



Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# Android Development

Libraries

# Introduction

- Why to use libraries
  - Lots of functions
  - Lots of problems
- Code repetition
  - Someone else already did it!
  - Possibly even better than us
  - Or similar to what we want
- Open source
  - Readable code
  - Understandable code
  - Repairable code
  - Improvable code

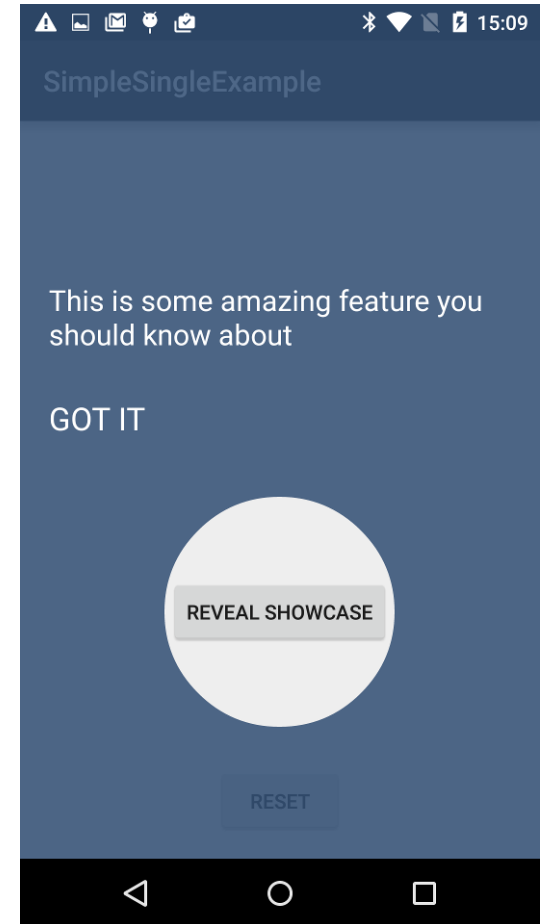
# MaterialShowcaseView

- Interactive help for the application: [link](#)
- Add the lib to the gradle file

```
allprojects {  
    repositories {  
        jcenter()  
        maven { url "https://jitpack.io" }  
    }  
}
```

- Dependency

```
compile 'com.github.deano2390  
        :MaterialShowcaseView:1.3.4'
```



# MaterialShowcaseView

- Example

```
new MaterialShowcaseView.Builder(this)
    .setTarget(mButtonShow)
    .setDismissText("GOT IT")
    .setContentText("This is amazing")
    .setDelay(withDelay) // optional
    .singleUse(SHOWCASE_ID)
        // unique ID used to ensure it is only shown once
    .show();
```

# JSON

- JavaScript Object Notation in short.
- It originated from JavaScript, as its name indicates also, but today it's one of the most popular text base data storing format.
  - The alternative of XML in the web communication.
- It was specified at 2006, in the [RFC4627](#).
- Mostly used for storing and sending structured data.
  - It's shorter than xml so takes less place and has smaller data usage. (Note as it would matter in today's world)
- We will use it to communicate with webservices.

# JSON Example

```
{
  "firstName": "Kovacs",
  "secondName": "Janos",
  "age": 25,
  "address":
  {
    "street": „Harsfa utca 11.",
    "city": „Budapest",
    "zipCode": „1162"
  },
  "weight": 75.3
...
  "phoneNumbers":
  [
    {
      "type": „home",
      "number": "06 1 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

# JSON

- Datatypes

- Number (double-precision floating-point format)
- String (double-quoted(") Unicode (UTF8) with backslash (\) escaping)
- Boolean (true or false)
- Object (an unordered collection of key : value pairs, where the colon (:) character separates the key and value, the pairs are separated with coma (,), the list is within curly braces. The keys have to be String type, and they have to be different from each other)
- null (empty)
- Value (it can be a string, a number, Boolean, object or null)
- Array (an ordered sequence of values separated by comma, within squared brackets)

# Gson

- Gson is a Java library that can be used to convert Java Objects into their JSON representation.
  - It can also be used to convert a JSON string to an equivalent Java object.
  - Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

```
dependencies { implementation 'com.google.code.gson:gson:2.8.2' }
```

- To JSON:

```
gson.toJson("abcd");
```

- From JSON:

```
Integer one = gson.fromJson("1", Integer.class);
```



# Glide

- **Glide** is a fast and efficient open source media management and image loading framework
  - media decoding,
  - memory and disk caching,
  - resource pooling
- Glide supports fetching, decoding, and displaying video stills, images, and animated GIFs.
  - implementation `'com.github.bumptech.glide:glide:4.6.1'`
  - annotationProcessor `'com.github.bumptech.glide:compiler:4.6.1'`
- Example:

```
Glide.with(this)
    .load(url)
    .into(imageView);
```

# Butter Knife

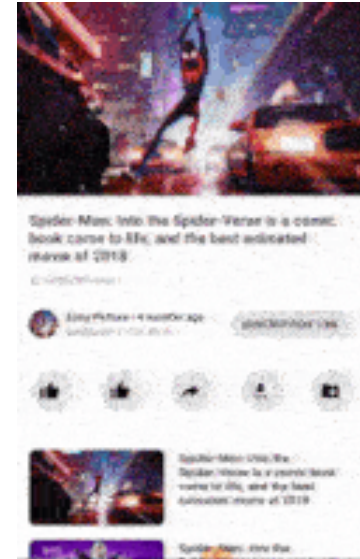
- Butterknife is a popular View "injection" library for Android.
  - This means that the library writes common boilerplate view code for you.
  - Based on annotations to save you time and significantly reduce the lines of boilerplate code written.
- Butterknife uses **compile-time annotations**
  - There is no additional cost at run-time.
  - Instead of slow reflection, code is generated ahead of time.
  - Calling bind delegates to this generated code that you can see and debug.
  - **Butterknife does not slow down your app at all!**

# Butter Knife example

```
dependencies {  
    implementation 'com.jakewharton:butterknife:10.2.1'  
    annotationProcessor 'com.jakewharton:butterknife-compiler:10.2.1'  
}  
  
class ExampleActivity extends Activity {  
    @BindView(R.id.user) EditText username;  
    @BindView(R.id.pass) EditText password;  
  
    @BindString(R.string.login_error) String loginErrorMessage;  
  
    @OnClick(R.id.submit) void submit() { // TODO call server... }  
  
    @Override public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.simple_activity);  
        ButterKnife.bind(this);  
        // TODO Use fields...  
    }  
}
```

# The list is not complete

- Check [Android Arsenal](#) for library search
- Advices for usage
  - Check what the big companies are using
    - Google, Facebook, ...
  - In production don't forget to mention the libraries that you are using in your app (most of the cases the license requires this!)
- And we didn't even talk about all of [these](#)...



# Android Jetpack



# Android Jetpack

- Jetpack is a suite of
  - Libraries
  - Tools
  - guidance.
- These components help you follow best practices, free you from writing boilerplate code, and simplify complex tasks.
- Jetpack comprises the `androidx.*` package libraries, unbundled from the platform APIs.
  - This means that it offers backward compatibility and is updated more frequently than the Android platform.
- Introduced with API 28.

# Android Jetpack Components

- Foundation
  - Foundation components provide cross-cutting functionality like backwards compatibility, testing and Kotlin language support.
- Architecture
  - Architecture components help you design robust, testable and maintainable apps.
- Behavior
  - Behavior components help your app integrate with standard Android services like notifications, permissions, sharing and the Assistant.
- UI
  - Provide widgets and helpers to make your app not only easy, but delightful to use.

# Android Jetpack Components: AndroidX

- Brief history

- When developing apps that support multiple API versions, you may want a standard way to provide newer features on earlier versions of Android or gracefully fall back to equivalent functionality.
- Rather than building code to handle earlier versions of the platform, you can leverage these libraries to provide that compatibility layer.
- To provide additional convenience classes and features not available in the standard Framework API for easier development and support across more devices.



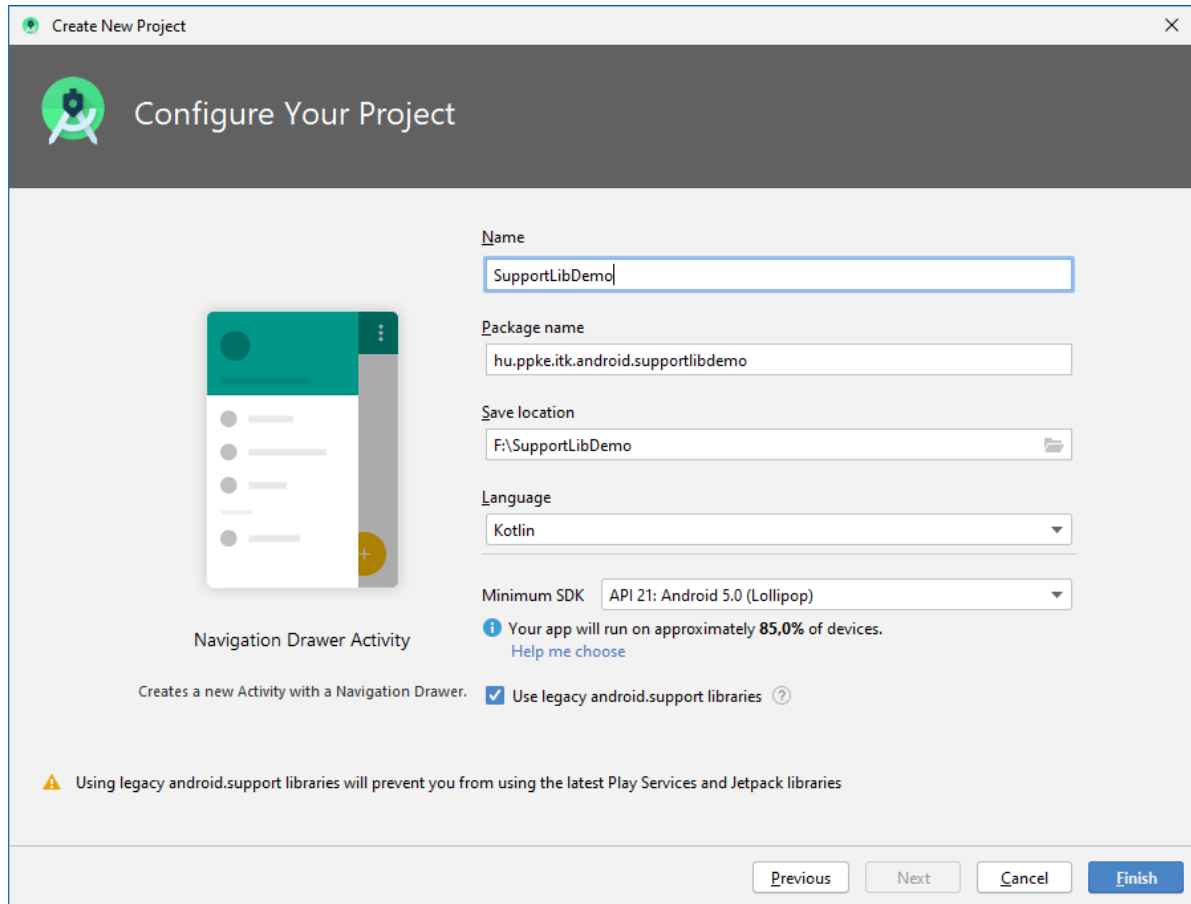
# Detour – Support library?

- Backward compatibility: backport the new functions to older android versions.
- Help developers to provide newer features on earlier API versions of Android or gracefully fall back to equivalent functionality.
  - Originally a single binary library for apps, the Android Support Library has evolved into a suite of libraries for app development.
  - Many of these libraries are now a strongly recommended, if not essential, part of app development.

# Detour – Support library?

- Why is it needed?
  - Without support libraries the developers either stop supporting the older versions or nobody would use the newer features, because they would need to implement it for himself for the older versions
  - Frequent Android version increase but the phones are not upgraded so fast. Lot of older phones cannot be upgraded because of the hardware or manufacturer.
- The (support) libraries will be part of the APK and will be downloaded and installed with it.

# Detour – Project with Support Lib



Create New Project

Configure Your Project

Name  
SupportLibDemo

Package name  
hu.ppke.itk.android.supportlibdemo

Save location  
F:\SupportLibDemo

Language  
Kotlin

Minimum SDK  
API 21: Android 5.0 (Lollipop)

Navigation Drawer Activity

Creates a new Activity with a Navigation Drawer.

☒ Use legacy android.support libraries

Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

Previous Next Cancel Finish

# Detour – Manifest

```
defaultConfig {  
    applicationId "hu.ppke.itk.android.supportlibdemo"  
    minSdkVersion 21  
    targetSdkVersion 29  
    versionCode 1  
    versionName "1.0"  
  
    testInstrumentationRunner  
        "android.support.test.runner.AndroidJUnitRunner"  
}  
  
...  
  
implementation 'com.android.support:appcompat-v7:28.0.0'  
implementation 'com.android.support:support-v4:28.0.0'  
implementation 'com.android.support:design:28.0.0'
```

# Detour – Main Activity

```
import android.support.design.widget.FloatingActionButton
import android.support.design.widget.Snackbar
import android.support.design.widget.NavigationView
import android.support.v4.widget.DrawerLayout
import android.support.v7.app.AppCompatActivity
import android.support.v7.widget.Toolbar

...

class MainActivity : AppCompatActivity() {

...

    val toolbar: Toolbar = findViewById(R.id.toolbar)
    setSupportActionBar(toolbar)
```

# Same with AndroidX

```
import com.google.android.material  
        .floatingactionbutton.FloatingActionButton  
import com.google.android.material.snackbar.Snackbar  
import com.google.android.material.navigation.NavigationView  
import androidx.drawerlayout.widget.DrawerLayout  
import androidx.appcompat.app.AppCompatActivity  
import androidx.appcompat.widget.Toolbar
```

# Using AndroidX libraries

- To use androidx-namespaced libraries
  - Set the compile SDK to Android 9.0 (API level 28) or higher
  - Set both of the following Android Gradle plugin flags to true in your gradle.properties file.
    - `android.useAndroidX`:  
When this flag is set to true, the Android plugin uses the appropriate AndroidX library instead of a Support Library.  
The flag is false by default if it is not specified.
    - `android.enableJetifier`: When this flag is set to true, the Android plugin automatically migrates existing third-party libraries to use AndroidX dependencies by rewriting their binaries.  
The flag is false by default if it is not specified

# Using AndroidX libraries

- All Jetpack components are available on the Google Maven repository.
- In build.gradle add the google() repository as shown below:

```
allprojects {  
    repositories {  
        google()  
        jcenter()  
    }  
}
```



# JetPack – Details

- Foundation
  - Android KTX
    - Write more concise, idiomatic Kotlin code
  - AppCompat
    - Degrade gracefully on older versions of Android
  - Car
    - Components to help develop Android apps for cars
  - Benchmark
    - Quickly benchmark your Kotlin-based or Java-based code from within Android Studio

# JetPack – Details

- Foundation
  - Multidex
    - Provide support for apps with multiple DEX files. (Prior to API 21)
  - Security
    - Read and write encrypted files and shared preferences by following security best practices.
  - Test
    - An Android testing framework for unit and runtime UI tests
  - TV
    - Components to help develop apps for Android TV
  - Wear OS by Google
    - Components to help develop apps for Wear

# JetPack – Detail – Foundation

- Android KTX is a set of Kotlin extensions
- KTX extensions provide concise, idiomatic Kotlin
- To do so, these extensions leverage several Kotlin language features, including the following:
  - Extension functions
  - Extension properties
  - Lambdas
  - Named parameters
  - Parameter default values
  - Coroutines

# JetPack – Details – Foundation

- Security library provides an implementation of the security best practices
  - reading and writing data at rest, as well as key creation and verification.
- The library uses the builder pattern to provide safe default settings for the following security levels:
  - **Strong security that balances great encryption and good performance.**
    - This level of security is appropriate for consumer apps, such as banking and chat apps, as well as enterprise apps that perform certificate revocation checking.
  - **Maximum security.**
    - This level of security is appropriate for apps that require a hardware-backed keystore and user presence for providing key access.

# JetPack – Details – Foundation

- Car, TV, Wear
  - Later
- Test
  - (Partially) last time

# JetPack – Details

- Architecture
  - Data Binding
    - Declaratively bind observable data to UI elements
  - Lifecycles
    - Manage your activity and fragment lifecycles
  - LiveData
    - Notify views when underlying database changes
  - Navigation
    - Handle everything needed for in-app navigation

# JetPack – Details

- Architecture
  - Paging
    - Gradually load information on demand from your data source
  - Room
    - Fluent SQLite database access
  - ViewModel
    - Manage UI-related data in a lifecycle-conscious way
  - WorkManager
    - Manage your Android background jobs

# JetPack – Details – Architecture

- The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.
- Layouts are often defined in activities with code that calls UI framework methods.



# Example – Architecture

- A code that calls `findViewById()` to find a `TextView` widget and bind it to the `userName` property of the `viewModel` variable.

```
findViewById<TextView>(R.id.sample_text).apply {  
    text = viewModel.userName  
}
```

- The following example shows how to use the Data Binding Library to assign text to the widget directly in the layout file.

```
<TextView  
    android:text="@{viewModel.userName}" />
```

# JetPack – Details – Architecture

- The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way.
  - The ViewModel class allows data to survive configuration changes such as screen rotations.
- Why?
  - The Android framework manages the lifecycles of UI controllers, such as activities and fragments.
    - The framework may decide to destroy or re-create a UI controller in response to certain user actions or device events that are completely out of your control.

# JetPack – Details – Architecture

- Why?
  - If the system destroys or re-creates a UI controller, any transient UI-related data you store in them is lost
    - For example, your app may include a list of users in one of its activities.
    - When the activity is re-created for a configuration change, the new activity has to re-fetch the list of users.
  - Another problem is that UI controllers frequently need to make asynchronous calls that may take some time to return.
    - The UI controller needs to manage these calls and ensure the system cleans them up after it's destroyed to avoid potential memory leaks.
    - This management requires a lot of maintenance, and in the case where the object is re-created for a configuration change, it's a waste of resources since the object may have to reissue calls it has already made.

# JetPack – Details – Architecture

- The WorkManager API makes it easy to schedule deferrable, asynchronous tasks that are expected to run even if the app exits or device restarts.
- Key features:
  - Backwards compatible up to API 14
    - Uses JobScheduler on devices with API 23+
    - Uses a combination of BroadcastReceiver + AlarmManager on devices with API 14-22
  - Add work constraints like network availability or charging status
  - Schedule asynchronous one-off or periodic tasks
  - Monitor and manage scheduled tasks
  - Chain tasks together
  - Ensures task execution, even if the app or device restarts
  - Adheres to power-saving features like Doze mode

# JetPack – Details

- Behavior
  - CameraX
    - Easily add camera capabilities to your apps
  - Media & playback
    - Backwards-compatible APIs for media playback and routing (including Google Cast)
  - Notifications
    - Provides a backwards-compatible notification API with support for Wear and Auto
  - Permissions
    - Compatibility APIs for checking and requesting app permissions

# JetPack – Details

- Behavior
  - Preferences
    - Create interactive settings screens
  - Sharing
    - Provides a share action suitable for an app's action bar
  - Slices
    - Create flexible UI elements that can display app data outside the app

# JetPack – Details

- UI
  - Animation & transitions
    - Move widgets and transition between screens
  - Emoji
    - Enable an up-to-date emoji font on older platforms
  - Fragment
    - A basic unit of composable UI
  - Layout
    - Lay out widgets using different algorithms

# JetPack – Details

- UI
  - Palette
    - Pull useful information out of color palettes
  - ViewPager2
    - Create swipe views between tabs
  - WebView
    - Deliver a web page or web application as a part of an app





Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# REST

# REST API

- **Representational State Transfer (REST)** is an architectural style that defines a set of constraints and properties based on HTTP.
- Web Services that conform to the REST architectural style, or **RESTful** web services, provide interoperability between computer systems on the Internet.

# REST API

- A REST style architecture have a client and a server.
  - The clients start requests to the server.
  - The servers process requests and they send back the right answer.
- 6 constraints for defining the architecture (the implementation is not constrained):
  - Client-Server Architecture
  - Statelessness
  - Cacheability
  - Layered system
  - Code on demand (optional)
  - Uniform interface

# REST API – constraints

- Client – server architecture
  - The client and the server are separated via a unified interface (client is not storing data, only shows it to the user)
  - Server and client are replaceable, can be developed separately, only the interface is given.
- Statelessness
  - The client–server communication is constrained by no client context being stored on the server between requests.
  - Each request from any client contains all the information necessary to service the request, and session state is held in the client.
- Cacheability
  - Clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from reusing stale or inappropriate data in response to further requests.

# REST API – constraints

- Layered system
  - A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.
  - Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.
- Code on demand (optional)
  - Servers can temporarily extend or customize the functionality of a client by transferring executable code.
- Uniform interface
  - The uniform interface constraint is fundamental to the design of any REST service. It simplifies and decouples the architecture, which enables each part to evolve independently.

# Uniform Interface

- There are four constraints for the uniform interface:
- **Resource identification in requests**
  - Individual resources are identified in requests, for example using URIs in Web-based REST systems. For example, the server may send data from its database as JSON objects.
- **Resource manipulation through representations**
  - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.

# Uniform Interface

- There are four constraints for the uniform interface:
- **Self-descriptive messages**
  - Each message includes enough information to describe how to process the message.
- **Hypermedia as the engine of application state**
  - Having accessed an initial URI for the REST application a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs.

# Structure of Web Services

- The HTTP based RESTful API-s have the following components:

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as <code>http://api.example.com/resources/</code>	<b>List</b> the URIs and perhaps other details of the collection's members.	<b>Replace</b> the entire collection with another collection.	<b>Create</b> a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	<b>Delete</b> the entire collection.
Element, such as <code>http://api.example.com/resources/item17</code>	<b>Retrieve</b> a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	<b>Replace</b> the addressed member of the collection, or if it does not exist, <b>create</b> it.	Not generally used. Treat the addressed member as a collection in its own right and <b>create</b> a new entry within it.	<b>Delete</b> the addressed member of the collection.



# Homework

- First checkpoint!
  - Deadline – next week! (04/08)
  - I'll assess your progress and provide feedback.



# Camera and Media management

Next week