# Mobile app development – week 10

Android components

    Services (cont.)

RESTful interfaces in Android

Android components

    Content Providers

Second mandatory assignment

Exercises

## Course evaluation: THIS WEEK

Jan Leschly, CEO /
Tennis pro (ATP 10)
If you are not counting points,
you are just warming up

## Changes to the schedule

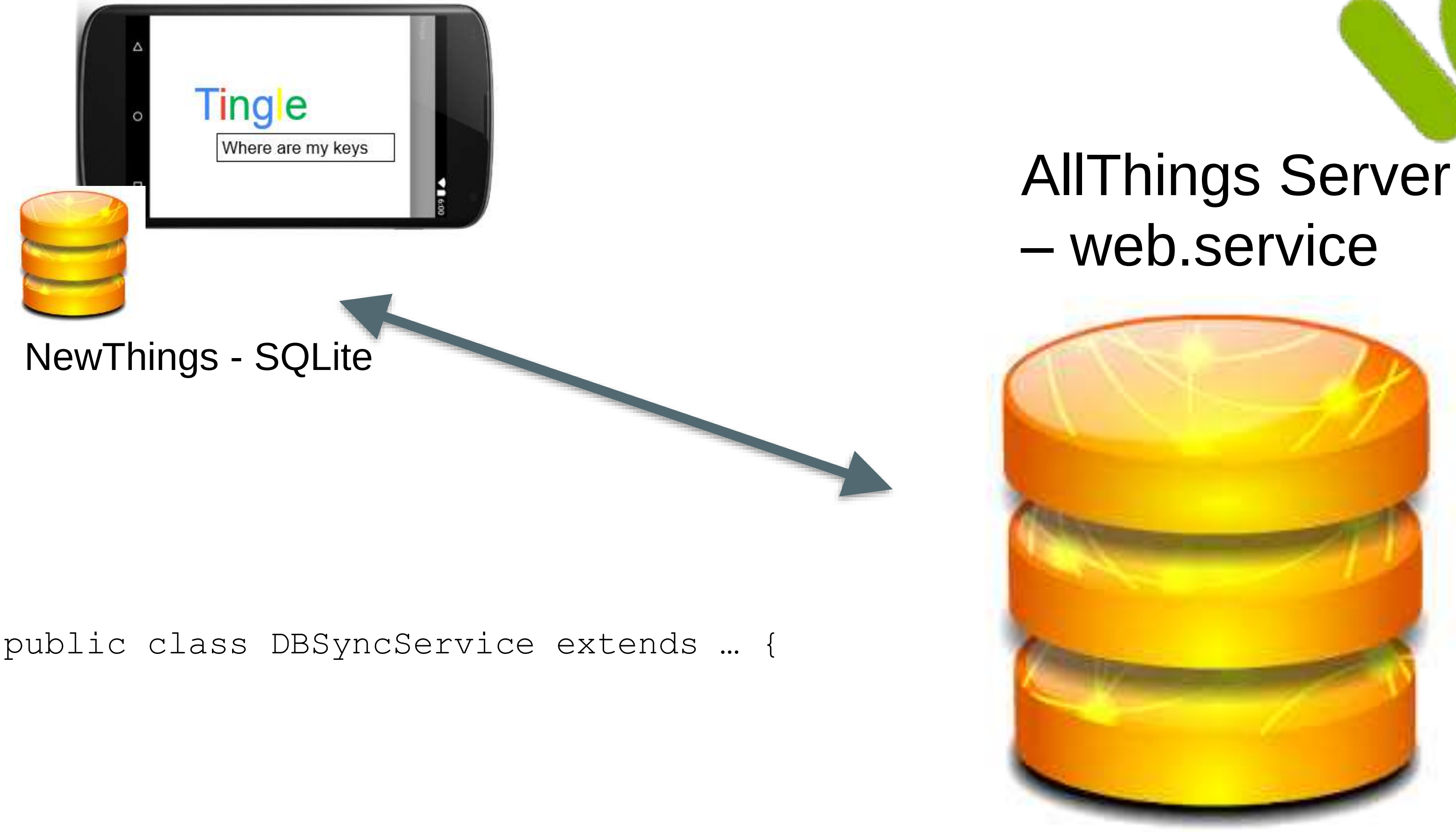
**No lecture** on April 20, but

exercises from 17 - 21

**Extra exercises** from April 13 to 27

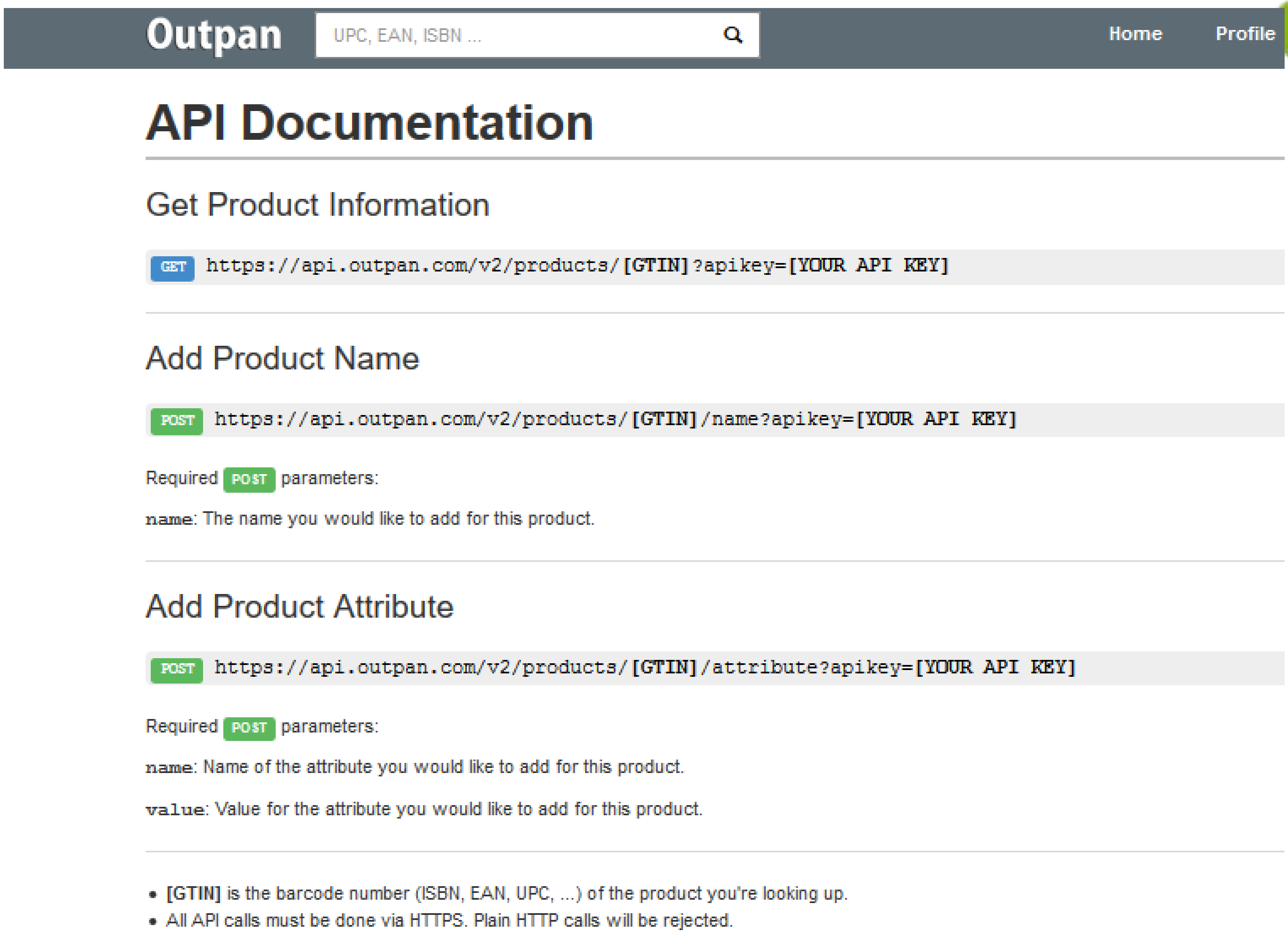
details in worplanW1112

## Synchronization service



```
public class DBSyncService extends … {



  }
```

# Web-service example: Outpan



Outpan | UPC, EAN, ISBN ... | Home | Profile

## API Documentation

### Get Product Information

GET `https://api.outpan.com/v2/products/[GTIN]?apikey=[YOUR API KEY]`

### Add Product Name

POST `https://api.outpan.com/v2/products/[GTIN]/name?apikey=[YOUR API KEY]`

Required POST parameters:

`name`: The name you would like to add for this product.

### Add Product Attribute

POST `https://api.outpan.com/v2/products/[GTIN]/attribute?apikey=[YOUR API KEY]`

Required POST parameters:

`name`: Name of the attribute you would like to add for this product.

`value`: Value for the attribute you would like to add for this product.

- [GTIN] is the barcode number (ISBN, EAN, UPC, ...) of the product you're looking up.
- All API calls must be done via HTTPS. Plain HTTP calls will be rejected.

## Web services

Exploiting HTTP (ment for browsing) for data exchange:

- SOAP-based services

  XML-based, strongly typed
  => tight coupling of server and client

- REST-based (“RESTful”) services

  Text (URL), JSON (XML), not typed, simple to implement widely used

https://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs--rest-challenges.html

http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/#_ga=1.24995194.1380490897.1460355635

# REST using HTTP protocol in Android (GET)

GET requests do not contain a body and should be idempotent. Parameters are sent in the request line e.g.

https://www.outpan.com/view_product.php?barcode=7311310025250

Listing 23.3 Basic networking code (FlickrFetchr.java)

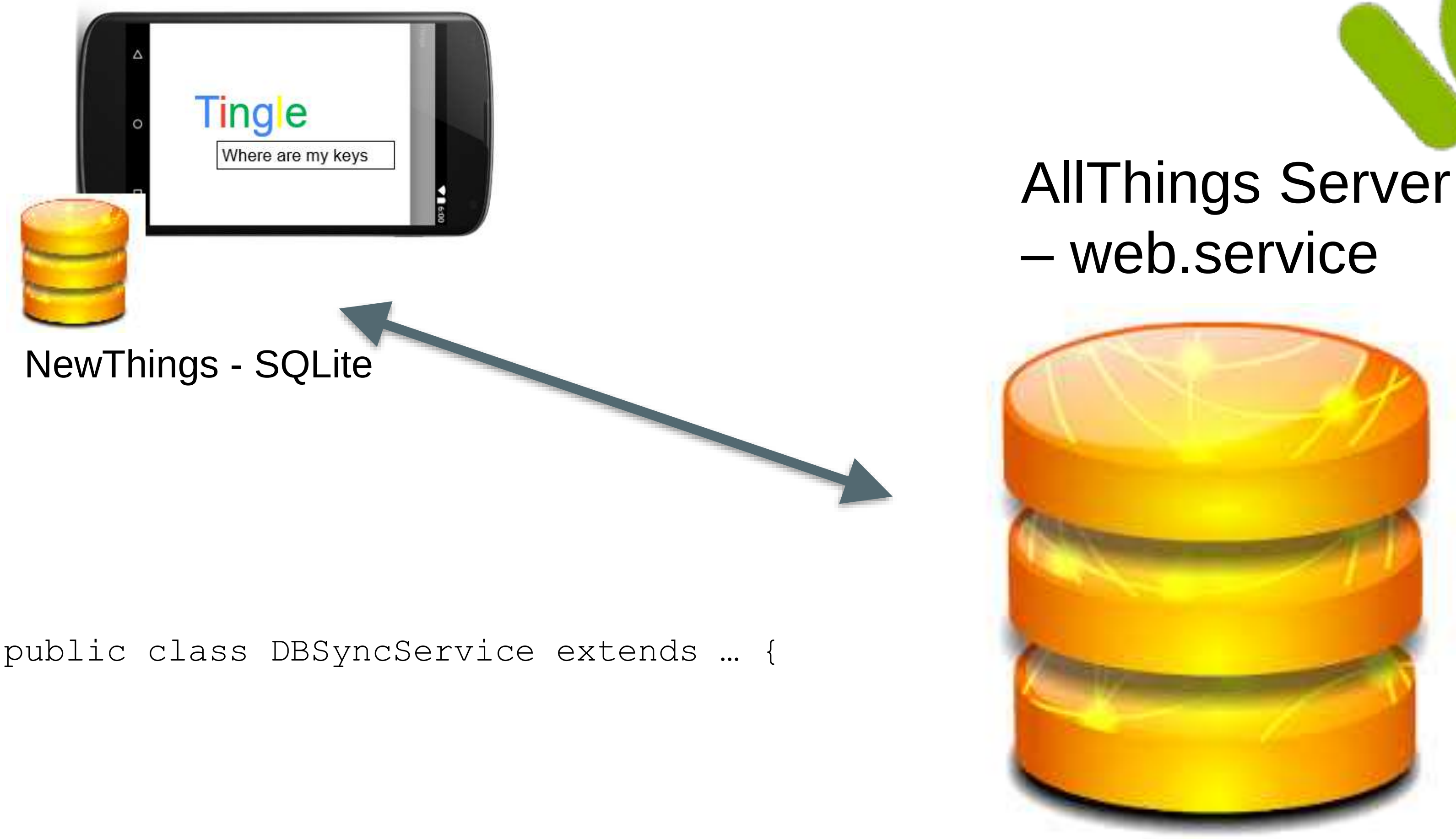
```
public class FlickrFetchr {
  public byte[] getUrlBytes(String urlSpec) throws IOException
   {
  URL url = new URL(urlSpec);
  HttpURLConnection connection =
   (HttpURLConnection)url.openConnection();
  try {
       ByteArrayOutputStream out = new ByteArrayOutputStream();
       InputStream in = connection.getInputStream();
            …
       return out.toByteArray();
  } finally { connection.disconnect(); }
  }    …
}
```

# REST using HTTP protocol in Android (POST)

POST requests may contain data in the body - even large amounts of binary data. Parameters are sent in the body.

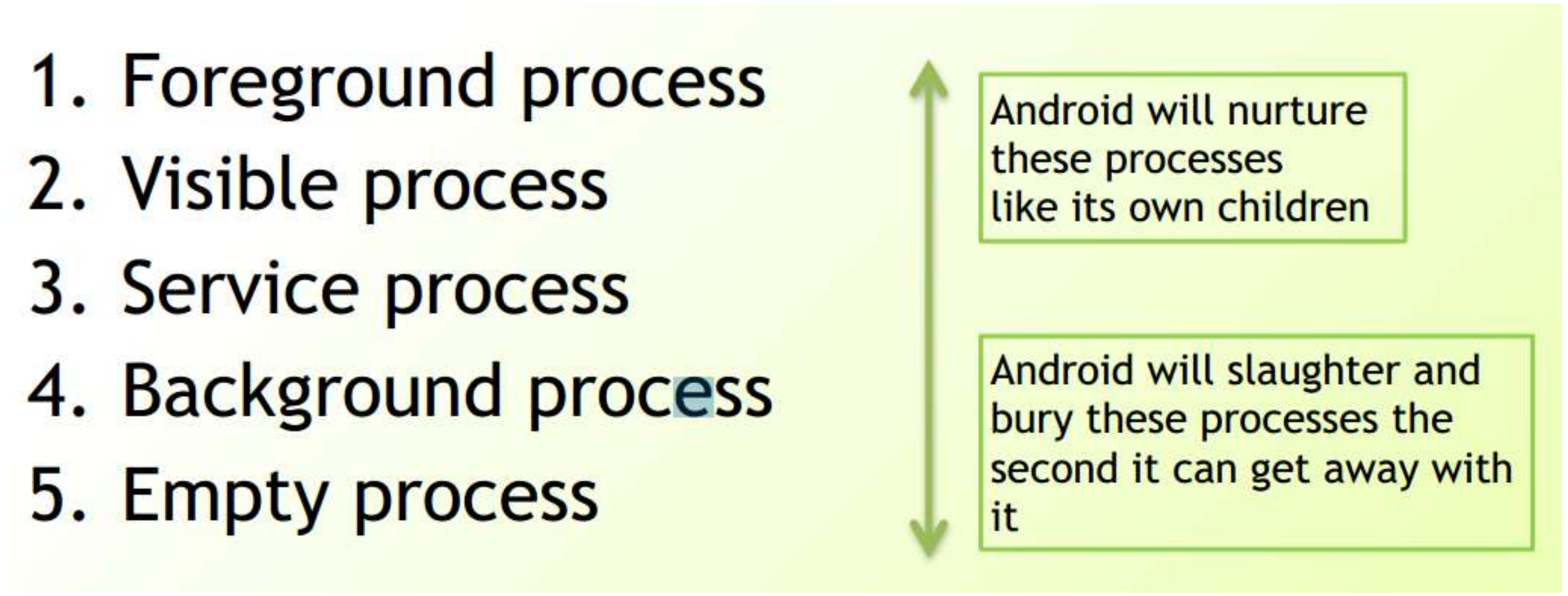
```
URL url = new URL(resource);
URLConnection connection = url.openConnection();

((HttpURLConnection)connection).setRequestMethod("POST");

OutputStream outputStream = connection.getOutputStream();
PrintWriter writer = new PrintWriter(outputStream);
writer.write(queryString);
writer.flush();
writer.close();

InputStream inputStream = connection.getInputStream();
BufferedReader reader = new BufferedReader(
    new InputStreamReader(inputStream));

String result = reader.readLine();
```

# Synchronization service



```
public class DBSyncService extends … {



  }
```

# Process states for an Android app



1. Foreground process
2. Visible process
3. Service process
4. Background process
5. Empty process

↕

Android will nurture these processes like its own children

Android will slaughter and bury these processes the second it can get away with it

## Synchronization service

### Introduce a syncButton

```
syncButton.setOnClickListener(new View.OnClickListener() {
  @Override public void onClick(View view) {
    Intent i= DBSyncService.newIntent(getActivity());
    getActivity.startservice(i);
  }
});
```

```
public class DBSyncService extends IntentService {

      @Override
      public void onHandleIntent(Intent i){
       …
      }
  }
```

# ThingsDB

```
public class ThingsDB {
  private static ThingsDB sThingsDB;

  private Context mContext;
  private SQLiteDatabase mDatabase;

  public static ThingsDB get(Context context) {
    if (sThingsDB == null) sThingsDB= new ThingsDB(context);
    return sThingsDB;
  }

  public void addThing(Thing thing) {
    …
  }
  public void deleteThing(ThingID id) {
    …
  }
  public void Sync() {
    …
  }
….
```

## Things to consider

User pressing Sync button several times

Network failure while syncing

User adding/deleting while syncing

…

# ThingsDB shared synchronized singleton ?

```
public class ThingsDB {
  private static ThingsDB sThingsDB;

  private Context mContext;
  private SQLiteDatabase mDatabase;

  public static synchronized ThingsDB get(Context context) {
    if (sThingsDB == null) sThingsDB= new ThingsDB(context);
    return sThingsDB;
  }

  public void synchronized addThing(Thing thing) {
    …
  }
  public void synchronized deleteThing(ThingID id) {
    …
  }
  public void synchronized Sync() {
    …
  }
…
}
```

# Android components

- Activities
- Services
- Broadcast Receivers
- Content Providers

An Android app consists of one or more components that all run in the same (Linux) process – separated from other processes/apps.

http://developer.android.com/guide/components/fundamentals.html

## Services

- runs in the background,
  often while no activity in the app is active

- roughly an activity without a UI.

- run on the *main* thread, *not* in a worker thread.

```
public class MyService extends Service {
  @Override
  public void onCreate() { … }
    …
  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    …
    return …;
  }
 …  more @Override
}
```

## Basic service functionality

- needs to be started by another component `startService // stopService` providing an intent

- a service can also call `stopSelf`

```
 startService(intent);    -------→   onCreate()


                                 -------→  onStartservice(intent)
```

# Implementing a service

A service must be registered in the manifest:

```
<service android:name= ".MyService">
```

`onCreate` and `onDestroy` lifecycle methods (optional)

`onStartCommand`

onStartCommand returns an integer value that dictates what should happen in case the service is killed by Android after the method has returned:

- START_NOT_STICKY: the service is *not* re-created, unless there are pending intents to deliver.
- START_STICKY: the service is re-created, and onStartCommand is called with a null intent (unless there are intents to deliver).
- START_REDELIVER_INTENT: the service is re-created, and onStartCommand is called with the last intent delivered.

## Service example

```
public class InternalService extends Service {


  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {

    Toast.makeText(getBaseContext(), "Running internal service.",
                   Toast.LENGTH_SHORT).show();

    return Service.START_NOT_STICKY;
    }
}
```

More complete example:

http://codetheory.in/understanding-android-started-bound-services/

## BREAK

Jan Leschly, CEO /
Tennis pro (ATP 10)
If you are not counting points,
you are just warming up

**Remember to do the course evaluation**

## Service binding

Instead of starting and stopping the service manually, components can *bind* to it.

Communication between components and the service is done through a custom interface, making a far richer data exchange with the service possible.

When all components bound to a service have been unbound, the service is destroyed.

Implement the `onBind` method in your service (instead of `onStartCommand`) to return the object your service communicates through.

If you do not want your service to support binding, then just let onBind return null.

## Example of Binding

```
public class MyService extends Service {

    private IBinder myBinder = new MyBinder();
    public MyService() {      }
    @Override
    public void onCreate() { …    }

    @Override
    public IBinder onBind(Intent intent) { return myBinder; }
    @Override
    public boolean onUnbind(Intent intent) {return false;    }

    public class MyBinder extends Binder {
        MyService getService() { return MyService.this; }
    }
    // Methods used by the binding client components

    public void methodOne() { // Some code...   }

    public void methodTwo() { // Some code...   }

}
```

http://codetheory.in/understanding-android-started-bound-services/

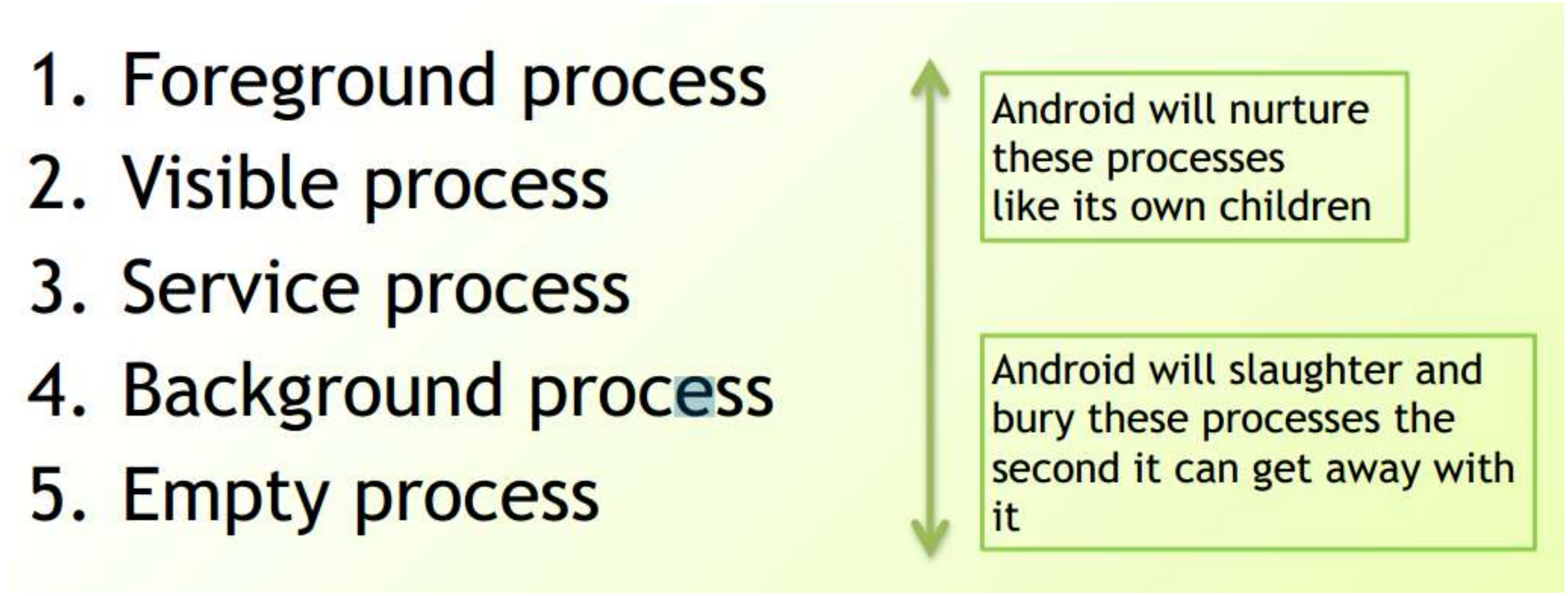
## Intent service (reminder)

To use an intent service, extend the *IntentService* class instead of the Service class, and implement the *onHandleIntent* method instead of *onStartCommand*.

Provides an easy way of creating a service that:

- Creates a worker thread
- Provides a queue that executes intents sent to it on the worker thread one at a time
- Shuts down the service when the last intent has been processed

# Process states for an Android app



1. Foreground process
2. Visible process
3. Service process
4. Background process
5. Empty process

Android will nurture these processes like its own children

Android will slaughter and bury these processes the second it can get away with it

## Foreground services

A service can forced into "foreground" state

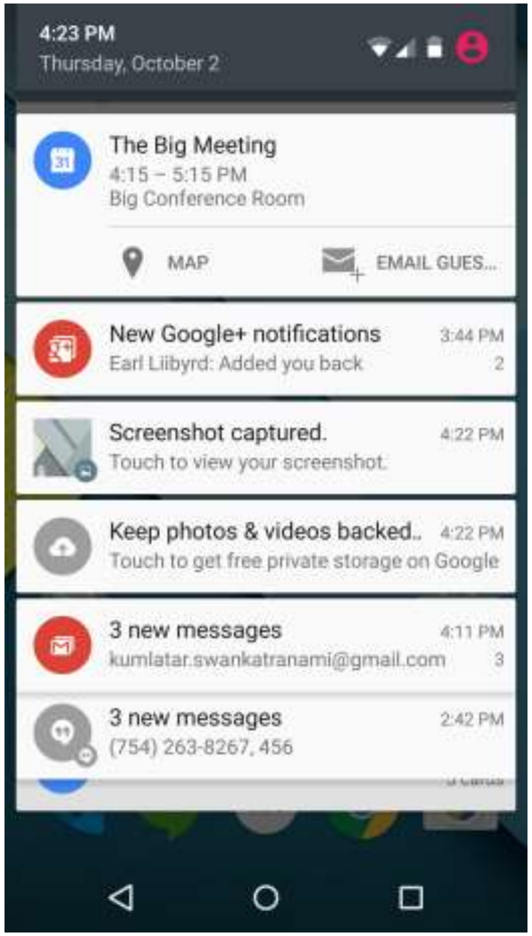
This means that the service has a higher priority and is less likely to be shut down

This *requires* that the service shows a notification so the user is aware of the foreground service

Use the methods *startForeground* and *stopForeground* in the service to toggle foreground status

# Notifications (1)

”persistent toast”



4:23 PM
Thursday, October 2

The Big Meeting
4:15 – 5:15 PM
Big Conference Room

New Google+ notifications 3:44 PM
Earl Liibyrd: Added you back 2

Screenshot captured. 4:22 PM
Touch to view your screenshot.

Keep photos & videos backed.. 4:22 PM
Touch to get free private storage on Google

3 new messages 4:11 PM
kumlatar.swankatranami@gmail.com 3

3 new messages 2:42 PM
(754) 263-8267, 456

## Notifications (1)

From listing 26.17 (textbook)

```
Notification notification = new NotificationCompat.Builder(this)
    .setTicker(resources.getString(R.string.new_pictures_title))
    .setSmallIcon(android.R.drawable.ic_menu_report_image)
    .setContentTitle(resources.getString(R.string.new_pictures_title))
    .setContentText(resources.getString(R.string.new_pictures_text))
    .setContentIntent(pi)
    .setAutoCancel(true)
    .build();

NotificationManagerCompat notificationManager =
                               NotificationManagerCompat.from(this);
notificationManager.notify(0, notification);
```

# Notifications (1)

From listing 26.17 (textbook)

```
Notification notification = new NotificationCompat.Builder(this)
    .setTicker(resources.getString(R.string.new_pictures_title))
    .setSmallIcon(android.R.drawable.ic_menu_report_image)
    .setContentTitle(resources.getString(R.string.new_pictures_title))
    .setContentText(resources.getString(R.string.new_pictures_text))
    .setContentIntent(pi)
    .setAutoCancel(true)
    .build();

NotificationManagerCompat notificationManager =
                               NotificationManagerCompat.from(this);
notificationManager.notify(0, notification);
```

## Pending intents

```
PendingIntent pendingIntent;
```

- used to fire an intent after the app is gone - which, for instance, is the case with notifications.

- wrapper around a “normal” intent. It:

- is tied to the context that created it and
- has permission to execute the contained intent as if the original app did it, even if the app has been killed by the Android system.

# Alarm manager

A system service that can be set up to fire a pending intent (and thus reach an Android component) at a given time or interval.

```
public final static int FIVE_SECONDS = 5 * 1000;
private AlarmManager alarmManager;
private PendingIntent pendingIntent;

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

    alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);

    Intent intent = new Intent(this, AlarmReceiver.class);
    pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() +
        FIVE_SECONDS, pendingIntent);
}
```

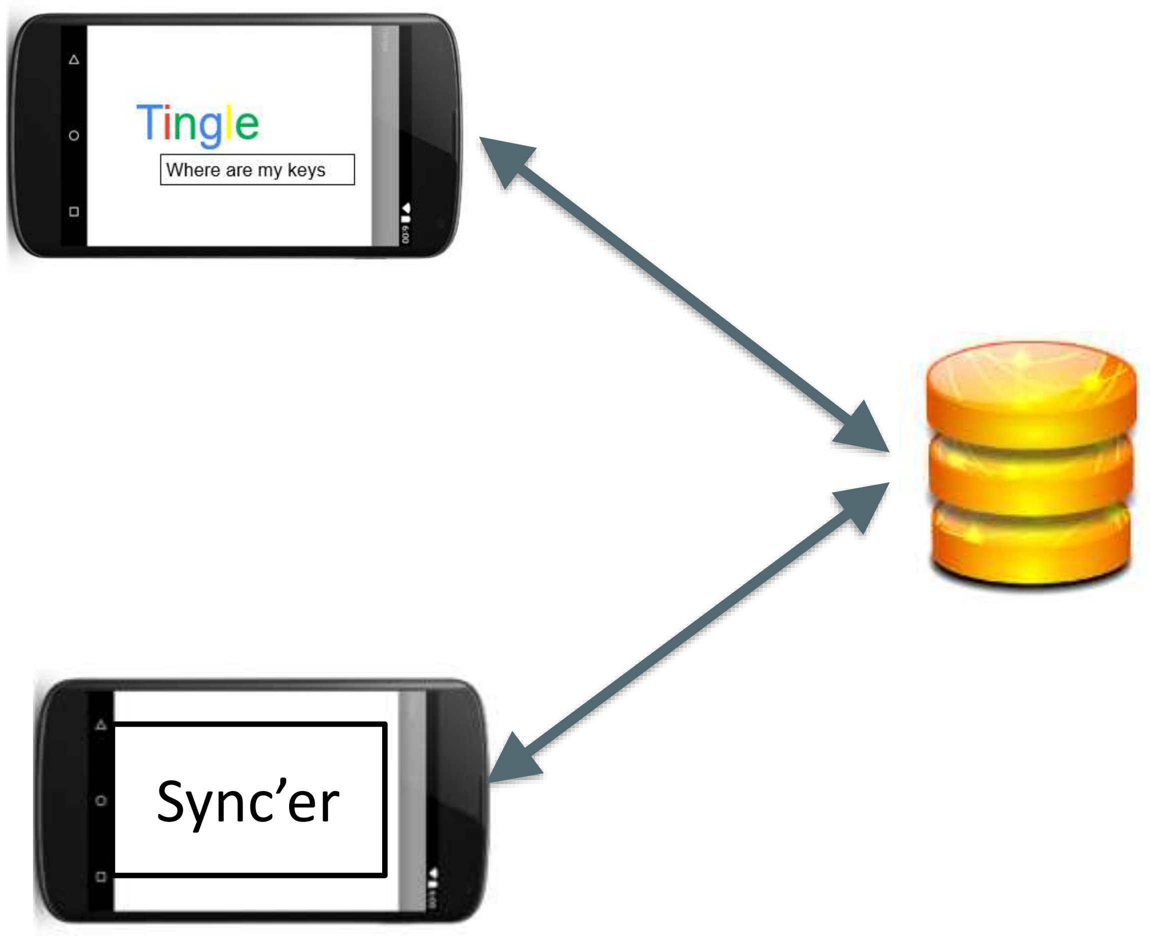
# Android components

- **Activities**
- **Services**
- **Broadcast Receivers**
- **Content Providers**

An Android app consists of one or more components that all run in the same (Linux) process – separated from other processes/apps.

[http://developer.android.com/guide/components/fundamentals.html](http://developer.android.com/guide/components/fundamentals.html)

# Synchronization in a separate app



# Content Providers

allows an app to expose information to other apps

uses concepts/syntax similar to a SQLite database

```
public class ThingProvider extends ContentProvider {

 @Override
 public boolean onCreate() { … }

 @Override
 public Cursor query(…)  { … }

 insert() update() delete()  getType()
}
```

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

http://developer.android.com/guide/topics/providers/content-providers.html

## Using a content provider

Know the URI of the provider, e.g.

```
content://com.android.contacts/contacts/lookup
content://user_dictionary/words
content://dk.staunstrups.tingle/database
```

Request permission for using the provider, e.g

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

Query the content resolver

# Content provider example

```
public class MainActivity extends ListActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);


  Cursor cursor= getContentResolver().query(Settings.System.CONTENT_URI,
                null, null, null, Settings.System.NAME);


   String[] from = {Settings.System.NAME, Settings.System.VALUE};

   int[] to = {android.R.id.text1, android.R.id.text2};
   SimpleCursorAdapter cursorAdapter= new SimpleCursorAdapter(this,
                  android.R.layout.simple_list_item_2, cursor, from, to, 0
   setListAdapter(cursorAdapter);
 }
}
```

## Second Mandatory Assignment

Any questions?