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
- Copy dimens.xml from the **03-parcelable-cardview-searchview-sharedpreferences** directory in the course materials repository
- Create an interface class called IRecyclerViewItem

Parcelable

Parcelable

- When we want to transfer data, i.e., a string from one activity to another we use a putExtra & intent
- When we want to transfer an object from one activity to another we can not do this
- An interface used to serialize a class so its properties can be transferred from one activity to another
    - Object can be represented as a sequence of bytes which includes:
        - Object data
        - Object type
        - Types of data stored in the object
    - After a serialized object has been written, it can be read & deserialized, i.e., the bytes that represent the object can be used to recreate the object in memory
- **writeToParcel** - flatten this object in to a Parcel
- **describeContents** - describes the objects contained in this Parcelable instance
- **createFromParcel** - create a new instance of the Parcelable class. Instantiating it from the given Parcel
- **newArray** - create a new array of the Parcelable class
- 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 in the RecyclerView

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

RecyclerItemClickListener.kt

- Extends RecyclerView.SimpleOnItemTouchListener
    - Listens for an item touch
- Pass in a reference to Context, RecyclerView & IRecyclerViewItem

```
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
    - MotionEvent - object used to report movement events (mouse, pen, finger, trackball)
- Notified when a tap occurs with the up MotionEvent that triggered it
- findChildViewUnder - find the topmost View under the given point
    - Params - horizontal position in pixels to search
    - Params - vertical position in pixels to search
- getChildAdapterPosition - get the adapter position of the View
    - Params - child view to query
- Resource: GestureDetectorCompat

```
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
- If true, the MotionEvent is intercepted & not passed into the child, rather to the onTouchEvent()
- Resource: Gestures ViewGroup

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

MainActivity.kt

- In the MainActivity onCreate()

```
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

```
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

```
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

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
}
```

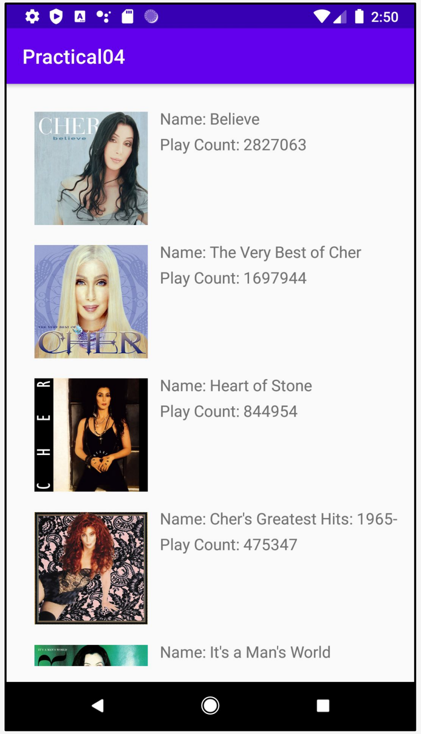
onCreate

```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


2:50

Practical04

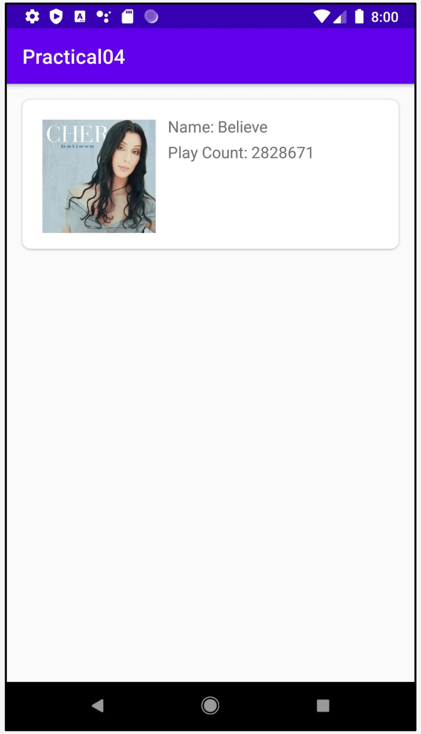
Name: Believe
Play Count: 2827063

Name: The Very Best of Cher
Play Count: 1697944

Name: Heart of Stone
Play Count: 844954

Name: Cher's Greatest Hits: 1965-
Play Count: 475347

Name: It's a Man's World


8:00

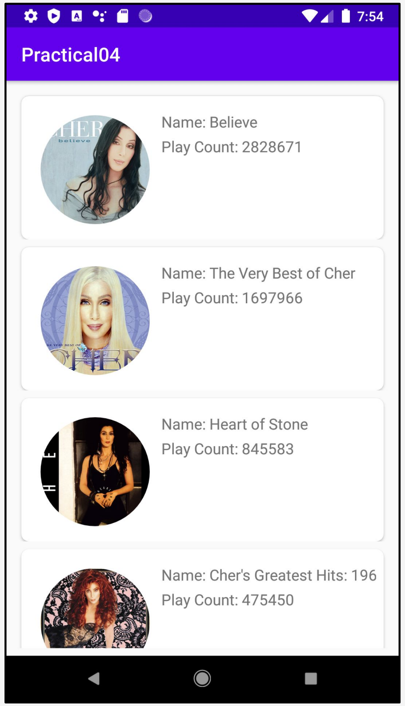
Practical04

Name: Believe
Play Count: 2828671

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 album_item.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



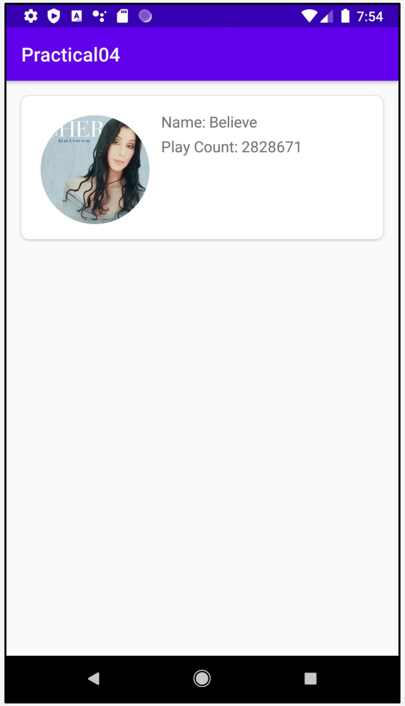
Practical04

Name: Believe
Play Count: 2828671

Name: The Very Best of Cher
Play Count: 1697966

Name: Heart of Stone
Play Count: 845583

Name: Cher's Greatest Hits: 196
Play Count: 475450



Practical04

Name: Believe
Play Count: 2828671

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
  - Classes, functions & variables are final by default...they can not be inherited from other class
  - To make a class, function or variable inheritable from other class, use the open keyword
- Extends AppCompatActivity - base class for activities
- Sets the toolbar as the app for the activity
- Set whether home should be displayed as an “up” affordance
- Resources: Setup App Bar & setDisplayHomeAsUpEnabled

```
open class BaseActivity : AppCompatActivity() {
    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 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

```
<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

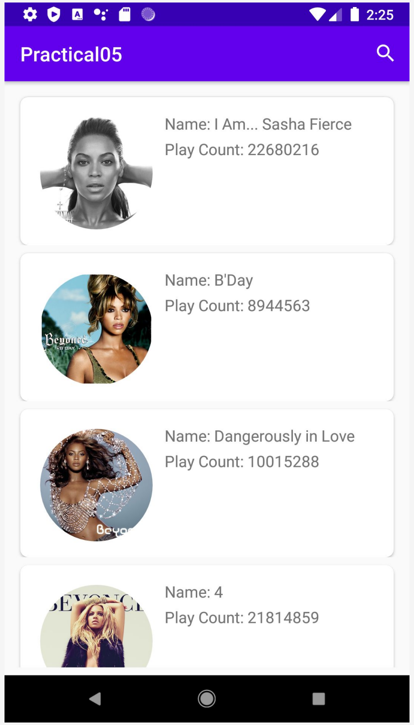
- Inflate menu_main.xml
- onCreateOptionsMenu()
- onOptionsItemSelected()
- Start activity - MainActivity.kt -> SearchActivity.kt
- Resource: Menus

```
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()



Name: I Am... Sasha Fierce
Play Count: 22680216

Name: B'Day
Play Count: 8944563

Name: Dangerously in Love
Play Count: 10015288

Name: 4
Play Count: 21814859

SharedPreferences

SharedPreferences

- There are primarily three ways to store data persistently:
  - SharedPreferences
  - Traditional file systems
  - Relational database through the support of SQLite databases
- An object which saves application data as key/value pairs
  - A name for your data is specified then saved automatically to an XML file for you
  - You can call the getDeafultSharedPreferences() method which returns a SharedPreference instance pointing to the file that contains the values
- Add the following to the build.gradle - implementation 'androidx.preference:preference:1.1.1'
- Resource: SharedPreferences

MainActivity.kt

- onResume()
  - When the activity enters the resumed state, it comes to the foreground then the system invokes the callback
  - This is the state in which the app interacts with the user
  - The app remains in this state until something happens, i.e., receiving a phone call or navigating to another activity
- 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

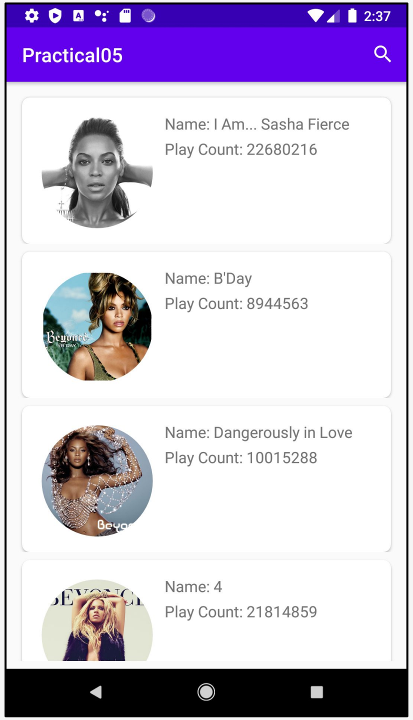
```kotlin
override fun onResume() {
    super.onResume()
    val sharedPref: SharedPreferences = PreferenceManager.getDefaultSharedPreferences(this@MainActivity)
    val queryResult: String? = sharedPref.getString("album_query", "")

    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

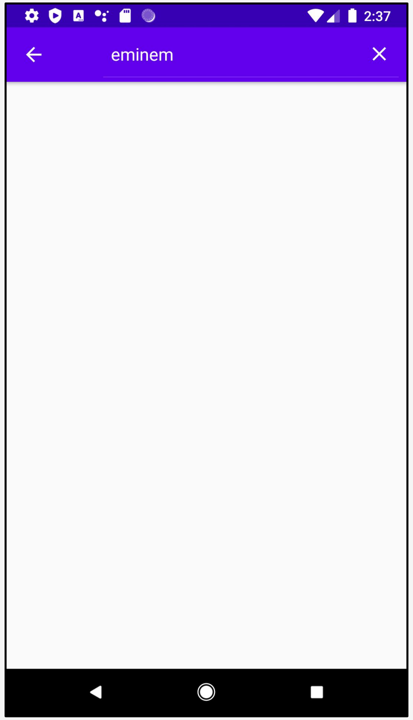


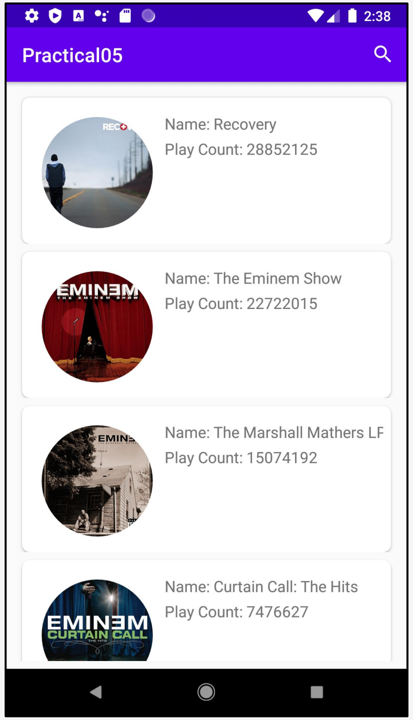
Name: I Am... Sasha Fierce
Play Count: 22680216

Name: B'Day
Play Count: 8944563

Name: Dangerously in Love
Play Count: 10015288

Name: 4
Play Count: 21814859





Name: Recovery
Play Count: 28852125

Name: The Eminem Show
Play Count: 22722015

Name: The Marshall Mathers LP
Play Count: 15074192

Name: Curtain Call: The Hits
Play Count: 7476627

Formative Assessment

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