Parcelable, CardView, SearchView & SharedPreferences

IN721: Mobile Application Development

Kaiako: Grayson Orr

Today's Content

- Parcelable
- CardView
- SearchView
- SharedPreferences

Preparation

- Copy & paste your practical from last session then rename it to Practical04
- Open Practical04 in Android Studio
- In strings.xml, change the app name to Practical04
- Create an activity class called DetailsActivity
- Create a class called RecyclerItemClickListener which extends RecyclerView.SimpleOnItemTouchListener
- Create an interface class called IRecyclerViewItem

Parcelable

Parcelable

- An interface used to serialize a class
- Properties can be transferred from one activity to another
- Legacy implementation - Serializable
- Resource: Parcelable

Album.kt

- How do we add a parcelable implementation to Album.kt?
- Right-click on the class definition > Show Context Actions > Add Parcelable Implementation
- This is Java-esque & a lot of boilerplate. In the practical, you will implement the Kotlin parcelable implementation

getAlbum

- Create a new function called getAlbum() in LastFmRecyclerViewAdapter.kt
- If the ArrayList of Album objects is not empty, return the Album object at nth position

```kotlin
fun getAlbum(position: Int): Album? {
    return if (albums.isNotEmpty()) albums[position] else null
}
```

RecyclerItemClickListener.kt

- Extends RecyclerView.SimpleOnItemTouchListener
- Pass in a reference to Context, RecyclerView & IRecyclerViewItem

```kotlin
class RecyclerItemClickListener(
    context: Context,
    private val recyclerView: RecyclerView,
    private val listener: IRecyclerViewItem
) : RecyclerView.SimpleOnItemTouchListener() {

    // private val gestureDetector

    // override fun onInterceptTouchEvent
}
```

gestureDetector

- Detects gestures & events using the supplied MotionEvent
- Notified when a tap occurs with the up MotionEvent that triggered it
- findChildViewUnder - find the topmost View under the given point
- getChildAdapterPosition - get the adapter position of the View
- Resource: GestureDetectorCompat

```kotlin
private val gestureDetector = GestureDetectorCompat(context, object :
GestureDetector.SimpleOnGestureListener() {
    override fun onSingleTapUp(e: MotionEvent): Boolean {
        val childView: View? = recyclerView.findChildViewUnder(e.x, e.y)
        if (childView != null)
            listener.onItemClick(childView, recyclerView.getChildAdapterPosition(childView))
        return true
    }
})
```

onInterceptTouchEvent

- Called whenever a touch event is detected on the surface of a ViewGroup
- What is the ViewGroup being used in our app?
- If true, the MotionEvent is intercepted & not passed into the child, rather to the onTouchEvent()
- Resource: Gestures ViewGroup

```kotlin
override fun onInterceptTouchEvent(rv: RecyclerView, e: MotionEvent): Boolean {
    return gestureDetector.onTouchEvent(e)
}
```

MainActivity.kt

- In the MainActivity onCreate() & below imageRecyclerView.addOnItemTouchListener

```
imageRecyclerView.addOnItemTouchListener(RecyclerItemClickListener(
    this@MainActivity,
    imageRecyclerView,
    this
))
```

IRecyclerViewItem.kt

- Passed a reference in RecyclerItemClickListener.kt
- Implemented in MainActivity.kt
- Get the View & the adapter position of the View

```kotlin
interface IRecyclerViewItem {
    fun onItemClick(view: View, position: Int)
}
```

onItemClick

- MainActivity.kt implements IRecyclerViewItem.kt
- Get the Album object at nth position in the RecyclerView
- Create a new Intent - MainActivity.kt -> DetailsActivity.kt
- Put Album object's data into the Intent
- Start activity

```kotlin
override fun onItemClick(view: View, position: Int) {
    val album: Album? = lastFmRecyclerViewAdapter.getAlbum(position)
    if (album != null) {
        val intent = Intent(this@MainActivity, DetailsActivity::class.java)
        intent.putExtra("album", album)
        startActivity(intent)
    }
}
```

CardView

activity_details.xml

- Copy activity_details.xml into the layout res directory
- Do not worry...we will discuss the layout structure

CardView

- Used in lists to hold each item's information
- Easy & consistent way to display information
- Cards have default elevation above their containing ViewGroup
- Easy way to contain a group of Views
- Resource: CardView

DetailsActivity.kt

- Extends AppCompatActivity

```kotlin
class DetailsActivity : AppCompatActivity() {

    private lateinit var albumNameText: TextView
    private lateinit var albumImageView: ImageView
    private lateinit var albumPlayCountText: TextView

    // override fun onCreate
}
```

onCreate

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_details)
    val album: Album? = intent.extras?.getParcelable("album")

    albumNameText = findViewById(R.id.album_name_text)
    albumImageView = findViewById(R.id.album_image_view)
    albumPlayCountText = findViewById(R.id.album_play_count_text)

    albumNameText.text = getString(R.string.album_name, album?.name)
    Picasso.with(this@DetailsActivity).load(album?.image)
        .error(R.drawable.ic_album_black_24)
        .placeholder(R.drawable.ic_album_black_24)
        .into(albumImageView)
    albumPlayCountText.text = getString(R.string.play_count, album?.playCount.toString())
}
```

Emulator
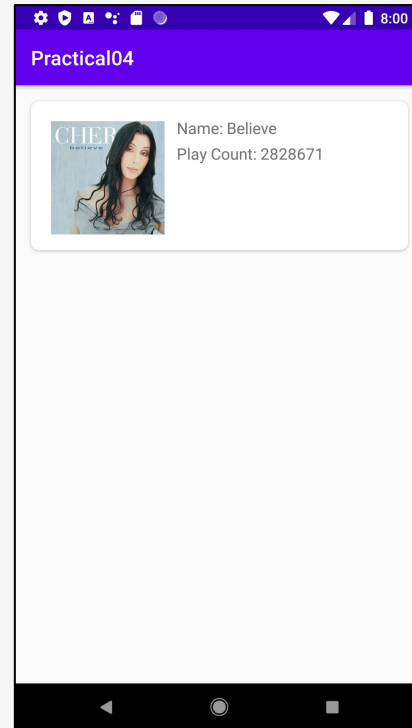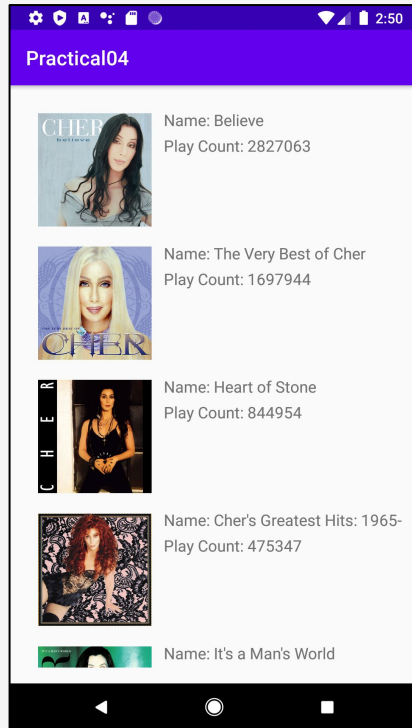
Practical Part 1

- Please use the current app
- Independent tasks:
  - Implement the code as specified in the previous lecture slides
  - To keep the UI consistent, in activity_main.xml, change the layout so that the ImageView & TextViews are contained within a CardView
  - Research: Change the current parcelable implementation using Kotlin's equivalent @Parcelize special annotation
    - Resource: Parcelize - Medium
  - Research: Change the all ImageView widgets to CircleImageView widgets
    - Resource: CircleImageView

Expected Output
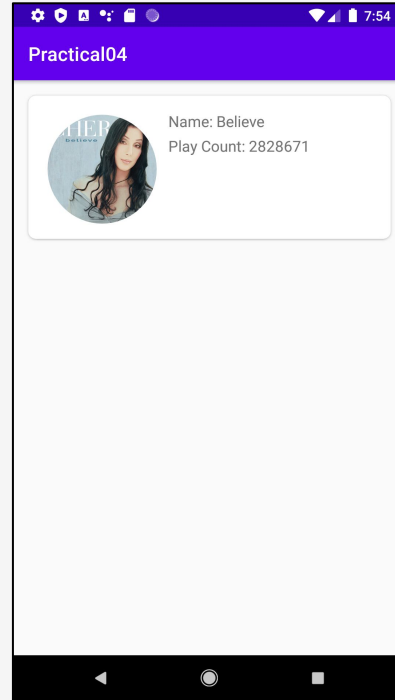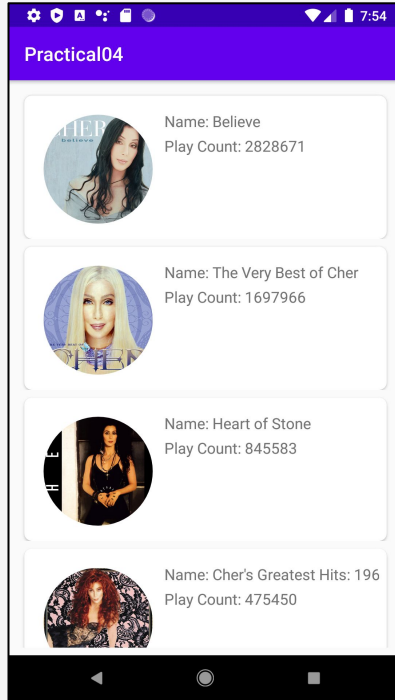
Formative Assessment

- Please write your answers to the following questions in your app:
  - What is a CardView?
  - What is the purpose of a Parcelable class?

SearchView

BaseActivity.kt

- Open class
- Extends AppCompatActivity
- Sets the toolbar as the app for the activity
- Set whether home should be displayed as an "up" affordance
- Resources: Setup App Bar & setDisplayHomeAsUpEnabled

```kotlin
@SuppressLint("Registered")
open class BaseActivity : AppCompatActivity() {
    internal fun displayToolbar(isHomeEnabled: Boolean) {
        setSupportActionBar(findViewById(R.id.toolbar))
        supportActionBar?.setDisplayHomeAsUpEnabled(isHomeEnabled)
    }
}
```

Searchable Configuration

- Create a new res directory called xml
- Create an XML file called searchable.xml. This is the traditional name for a search config file
- The searchable config file must include the <searchable> element as the root node
- We have specified a label & hint attribute
- The label attribute is only required attribute
- @string/search_hint & @string/search_label = Search Artist in strings.xml
- Resource: Creating a Searchable Configuration

```xml
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:hint="@string/search_hint"
    android:label="@string/search_label" />
```

styles.xml

- Three new styles
- windowActionBar - indicates whether this window should have an Action Bar in place of the default title bar
- windowNoTitle - indicates whether there should be no title on this window

```xml
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
<style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar"/>
<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light"/>
```

SearchActivity.kt & activity_search.xml

- Copy SearchActivity.kt into the activities package directory
  - This Activity file has been commented to help you understand how everything works
- Copy activity_search.xml into the layout res directory

AndroidManifest.xml

- SearchView
  - Provides a UI for entering a search query
  - Submitting a request to a search provider
- Reference AppTheme.NoActionBar
- Reference searchable.xml
- Resource: SearchView

```
<activity
  android:name=".activities.SearchActivity"
  android:theme="@style/AppTheme.NoActionBar">
  <intent-filter>
      <action android:name="android.intent.action.SEARCH" />
  </intent-filter>
  <meta-data android:name="android.app.searchable"
      android:resource="@xml/searchable"/>
</activity>
```

Menu Preparation

- Create a new res directory called menu
- Copy menu_main.xml & menu_search.xml into the menu res directory
- Copy ic_search_white_24.xml into the drawables res directory

MainActivity.kt

- Inflate menu_main.xml
- onCreateOptionsMenu()
- onOptionsItemSelected()
- Start activity - MainActivity.kt -> SearchActivity.kt
- Resource: [Menus](#)

```kotlin
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.action_main -> {
            startActivity(Intent(this@MainActivity, SearchActivity::class.java))
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Emulator

- Run app
- The system has called onCreateOptionsMenu()
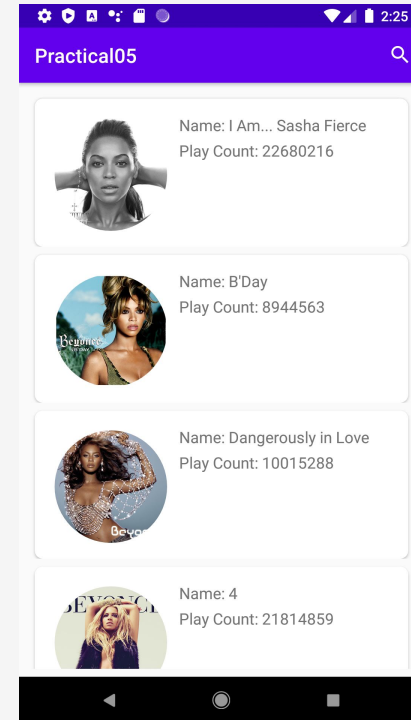
SharedPreferences

SharedPreferences

- An interface for accessing & modifying preference data return by getSharedPreferences
- There is a single instance of this class that all clients share
- Modifications must go through an Editor object
- Provides string consistency guarantees
- Uses expensive operations - this may slow down your app
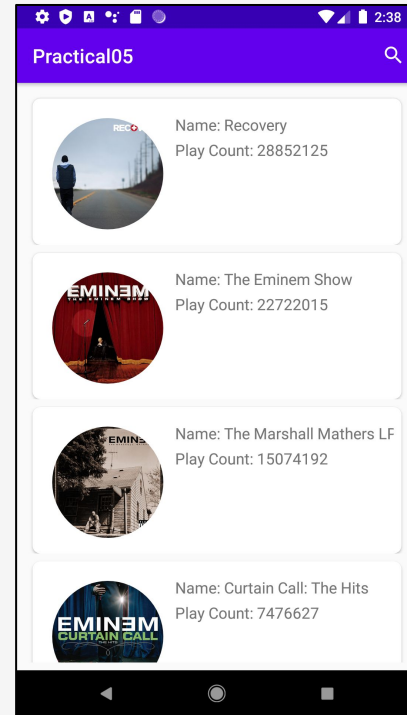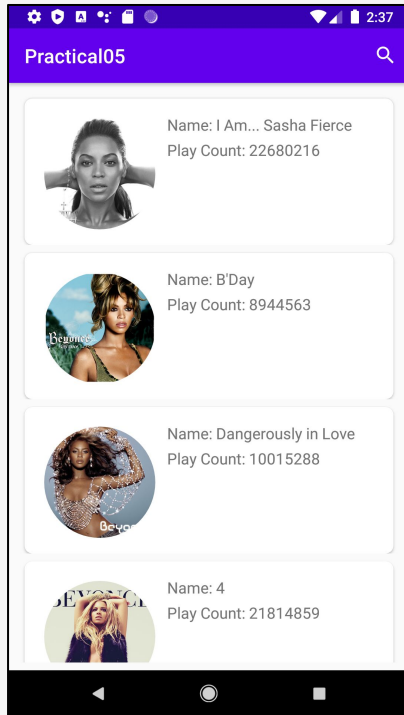- Resource: SharedPreferences

MainActivity.kt

- Get the value with the key album_query from SharedPreferences
- Check if the value is not empty
    - If true, build URI & execute the async task

```kotlin
override fun onResume() {
    super.onResume()
    val sharedPref: SharedPreferences = PreferenceManager.getDefaultSharedPreferences(applicationContext)
    val queryResult: String? = sharedPref.getString("album_query", "")

    if (queryResult!!.isNotEmpty()) {
        val url: String = buildUri(
            getString(R.string.base_url), getString(R.string.method),
            queryResult, getString(R.string.api_key), getString(R.string.format)
        )
        rawDataAsyncTask = RawDataAsyncTask(this)
        rawDataAsyncTask.execute(url)
    }
}
```

Practical Part 2

- Please use the current app
- Independent tasks:
  - Implement the code as specified in the previous lecture slides
  - Once you have completed the practical, create a branch named 03-submission, push the app to the branch, make a pull request & set Grayson-Orr as the reviewer
  - If you do not set Grayson-Orr as a reviewer, I will not mark off your practical
  - DO NOT MERGE YOUR OWN PULL REQUEST!

Expected Output

Formative Assessment

- Please write your answers to the following questions in your app:
    - What does setSearchableInfo do?
    - What is SharedPreferences?