Broadcast Receivers, Content Providers, Adapters & Lists
Last lecture

• Fragments
• Intents & Components
  • Basic introduction to application components
  • Intents & Intent Filters
  • Launching Activities
• Context
This Lecture

• Application Components:
  • Activities
  • Broadcast Receivers
  • Content Providers
  • Services

• UI – List handling
  • Adapters
Broadcasts

• Apps can send/receive broadcast messages from the system or other apps
  • Publish-subscribe pattern

• Allows your app to receive events outside of user flow
  • A set of pre-defined system event broadcasts
    • Screen turned on, system boots up, charger connected
  • Custom, app-defined broadcasts
    • E.g. Data downloaded notification

https://developer.android.com/guide/components/broadcasts
Broadcast Receivers

• Apps can register receivers for specific broadcasts (subscribe)

• Also uses *implicit vs explicit* concept
  • Explicit broadcasts target some specific app
  • System broadcasts are implicit, Android runtime notifies all *receivers* registered to that event
    • E.g. register for ACTION_BOOT_COMPLETED
  • Broadcasts can be delivered even to non-running apps!

https://developer.android.com/guide/components/broadcasts
System Broadcast Events

• Message is broadcast as an Intent object
  • *Action* property identifies the event, e.g. :
    • `android.intent.action.AIRPLANE_MODE`
  • Can contain other Intent Extras

• System Broadcast action list can be found in `broadcast_actions.txt` of SDK

• To see which data is attached to a broadcasts Intent, use the documentation
  • E.g. see `ACTION_AIRPLANE_MODE_CHANGED`
Receiving broadcasts (1)

1. Create receiver – BroadcastReceiver subclass
2. Register for a given type of broadcast via Context:

```kotlin
val receiver = MyBroadcastReceiver()
val filter = IntentFilter(Intent.ACTION_POWER_CONNECTED)
registerReceiver(receiver, filter)

// Later:
unregisterReceiver(receiver)
```

- In this case, the lifecycle of the context influences how long the receiver can get events!
  - Activity context vs application context
- Always call unregister as well!

https://developer.android.com/guide/components/broadcasts#context-registered-receivers
Receiving broadcasts (2)

Alternatively:

• Declare the BroadcastReceiver in the Manifest:

```
<receiver
  android:name="MyReceiver"
  android:enabled="true"
  android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.BATTERY_LOW"/>
  </intent-filter>
</receiver>
```

• The system launches your app if not running when broadcast is sent

https://developer.android.com/guide/components/broadcasts#receiving-broadcasts
Implementing a Broadcast receiver

• Implement as subclass of `BroadcastReceiver`
• Receive Intent broadcast objects by overriding `onReceive(context: Context?, intent: Intent?)`
  
• The Intent object contains additional info
  • Action name – name of the broadcast event
  • E.g. the Intent for Airplane mode changing has a Boolean extra

• Upon app installation, the system package manager registers the receiver

https://developer.android.com/guide/components/broadcasts
Implementing a Broadcast Receiver: Kotlin

class MyBroadcastReceiver: BroadcastReceiver(){
    override fun onReceive(context: Context?, intent: Intent?) {
        // react to the event
    }
}

• The BroadcastReceiver is considered active only during executing onReceive()

• DEMO – receiving charger (dis)connected broadcasts

Receiving *implicit* broadcasts

• Since API ver. 26, *implicit* broadcast receivers should not be declared in the manifest
  • Instead, use the programmatic approach

• Currently, there’s a list of exceptions to this:

• Alternative approach is to use scheduled jobs and [JobSchedulers](https://developer.android.com/reference/android/app/ScheduledWorkListener)
Implicit broadcast example

• For certain situations, implicit broadcasts are necessary

ACTION_DEVICE_OWNER_CHANGED

This broadcast is not sent very often; some apps need to receive it, so that they know that the device's security status has changed.

ACTION_EVENT_REMINDER

Sent by the calendar provider to post an event reminder to the calendar app. Since the calendar provider doesn't know what the calendar app is, this broadcast has to be implicit.

ACTION_MEDIA_MOUNTED, ACTION_MEDIA_CHECKING, ACTION_MEDIA_UNMOUNTED, ACTION_MEDIA_EJECT, ACTION_MEDIA_UNMOUNTABLE, ACTION_MEDIA_REMOVED

https://developer.android.com/guide/components/broadcast-exceptions
Sending Broadcasts

- **LocalBroadcastManager.getInstance(context).sendBroadcast()**
  - Sent to receivers within your app
  - More performant
  ```kotlin
  val intent = Intent()
  intent.action = "com.example.broadcast.MY_MESSAGE"
  intent.putExtra("data", "My custom message data")
  sendBroadcast(intent)
  ```
- **sendBroadcast(Intent)**
  - Sent to all receivers
- **sendOrderedBroadcast(Intent, String)**
  - Sent to all receivers, but one at a time
  - Order is specified with `android:priority` attribute of the matching intent-filter
  - Can chain the broadcast

https://developer.android.com/guide/components/broadcasts#sending-broadcasts
Best practices

• Try to use context-registered broadcast receivers
  • Their lifecycle is shorter
  • If many apps are registered to the same implicit broadcast, the system may launch a lot of apps
    • E.g. imagine if several apps try to start synchronizing data when WiFi state changes

• A receivers onReceive(..) method should not do heavy/long-running work
  • goAsync() or JobSchedulers can be used if you still need to

• To limit which apps can send broadcasts to you, use permission management of broadcasts

Example: logging the number of a call

Here we use an Kotlin anonymous object instead of creating new Class

```kotlin
val TAG: String = "CallReceiver"

val callReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {

        if (intent.extras != null) {
            val state = intent.getStringExtra(TelephonyManager.EXTRA_STATE)
            Log.i(TAG, state)

            // System sends 2 "RINGING" events: 1 with number, 1 without
            if (intent.hasExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)) {
                val phoneNo = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER)
                Log.i(TAG, phoneNo)
            }
        }
    }
}

registerReceiver(callReceiver, IntentFilter(TelephonyManager.ACTION_PHONE_STATE_CHANGED))

https://developer.android.com/reference/android/telephony/TelephonyManager#ACTION_PHONE_STATE_CHANGED
```
Demo

• It didn’t work.. What was missing?
• Permissions!
System Permissions

• Recall – apps must declare required permissions in the Manifest file
  • `<uses-permission>`

Divided into 2 categories: normal and dangerous

• Normal:
  • Do not directly risk user’s privacy
  • E.g. setting the time zone
  • If declared in manifest, system grants the permissions automatically

• Dangerous:
  • May give app access to user’s confidential data
  • In addition to listing the permissions in the manifest, app has to ask user for each individual permission when running

https://developer.android.com/guide/topics/permissions/overview#normal-dangerous
Requesting permissions

1. Declare in manifest:

```xml
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

2. Request permission from user (e.g. in Activity):
   - Applies to dangerous permissions

```kotlin
val requestedPermissions: Array<String> = arrayOf(
    Manifest.permission.READ_CALL_LOG,
    Manifest.permission.READ_PHONE_STATE)

ActivityCompat.requestPermissions(
    activity: this, requestedPermissions, requestCode: 1)
```

https://developer.android.com/training/permissions/requesting.html
https://developer.android.com/reference/android/telephony/TelephonyManager#ACTION_PHONE_STATE_CHANGED
Requesting permissions (2)

3) Handle results in callback function:

```kotlin
override fun onRequestPermissionsResult( .. ) {
    //user has made a decision about permissions, react
}
```

Note: `requestPermissions()` is available from Activity, Context directly, however we are using ActivityCompat support library helper:

```kotlin
this.requestPermissions(requestedPermissions, requestCode: 1)
```

• Handles a lot of backwards compatibility issues for you
Checking for permissions

- `checkSelfPermission()`
  - `if` (ActivityCompat.checkSelfPermission(this, Manifest.permission.READ_CALL_LOG) != PackageManager.PERMISSION_GRANTED) {
    // request permissions,
  } else {
    // already granted
  }

- `shouldShowRequestPermissionRationale()`
  - Use it to determine whether to show explanatory text about how your app uses permissions
  - True if user has previously denied the request, false if user has denied the request and selected “Don’t ask again”

https://developer.android.com/training/permissions/requesting#perm-check
Content Providers
Content Providers

- Content providers manage access to a structured set of data
- Enable sharing of data across applications
  - Address book, Calendar, Photo Gallery, ...
- By default, applications can’t access the data and files of other applications
- In effect, a Content Provider is an abstraction interface to data storage mechanisms
  - Uniform APIs for CRUD
    - Create, Read, Update, Delete
  - Files, SQLite databases, in-memory hash maps
  - Secure

https://developer.android.com/guide/topics/providers/content-providers
**Content Provider**

- Presents data to external applications as one or more **tables**
  - Row – instance of some data type
  - Column – individual piece of data for an instance
- Manages access to the stored data for various APIs and components:
  - Sharing data with other apps
  - Sending data to a widget
  - Loading data into UI
  - Returning custom search suggestions
  - ...

https://developer.android.com/guide/topics/providers/content-provider-basics
Accessing Content Providers

• To access data in a Content Provider, you use a **ContentResolver**
  
  ```java
  var cr = getContentResolver()
  ```

  ContentResolver provides methods to perform basic CRUD operations

https://developer.android.com/guide/topics/providers/content-provider-basics#ClientProvider
Example: user dictionary

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>frequency</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr_FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>

- Stores the spellings of non-standard words that the user wants to keep
- _ID column acts as *primary key*, maintained by the provider
Using ContentResolver example

• Call ContentResolver.query():

```java
// Queries the user dictionary and returns results
getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,  // Content URI of the words table
    mProjection,  // Columns to return for each row
    mSelectionClause,  // Selection criteria
    mSelectionArgs,  // Selection criteria
    mSortOrder);  // Sort order for returned rows
```
Content URIs

A content URI is a URI that identifies data in a provider.

Content URIs include

- The fixed scheme value for content URI-s (content://)
- Symbolic name of the entire provider (its authority)
- a name that points to a table or file (a path).
- Optional: id part points to an individual row in a table (Query).

https://developer.android.com/guide/topics/providers/content-provider-basics#ContentURIs
Example: URI

• Note: The User Dictionary Provider defines the `android.permission.READ_USER_DICTIONARY` in it’s manifest, so clients of it must request this permission

```java
getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,
    mProjection,
    mSelectionClause
    mSelectionArgs,
    mSortOrder);

// SQL: SELECT ... FROM words WHERE...
```

• Next, constructing the rest of the query
Projection defines which columns to return

// String array values are contract class constants

```javascript
var mProjection = arrayOf(
  UserDictionary.Words. ID,
  UserDictionary.Words. WORD,
  UserDictionary.Words. LOCALE
)
```

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>frequency</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr.FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>

// SQL: SELECT ID, WORDS, LOCALE FROM ...

https://developer.android.com/guide/topics/providers/content-provider-basics#Query
var mSelectionClause: String?
var mSelectionArgs: Array<String>

// SQL: SELECT .. FROM words WHERE word = "mobile"

mSelectionClause = "${UserDictionary.Words.\$\{WORD\} = ?"

mSelectionArgs = arrayOf("mobile")

• Selection Clause – logical statement for filtering rows

• Selection Args – arguments for selection clause, if clause uses replaceable parameter “?”

• No filtering:

mSelectionClause = null
mSelectionArgs = arrayOf(""")

getContentResolver().query(  
mURI,  
mProjection,  
mSelectionClause  
mSelectionArgs,  
mSortOrder);
Content Provider Query Results

• As a result of the query, a **Cursor** object is returned
• Cursor provides random read access to returned rows & columns
• `Cursor.getCount()` returns no. of matches
• You can iterate over the Cursor’s rows

https://developer.android.com/guide/topics/providers/content-provider-basics#DisplayResults
Example - Contacts Content Provider

```kotlin
val uri = ContactsContract.CommonDataKinds.Phone.CONTENT_URI
val SELECTED_COLUMNS = arrayOf(
    ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME,
    ContactsContract.CommonDataKinds.Phone.NUMBER
)

val cursor: Cursor? = contentResolver.query(uri, SELECTED_COLUMNS, null, null, null)
if (cursor != null) {
    Log.i(TAG, "Rows: ${cursor.getCount()}")

    val indexName = cursor.getColumnIndex(SELECTED_COLUMNS[0])
    val indexNumber = cursor.getColumnIndex(SELECTED_COLUMNS[1])

    while (cursor.moveToNext()) {
        val name = cursor.getString(indexName)
        val number = cursor.getString(indexNumber)

        Log.i(TAG, "Name: $name; Phone: $number")
    }
}
cursor.close()
```
ListView & Adapter
Adapters

• **Adapter** – bridge between AdapterView and data
  - Binds the view to external data source
  - Initializes and populates the View

• **AdapterView** – view that displays items loaded into an adapter

• Call `Adapter.notifyDataSetChanged()` to refresh the UI

https://developer.android.com/guide/topics/providers/content-provider-basics#DisplayResults
https://developer.android.com/guide/topics/ui/binding
Example with ArrayAdapter

```kotlin
val nameList = mutableListOf<String>()

// Note: simple_list_item_1 is a Android OS default file
val adapter = ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, //layout showing 1 textview
    nameList)

//bind array to the list view
main_listview.setAdapter(adapter)

// update the data
nameList.add("Johnny")

adapter.notifyDataSetChanged()
```

https://developer.android.com/guide/topics/ui/declaring-layout#AdapterViews
Cursor Adapter

- **SimpleCursorAdapter** – An AdapterView implementation for use with Cursors.
  - Specify layout to use for rows
  - Specify which columns to use
  - Cursor *must* include the _ID column!

Example – ContactsProvider, ListView and CursorAdapter

```kotlin
val SELECTED_COLUMNS = arrayOf(
    ContactsContract.CommonDataKinds.Phone._ID,
    ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME,
    ContactsContract.CommonDataKinds.Phone.NUMBER
)

val toViews = intArrayOf(R.id.name_textview, R.id.number_textview)

val cursorAdapter = SimpleCursorAdapter(
    this, // context
    R.layout.list_row, // Layout that defines the views for this list item
    cursor, // the database cursor
    SELECTED_COLUMNS, // "From" - column names of the data to bind to the UI
toViews, // TextViews that should display selected columns
0) // flags used to determine the behaviour of the adapter

main_listview.setAdapter(cursorAdapter)
```

https://developer.android.com/guide/topics/providers/content-provider-basics#DisplayResults
Modifying data

• `ContentResolver.insert()` – returns URI of inserted row

```kotlin
var values = ContentValues()
values.put(Data.RAW_CONTACT_ID, rawContactId)
values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE)
values.put(Phone.NUMBER, "1-800-G00G-411")
values.put(Phone.TYPE, Phone.TYPE_CUSTOM)
values.put(Phone.LABEL, "free directory assistance")
val dataUri = contentResolver.insert(Data.CONTENT_URI, values)
```

• `ContentResolver.update()`
  • Specify query selection criteria
  • Use `ContentValues` to set new values

• `ContentResolver.delete()`
  • Specify query selection criteria

[https://developer.android.com/guide/topics/providers/content-provider-basics#Modifications](https://developer.android.com/guide/topics/providers/content-provider-basics#Modifications)
Finding existing and creating new Content Providers

• To find out about the existing various providers available on Android, check out:

• Create your own Content Provider if
  • You want to share data with other applications
    • E.g. widgets
  • You want to implement custom search suggestions
Creating Content Providers

• Extend ContentProvider class
• Override the necessary methods
  • Insert, delete, update
  • Query handling

• Declare provider in manifest
• For more info refer to the developer portal

https://developer.android.com/guide/topics/providers/content-provider-creating
• This week in lab
  • Broadcast receivers
  • ListView & Adapters