

Mobila applikationer och trådlösa nät, HT14

Lecturer: Anders Lindström,
anders.lindstrom@sth.kth.se

Lecture 4

Today's topics

- Intents
- BroadcastReceivers
- Introduction to Networking in Android



Configuration changes runtime

- Unless you specify otherwise, a configuration change at runtime will cause your current activity to be destroyed
- Examples:
 - Change in screen orientation
 - Language
 - Input devices, ...(defined in `android.content.res.Configuration`)
- If the activity had been in the foreground or visible to the user, a new instance of the activity will be created, and resource values reloaded
- Save UI-state in `onPause()` or `onSaveInstanceState`

Handle configuration changes in code

- This suppresses the Activity to be destroyed and restarted
- The application manifest:

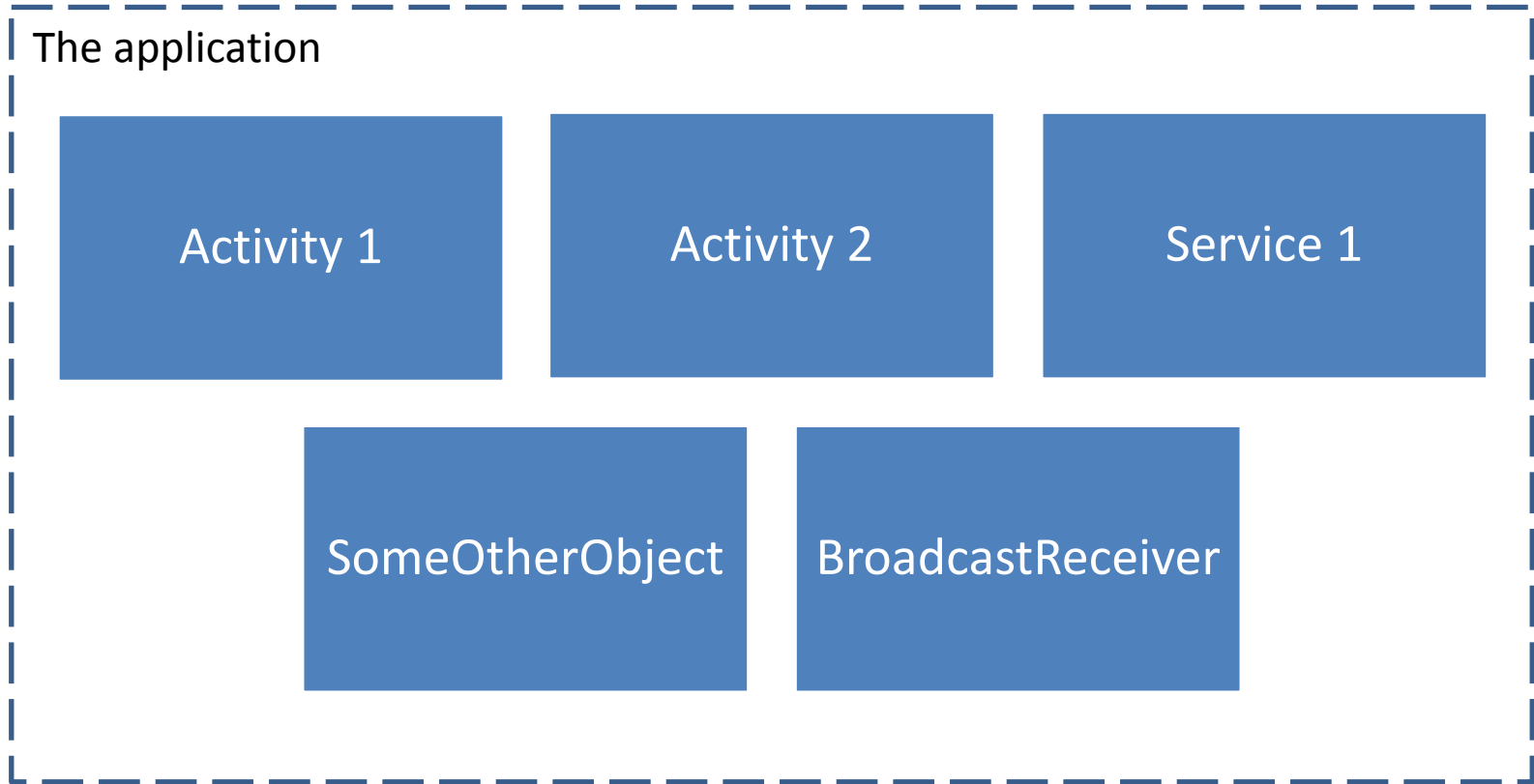
```
<activity
    ...
    android:configChanges="orientation|keyboardHidden"
/>
```
- In the activity:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

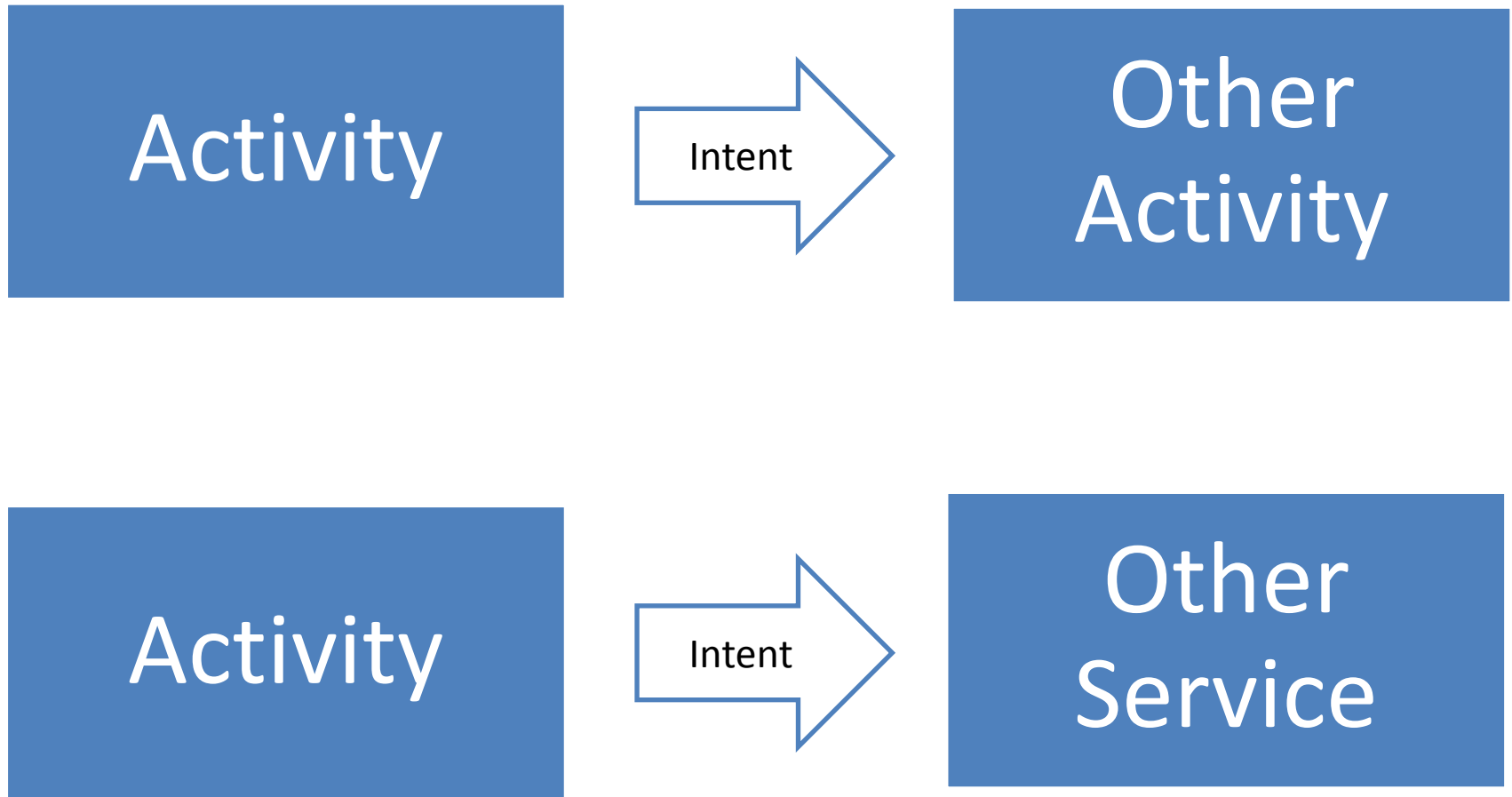
    // Update any UI based on resource values, they might
    // have changed

    if(newConfig.orientation ==
        Configuration.ORIENTATION_LANDSCAPE) { . . .
```

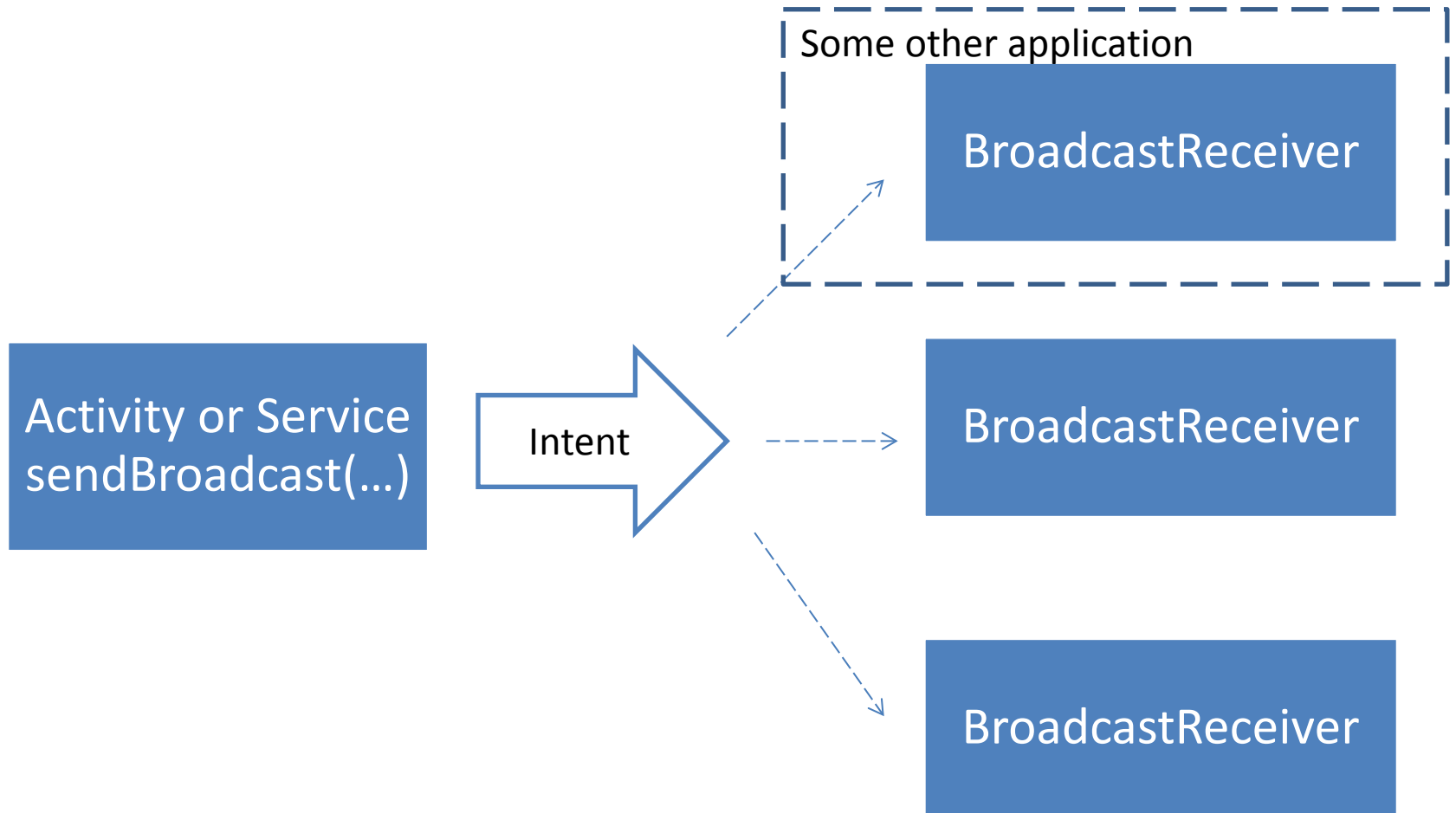
An Android application



Intents – send message or start other component



Broadcast Intents



Intents

- An Intent (meaning aim, purpose, intention) is an *asynchronous* message sent between application components

Intents can be used to

1. Explicitly start a *specific Activity* or Service (together with some data)
2. Implicitly request that *the best suited Activity* or Service perform an action (together with some data)
3. Broadcast that something has occurred

Explicit Intents

- To start a new Activity explicitly, call `startActivity(intent)`
- ```
Intent intent = new Intent(MyCurrentActivity.this, MyNextActivity.class);
startActivity(intent);
```
- The new Activity is moved to the top of the Activity stack
- Back button or `finish()` removes the Activity from the stack
- NB! By default, the application doesn't receive any notification when the newly started Activity finishes



# Explicit Intents

- All Activities must be registered in the AndroidManifest.xml file

```
<application . . .>
 <activity android:name=".MyMainActivity"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category
 android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>

 <activity
 android:name=".MyOtherActivity">
 </activity>
</application>
```

- Example: ExplicitIntent.zip

# Implicit Intents

- Intent to perform a task, without specifying an Activity, Service or such
- State the action to perform and, optionally, the data to act upon – as an URI
- The Android OS finds the appropriate Activity!
- Native actions: ACTION\_DIAL, ACTION\_EDIT, ACTION\_PICK, ACTION\_WEB\_SEARCH, ..., *ACTION\_VIEW*
- URI's:  
“tel:+4687904813”  
“content://contacts/people”,  
“http://google.com”,  
...

# Implicit Intents

- Start a new Activity implicitly
- ```
Intent intent = new Intent(
    Intent.ACTION_VIEW,
    Uri.parse("content://contacts/people/"));
startActivity(intent);
```
- ```
if(emergency && !helpInSight) {
 Intent intent = new Intent(
 Intent.ACTION_DIAL,
 Uri.parse("tel:112"));
 startActivity(intent);
}
```
- Need not be a native Activity, third party apps can be registered to support actions (native or new ones)

# Returning results, Sub-Activities

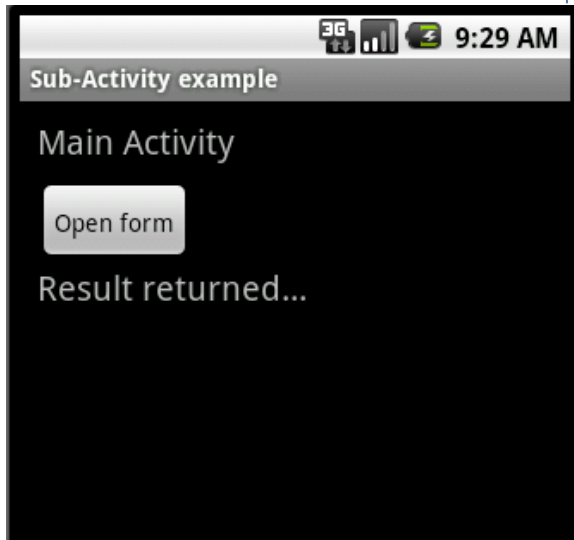
## Sub-Activity

- E.g. a (sub) Activity for user input (a form)
- Started with *startActivityForResult*
- Triggers an event in the parent Activity on finish
- May return result to the parent Activity
- Returned from the Sub-Activity: Intent with data, request code, result code

# Returning results, Sub-Activities

1)

```
startActivityForResult(
 intent,
 requestCode);
```

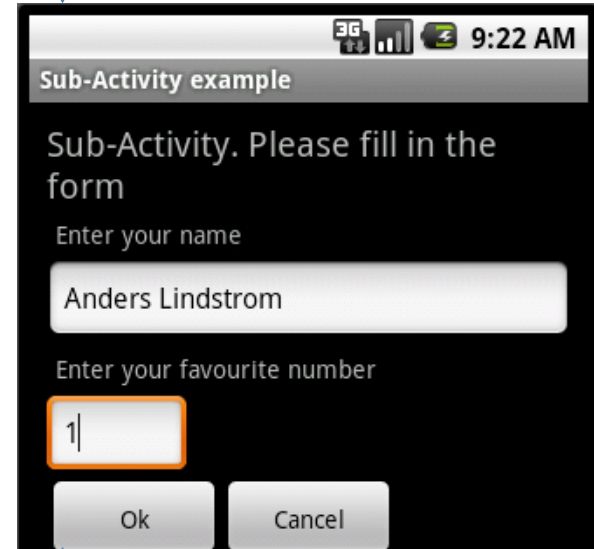


3) Call back:

```
onActivityResult(
 int requestCode,
 int resultCode,
 Intent result)
{ . . .
```

2)

```
setResult(
 resultCode,
 result);
finish();
```



# Returning results, Sub-Activities

- Starting a sub activity from parent Activity:

```
private static final int SHOW_FORM = 0; // Request code(s)
. . .

private class onStartSubListener implements View.OnClickListener {
 public void onClick(View v) {
 Intent intent = new
 Intent(MainActivity.this,
 SubActivity.class);
 startActivityForResult(intent, SHOW_FORM);
 }
}
```

# Returning results, Sub-Activities

- Call-back implementation in parent Activity:

```
public void onActivityResult(int requestCode,
 int resultCode, Intent result) {
 super.onActivityResult(requestCode, resultCode, result);

 if(resultCode == Activity.RESULT_OK) {
 switch(requestCode) {
 case SHOW_FORM:
 String name = result.getStringExtra(NAME);
 int number = result.getIntExtra(FAVOURITE_NUMBER, 42);
 // Process result and update view . . .
 break;
 case SHOW_SOMETHING_ELSE:
 // ...
 break;
 }
 . . .
 }
 . . .
}
```

# Returning results, Sub-Activities

- Returning result from Sub-Activity:

```
private class OnOkClickListener implements View.OnClickListener {
 public void onClick(View v) {
 String name = editName.getText().toString();
 int number =
 Integer.parseInt(editNumber.getText().toString());

 Intent result = new Intent();
 result.putExtra(MainActivity.NAME, name);
 result.putExtra(MainActivity.FAVOURITE_NUMBER, number);

 setResult(Activity.RESULT_OK, result);
 finish();
 }
}
```

- Example code: SubActivity.zip



# Intent filters

- To inform the system which implicit intents they can handle, activities, services, and broadcast receivers can have one or more intent filters
- Each filter describes a set of *implicit* intents that the component is willing to *receive*
- An implicit intent is delivered to a component only if it can pass through one of the component's filters
- *An explicit intent is always delivered to its target, no matter what it contains; the filter is not consulted*

# Intent filters

- AndroidManifest.xml file with intent-filter

```
<application . . .>
 <activity android:name=".MyMainActivity"
 android:label="@string/app_name">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category
 android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>

 <activity
 android:name=".MyOtherActivity">
 </activity>
</application>
```

# Broadcasting Intents

- Broadcast messages between components with the `sendBroadcast(...)` method
- This makes it possible to broadcast system wide
- ```
public static final String SOME_CUSTOM_EVENT =  
    "se.kth.anderslm.SOME_CUSTOM_EVENT";
```
- ```
Intent intent = new Intent(
 SOME_CUSTOM_EVENT);
intent.putExtra(...);
context.sendBroadcast(intent);
```

# Broadcasting Intents

Native Android broadcast events, examples

- ACTION\_TIME/DATE\_CHANGED  
ACTION\_MEDIA\_BUTTON  
ACTION\_CAMERA\_BUTTON  
ACTION\_NEW\_OUTGOING\_CALL  
ACTION\_SCREEN\_ON/OFF  
ACTION\_TIMEZONE\_CHANGED  
ACTION\_PACKAGE\_ADDED . . .
- ACTION\_MEDIA\_EJECT  
ACTION\_MEDIA\_MOUNTED/UNMOUNTED  
ACTION\_BATTERY\_CHANGED  
ACTION\_POWER\_CONNECTED/DISCONNECTED
- ACTION\_BOOT\_COMPLETED  
ACTION\_SHUTDOWN

# BroadcastReceivers

- A BroadcastReceiver listens to broadcasted Intents

- ```
public class CameraButtonReceiver
    extends BroadcastReceiver {
    @Override
    public void onReceive(
        Context context, Intent intent) {
        // To do...
    }
}
```

- Must be registered in the application manifest (or in code)
- Intent Filters are used to specify which Intents the BroadcastReceiver is listening for

BroadcastReceivers

- Register the BroadcastReceiver in the application manifest or in the source code
- ```
<receiver android:name=".CameraButtonReceiver">
 <intent-filter>
 <action android:name=
 "android.intent.action.CAMERA_BUTTON"/>
 </intent-filter>
</receiver>
```
- Receivers registered in the manifest file are always active, whether or not the application is running at the moment

# BroadcastReceivers

- Registering a BroadcastReceiver in the source code
- ```
IntentFilter filter = new IntentFilter (  
    "se.kth.anderslm.CameraButtonReceiver");  
CameraButtonReceiver receiver = new  
CameraButtonReceiver();  
registerReceiver(receiver, filter);
```
- Receivers registered in the source code are active only when the corresponding application is running, or until unregistered
- ```
unRegisterReceiver(receiver);
```
- Useful e.g. when using it for updating the UI (unregister when the activity isn't visible or active - in onPause)

# The Application class

- The Application class may be sub-classed to
  - Maintain application state
  - Transfer/share objects between application components
  - Manage and maintain other resources used by multiple application components
- Register a custom Application class in the manifest file

```
<application
 android:icon="@drawable/ic_launcher"
 android:name="CustomApplication" >
 <activity ...>
 ...
</application>
```
- Objects might also be shared between application components using the Singleton Design Pattern



# The Application class

- Implement the Application sub-class in a way similar to the Singleton design pattern

- `public class CustomApplication extends Application {`  
    `public static final CustomApplication singleton;`  
    `private SomeSortOfSharedData data;`  
  
    `public static CustomApplication getInstance() {`  
        `return singleton;`  
    `}`  
  
    `public final void onCreate() {`  
        `super.onCreate();`  
        `singleton = this;`  
        `// Create other, shared objects`  
        `data = new ...;`  
    `}`  
  
    `public SomeSortOfSharedData getData() { ...`  
    `public void setData(...) { ...`  
  
    `}`

# Android Networking

## Packages

- java.net: URL, URLConnection, HttpURLConnection, Socket, DatagramSocket...
- java.io: InputStream, OutputStream (and wrapper/filter streams), IOException, ...
- java.nio – unblocking IO, channels
- android.bluetooth: BluetoothSocket, BluetoothServerSocket, ...

# Android Networking

- Add uses permission to the Android Manifest file, at least android.permission.INTERNET

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

# URLConnection

- A connection to a URL for reading or writing
- `BufferedReader in = null;`

```
try {
 URL url = new URL("ftp://xxx.yyy.zzz/index.html");
 URLConnection urlConnection = url.openConnection();

 in = new BufferedReader(new
 InputStreamReader(urlConnection.getInputStream()));

 String line = in.readLine();
 while(line != null) { ...
}
finally {
 in.close();
}
```

# URLConnection

- Used to send and receive data over the web. Data may be of any type and length (here: downloading an image from the net, example code from Networking1.zip)

```
URLConnection http = null;
InputStream istream = null;
try {
 URL url= new URL(http://www.google.com/);
 http = (URLConnection) url.openConnection();
 istream = http.getInputStream();
 Bitmap bmImg = BitmapFactory.decodeStream(istream);
 imageView.setImageBitmap(bmImg);
}
finally {
 if(istream != null) istream.close();
 if(http != null) http.disconnect();
}
```

# URLConnection

- Reading from a HttpURLConnection, saving to file

```
try {
 URL url= new URL(httpUrIstr);
 http = (URLConnection) url.openConnection();

 istream = http.getInputStream();
 byte[] buffer = new byte[1024];
 fos = this.openFileOutput (fileName, Activity.MODE_PRIVATE);

 int readSize = 0;
 while (readSize != -1) {
 readSize = istream.read(buffer);
 if (readSize > 0) {
 fos.write(buffer, 0, readSize);
 }
 }
}
```

# XML

- Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form
- Meta-data
- Widely used for the representation of arbitrary data structures, for example in web services
- An element can contain:
  - other elements
  - text
  - *attributes*(or a mix of all of the above...)

# XML

- `<bookstore>`
  - `<book category="WEB">`
    - `<title>Learning XML</title>`
    - `<author>Erik T. Ray</author>`
    - `<year>2003</year>`
    - `<price>39.95</price>`
  - `</book>`
  - `<book>`
    - `. . .`
  - `</book>`
- `</bookstore>`



# Parsing XML

Slogan: "Don't Superize Me"

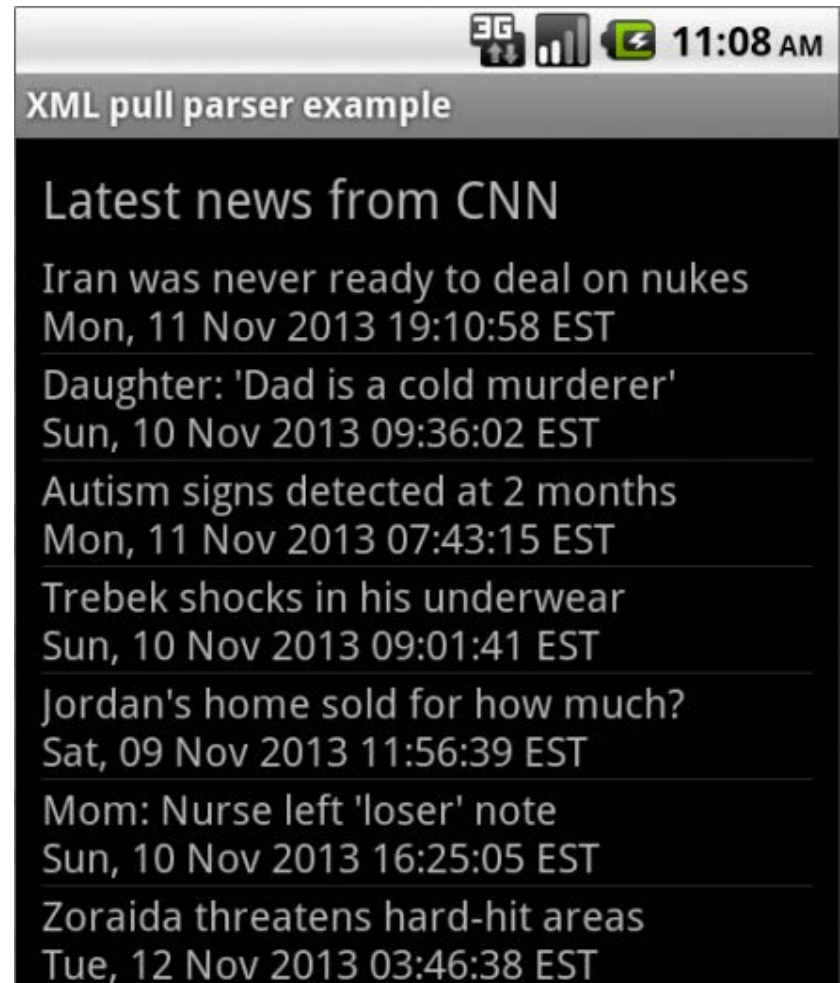
- Model parser – Reads the entire document and stores it as a tree-structure  
Android: DocumentBuilder
- Push parser - Set up callbacks, the SAX component *pushes* document events to you  
Android: org.xml.sax
- Pull parser – *Pull* data from the document using the pull parser component  
Android: XMLPullParser
- Pull/Push reads only the parts of the document needed

# Parsing XML (pull parser)

- `XmlPullParser parser = XmlPullParserFactory.newInstance().newPullParser() parser.setInput(inputStream, encoding);`
- `int parseEvent = parser.next();`
- Event codes:  
`XmlPullParser.START_DOCUMENT, END_DOCUMENT, START_TAG, END_TAG, TEXT, ...`
- Getting attributes:  
`int n = parser.getAttributeCount();`  
`String attributeName = parser.getAttributeName(i);`  
`String value = parser.getAttributeValue(i);`

# Parsing XML, example

- RSS (Really Simple Syndication or Rich Site Summary) is an XML-based format for sharing and distributing Web content, such as news headlines
- Example: CNN news RSS,  
[http://rss.cnn.com/rss/cnn\\_topstories.rss](http://rss.cnn.com/rss/cnn_topstories.rss)



# Parsing XML, example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>
 <title>CNN.com</title>
 <link>http://edition.cnn.com/?eref=edition</link>
 <description>CNN.com delivers up-to-the-minute news.</description>
 <item>
 <title> Mubarak's VP says talks with opposition under way</title>
 <link>
 http://rss.cnn.com/~r/rss/cnn_topstories/~3/xXT6mhyGJWI/index.html</link>
 <description> Talks between opposition parties and Egyptian Vice. .
 .</description>
 <pubDate> Mon, 31 Jan 2011 16:27:37 EST</pubDate>
 </item>
 <item>
 <title>Why Nicole Kidman kept Faith a secret</title>
 <link>. . .
 </item>
</channel>
</rss>
```

# Parsing XML

```
private void updateNews() throws Exception {
 HttpURLConnection http = null;
 InputStream xmlStream = null;
 try {
 URL url = new URL(cnnUrl);
 http = (HttpURLConnection) url.openConnection();
 RSSParser parser = new RSSParser();
 parser.parse(http.getInputStream(), newsItems);
 adapter.notifyDataSetChanged();
 }
 finally {
 if(xmlStream != null) xmlStream.close();
 if(http != null) http.disconnect();
 }
}
```

# Parsing XML, the parser class

```
XmlPullParser parser =
 XmlPullParserFactory.newInstance().newPullParser();
parser.setInput(inputStream, encoding);

int parseEvent = parser.getEventType();
while(parseEvent != XmlPullParser.END_DOCUMENT) {
 switch(parseEvent) {
 case XmlPullParser.START_DOCUMENT: . . .
 case XmlPullParser.START_TAG:
 String tagName = parser.getName();
 if(tagName.equalsIgnoreCase(ITEM)) {
 parseItem();
 }
 break;
 case XmlPullParser.END_TAG: . . .
 }
 parseEvent = parser.next();
}
```

# Parsing XML, the parser class

```
private void parseItem() throws IOException, XmlPullParserException {
 int parseEvent;
 String name, item = "";
 // Continue until end of </item>
 do {
 parseEvent = parser.next();
 name = parser.getName();
 if(parseEvent == XmlPullParser.START_TAG) {
 if(name.equalsIgnoreCase(TITLE)) {
 item += parser.nextText() + "\n";
 }
 else if(name.equalsIgnoreCase(PUBDATE)) {
 item += parser.nextText();
 }
 }
 } while(parseEvent != XmlPullParser.END_TAG // !name.equals(ITEM));

 itemList.add(item);
}
```

# Parsing XML

- Example with push parser, SAX, in Meier chapter 5: Creating an Earthquake viewer