Mobile app development – week 6

Persistence/Databases
- General introduction
- SQLite

Threading (continued)

Coordination concepts: monitors and message passing

Comments related to the mandatory assignment
- Feedback
- Future of Tingle

Exercises

Persistence

- SharedPreferences
- Java files
- SQLite database
- External database (cloud)

http://developer.android.com/guide/topics/data/data-storage.html

SharedPreferences

by far the easiest way to persist data in your app

a key/value-map to store and access values from

key	value
what1	Mouse
what2	Keys
what3	Book
where1	Desktop
where2	Drawer
where3	Desktop

values are stored in an XML file located on the device in:
/data/data/[app package name]/shared_prefs/[prefs_name].xml

Example

Retrieve shared preferences object:

```
SharedPreferences prefs =
                    getSharedPreferences("my_prefs", MODE_PRIVATE);
```

Save value to shared preferences:

```
Editor editor = prefs.edit();
editor.putString("what1", "Book");
editor.commit();
```

Get specific value:

```
String prefsValue= prefs.getString("where2", null);
```

Files

- The folder res/raw may contains files

- These files can be read from using the standard Java streams and readers:

```
InputStream in = getResources().openRawResource(R.raw.myfile);
InputStreamReader reader = new InputStreamReader(in);
BufferedReader br = new BufferedReader(reader);
String txt;
try {
  while((txt = br.readLine()) != null) {
    Toast.makeText(this, txt, Toast.LENGTH_SHORT).show();
  }
} catch (Exception e) {
  Log.e("ReadFile", e.getMessage());
}
```

Internal/external storage

All Android devices have the concepts of internal and external storage

- Internal storage is (as default) private to the app and cannot be accessed from the outside

- External storage is accessible from the outside. It may be an SD card, but may also be non-removable storage

- Both types of storage can be handled with Java file handling classes e.g. File, FileInputStream and FileOutputStream

- To access the external storage, the app needs **permission** from the user

Permissions

In order to read or write files on the external storage, your app must acquire a uses-permission

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

If you need to both read and write files, then you need to request only the write permission

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="staunstrups.dk.tingle"
xmlns:android="http://schemas.android.com/apk/res/android">

<uses-permission android: name=”android.permission.WRITE_EXTERNAL_STORAGE”/>

<application …
   <activity android:name=".TingleActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
…
</activity> </application></manifest>
```

SQLite

SQLite database built into Android

• SQLite is an open source file-based relational database that is particularly popular in small, embedded systems due to its low footprint and great portability

• You'll find the database file here:

/data/data/[app package name]/databases/[database_name]

Tables

An SQLite database consists of a number of tables

_id	what	where
1	Mouse	Desktop
2	Glasses	Desktop
3	Keys	Drawer
4	Book	Desktop

That are manipulated by queries:

```
insert into things values('Pencil', 'Desktop')
```

SQLite – Android style

Helper class (creation and updating)

Cursors (results from a query)

ContentValues (Java concept: set of values)

Tingle – helper class

```
public class ThingsBaseHelper extends SQLiteOpenHelper {
  private static final String TAG = "ThingBaseHelper";
  private static final int VERSION = 1;
  private static final String DATABASE_NAME = "thingBase.db";
  public ThingsBaseHelper(Context context) {
    super(context, DATABASE_NAME, null, VERSION);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + ThingsDbSchema.ThingTable.NAME + "(" +
        " _id integer primary key autoincrement, " +
        ThingsDbSchema.ThingTable.Cols.WHAT + ", ” +
        ThingsDbSchema.ThingTable.Cols.WHERE  + ")”    );
  }

  @Override
  public void onUpgrade(SQLiteDatabase db,
                int oldVersion, int newVersion) {  }
}
```

SQLite – Android style

Helper class (creation and updating)

Cursors (results from a query)

ContentValues (Java concept: set of values)

Cursor

```
private ThingsCursorWrapper queryThings(String
whereClause, String[] whereArgs) {
  Cursor cursor = mDatabase.query(
      ThingTable.NAME,
      null, // Columns - null selects all columns
      whereClause, whereArgs,
      null, // groupBy
      null, // having
      null  // orderBy
  );
  return new ThingsCursorWrapper(cursor);
}
```

id	what	where
2	Mouse	Desktop
8	Book	Desktop

Multithreading / concurrency in Android

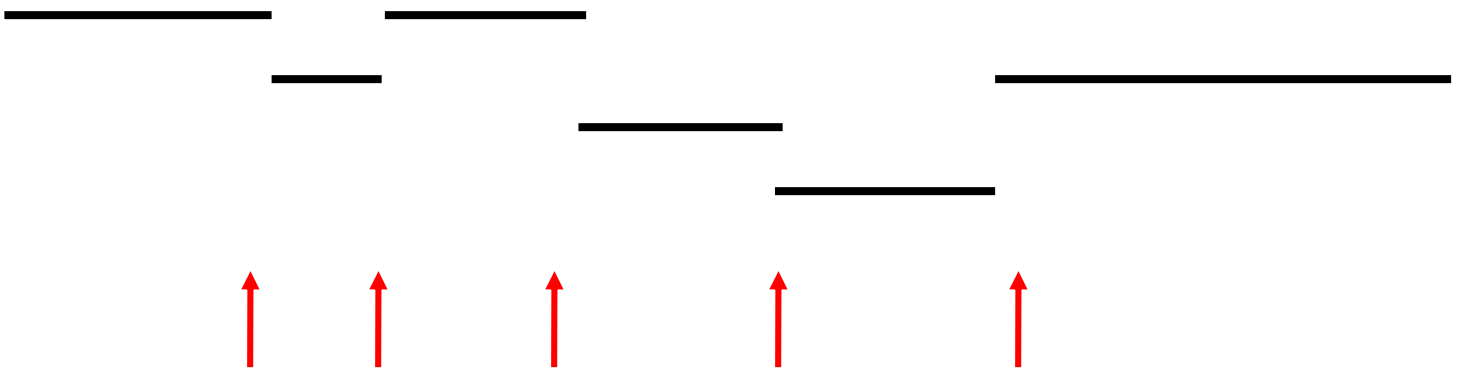
Java threads

```
class t extends … implements Runnable {
    public void run() {
            …
    }
}
```

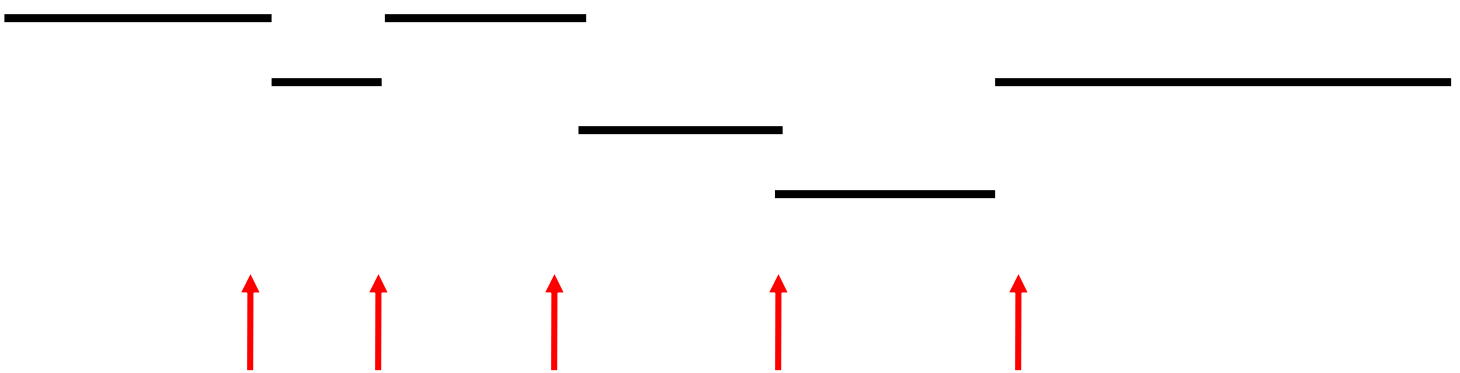
Asynctask

```
private class aT extends AsyncTask<Params, Progress, Result> {
    protected abstract Result doInBackground(Params... param) {    ….   }
    protected void onProgressUpdate(Progress  … prog  {   …  }
    protected void onPostExecute(Result  ….  result) {  …  }
}
```

Scheduling

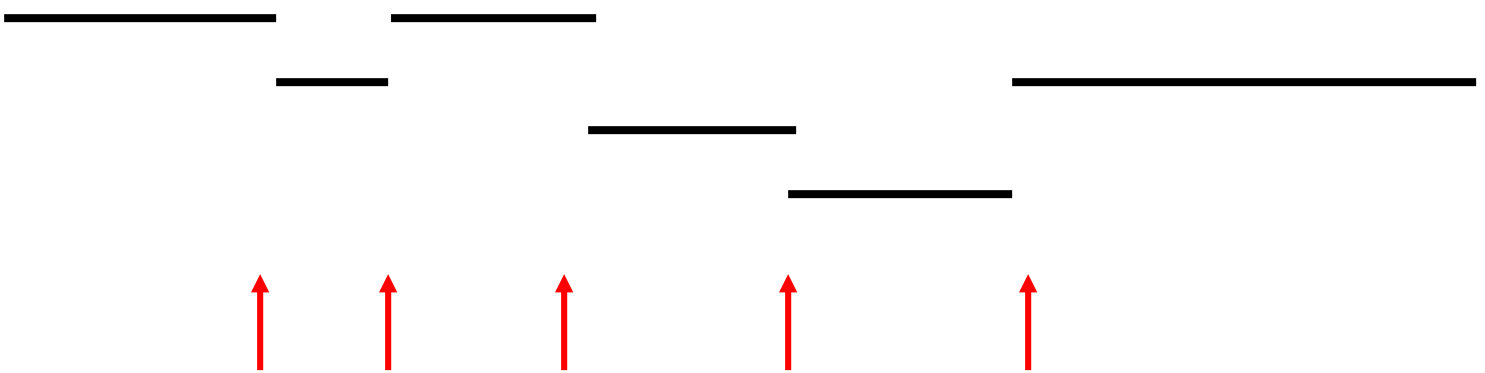


Preemptive scheduling



Enforced by underlying software (an operating system)

Cooperative scheduling



Performed by the application code

In Java: ”yield” or ”sleep”

Thread safety

```
threadA:  {
     acc1= acc1-amount;
     acc2= acc2+amount;
}
```

scheduling

```
threadB:  {
     sum= acc1+acc2;
}
```

In Android only the main thread (UI thread) can touch the user interface

Mutual exclusion / critical region

```
threadA:  synchronized (lock) {
      acc1= acc1-amount;
      acc2= acc2+amount;
}
```

```
threadB: synchronized (lock) {
      sum= acc1+acc2;
}
```

Synchronized classes in Java (monitors)

```
public class  javaMonitor {
    private int acc1=0;
    private int acc2=0;

    public synchronized void move(int amount) {
        acc1= acc1-amount;
        acc2= acc2+amount;
    }

    public synchronized int sum() {
        return acc1+acc2;
    }
}
```

Message passing

```
threadA:  {
    send(amount);
}
```

```
threadSum:  {
  int acc1=0; int acc2=0;

    wait(amount);
    acc1= acc1-amount;
    acc2= acc2+amount;
    send(acc1+acc2);
}
```

```
threadB:  {
    wait(sum);
}
```

Monitors versus message passing

Beware much nonsense written about the two

Theoretically equally powerful

Both can be implemented using shared memory or networking

Message passing is easy to mess up

Monitors can become a bottleneck

Be modest and use simple solutions

stay away from `setPriority`

Feedback mandatory assignment

Tabbing between textViews

Refreshing list (insert and delete)

Trailing spaces in textViews

Future of Tingle

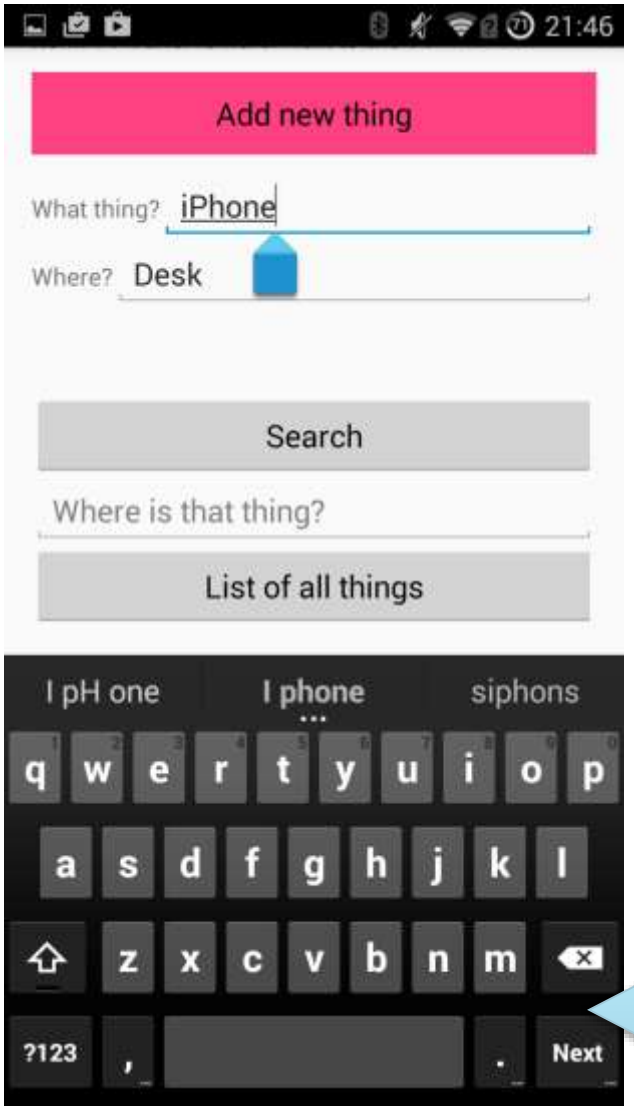
Resubmission of first mandatory assignment

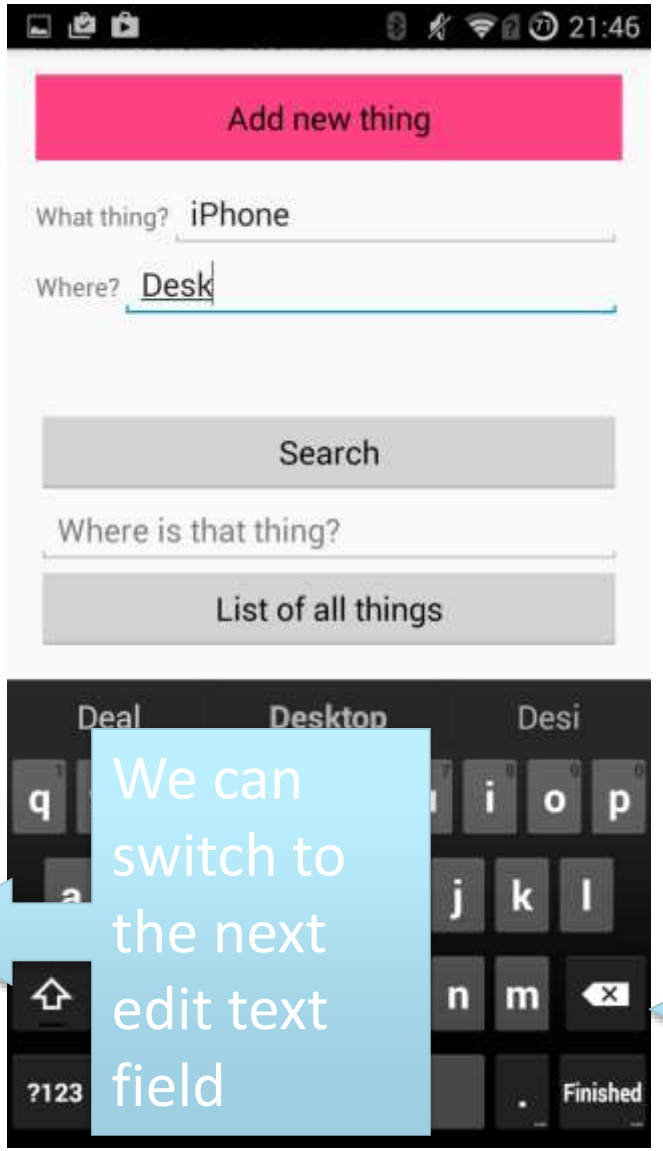
March 18

<u>PROBLEM:</u> THE EDIT FIELDS ALLOW TWO LINES OF TEXT AND DOES NOT SUPPORT TABBING BETWEEN THE FIELDS.

The multi-lining breaks the design

We can't continue. The keyboard hides the button. We are stuck





We can switch to the next edit text field

We can exit the form and continue further

http://developer.android.com/guide/topics/ui/controls/text.html

You can set the keyboard type

You can set and control the keyboard actions

You can set custom button label

You can provide auto-complete

```
<EditText
    android:id="@+id/launch_codes"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/enter_launch_codes"
    android:inputType="number"
    android:imeActionLabel="@string/launch" />
```

```
<EditText
    android:id="@+id/postal_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/postal_address_hint"
    android:inputType="textPostalAddress|
                       textCapWords|
                       textNoSuggestions" />
```

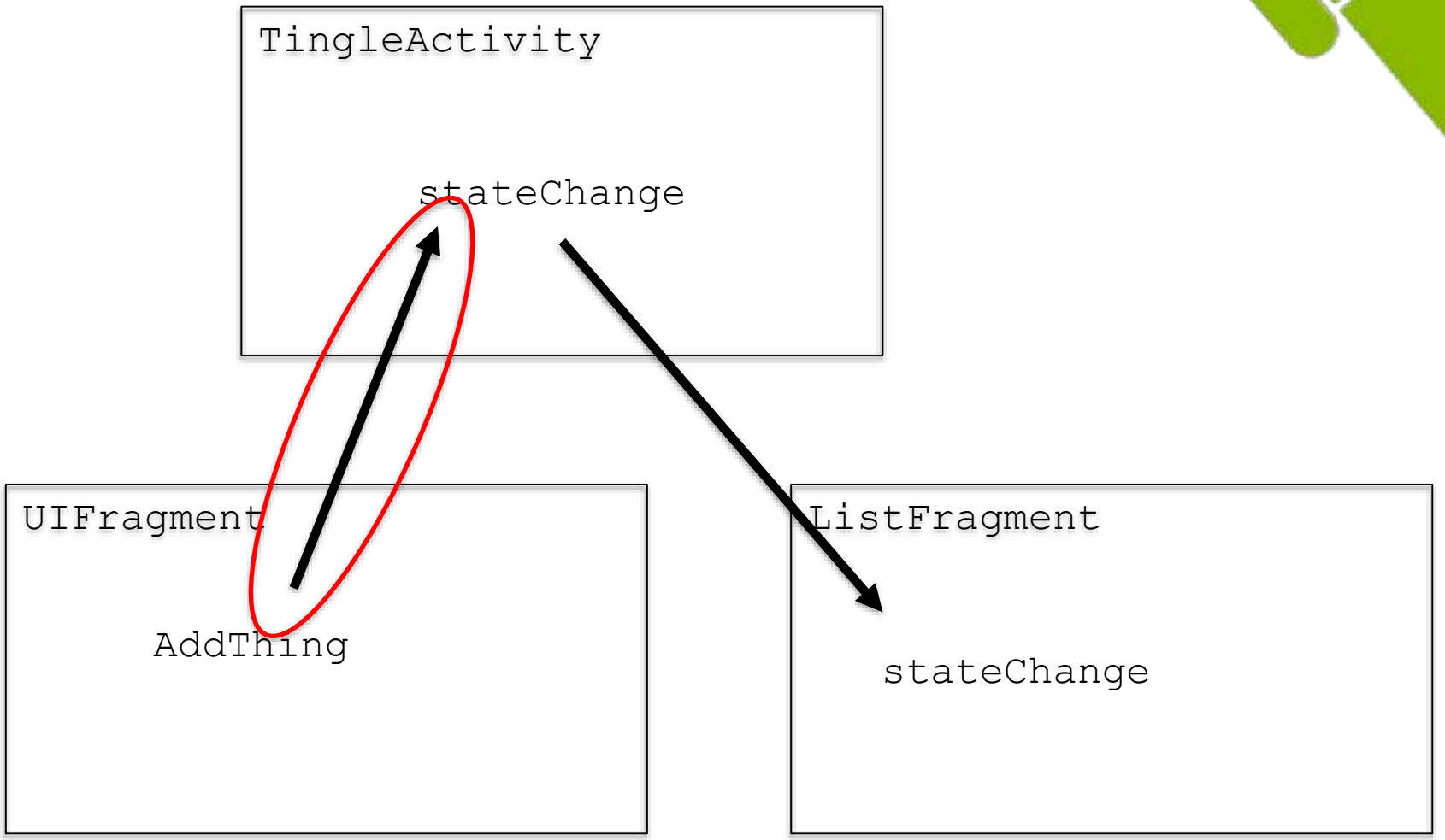
```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
    android:inputType="text"
    android:imeOptions="actionSend" />
```

```
ns="a"/>
```

```
actionDone
actionGo
actionNext
actionNone
actionPrevious
actionSearch
actionSend
actionUnspecified
flagForceAscii
flagNavigateNext
flagNavigatePrevious
flagNoAccessoryAction
flagNoEnterAction
flagNoExtractUi
```

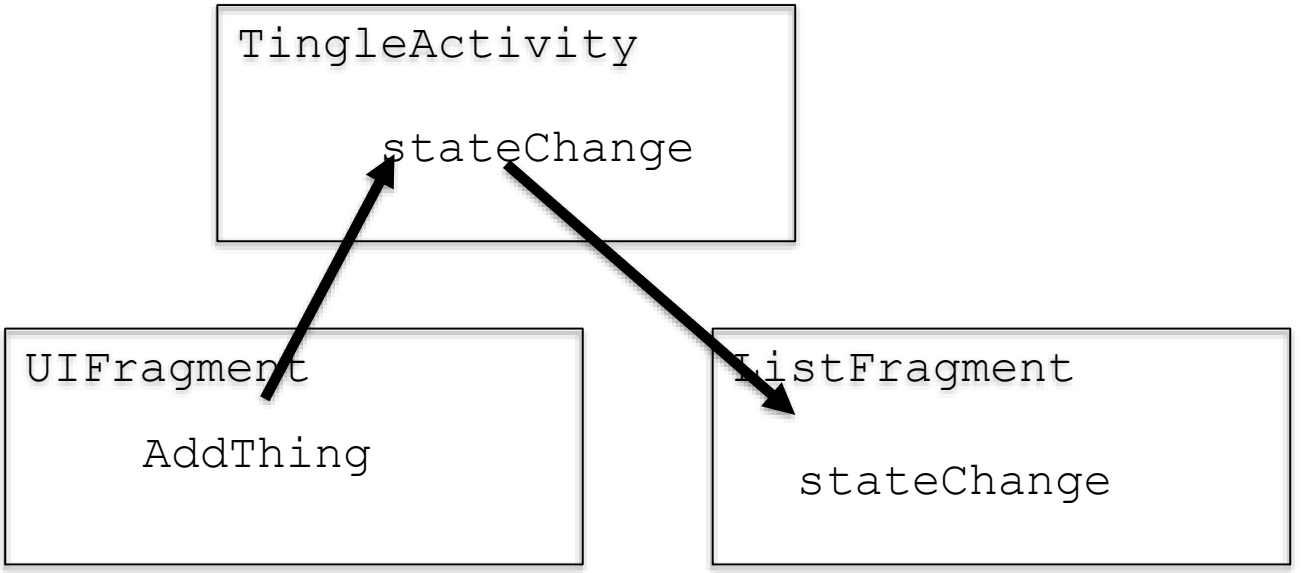
```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

Communicating between fragments and activities



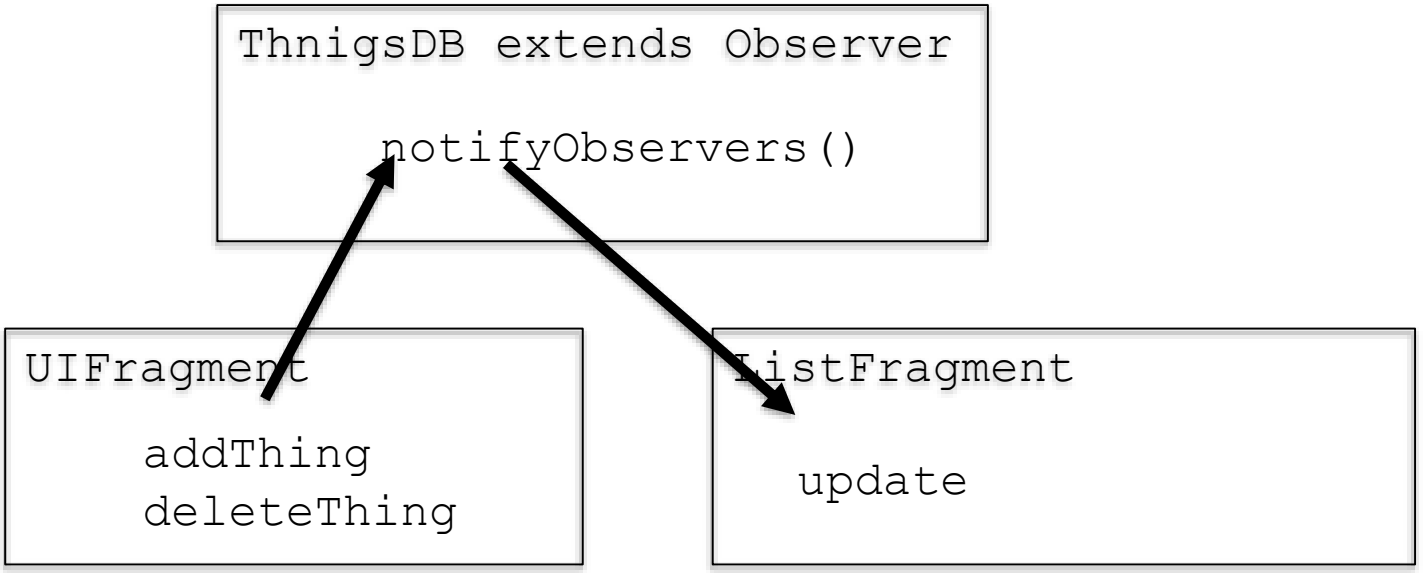
Nofifying about statechanges with an interface

```
public class ListFragment extends Fragment {

  static private ThingArrayAdapter listAdapter;

  public void stateChange() { listAdapter.notifyDataSetChanged(); }

  …
```

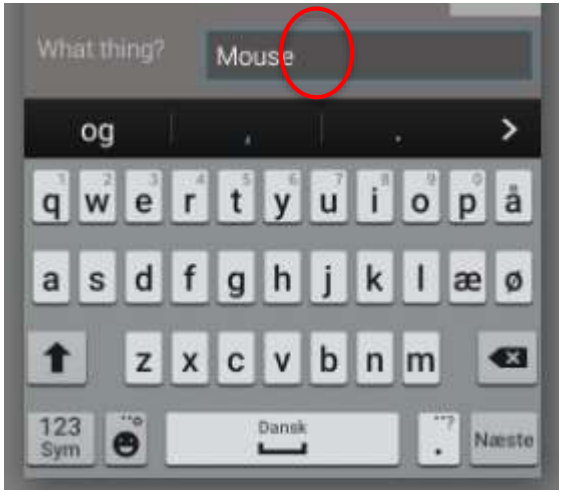


A more general solution: observer pattern/class

```
public class ListFragment extends Fragment implements Observer {

  static private ThingArrayAdapter listAdapter;

  @Override
  public void update(…) { listAdapter.notifyDataSetChanged(); }

  …
```



Extra spaces



1. newWhat.getText.toString().trim();

2. newWhat.getText.toString().replaceAll(”\\s+$”, ””);

Tingle – the dream

Many challenges (opportunities)

- registration of things (camera, barcode/rfid/beacon, …)
- searching (ranking)

- business model
- probably many other

Tingle – second mandatory assignment

Many challenges (opportunities)

- registration of things (camera, barcode/rfid/beacon, …)
- searching (ranking)

- business model
- probably many other