Android

Services, AIDL

Contents

PDM
sayHello();

- Broadcast Receivers
  - System
  - Custom
  - Static
  - Dynamic
- Shared Preferences

Broadcast Receiver

- Android component
- Similar to a publish/subscribe design pattern
- Listen for events
  - Eg: battery low
- Event is described through an **Intent**

Broadcast types

- System
  - Sent by the Android System
    - Public events defined in Intent class
    - Eg: battery low
- Custom
  - Sent by apps
    - Defined inside the application

System broadcast intents: link

Creating a broadcast receiver


```
class MyBroadcastReceiver: BroadcastReceiver() {
    override fun onReceive(p0: Context?, p1: Intent?) {
        Log.d(TAG,  msg: "Received intent $p1")

        // TODO: do actions specific to the intent received
    }

    companion object {
        val TAG: String = this::class.java.toString()
    }
}
```

Declaring a broadcast

- Static registration (AndroidManifest)
  - Always active
- Dynamic registration
  - Registered through Context
  - Active while the registering context is active

Declare static broadcast


```
<!-- If this receiver listens for broadcasts sent from the system or from
     other apps, even other apps that you own, set android:exported to "true". -->
<receiver android:name=".MyBroadcastReceiver" android:exported="false">
    <intent-filter>
        <action android:name="APP_SPECIFIC_BROADCAST" />
    </intent-filter>
</receiver>
```

Steps:

- Declare the receiver using a <receiver> tag
- Declare an intent filter for the receiver

Note: if you want to receive events from other components than your application (from the system, from other apps) you **must** set the **android:exported** property to **true**

Declare dynamic broadcasts


```
val broadcastReceiver = MyBroadcastReceiver()
val intentFilter = IntentFilter(Intent.ACTION_BATTERY_LOW)
val receiverFlags = Context.RECEIVER_EXPORTED;

this.registerReceiver(broadcastReceiver, intentFilter, receiverFlags)
```

Steps:

- Create an instance of the receiver
- Create an instance of the IntentFilter
- Choose wheter the receiver should be exported or not
- Register the receiver

Note: context registered receivers will receive broadcasts as long as their registering context is valid

Sending broadcasts


```
Intent().also { intent ->
    intent.setAction("com.example.broadcast.MY_NOTIFICATION")
    intent.putExtra("data", "Nothing to see here, move along.")
    sendBroadcast(intent)
}
```

Receive with permission

Declaration

```
<receiver android:name=".MyBroadcastReceiver"
          android:permission="android.permission.BLUETOOTH_CONNECT">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_FOUND"/>
    </intent-filter>
</receiver>
```

```
var filter = IntentFilter(Intent.ACTION_FOUND)
registerReceiver(receiver, filter, Manifest.permission.BLUETOOTH_CONNECT, null )
```

Sender

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

Send with permission

Sender:

```
sendBroadcast(Intent(BluetoothDevice.ACTION_FOUND),
              Manifest.permission.BLUETOOTH_CONNECT)
```

Receiver:

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

Shared Preferences

- Save small collections of data
  - Eg: Settings, configurations
- Persistant across app sessions
- Data is saved as **key-value pairs**
- File-based storage
  - XML file
  - In app's private storage area
  - One app can store multiple files
- Public or private

Why use multiple files?

- **Organizational purposes:** to group related data
  - Eg: settings, user data, cache
- **Modular design:** different parts of the app can manage its data
- Performance optimisations: smaller files are faster to read
  - Note: too many files could be hard to manage as well

How to: retrieve shared prefs

- Get preference by file name

```
val sharedPref = activity?.getSharedPreferences(
        getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

- Get activity default shared preferences

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE)
```

How to: write to shared prefs


```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPref.edit()) {
    putInt(getString(R.string.saved_high_score_key), newHighScore)
    apply()
}
```

Steps:

- Create an editor
- Add a new value
  - putInt, putString, etc
- Call apply() or commit() to save

Note: apply() will save the in-memory reference
and write to disk later while commit() will write to disk synchronously

How to: read from shared prefs


```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
val defaultValue = resources.getInteger(R.integer.saved_high_score_default_key)
val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key), defaultValue)
```

Questions

PDM
sayHello();