Lecture
Running Background Tasks

TU856/3

Dr. Susan McKeever

Background tasks?

What sort of
tasks on a phone
delay / hang the screen?

Background tasks

- Network operations
- dB connections (if updating)
- Downloads
- Geocoding in an app that uses maps – any yesterday?
- and many more....

Android API flags this

getWritableDatabase

Added in API level 1

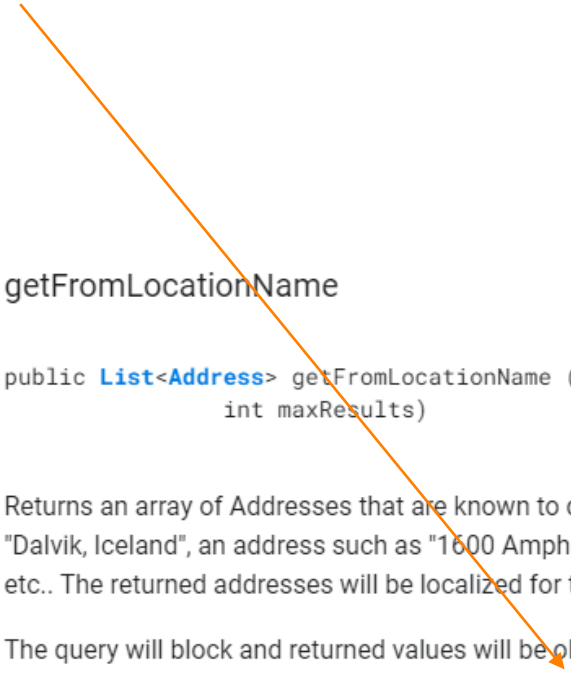
```
public SQLiteDatabase getWritableDatabase ()
```

Create and/or open a database that will be used for reading and writing. The first time this is called, the database will be opened and `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and/or `onOpen(SQLiteDatabase)` will be called.

Once opened successfully, the database is cached, so you can call this method every time you need to write to the database. (Make sure to call `close()` when you no longer need the database.) Errors such as bad permissions or a full disk may cause this method to fail, but future attempts may succeed if the problem is fixed.

> **!** Database upgrade may take a long time, you should not call this method from the application main thread, including from `ContentProvider.onCreate()`.

Sort of..



getFromLocationName

Added in API level 1

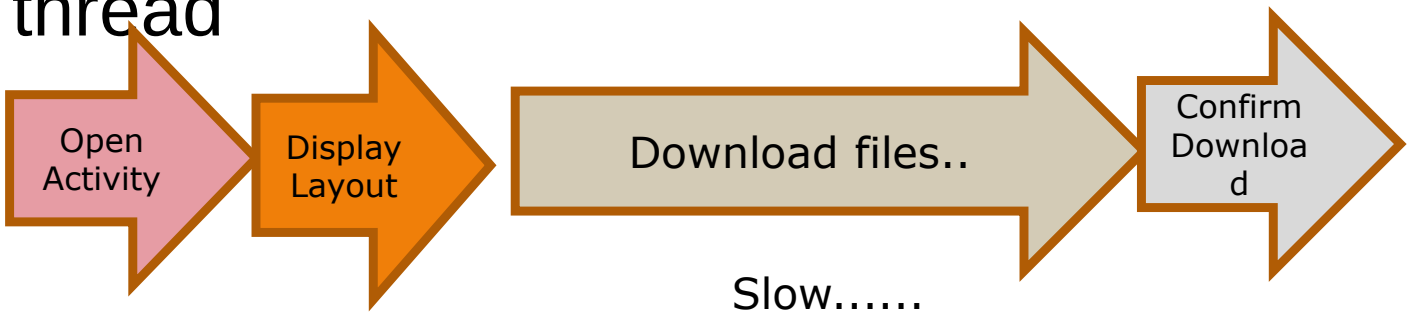
```
public List<Address> getFromLocationName (String locationName,
                int maxResults)
```

Returns an array of Addresses that are known to describe the named location, which may be a place name such as "Dalvik, Iceland", an address such as "1600 Amphitheatre Parkway, Mountain View, CA", an airport code such as "SFO", etc.. The returned addresses will be localized for the locale provided to this class's constructor.

The query will block and returned values will be obtained by means of a network lookup. The results are a best guess and are not guaranteed to be meaningful or correct. It may be useful to call this method from a thread separate from your primary UI thread.

Parameters

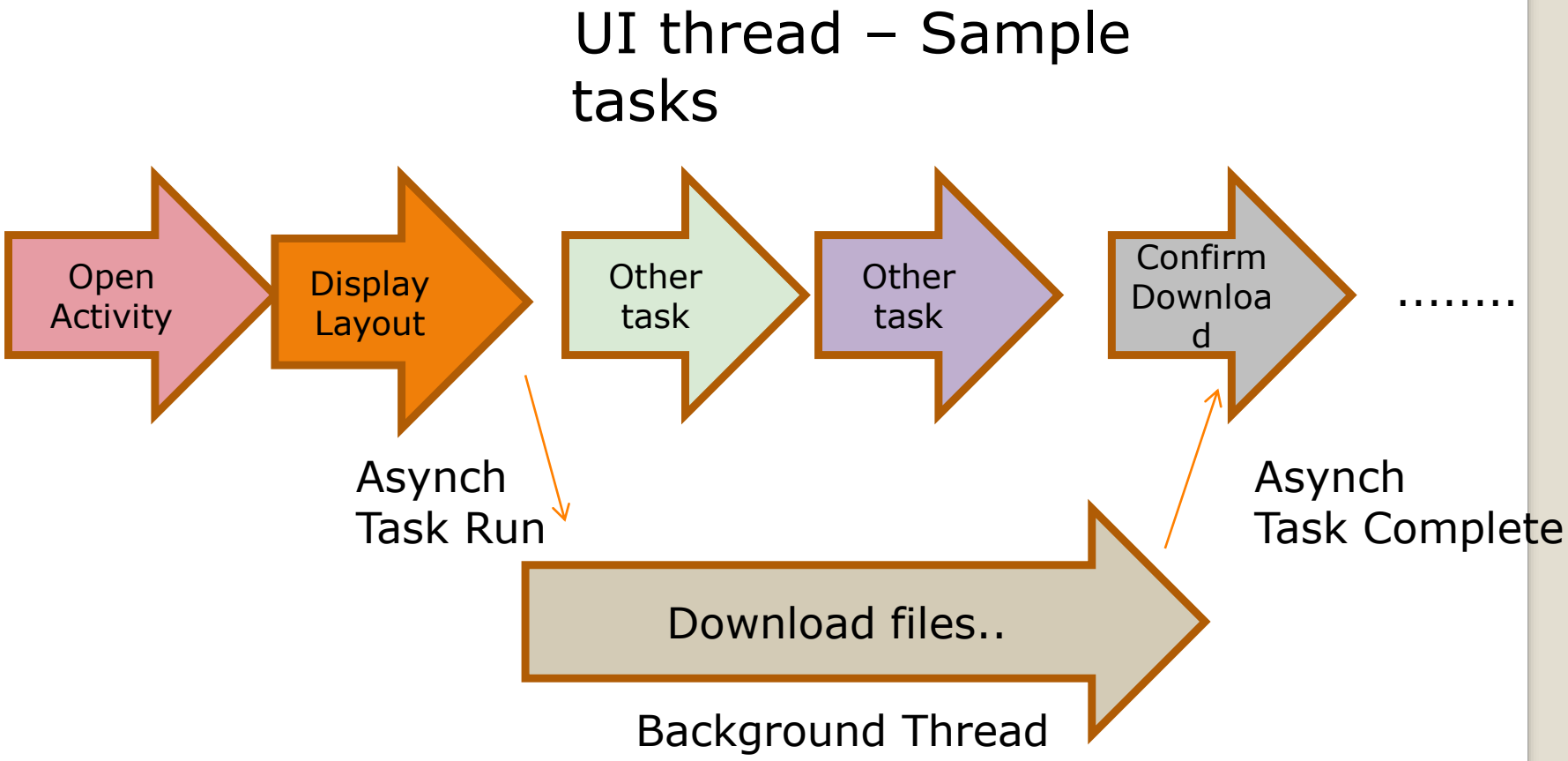
To speed up

- General threading concept
  - Move the “slow” task(s) off the main UI thread
  - 

  - Various ways in Android – including:

1. Asynchronous tasks
2. Intent Services

Pushing delays to another thread



Implementing

- AsynchTask class
- Subclass
- Background method
- Use from calling class
  - (note execute() method)..

To implement? - example

```
 private class SomeSlowTaskName extends AsyncTask <URL, Integer,
      Long>  // NOTE: Use of java generic  types
{
     protected Long doInBackground(URL... urls) {
         int count = urls.length;
         long totalSize = 0;
         for (int i = 0; i < count; i++) {
             totalSize += Downloader.downloadFile(urls[i]);
             publishProgress((int) ((i / (float) count) * 100));
             // Escape early if cancel() is called
             if (isCancelled()) break;
         }
         return totalSize;
     }

     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }
     protected void onPostExecute(Long result) {
         showDialog("Finished " + result ;
     }
 }
```

Asynch Task <generic types>

The three types used by an asynchronous task are the following:

`1.Params`, the type of the parameters sent to the task upon execution.

`2.Progress`, the type of the progress units published during the background computation.

`3.Result`, the type of the result of the background computation.

Asynch Task versus Intent Service

Use an `AsyncTask` for short repetitive tasks that are tightly bound to an activity, like what you're currently trying to do.

e.g. dB Connection

`IntentService` are more geared towards scheduled tasks (repetitive or not) that should run on the background, independent of your activity.

e.g.Sports app getting scheduled updates of info from a remote dB

Remainder of slides are for reference only

Services in Android

A `Service` is an application component that can perform long-running operations in the background - no user interface.

Continues to run - *even if the user switches to another application.*

Runs on the main (UI) thread, *unless you create another thread for it*

Examples : handle network transactions, play music, perform file I/O, interact with a content provider, all from the background.

Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).

Intent Service

The IntentService - a special type of Service used to perform a certain task in the background.

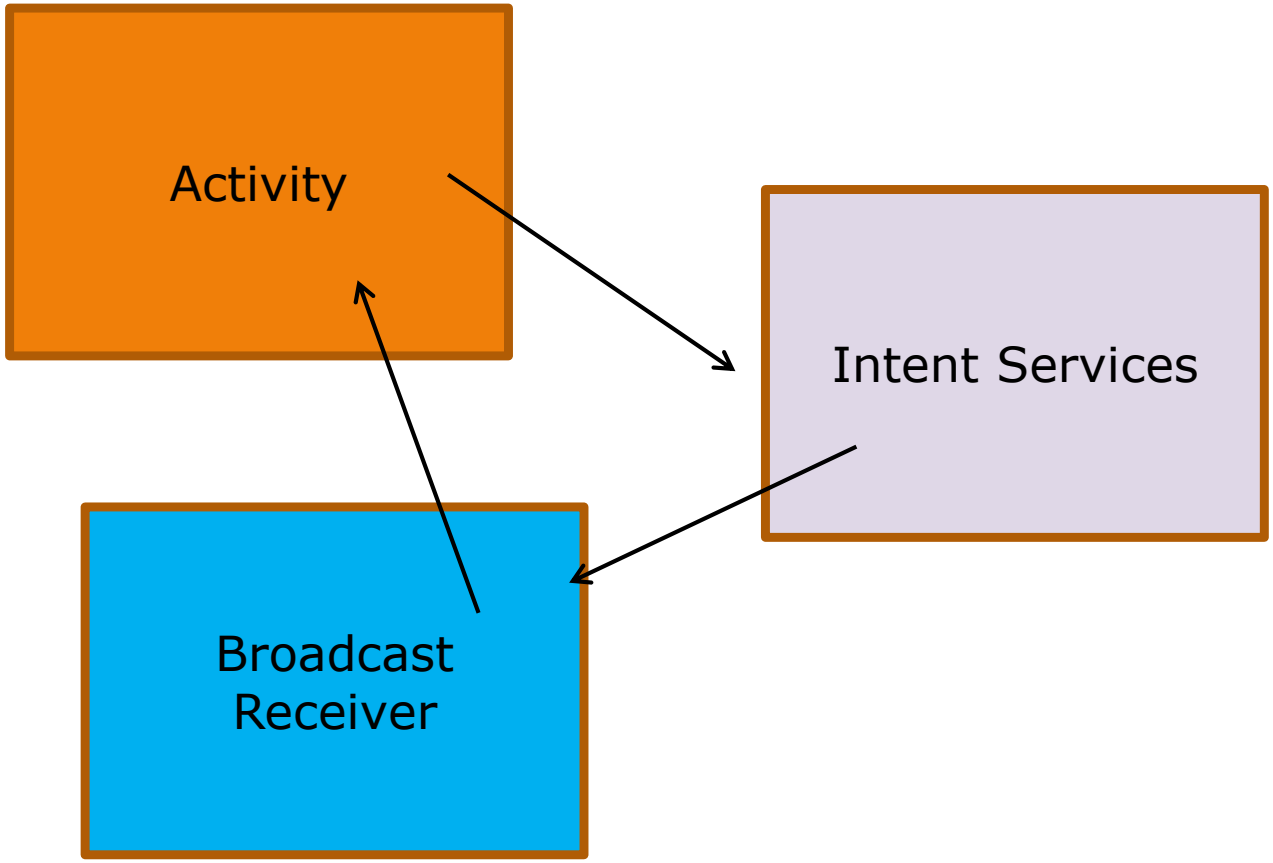
Once done, the instance of IntentService *terminate itself* automatically (Services don't)

Runs on its own thread so doesn't block the user interface (Services don't by default)

Examples for its usage would be to download a certain resources from the Internet.

Intent Service

Activity.. offloads to **Intent service**.. which can broadcast when it has finished.. **Broadcast receiver picks** that up...and notifies the activity



Intent Service – to implement

```
public class ExampleIntentService extends IntentService {
    public static final String PARAM_IN_MSG = "some input";

    public ExampleIntentService() {
        super("ExampleIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {

    // Put background processing here..
    }
}
```

Intent Service – and call from an activity

Instantiate an intent – and start it.. familiar?

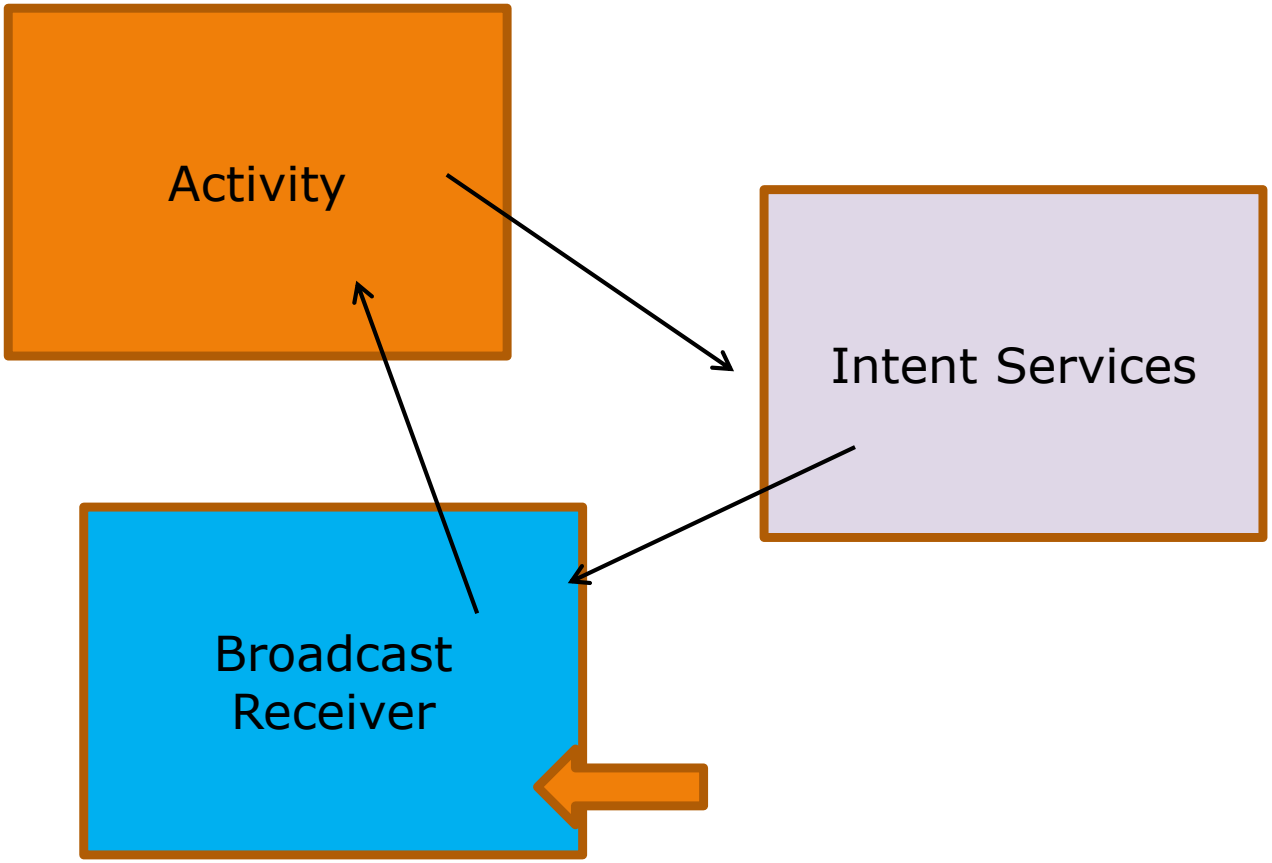
```
Intent myIntent = new Intent(this,
     ExampleIntentService.class);

// send in some data if needed..
myIntent.putExtra(SimpleIntentService.PARAM_I
N_MSG, strInputMsg);

startService(myIntent);
```

What happens if Activity needs to know what happened?

Would need a broadcast receiver..




```
public class ResponseReceiver extends BroadcastReceiver {
      @Override
    public void onReceive(Context context, Intent intent) {
    etc
 }
}
```

Asynch Task versus Intent Service

Use an `AsyncTask` for short repetitive tasks that are tightly bound to an activity, like what you're currently trying to do.

e.g. dB Connection

!!! Activities and asynchtask can become detached – see overleaf.

`IntentService` are more geared towards scheduled tasks (repetitive or not) that should run on the background, independent of your activity.

e.g.Sports app getting scheduled updates of info from a remote dB

!!! If your activity closes or refreshes, your service is not ended because of that!!!!

Activity lifecycle vs Asynch Task

AsyncTasks don't follow Activity instances' life cycle.

Unexpected behaviour

e.g. If you start an AsyncTask inside an Activity and you rotate the device, the Activity will be destroyed and a new instance will be created. But the AsyncTask will not die. It will go on living until it completes. And when it completes, the AsyncTask won't update the UI of the new Activity. Indeed it updates the former instance of the activity that is not displayed anymore.

Memory leak issue

Usually create AsyncTasks as inner classes of your Activities.
Inner classes can access directly any field of the outer class.

Downside, it means the inner class will hold an invisible reference on its outer class instance : the Activity.
Potential for memory leak if if the AsyncTask lasts for long, it keeps the activity "alive" - The activity can't be garbage collected

Questions…

What's the name of the default thread used by activites in Android

Give three examples of slow processing that could benefit from asynchronous processing

List two way so implementing background processing in Android

What does a broadcast receiver do?

Which mechanism for background processing would you use for:

- An activity to download a file, showing the progress bar, and use the downloaded data straightaway;
- A background upload of data to a remote server, kicked off when the user opens a screen. User is unaware of it.
- an activity to kick off a piece of music to play in the background – until the music is finished