ITSMAPF13 Lesson 6

Service
Threading and task.
Process, Application..
Jesper Rosholm Tørresø

Where is the main method?

- How do I control the execution of my App?
- How does Android FW control my App?
- Once again!
  - ***You should know the Android App Architecture!***

http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html

Concurrency

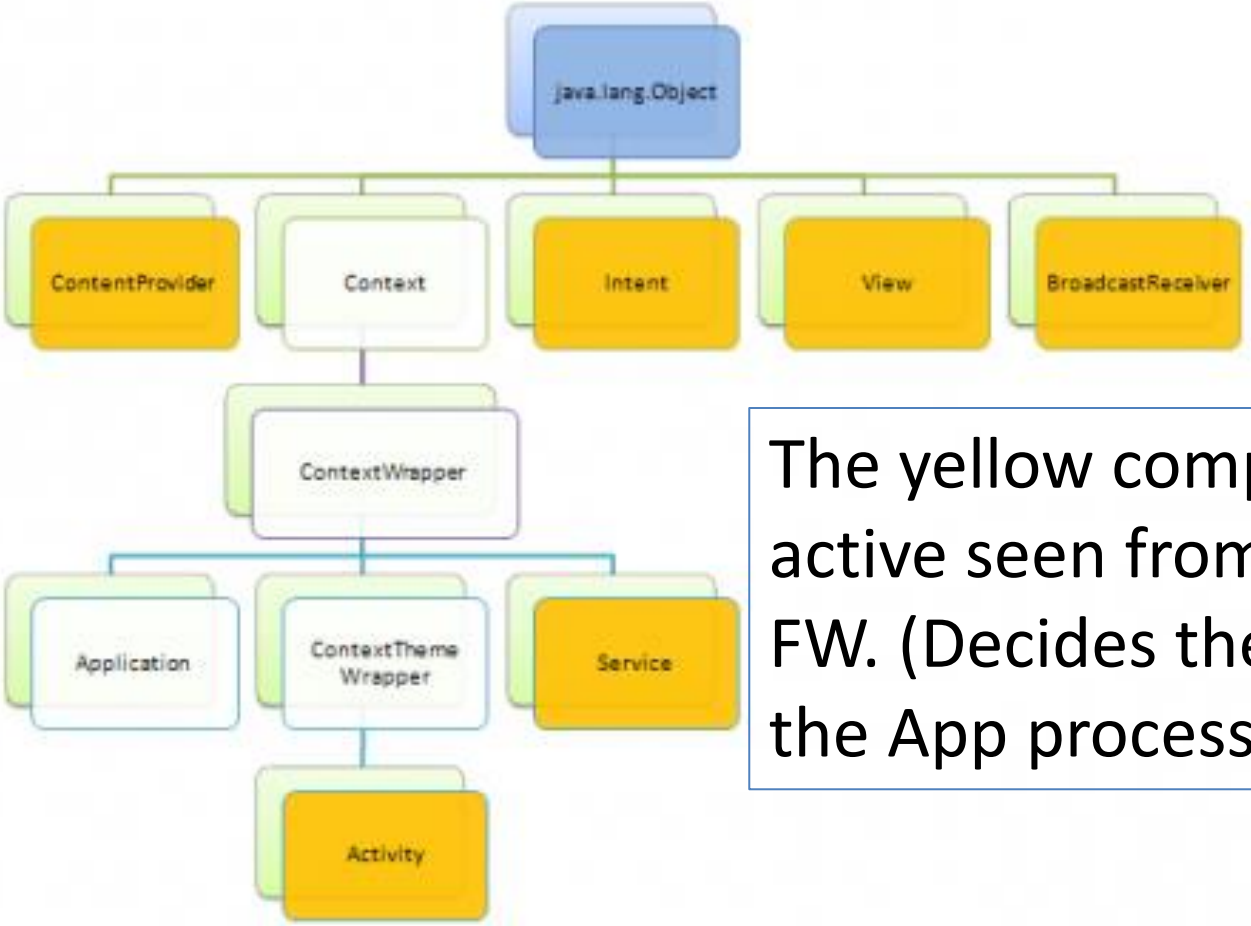
- When writing a SW program we normally got two ways of handling concurrency/multitasking
  - A process or more processes
  - A thread or some more threads in a process

The APP overall Architecture
The Process

- An App runs as one process or if you want more processes on the Linux OS .
- You can specify the process scope to others App on the Smartphone/Tablet and Apps can share a process.
  - The Dalvik JVM has build in features for inter process communication.
- You can share data between Apps ex by using a ContentProvider.

Android App Class Architecture

The running components



The yellow components are active seen from the Android FW. (Decides the active state of the App process)

http://vladnevzorov.com/2011/05/18/android-application-architecture-part-iii-main-parts-of-the-applications/

Here we see at Activity with Service

- A Service may be used for
  - Moving different task out of user focus.
  - Share logic
    - between an Apps Activities (Bound Service typically)
    - to other Apps. (Started Service typically)
- Starting up a long term Background Task
  - You may also start up short term Background Task in an Activity
- Short Term / Long Term? specific to App Use Cases
  - Android had some requirements for responsiveness (5 sec)

The APP android:process

```
<!-- From Manifest -->
<application|activity|service|receiver|provider
        android:name="ThemeApplication"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:process=":ThemeApp“|"themeApp“ >
```

- **The name of a process where all components of the application should run**. Each component can override this default by setting its own process attribute. By default, Android creates a process for an application when the first of its components needs to run. All components then run in that process. **By setting this attribute to a process name that's shared with another application,** you can arrange for components of both applications to run in the same process — but only if the two applications also share a user ID and be signed with the same certificate.
- **If the name assigned to this attribute begins with a colon (':'),** a new process, **private** to the application, is created when it's needed.
- **If the process name begins with a lowercase character, a global** process of that name is created. A global process can be shared with other applications, reducing resource usage.

http://developer.android.com/guide/topics/manifest/application-element.html

Android Process lifecycle

Processes are ranked according to components running in it

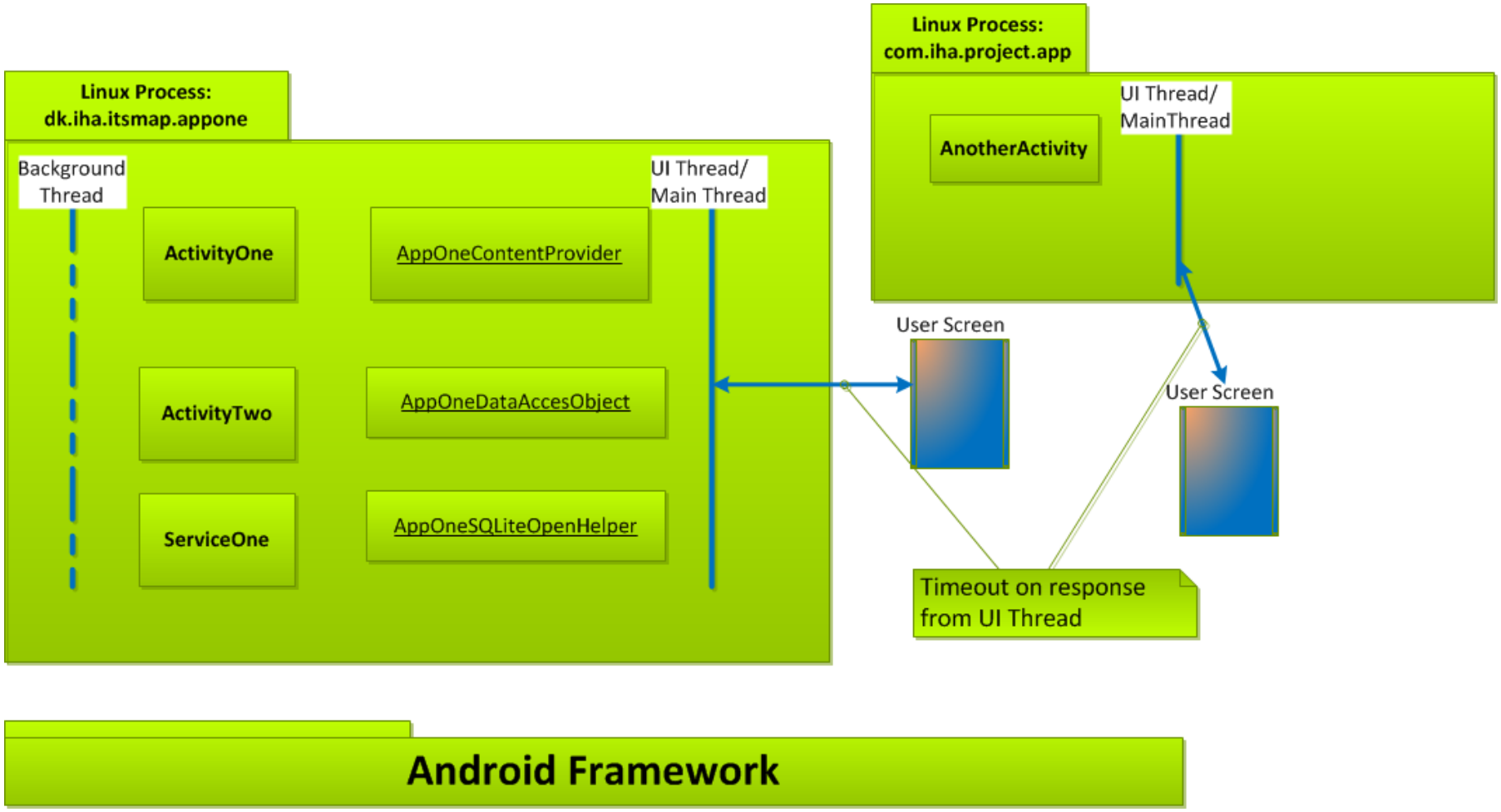
1. **Foreground process** A process that is required for what the user is currently doing. Ex an Activity.
2. **Visible process** A process that doesn't have any foreground components, but still can affect what the user sees on screen. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. (Rare)
3. **Service process** A process that is running a service that has been started with the startService() method and does not fall into either of the two higher categories.
4. **Background process** A process holding an activity that's not currently visible to the user (the activity's onStop() method has been called). Has no running Services
5. **Empty process.** A process that doesn't hold any active application components (Cached process)

One reason why all changes between component are done through the Framework

http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html

In the Process: Thread(s)+components

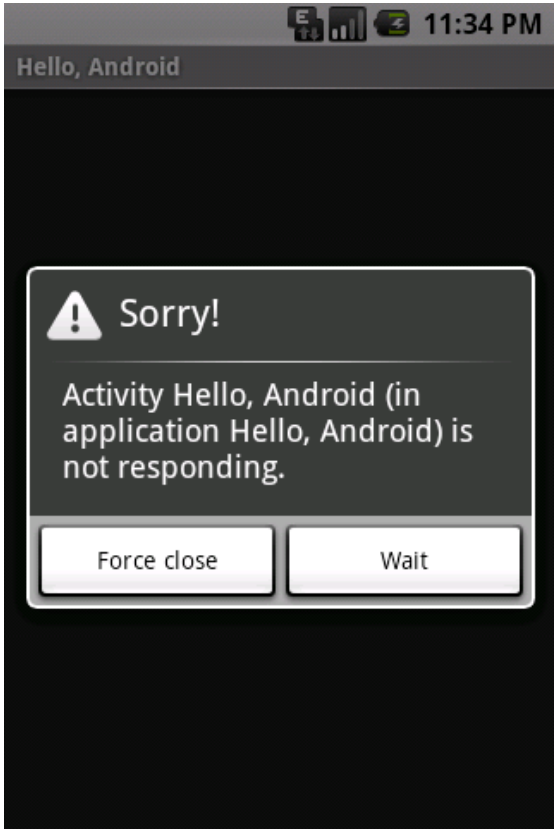
Normally one thread is assumed appropriate:
UI/Main thread



Decoupling the Main/UI Thread

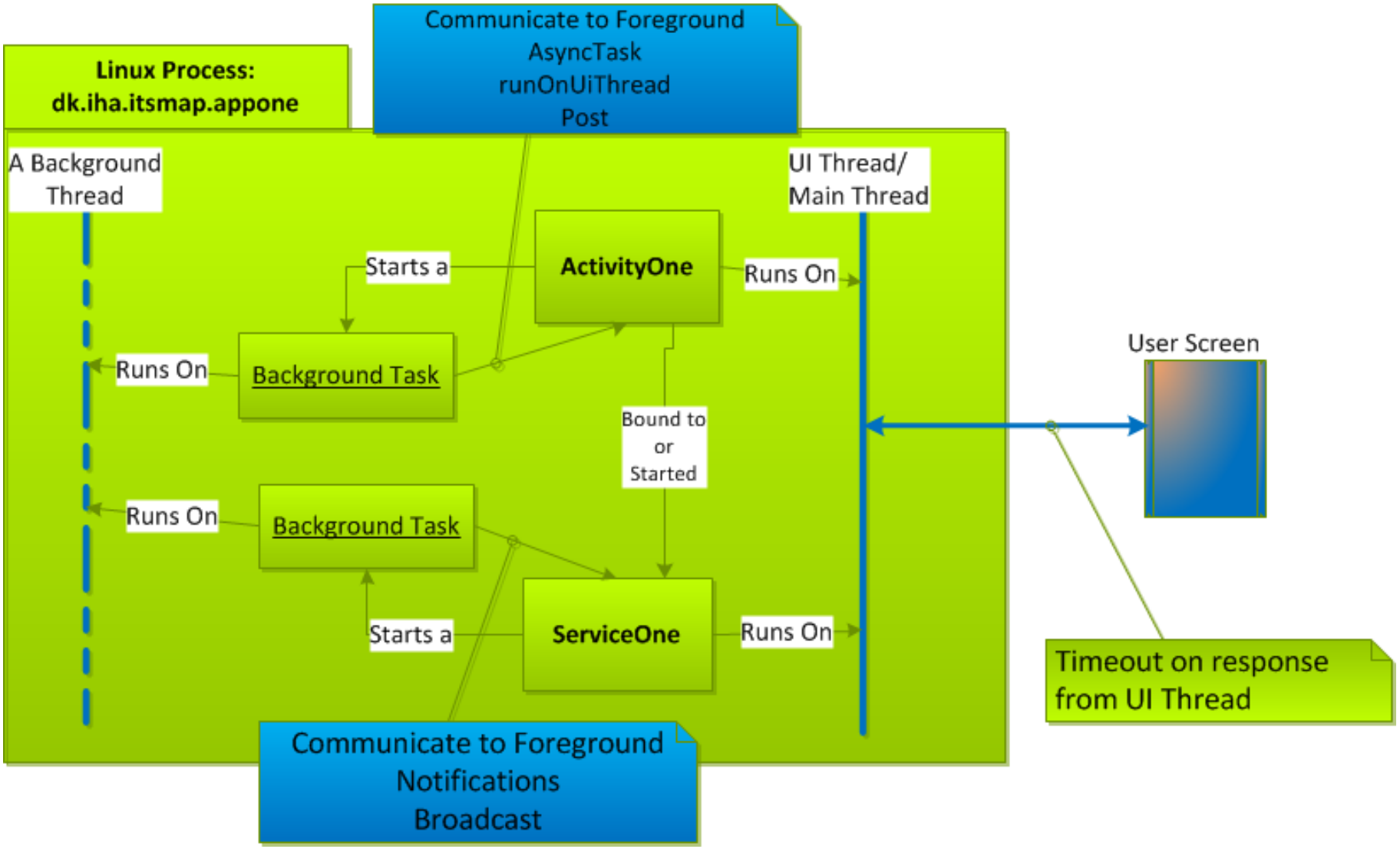
Background tasks

- Time consuming tasks must be decoupled from UI thread otherwise a Application Not Responding (ANR) dialog appears to user if the UI thread not responses within 5 sec.
- A lot of actions in a App depends not on the Users events. Service is a candidate for handling these actions.



http://developer.android.com/guide/components/processes-and-threads.html
http://developer.android.com/guide/practices/design/responsiveness.html

Foreground Background
Move of control to Background



Time consuming tasks in Activities or “no threaded” Services

- Use the Android AsynchTask.
  - The “*Single Thread approach*” means the UI task **is not synchronized** (.. Avoid normal threading!!!)
  - The AsyncTask has full synchronized access to GUI/Views

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
     protected Bitmap doInBackground(String... urls) {
         return loadImageFromNetwork(urls[0]);
     }

     protected void onPostExecute(Bitmap result) {
         mImageView.setImageBitmap(result);
     }
 }
```

http://developer.android.com/guide/components/processes-and-threads.html

Use a service for

Non user driven tasks in background
Sharing common functionality

- Two ways to start and contact a Service
  - Use startService() method, “Run Along”, and sent an Intent telling the Service what to do. Service may run forever until stopped by itself or another component.
    - The Started Service Approach
  - Bind to a Service interface bindService(). Use a reference to the IF, “Keep in touch”.
    - The Bound Service Approach. Threaded or Not Threaded
- A Service may provide both approaches!

http://developer.android.com/guide/topics/fundamentals/services.html

Service Scope: Declaration

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Enabling or disabling export of Service
android:exported =“true”
android:exported =“false”

Intent Filter may be used

Service Scope: Call

```
@Override
protected void onStart() {
   super.onStart();
   Intent intent = new Intent(this,
                              ExampleService.class);

// Eventually Bind to LocalService
   bindService(intent, mConnection,
               Context.BIND_AUTO_CREATE);
}
```

- Home App.

- Other Apps

```
// Make sure the service is started.  It will continue running
// until someone calls stopService().
// We use an action code here, instead of explictly supplying
// the component name, so that other packages can replace
// the service.
startService(new Intent(
                  "com.example.android.apis.app.REMOTE_SERVICE"));
```

Sending an Intent

(“Started Service”)

Extend the Android IntentService, if you do not need multi threaded Service request handling.

Deal with Constructor an onHandleIntent() methods

```
public class HelloIntentService extends IntentService {

  /**
   * A constructor is required, and must call the super IntentService(String)
   * constructor with a name for the worker thread.
   */
  public HelloIntentService() {
      super("HelloIntentService");
  }

  /**
   * The IntentService calls this method from the default worker thread with
   * the intent that started the service. When this method returns, IntentService
   * stops the service, as appropriate.
   */
  @Override
  protected void onHandleIntent(Intent intent) {
      // Normally we would do some work here, like download a file.
      // For our sample, we just sleep for 5 seconds.
      long endTime = System.currentTimeMillis() + 5*1000;
      while (System.currentTimeMillis() < endTime) {
          synchronized (this) {
              try {
                  wait(endTime - System.currentTimeMillis());
              } catch (Exception e) {
              }
          }
      }
  }
}
```

Here: Parse Intent Sent

Replying by sendBroadcast(retint);

IntentService

The `IntentService` does the following:

- Creates a default worker thread that executes all intents delivered to `onStartCommand()` **separate from your application's main thread**.
- Creates a **work queue** that passes one intent at a time to your `onHandleIntent()` implementation, so you never have to worry about multi-threading.
- **Stops the service** after all start requests have been handled, so you **never have to call** `stopSelf()`.
- Provides default implementation of `onBind()` that returns null.
- Provides a **default implementation** of `onStartCommand()` that sends the Intent to the work queue and then to your `onHandleIntent()` implementation.

Standard Service

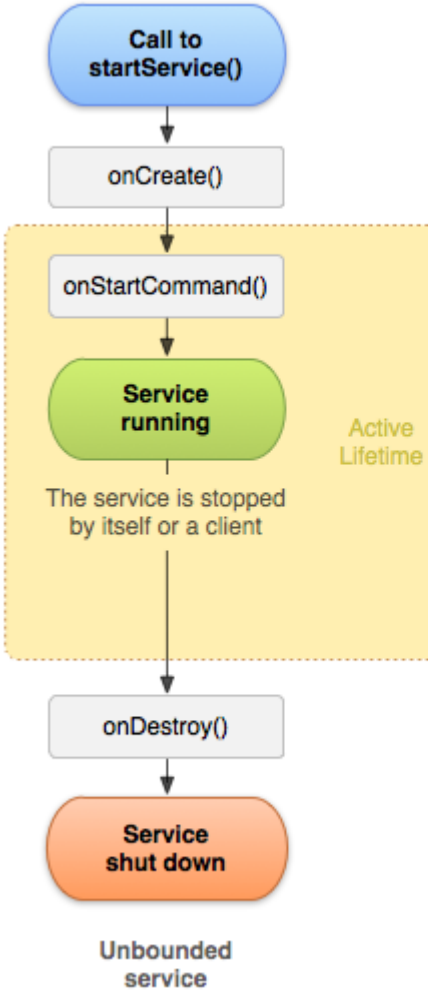
(If multi threading is needed)

Implement methods as shown in state diagram

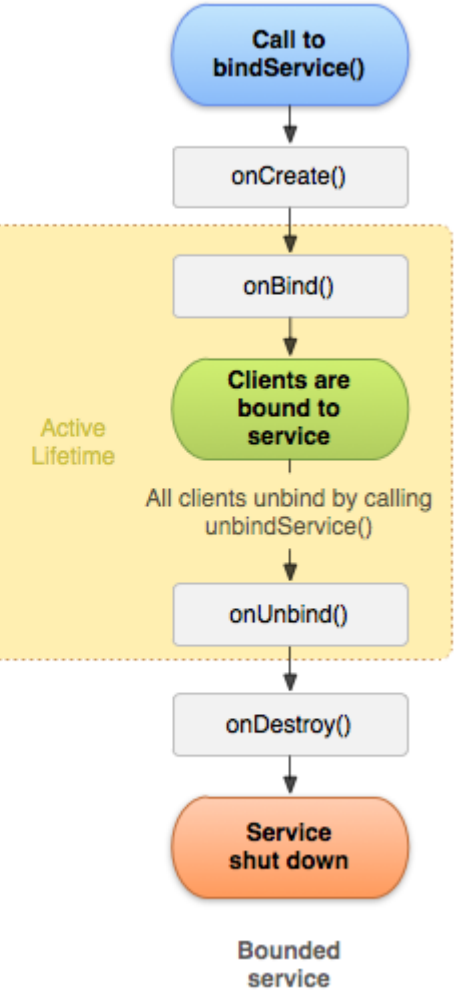
Follow one of the many guidelines like

http://developer.android.com/guide/topics/fundamentals/services.html#ExtendingService

Use ServiceHandler and HandlerThread clases



Bound Service



Implement methods as shown in state diagram

Follow one of the many guidelines like

http://developer.android.com/guide/topics/fundamentals/bound-services.html

Differ between App local or APP global Service

- Local: Extend Binder class
- Global: Use Messenger class

Foreground/Background Service

Use startForeground() + a notification
“Keep users focus”

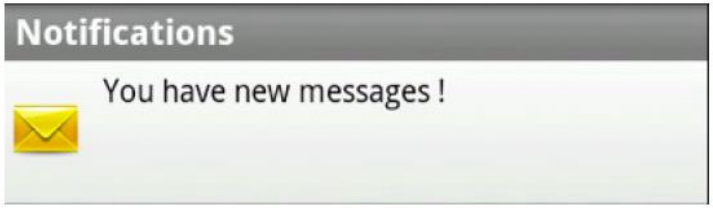
```
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker_text),
        System.currentTimeMillis());
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(this, getText(R.string.notification_title),
        getText(R.string.notification_message), pendingIntent);
startForeground(ONGOING_NOTIFICATION, notification);
```

Move in background
Use stopForeground(true)
“Remove users focus”

Services communicating to user

Toast or StatusBar Notifications

- Toast .. We have been there
- Notifications
  - Status bar popups
  - Message in notification leads to activity
  - Can be simple text or a custom view
  - No UI interaction in view



Notifications

- *NotificationManager*

```
mManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
```

- *Notification(int, CharSequence, long)*
- Updating: *setLatestEventInfo()*
- Notifying: *notify(int, Notification)*
- Cleaning up.. *cancel(int)* method or use flags

Lesson 6 Exercise 1

- Create a service which is bindable and has a method to update a value
- Create activity and bind to the service and set value
- Service should show notification when value is set and update the notification text when values are updated