on a **background thread**

- AsyncTask
- The Loader Framework
- Services



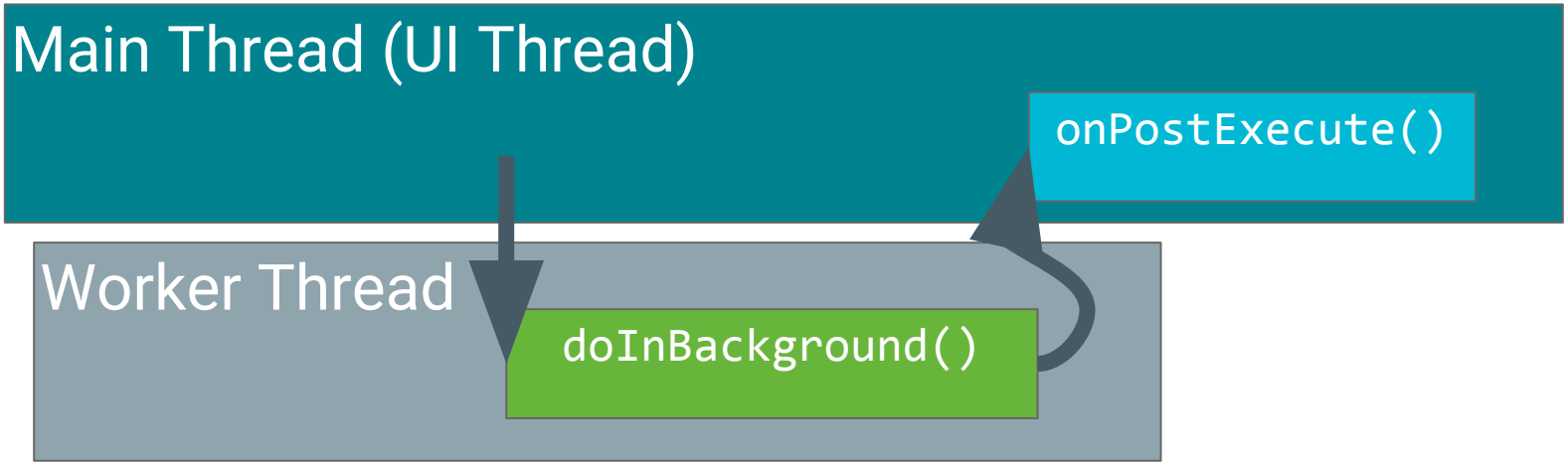
Two rules for Android threads

- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside the UI thread
  - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

Use AsyncTask to implement basic background tasks



Override two methods

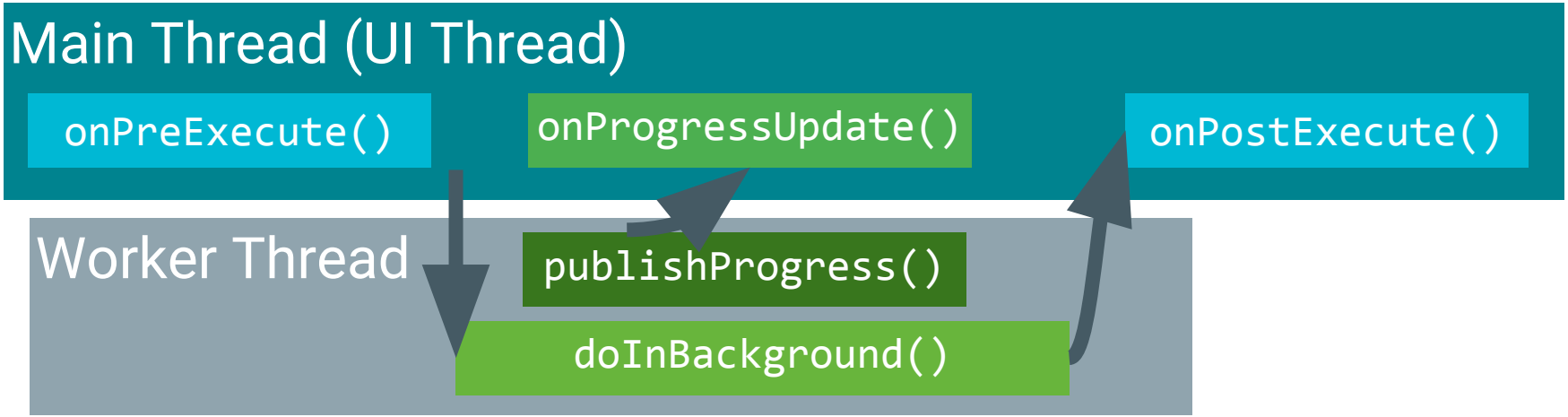
- `doInBackground()`—runs on a background thread
  - All the work to happen in the background
- `onPostExecute()`—runs on main thread when work done
  - Process results
  - Publish results to the UI

AsyncTask helper methods

- `onPreExecute()`
  - Runs on the main thread
  - Sets up the task

- `onProgressUpdate()`
  - Runs on the main thread
  - receives calls from `publishProgress()` from background thread

AsyncTask helper methods



Creating an AsyncTask

1. Subclass AsyncTask
2. Provide data type sent to `doInBackground()`
3. Provide data type of progress units for `onProgressUpdate()`
4. Provide data type of result for `onPostExecute()`

```
private class MyAsyncTask
    extends AsyncTask<URL, Integer, Bitmap> {...}
```

MyAsyncTask class definition

```
private class MyAsyncTask
    extends AsyncTask<String, Integer, Bitmap> {...}
```

- String—could be query, URI for filename
- Integer—percentage completed, steps done
- Bitmap—an image to be displayed
- Use Void if no data passed

onPreExecute()

```
protected void onPreExecute() {
    // display a progress bar
    // show a toast
}
```

doInBackground()

```
protected Bitmap doInBackground(String... query) {
    // Get the bitmap
    return bitmap;
}
```

onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
```

onPostExecute()

```
protected void onPostExecute(Bitmap result) {
    // Do something with the bitmap
}
```

Start background work

```
public void loadImage (View view) {
    String query = mEditText.getText().toString();
    new MyAsyncTask(query).execute();
   }
```

Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
- AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
- Old AsyncTasks stay around
- App may run out of memory or crash

When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise

Internet connection

Steps to connect to the Internet

1. Add permissions to Android Manifest
2. Check Network Connection
3. Create Worker Thread
4. Implement background task
    a. Create URI
    b. Make HTTP Connection
    c. Connect and GET Data
5. Process results
    a. Parse Results

Permissions

Permissions in AndroidManifest

Internet

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Check Network State

```
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Worker Thread

Use Worker Thread

- AsyncTask—very short task, or no result returned to UI
- AsyncTaskLoader—for longer tasks, returns result to UI
- Background Service—covered in a later week

Background work

In the background task (for example in `doInBackground()`)

1. Create URI
2. Make HTTP Connection
3. Download Data

Create URI

URI = Uniform Resource Identifier

String that names or locates a particular resource

- file://
- http:// and https://
- content://

Sample URL for Google Books API

https://www.googleapis.com/books/v1/volumes?q=pride+prejudice&maxResults=5&printType=books

Constants for Parameters

```
final String BASE_URL =
    "https://www.googleapis.com/books/v1/volumes?";

final String QUERY_PARAM = "q";

final String MAX_RESULTS = "maxResults";

final String PRINT_TYPE = "printType";
```

Build a URI for the request

```
Uri builtURI = Uri.parse(BASE_URL).buildUpon()
        .appendQueryParameter(QUERY_PARAM, "pride+prejudice")
        .appendQueryParameter(MAX_RESULTS, "10")
        .appendQueryParameter(PRINT_TYPE, "books")
        .build();
URL requestURL = new URL(builtURI.toString());
```

HTTP Client Connection

Make a connection from scratch

- Use `HttpURLConnection`
- Must be done on a separate thread
- Requires InputStreams and try/catch blocks

Create a HttpURLConnection

```
HttpURLConnection conn =
   (HttpURLConnection) requestURL.openConnection();
```

Configure connection

```
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
```

Connect and get response

```
conn.connect();
int response = conn.getResponseCode();

InputStream is = conn.getInputStream();
String contentAsString = convertIsToString(is, len);
return contentAsString;
```

Close connection and stream

```
} finally {
    conn.disconnect();
        if (is != null) {
            is.close();
        }
}
```

Convert Response to String

Convert input stream into a string

```
public String convertIsToString(InputStream stream, int len)
    throws IOException, UnsupportedEncodingException {

    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

BufferedReader is more efficient

```
StringBuilder builder = new StringBuilder();
BufferedReader reader =
    new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
   builder.append(line + "\n");
}
if (builder.length() == 0) {
   return null;
}
resultString = builder.toString();
```

Parse Results

Parsing the results

- Implement method to receive and handle results (`onPostExecute()`)
- Response is often JSON or XML

Parse results using helper classes

- JSONObject, JSONArray
- XMLPullParser—parses XML

JSON basics

```
{
 "population":1,252,000,000,
 "country":"India",
 "cities":["New Delhi","Mumbai","Kolkata","Chennai"]
}
```

JSONObject basics

```
JSONObject jsonObject = new JSONObject(response);

String nameOfCountry = (String) jsonObject.get("country");
long population = (Long) jsonObject.get("population");
JSONArray listOfCities = (JSONArray) jsonObject.get("cities");

Iterator<String> iterator = listOfCities.iterator();
while (iterator.hasNext()) {
    // do something
}
```

Another JSON example

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}
```

Another JSON example

Get "onclick" value of the 3rd item in the "menuitem" array

```
JSONObject data = new JSONObject(responseString);
JSONArray menuItemArray =
        data.getJSONArray("menuitem");
JSONObject thirdItem =
        menuItemArray.getJSONObject(2);
String onClick = thirdItem.getString("onclick");
```

Manage Network Connection

Getting Network information

- ConnectivityManager
  - Answers queries about the state of network connectivity
  - Notifies applications when network connectivity changes
- NetworkInfo
  - Describes status of a network interface of a given type
  - Mobile or Wi-Fi

Check if network is available

```
ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

if (networkInfo != null && networkInfo.isConnected()) {
    // Create background thread to connect and get data
    new DownloadWebpageTask().execute(stringUrl);
} else {
    textView.setText("No network connection available.");
}
```

Check for WiFi & Mobile

```
NetworkInfo networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();

networkInfo =
    connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
```

Loaders

What is a Loader?

- Provides asynchronous loading of data
- **Reconnects to Activity after configuration change**
- Can monitor changes in data source and deliver new data
- Callbacks implemented in Activity
- Many types of loaders available
  - AsyncTaskLoader, CursorLoader

Why use loaders?

- Execute tasks OFF the UI thread
- LoaderManager handles configuration changes for you
- Efficiently implemented by the framework
- Users don't have to wait for data to load

What is a LoaderManager?

- Manages loader functions via callbacks
- Can manage multiple loaders
  - loader for database data, for AsyncTask data, for internet data…

Get a loader with initLoader()

- Creates and starts a loader, or reuses an existing one, including its data
- Use restartLoader() to clear data in existing loader

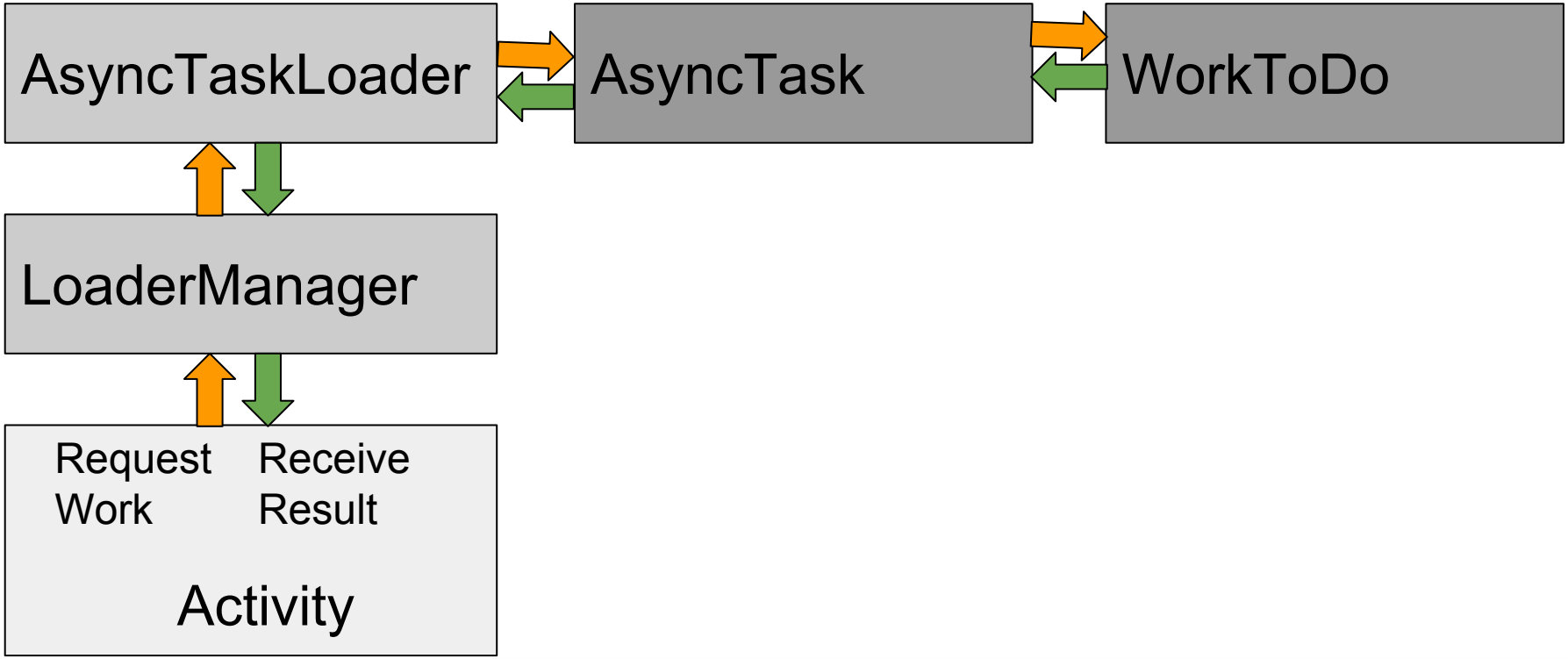
```
LoaderManager.getInstance(this).initLoader(Id, args,callback);

LoaderManager.getInstance(this).initLoader(0, null, this);

LoaderManager.getInstance(this).initLoader(0, null, this);
```

Implementing AsyncTaskLoader

AsyncTaskLoader Overview



AsyncTask ⟶ AsyncTaskLoader

`doInBackground()` ⟶ `loadInBackground()`

`onPostExecute()` ⟶ `onLoadFinished()`

Steps for AsyncTaskLoader subclass

1. Subclass AsyncTaskLoader
2. Implement constructor
3. `loadInBackground()`
4. `onStartLoading()`

Subclass AsyncTaskLoader

```
public static class StringListLoader
        extends AsyncTaskLoader<List<String>> {

    public StringListLoader(Context context, String queryString) {
        super(context);
        mQueryString = queryString;
    }
}
```

loadInBackground()

```
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Load the data from the network or from a database
    return data;
}
```

onStartLoading()

When `restartLoader()` or `initLoader()` is called, the LoaderManager invokes the `onStartLoading()` callback

- Check for cached data
- Start observing the data source (if needed)
- Call `forceLoad()` to load the data if there are changes or no cached data

```
protected void onStartLoading() {  forceLoad();  }
```

Implement loader callbacks in Activity

- `onCreateLoader()` — Create and return a new Loader for the given ID
- `onLoadFinished()` — Called when a previously created loader has finished its load
- `onLoaderReset()` — Called when a previously created loader is being reset making its data unavailable

onCreateLoader()

```
@Override
public Loader<List<String>> onCreateLoader(int id, Bundle args) {
    return new StringListLoader(this,args.getString("queryString"));
}
```

onLoadFinished()

Results of `loadInBackground()` are passed to `onLoadFinished()` where you can display them

```
public void onLoadFinished(Loader<List<String>> loader,
List<String> data) {
    mAdapter.setData(data);
}
```

onLoaderReset()

- Only called when loader is destroyed
- Leave blank most of the time

```
@Override
public void onLoaderReset(final LoaderList<String>> loader) { }
```

Get a loader with initLoader()

- In Activity
- Use support library to be compatible with more devices

```
LoaderManager.getInstance(this).initLoader(0, null, this);
```

Sending Code to execute on the UI Thread

If a different thread would like to update the graphical components of a screen (`Views`) it needs to do in the UI thread.

- This can be done by sending the code for execution to the UI thread using the `runOnUiThread` method found in the `Activity` class:

```
runOnUiThread(new Runnable(){
                    @Override
                    public void run(){
                        // code to be executed on UI thread
                        // i.e. widget updates
                    }
                });
```

- Alternatively call:

```
View.post(Runnable)
```

- or

```
View.postDelayed(Runnable, long)
```