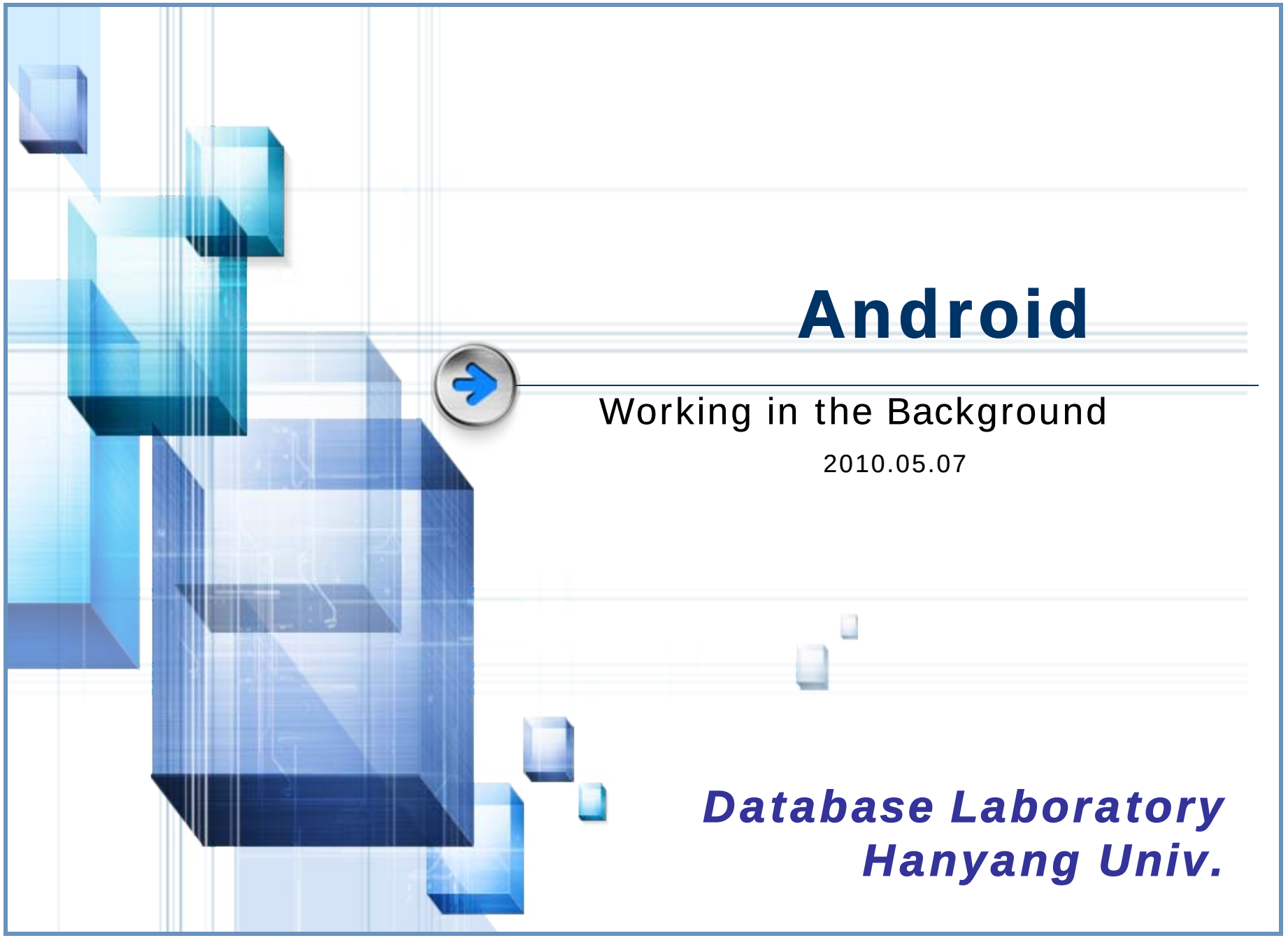


Android

Working in the Background

2010.05.07

Database Laboratory
Hanyang Univ.

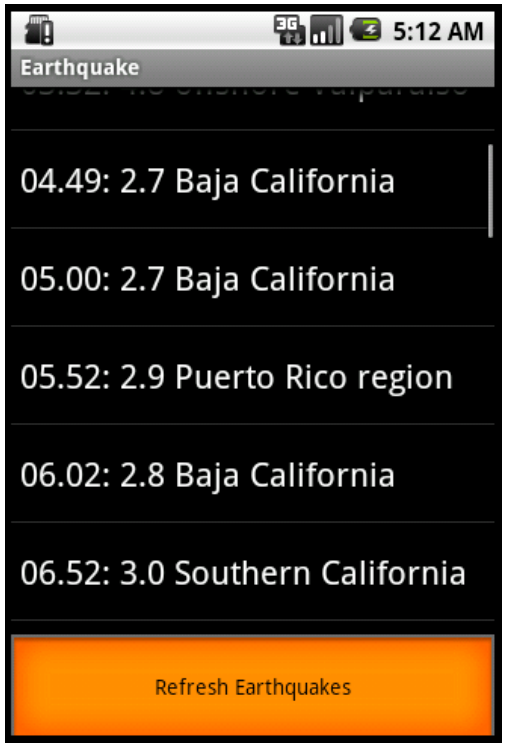
Introducing Service

- **Android offers the *Service class* to create application components specifically to handle operations and functionality that should run silently, without a User Interface.**

- **Services run in the background**
  - Updating your Content Providers, firing Intents, and triggering Notifications.
  - ***Services*** are started, stopped, and controlled from other application components.
  - Started ***Services*** receive higher priority than inactive or invisible Activities.
  - Applications that update regularly but only rarely or intermittently need user interaction are good candidates for implementation as ***Services***.

An Earthquake Monitoring Service Example

- In this Example, you'll modify the Earthquake example
- you'll move the earthquake updating and processing functionality into a **separate Service component**



An Earthquake Monitoring Service Example

1. Start by creating a new *EarthquakeService* that extends *Service*

```
package com.paad.earthquake;

import java.io.IOException;

public class EarthquakeService extends Service {

  public static final String NEW_EARTHQUAKE_FOUND = "New_Earthquake_Found";

    @Override
    public void onCreate() {
        // TODO: Initialize variables, get references to GUI objects
    }

    @Override
    public void onStart(Intent intent, int startId) {
        // TODO: Actions to perform when service is started.
    };

  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }
```

An Earthquake Monitoring Service Example

2. **Add this new *Service* to the manifest by adding a new service tag within the *application* node.**

```
<service android:enabled="true" android:name=".EarthquakeService"></service>
```

3. **Move the *refreshEarthquakes* and *addNewQuake* methods out of the Earthquake Activity and into the *EarthquakeService*.**
   - You'll need to remove the calls to *addQuakeToArray* and *loadQuakesFromProvider*
   - In the *EarthquakeService* remove all references to the earthquakes ArrayList

An Earthquake Monitoring Service Example

addNewQuake()

```
private void addNewQuake(Quake _quake) {
  ContentResolver cr = getContentResolver();

  // Construct a where clause to make sure we don't already have this
  // earthquake in the provider.
  String w = EarthquakeProvider.KEY_DATE + " = " + _quake.getDate().getTime();

  // If the earthquake is new, insert it into the provider.
  if (cr.query(EarthquakeProvider.CONTENT_URI, null, w, null, null).getCount() == 0){
    ContentValues values = new ContentValues();

    values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
    values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

    double lat = _quake.getLocation().getLatitude();
    double lng = _quake.getLocation().getLongitude();
    values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
    values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
    values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
    values.put(EarthquakeProvider.KEY_MAGNITUDE, _quake.getMagnitude());

    cr.insert(EarthquakeProvider.CONTENT_URI, values);
  }
}
```

An Earthquake Monitoring Service Example

refreshEarthquakes()

```
private void refreshEarthquakes() {
    // Get the XML
    URL url;
    try {
      String quakeFeed = getString(R.string.quake_feed);
      url = new URL(quakeFeed);

      URLConnection connection;
      connection = url.openConnection();

      HttpURLConnection httpConnection = (HttpURLConnection)connection;
      int responseCode = httpConnection.getResponseCode();

      if (responseCode == HttpURLConnection.HTTP_OK) {
        InputStream in = httpConnection.getInputStream();

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();

        // Parse the earthquake feed.
        Document dom = db.parse(in);
        Element docEle = dom.getDocumentElement();
```

An Earthquake Monitoring Service Example

Cont'D

```
// Get a list of each earthquake entry.
NodeList nl = docEle.getElementsByTagName("entry");
if (nl != null && nl.getLength() > 0) {
  for (int i = 0 ; i < nl.getLength(); i++) {
    Element entry = (Element)nl.item(i);
    Element title = (Element)entry.getElementsByTagName("title").item(0);
    Element g = (Element)entry.getElementsByTagName("georss:point").item(0);
    Element when = (Element)entry.getElementsByTagName("updated").item(0);
    Element link = (Element)entry.getElementsByTagName("link").item(0);

    String details = title.getFirstChild().getNodeValue();
    String hostname = "http://earthquake.usgs.gov";
    String linkString = hostname + link.getAttribute("href");

    String point = g.getFirstChild().getNodeValue();
    String dt = when.getFirstChild().getNodeValue();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss'Z'");
    Date qdate = new GregorianCalendar(0,0,0).getTime();
    try {
      qdate = sdf.parse(dt);
    } catch (ParseException e) {
      e.printStackTrace();
    }

    String[] location = point.split(" ");
    Location l = new Location("dummyGPS");
    l.setLatitude(Double.parseDouble(location[0]));
    l.setLongitude(Double.parseDouble(location[1]));
```

An Earthquake Monitoring Service Example

Cont'D

```
                String magnitudeString = details.split(" ")[1];
                int end =  magnitudeString.length()-1;
                double magnitude = Double.parseDouble(magnitudeString.substring(0, end));

                details = details.split(",")[1].trim();

                Quake quake = new Quake(qdate, details, l, magnitude, linkString);

                // Process a newly found earthquake
                addNewQuake(quake);
              }
            }
          }
        } catch (MalformedURLException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        } catch (ParserConfigurationException e) {
          e.printStackTrace();
        } catch (SAXException e) {
          e.printStackTrace();
        }
        finally {}
    }
}
```

An Earthquake Monitoring Service Example

4. Within the Earthquake Activity, create a new *refreshEarthquakes* method

- It should explicitly start the *EarthquakeService*

```
/** Refresh data from the earthquake feed using the Service*/
private void refreshEarthquakes() {
    startService(new Intent(this, EarthquakeService.class));
}
```

An Earthquake Monitoring Service Example

5. **The *EarthquakeService* will now update on an automated schedule(if one is specified).**
   - This information is not yet passed back to the Earthquake Activity
   - To alert those components, and any other applications interested in earthquake data, modify the *EarthquakeService* to broadcast a new Intent whenever a new earthquake is added

① **Modify the addNewQuake method to call a new announceNewQuake method**

An Earthquake Monitoring Service Example

```
public static final String NEW_EARTHQUAKE_FOUND = "New_Earthquake_Found";

private void addNewQuake(Quake _quake) {
  ContentResolver cr = getContentResolver();
  // Construct a where clause to make sure we don't already have this
  // earthquake in the provider.
  String w = EarthquakeProvider.KEY_DATE + " = " + _quake.getDate().getTime();

  // If the earthquake is new, insert it into the provider.
  if (cr.query(EarthquakeProvider.CONTENT_URI, null, w, null, null).getCount()==0){
    ContentValues values = new ContentValues();

    values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
    values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

    double lat = _quake.getLocation().getLatitude();
    double lng = _quake.getLocation().getLongitude();
    values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
    values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
    values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
    values.put(EarthquakeProvider.KEY_MAGNITUDE, _quake.getMagnitude());

    cr.insert(EarthquakeProvider.CONTENT_URI, values);
     announceNewQuake(_quake);
  }
}
```

```
private void announceNewQuake(Quake quake) {
  }
```

An Earthquake Monitoring Service Example

② Within *announceNewQuake*, broadcast a new Intent whenever a new earthquake is found.

```
private void announceNewQuake(Quake quake) {
    Intent intent = new Intent(NEW_EARTHQUAKE_FOUND);
    intent.putExtra("date", quake.getDate().getTime());
    intent.putExtra("details", quake.getDetails());
    intent.putExtra("longitude", quake.getLocation().getLongitude());
    intent.putExtra("latitude", quake.getLocation().getLatitude());
    intent.putExtra("magnitude", quake.getMagnitude());

    sendBroadcast(intent);
  }
```

❖ That completes the *EarthquakeService* implementation

6. You still need to modify the two Activity components to listen for the Service Intent broadcasts and refresh their displays accordingly

An Earthquake Monitoring Service Example

① **Within the Earthquake Activity, create a new internal *EarthquakeReceiver* class that extends *BroadcastReceiver***

- Override the *onReceive* method to call *loadFromProviders* to update the earthquake array and refresh the list

```
public class EarthquakeReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    loadQuakesFromProvider();
  }
}
```

An Earthquake Monitoring Service Example

② Override the *onResume* method to register the new Receiver and update the LiveView contents when the Activity becomes active

- Override *onPause* to unregister it when the Activity moves out of the foreground

```
EarthquakeReceiver receiver;
```

```
@Override
public void onResume() {
  super.onResume();
  IntentFilter filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
  receiver = new EarthquakeReceiver();
  registerReceiver(receiver, filter);
  loadQuakesFromProvider();
}

@Override
public void onPause() {
  unregisterReceiver(receiver);
  super.onPause();
}
```

An Earthquake Monitoring Service Example

- Now when the Earthquake Activity is launched, it will start the Earthquake Service
- This Service will then continue to run, updating the earthquake Content Provider in the background, even after the Activity is suspended or closed

More Trying!!

- At this stage, the earthquake processing is done in a Service
- but it's still being **executed on the main thread!!**
- Now, you need to **move time-consuming operations onto background threads** to improve performance
- **move the network lookup and XML processing done in the *EarthquakeService* onto a background thread**