

Material Design, AsyncTask & RecyclerView

IN721: Mobile Application Development

Kaiako: Grayson Orr

Today's Content

- Material Design
- AsyncTask
- RecyclerView

Preparation

- Instead of creating a new app, copy the app from the last session & rename it to Practical02
- Open Practical02 in Android Studio
- In strings.xml, change the app name to Practcial02

Material Design

Material Design

- Developed by Google in 2014
- Design system
- Inspired by the physical world
- Applied to many Google mobile apps for Android, iOS & the web

Material Design - Components

- Cards, lists & sheets
- Navigation drawers & tabs
- Floating action button
- Text fields, chips & selection controls
- Snackbars, banners & dialogs
- Resource: [Material Design](#)

Material Design - Theming

- Color
- Typography
- Shape
- Resource: [Components](#)

Material Design - Elevation

- Three-dimensional space
- X, Y & Z axis
- Resource: [Elevation](#)

Material Design - Dependencies

- Open up the app module's build.gradle file
- Add the following MDC Android support library to the dependencies block
- Click Sync Now in the top right-hand corner

```
api 'com.google.android.material:material:1.3.0-alpha01'
```

Material Design - Dependencies

- If a dependency is highlighted orange (new version available), update it
- The dependency block should look similar to the following:

```
dependencies {  
    implementation fileTree(dir: "libs", include: ["*.jar"])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"  
    implementation 'androidx.core:core-ktx:1.3.0'  
    implementation 'androidx.appcompat:appcompat:1.1.0'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
    api 'com.google.android.material:material:1.3.0-alpha01'  
}
```

styles.xml

- Go to styles.xml in res > values
- You should only have one style with the name attribute AppTheme
- In the AppTheme's parent attribute, change it to the following:

```
<style name="AppTheme" parent="Theme.MaterialComponents.Light.DarkActionBar">  
    <item name="colorPrimary">@color/colorPrimary</item>  
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>  
    <item name="colorAccent">@color/colorAccent</item>  
</style>
```

activity_main.xml

- Replace each widget except for the EditText widgets with its Material Design equivalent, for example, TextView should be replaced with `com.google.android.material.textview.MaterialTextView`
- There is no Material Design element for a Spinner

```
<com.google.android.material.textview.MaterialTextView
    android:id="@+id/output_text"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginRight="16dp"
    android:gravity="center"
    android:text="Hello World!"
    android:textSize="24sp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

activity_main.xml

- EditText widgets are slightly more complicated
- TextInputLayout
- TextInputEditText

activity_main.xml

- TextInputLayout

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/email_text_input"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:hint="Email Address"
    app:errorEnabled="true"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/months_spinner">

    <!-- TextInputEditText goes here -->

</com.google.android.material.textfield.TextInputLayout>
```

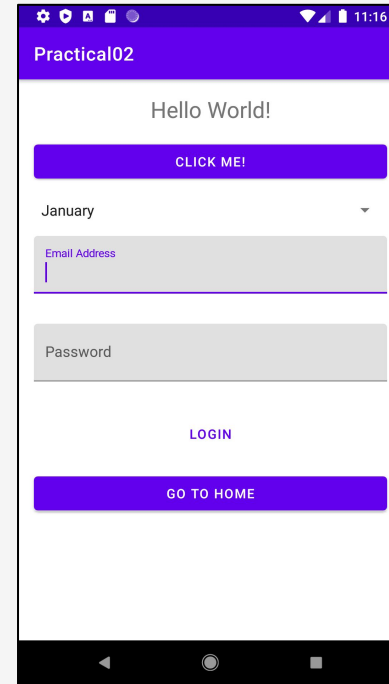
activity_main.xml

- TextInputEditText

```
<com.google.android.material.textfield.TextInputEditText  
    android:id="@+id/email_edit_text"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:inputType="textEmailAddress" />
```

Emulator

- Run app
- You will notice the styles of the default widgets have changed
- Experiment with the colours. The default colours are not visually appealing
- EditText widgets look odd. Lets style even more



styles.xml

- Add the following style to styles.xml

```
<style name="Widget.TextInputLayout" parent="Widget.MaterialComponents.TextInputLayout.OutlinedBox">  
    <item name="boxStrokeColor">@color/colorPrimary</item>  
</style>
```

Emulator

- Run app
- Is there is any visible change?
- Why?

activity_main.xml

- Add style attribute to TextInputLayout

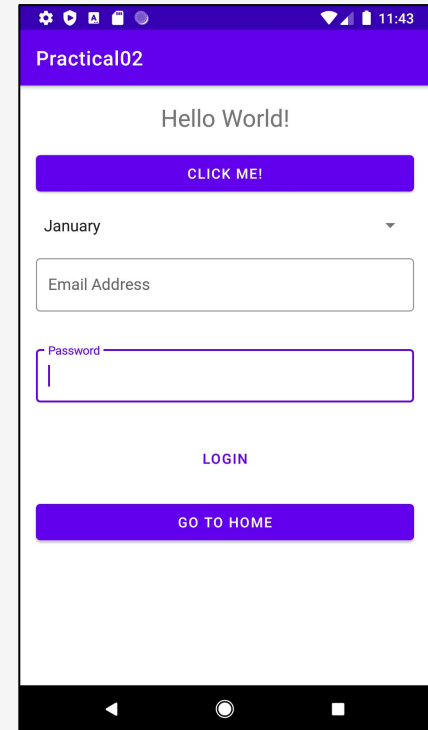
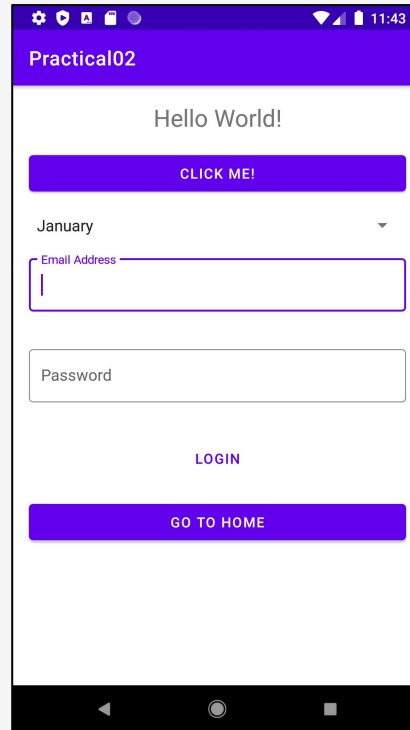
```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/email_text_input"
    style="@style/Widget.TextInputLayout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:hint="Email Address"
    app:errorEnabled="true"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/months_spinner">

    <!-- TextInputEditText goes here -->

</com.google.android.material.textfield.TextInputLayout>
```

Emulator

- Run app
- EditText widgets look better
- No more grey background



Preparation

- Open Practical03 app in Android Studio
- View the res directory, in particular, the layout directory & strings.xml
- View AndroidManifest.xml. What is different about this file?
- Create a data class called Album
- Create two classes called RawDataAsyncTask & LastFmAsyncTask which extends AsyncTask
- Create a class called LastFmRecyclerViewAdapter which extends RecyclerView.Adapter
- Create a class called LastFmViewHolder which extends RecyclerView.ViewHolder
- Create one enum class called DownloadStatus
- Create two interfaces classes called IDataDownloadComplete & IDataDownloadAvailable

Permissions

- What permission have we used?
- If the permission is not declared, the `java.lang.SecurityException` is raised
- Protect privacy of an Android user
- Apps must request permission to sensitive user data
- The system may grant the permission automatically or might prompt the user to approve the request
- We will look more into in a couple of weeks
- Resource: [Permissions](#)

Picasso

- Android Picasso
- An image loading/processing library developed & maintained by Square Inc.
- Simplifies the process of displaying images from external locations
- Most third-party services need to be referenced in the build.gradle (module)
- Resource: [Picasso](#)

```
implementation 'com.squareup.picasso:picasso:2.5.2'
```

Album.kt

- Data class
- Two properties
 - Name
 - Image
- Resource: [Data Classes](#)

```
data class Album(  
    var name: String?,  
    var image: String?  
)
```


DownloadStatus.kt

- Enum class
- Two enums
 - If the response code is 200, the status is OK. If not, the status is NONE
 - You should be checking for different exceptions. Something you all should be thinking about
- Resource: [Enum Classes](#)

```
enum class DownloadStatus {  
    NONE,  
    OK  
}
```

IDataDownloadComplete.kt

- onDownloadComplete
 - Return a JSON response from the API
 - Reference to DownloadStatus.kt

```
interface IDataDownloadComplete {  
    fun onDownloadComplete(data: String, status: DownloadStatus)  
}
```

IDataDownloadAvailable.kt

- onDataAvailable
 - Return an ArrayList of Album objects
- onError

```
interface IDataDownloadAvailable {  
    fun onDataAvailable(data: ArrayList<Album>)  
    fun onError(e: Exception)  
}
```

AsyncTask

Last.fm API

- Last.fm API
- What happens when you click on the resource link below?
- I have given you an API key...is this secure?
- Resource: [Last.fm API](#)

AsyncTask

- Defined by a computation that runs on a background thread
- Whose result is published on the UI thread
- Should ideally be used for short operations, for example, fetching data from an API
- When an async task is executed, the task goes through four steps:
 - `onPreExecute()`
 - `doInBackground()`
 - `onProgressUpdate()`
 - `onPostExecute()`
- Today we will look at `doInBackground()` & `onPostExecute()`
- In week four, we will look at the other two
- Resource: [AsyncTask](#)

RawDataAsyncTask.kt

- Extends AsyncTask
- Reference to IDataDownloadComplete.kt
- Three generic types used by an async task
 - Params - parameters sent to the task upon execution
 - Progress - progress units published during the background computation
 - Result - result of the background computation

```
class RawDataAsyncTask(private val listener: IDataDownloadComplete) :  
    AsyncTask<String, Void, String>() {  
  
    // override fun downloadData  
  
    // override fun doInBackground  
  
    // override fun onPostExecute  
}
```

downloadData

- Private function
- Returns the entire contents of the URL as a String using UTF-8
- Resource: [readText](#)

```
private fun downloadData(urlPath: String): String {  
    return URL(urlPath).readText()  
}
```


doInBackground

- doInBackground
 - Invoked on the background thread immediately after onPreExecute() finishes executing
 - Params of the async task are passed to this step
 - The result must be return by this step & be passed back to the last step

```
override fun doInBackground(vararg url: String): String {  
    var data = ""  
    try {  
        downloadStatus = OK  
        data = downloadData(url[0])  
    } catch (e: Exception) {  
        e.printStackTrace()  
    }  
    return data  
}
```

onPostExecute

- Invoked on the UI thread after the background computation finishes
- The result is passed to this step as a parameter
- result - data returned by the last step
- downloadStatus - OK

```
override fun onPostExecute(result: String) {  
    listener.onDownloadComplete(result, downloadStatus)  
}
```

LastFmAsyncTask.kt

- Extends AsyncTask
- Reference to IDataDownloadAvailable.kt

```
class LastFmAsyncTask(private val listener: IDataDownloadAvailable) :  
    AsyncTask<String, Void, ArrayList<Album>>() {  
  
    // override fun doInBackground  
  
    // override fun onPostExecute  
}
```

doInBackground

- ArrayList of Album objects
- Get data from the JSON response
 - Copy & paste the contents of **try-catch-async-task.txt**
- Return ArrayList of Album
- Resource: [JSON](#)

```
override fun doInBackground(vararg url: String): ArrayList<Album> {  
    val albums = ArrayList<Album>()  
    // try/catch - get data from the JSON response  
    return albums  
}
```

onPostExecute

- result - ArrayList of Albums returned by the last step

```
override fun onPostExecute(result: ArrayList<Album>) {  
    listener.onDataAvailable(result)  
}
```

RecyclerView

LastFmViewHolder.kt

- Describes an item view & metadata about its place within the RecyclerView
- Extends RecyclerView.ViewHolder
- Add fields for caching potentially expensive View.findViewById results
- ViewHolder will hold a TextView & ImageView. Refer to album_item.xml
- Resource: [RecyclerView.ViewHolder](#)

```
class LastFmViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
    var albumNameText: TextView = view.findViewById(R.id.album_name_text)  
    var albumImageView: ImageView = view.findViewById(R.id.album_image_view)  
}
```

LastFmRecyclerViewAdapter.kt

- Extends RecyclerView.Adapter
- Resource: [RecyclerView](#)

```
class LastFmRecyclerViewAdapter(private var albums: ArrayList<Album>) :  
    RecyclerView.Adapter<LastFmViewHolder>() {  
  
    // fun loadNewData  
  
    // override fun getItemCount  
  
    // override fun onCreateViewHolder  
  
    // override fun onBindViewHolder  
  
}
```


loadNewData

- Notify any registered observers that the data has changed
- Two classes of data change events
 - Item changes
 - Structural changes
- Which OO design pattern is being used?
- Resource: [notifyDataSetChanged](#)

```
fun loadNewData(newAlbums: ArrayList<Album>) {  
    albums = newAlbums  
    notifyDataSetChanged()  
}
```

getItemCount

- If the ArrayList of Album objects, return the total number of items held by the adapter
- Resource: [getItemCount](#)

```
override fun getItemCount(): Int {  
    return if (albums.isNotEmpty()) albums.size else 0  
}
```

onCreateViewHolder

- Called when the RecyclerView needs a new RecyclerView.ViewHolder of the given type to represent an item
- Inflating album_item.xml
- Resource: [onCreateViewHolder](#)

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LastFmViewHolder {  
    val view: View =  
        LayoutInflater.from(parent.context).inflate(R.layout.album_item, parent, false)  
    return LastFmViewHolder(view)  
}
```

onBindViewHolder

- Called by the RecyclerView to display the data at the specific position
- ic_baseline_album_black_24.xml has been provided for you
- What is `viewHolder.albumNameText.text` highlighted orange?
 - Resource: [itemView](#)
- Resource: [onBindViewHolder](#)

```
override fun onBindViewHolder(viewHolder: LastFmViewHolder, position: Int) {  
    val album: Album = albums[position]  
  
    viewHolder.albumNameText.text = "Name: " + album.name  
    Picasso.with(viewHolder.albumImageView.context)  
        .load(album.image)  
        .placeholder(R.drawable.ic_baseline_album_black_24)  
        .error(R.drawable.ic_baseline_album_black_24)  
        .into(viewHolder.albumImageView)  
}
```

MainActivity.kt

- Extends AppCompatActivity
- Implement IDataDownloadAvailable.kt & IDataDownloadComplete.kt

```
class MainActivity : AppCompatActivity(), IDataDownloadComplete,
    IDataDownloadAvailable {

    private lateinit var imageRecyclerView: RecyclerView
    private lateinit var rawDataAsyncTask: RawDataAsyncTask
    private lateinit var lastFmRecyclerViewAdapter: LastFmRecyclerViewAdapter

    // private fun buildUri

    // override fun onCreate

    // override fun onDownloadComplete

    // override fun onDataAvailable

    // override fun onError
}
```

buildURI

- Constructing a complex object, i.e. URI
- Which OO design pattern is being used?
- Resource: [URI](#)

```
private fun buildUri(
    baseUrl: String, method: String, artist: String,
    apiKey: String, format: String
): String {
    return Uri.parse(baseUrl)
        .buildUpon()
        .appendQueryParameter("method", method)
        .appendQueryParameter("artist", artist)
        .appendQueryParameter("api_key", apiKey)
        .appendQueryParameter("format", format)
        .build().toString()
}
```

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    lastFmRecyclerViewAdapter = LastFmRecyclerViewAdapter(ArrayList())  
    imageRecyclerView = findViewById(R.id.album_recycler)  
    imageRecyclerView.layoutManager = LinearLayoutManager(this@MainActivity)  
    imageRecyclerView.adapter = lastFmRecyclerViewAdapter  
  
    val url: String = buildUri(  
        getString(R.string.base_url), getString(R.string.method),  
        "cher", getString(R.string.api_key), getString(R.string.format)  
    )  
    rawDataAsyncTask = RawDataAsyncTask(this)  
    rawDataAsyncTask.execute(url)  
}
```

onDownloadComplete

- Check if the status is OK, execute the AsyncTask

```
override fun onDownloadComplete(data: String, status: DownloadStatus) {  
    if (status == OK) {  
        val lastFmAsyncTask = LastFmAsyncTask(this@MainActivity)  
        lastFmAsyncTask.execute(data)  
    }  
}
```


onDataAvailable

- Load the available data into the RecyclerView

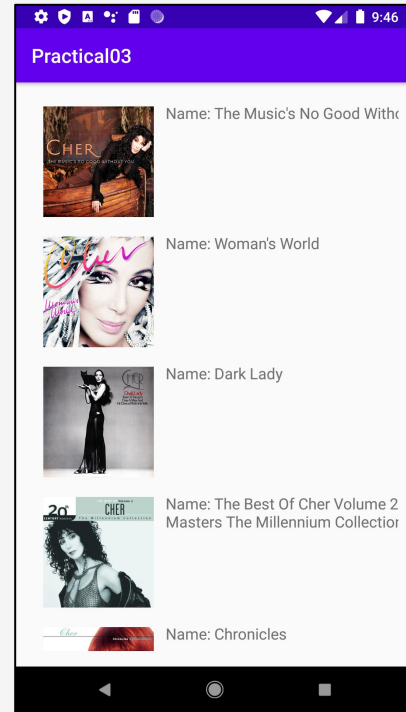
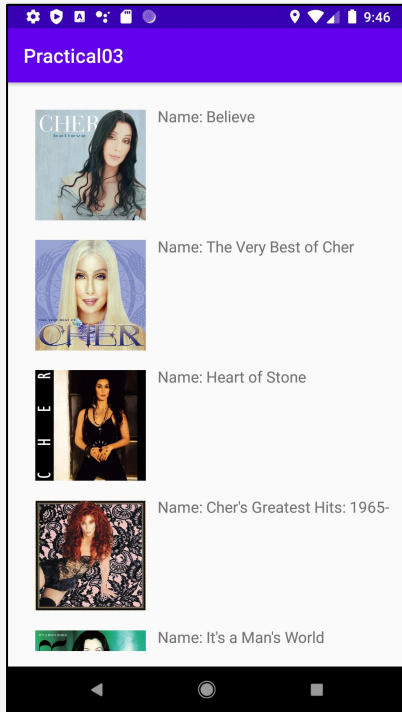
```
override fun onDataAvailable(data: ArrayList<Album>) {  
    Log.d(getString(R.string.TAG), getString(R.string.on_data_available, data))  
    lastFmRecyclerViewAdapter.loadNewData(data)  
}
```

onError

- Log any errors that may occur
- Resource: [Logcat](#)

```
override fun onError(e: Exception) {  
    Log.d(getString(R.string.TAG), getString(R.string.on_error, e.message))  
}
```

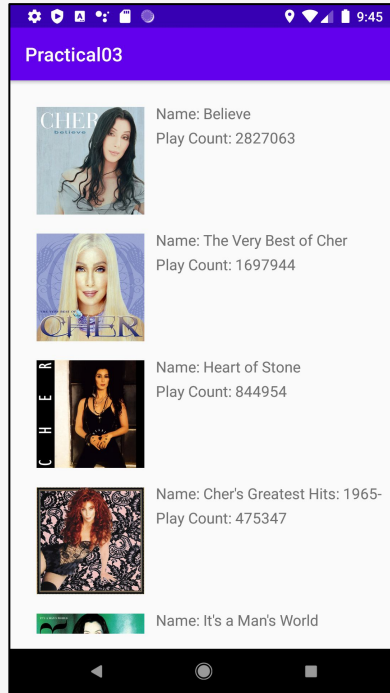
Emulator



Practical

- Please use the current app
- Independent tasks:
 - Implement the code as specified in the previous lecture slides
 - In album_item.xml, add a TextView. The TextView will display the album's play count
 - In Album.kt, add a property for play count
 - In LastFmAsyncTask.kt, get the play count data from the JSON response
 - In LastFmViewHolder.kt, add the View from album_item.xml
 - In LastFmRecyclerViewAdapter.kt, bind the appropriate data to the View in LastFmViewHolder.kt
 - Make sure to use Material Design
 - Once you have completed the practical, create a branch named 02-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!
 - Due Date/Time: 5/8/2020 @ 7.59 am

Expected Output



Formative Assessment

- Please write your answers to the following questions in your app:
 - What are the three generic types used by an async task?
 - What are the four steps that an async task goes through when executed?