Project: Innovative Open Source Courses for Computer Science

# Mobile Application Development Teaching Material

Radosław Maciaszczyk
West Pomeranian University of Technology in Szczecin

30.05.2021

Innovative Open Source Courses for Computer Science

This teaching material was written as one of the outputs of the project "Innovative Open Source Courses for Computer Science", funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

## Project information

Project was implemented under the Erasmus+.
Project name: **"Innovative Open Source courses for Computer Science curriculum"**
Project nr: **2019-1-PL01-KA203-065564**
Key Action: **KA2 – Cooperation for innovation and the exchange of good practices**
Action Type: **KA203 – Strategic Partnerships for higher education**

### Consortium

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
MENDELOVA UNIVERZITA V BRNE
ZILINSKA UNIVERZITA V ZILINE

### Erasmus+ Disclaimer

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

### Copyright Notice

Co-funded by the
Erasmus+ Programme
of the European Union

# MOBILE APPLICATION DEVELOPMENT

## Introduction

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

# What is mobile device

# What is mobile device

Main differences ?

- CPU
- Battery

# What is mobile device

Main differences ?

- CPU
- Battery
- Sensors

# What is mobile device

Main differences ?

- CPU
- Battery
- Sensors
- Connectivity

# What do you expect from a mobile devices

- ??

# What do you expect from a mobile devices

- ??
- Hom many devices do you have

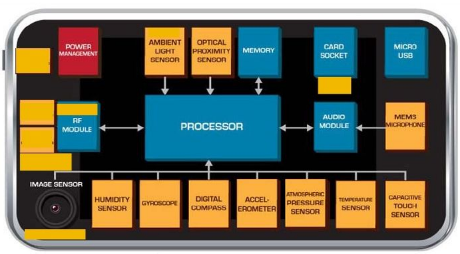
# What do you expect from a mobile devices

- ??
- Hom many devices do you have
- In future - one device many applications

# What do you expect from a mobile devices

- ??
- Hom many devices do you have
- In future - one device many applications
- ~~In future~~ no NOW

# Device for Developers



http://pl.scribd.com/doc/98309084/
Fusing-Sensors-Into-Mobile-OSes-Innovative-Use-Cases-Submitted-5-23-12

# Main vendors

- May 2021 - `gs.statcounter.com/vendor-market-share/mobile`
- Samsung 27,84 %
- Apple 26,47 %
- Xioami 10,62 %
- Huawei 8,85 %
- Oppo 5,39 %

# Mobile operating system

- May 2021 - `gs.statcounter.com/os-market-share/mobile/worldwide`
- Android 72,72 %
- iOS 26,47 %
- Samsung 0,4 %
- KaiOS 0,17 %
- Unknown 0,17 %

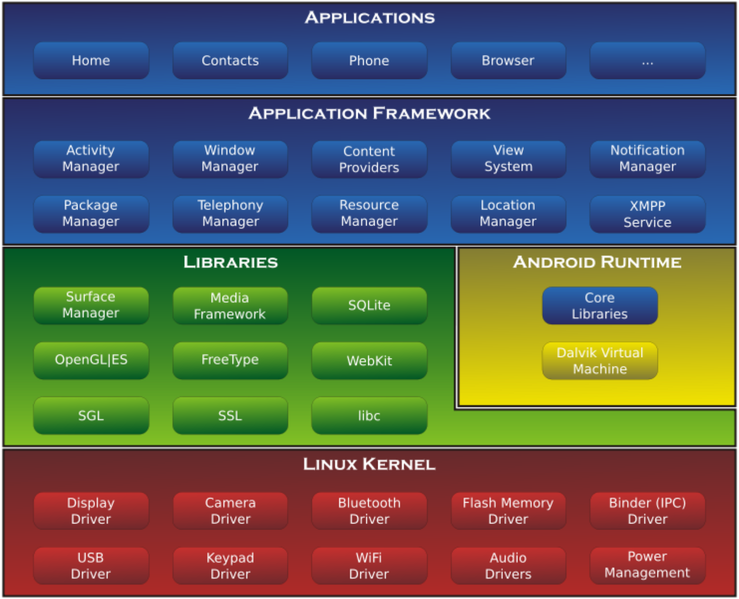
# What is Android?

- An open source software stack that includes
  - Operating system
  - Middleware
  - Key mobile applications (Web browser, PIM, SMS, Email...)
  - API libraries for writing mobile applications
- Open-source development platform for creating mobile applications
- Linux based operating system
- Generally for touchscreen mobile devices such as smartphones and tablet computers

# Where is Android

- Phones
- Tablets
- TVs
- STB - set-top-box
- robots
- will be in cars
- will be in flight entertainment systems on planes
- Android is everywhere !

# Android Software Stack



APPLICATIONS

Home

Contacts

Phone

Browser

...

APPLICATION FRAMEWORK

Activity Manager

Window Manager

Content Providers

View System

Notification Manager

Package Manager

Telephony Manager

Resource Manager

Location Manager

XMPP Service

LIBRARIES

Surface Manager

Media Framework

SQLite

OpenGL|ES

FreeType

WebKit

SGL

SSL

libc

ANDROID RUNTIME

Core Libraries

Dalvik Virtual Machine

LINUX KERNEL

Display Driver

Camera Driver

Bluetooth Driver

Flash Memory Driver

Binder (IPC) Driver

USB Driver

Keypad Driver

WiFi Driver

Audio Drivers

Power Management

# Application Fundamentals

- Android applications are written in the Java or Kotlin programming language. - Generally
- The Android SDK tools compile the code (along with any data and resource files) into an Android package, an archive file with an .apk suffix
- All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

# Application Fundamentals

- Android applications are written in the Java or Kotlin programming language. - Generally
- The Android SDK tools compile the code (along with any data and resource files) into an Android package, an archive file with an .apk suffix
- All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

# Application Fundamentals

- Android applications are written in the Java or Kotlin programming language. - Generally
- The Android SDK tools compile the code (along with any data and resource files) into an Android package, an archive file with an .apk suffix
- All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

# Android application

- Android application lives in its own security sandbox,
- Andorid OS is multi-user Linux system,
- Each application is a different user, (by default)
- Each process has its own virtual machine (VM)
- By default, every application runs in its own Linux process.
- Android starts the process when any of the application's components need to be executed
- Shuts down the process when it's no longer needed or when the system must recover memory for other applications.

# “Principle of least privilege”

- Each application, by default, has access only to the components that it requires to do its work and no more.
- An application cannot access parts of the system for which it is not given permission
- Question: How share data with other applications ?

# Share data with other applications

- Share the same Linux user ID, in which case they are able to access each other's files
- Create or use Content Provider
- Store data on SDCard
- Important: An application can request permission to access device data such as: the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more

# Application Components

Four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle.

- Activities
- Services
- Content providers
- Broadcast receivers

# Activity

- Represents a single screen with a user interface
- Multi activities in one app
- All activity has own lifecycle
- An activity is implemented as a subclass of Activity
- eg. Mail app: read mail, compose mail...

# Service

- Service runs in the background to perform long-running operations or to perform work for remote processes
- service does not provide a user interface
- has own lifecycle
- e.g. Music app: service might play music in the background while the user is in a different application
-

# Content providers

- Content Provider manages a shared set of application data
- Other apps can modify data, without knowledge of the detailed architecture
- It is also good practice to use content provider as internal
- System Content Providers store information line Contact, Photos, Video...

# Broadcast receivers

- Mechanism to send or receive broadcast messages from the Android system and other Android apps
- Similar to ”publish-subscribe” design pattern
- When send broadcast message other apps must subscribe this type of message
- There are many system messages e.g. System send message after boot is completed, battery is low...

# Intent

- This is seperate mechanism which we use to runs three of the four component types-activities, services, and broadcast receivers
- With intent me send informatio about action and data
- Depending on the component, we define actions differently

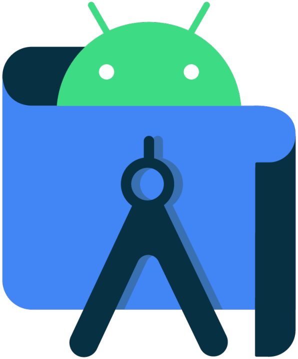
# The Manifest File

- All information about compoment must exist in a *AndroidManifest.xml* file
- Especial we must publish information about main activity
- We also must publish information about permission
- Application requires
- Declare the minimum API Level
- Declare hardware and software features used or required (camera, bluetooth services, or a multitouch screen etc)
- API libraries (other than the Android framework APIs), such as the Google Maps library.

# AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
^^I<uses-permission android:name="android.permission. ..."/>
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        <service>
        </service>
        <receiver>
        </receiver>
        <provider>
        </provider>
        ...
    </application>
</manifest>
^^I
```

# Live Example - Hello World



# Introduction to Android Studio

- https://www.youtube.com/watch?v=K2dodTXARqc
- https://www.youtube.com/user/androiddevelopers/

# Solutions for common Android development problems

- http://www.vogella.com/articles/AndroidDevelopmentProblems/article.html
- http://d.android.com
- http://stackoverflow.com/questions/tagged/android

# MOBILE APPLICATION DEVELOPMENT

## Component Lifecycle

Innovative Open Source courses for Computer Science

30.05.2021

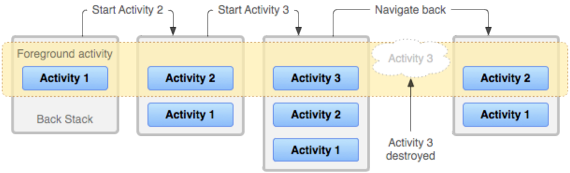
Funded by the European Union

# Activity

- An Activity is a component that provides a screen with which users can interact in order to do something
- Each activity is given a window in which to draw its user interface.
- The window typically fills the screen, sometimes may be smaller
- An application consists of multiple activities that are loosely bound to each other.

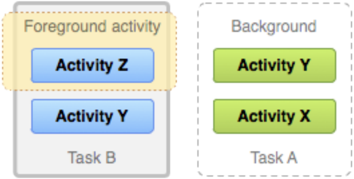
# Multiply Activities - how arrange

- Each activity can then start another activity in order to perform different actions.
- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack").
- When a new activity starts, it is pushed onto the back stack and takes user focus.
- The back stack is a basic "last in, first out" stack mechanism



# Multiply Task - how arrange

- When user start app first time or app is destroyed, the new task is created.
- When app exist, that application's task comes to the foreground



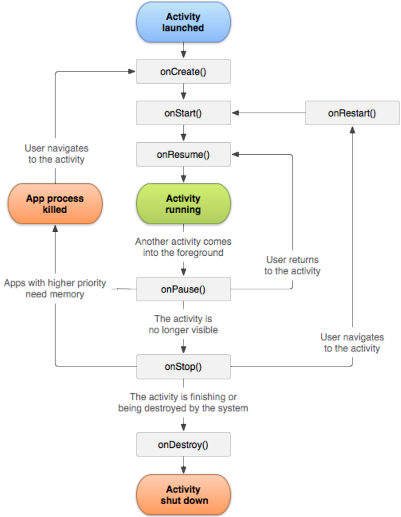
# Creating an Activity

- To create an activity, you must create a subclass of Activity (or an existing subclass of it)
- Activity has seven callback methods
- We must declare only one *onCreate()*
- Other depends on the application requirements
- Predictability activity depends on understanding the activity lifecycle

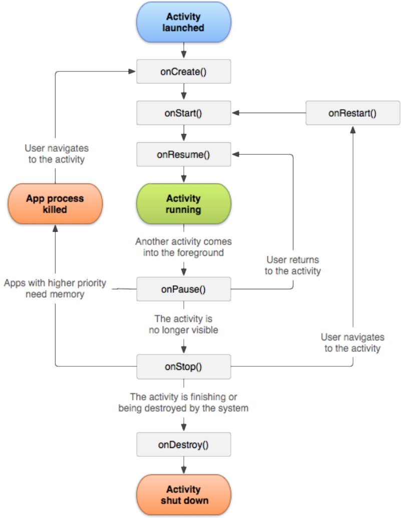
# Implementing the lifecycle callbacks - skeleton

```kotlin
class NewActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_new)
        // The activity is being created.
    }
    override fun onPause() {
        super.onPause()
        // Another activity is taking focus (this activity is about to be "paused").
    }
    override fun onRestart() {
        super.onRestart()
    }
    override fun onResume() {
        super.onResume()
        // The activity has become visible (it is now "resumed").
    }
    override fun onStart() {
        super.onStart()
        // The activity is about to become visible.
    }
    override fun onStop() {
        super.onStop()
        // The activity is no longer visible (it is now "stopped")
    }
    override fun onDestroy() {
        super.onDestroy()
        // The activity is about to be destroyed.
    }
}
```

# Activity Lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

# Activity Lifecycle

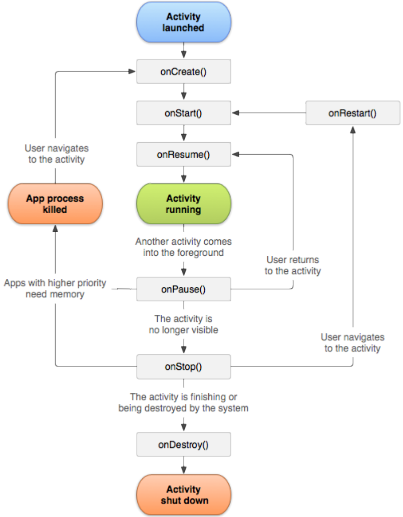


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

## onCreate()

Called when the activity is first created. You should set up - create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state. Always followed by *onStart()*.

# Activity Lifecycle



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

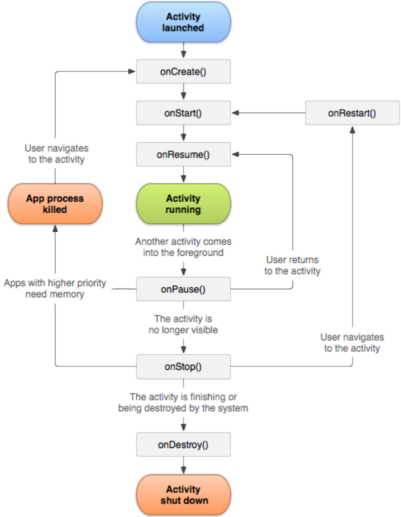
## onRestart()

Called after the activity has been stopped, just prior to it being started again.
Always followed by *onStart()*.

## onStart()

Called just before the activity becomes visible to the user.
Followed by *onResume()* if the activity comes to the foreground, or *onStop()* if it becomes hidden.

# Activity Lifecycle



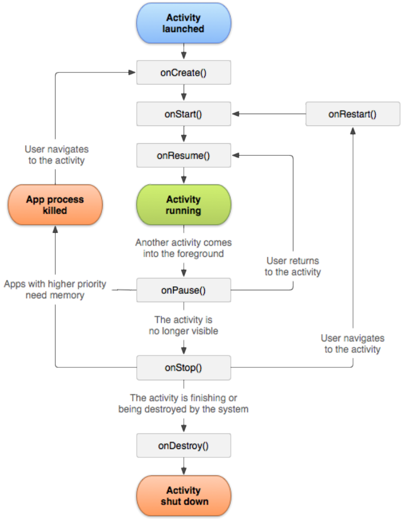
Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

## onResume()

Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it.

Always followed by *onPause()*.

# Activity Lifecycle

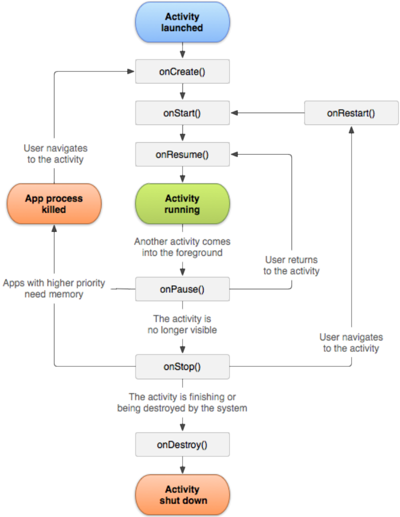

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

## onPause()

Called when the system is about to start resuming another activity. You should: commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It shoulddo whatever it does very quickly, the next activity will not be resumed until it returns.

Followed either by *onResume()* if the activity returns back to the front, or by *onStop()* if it becomes invisible to the user.

# Activity Lifecycle

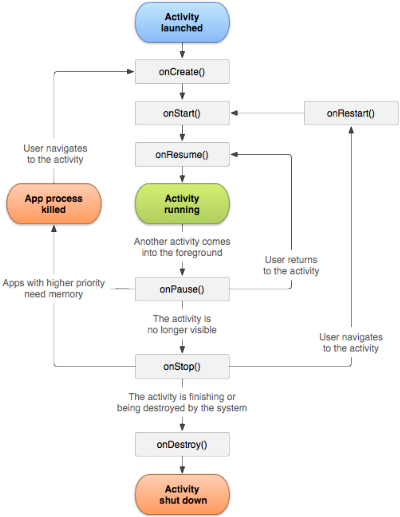


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

## onStop()

Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
Followed either by *onRestart()* if the activity is coming back to interact with the user, or by *onDestroy()* if this activity is going away.

# Activity Lifecycle



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

## onDestroy()

Called before the activity is destroyed. This is the final call that the activity will receive.

# More information

- https://developer.android.com/guide/components/activities/activity-lifecycle

# Saving Activity State

- By default, the system uses the Bundle instance state to save information about each View object in your activity layout, but not save all information.
- However, you can (and **should**)proactively retain the state of your activities using *onSaveInstanceState()* method.
- As your activity begins to stop, the system calls the onSaveInstanceState()
- This methods use key-value pairs to save state

```
override fun onSaveInstanceState(outState: Bundle?) {
    // Save the user's current game state
    outState?.run {
        putInt(STATE_SCORE, currentScore)
        putInt(STATE_LEVEL, currentLevel)
    }

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(outState)
}

companion object {
    val STATE_SCORE = "playerScore"
    val STATE_LEVEL = "playerLevel"
}
```

# Restore Activity State

- You can use *onCreate()* or *onRestoreInstanceState()*.
- This methods receive the same Bundle that contains the instance state information.

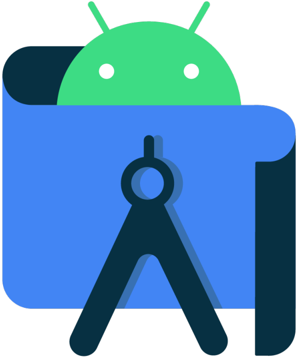
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState) // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        with(savedInstanceState) {
            // Restore value of members from saved state
            currentScore = getInt(STATE_SCORE)
            currentLevel = getInt(STATE_LEVEL)
        }
    } else {
        // Probably initialize members with default values for a new instance
    }
}
```

```
override fun onRestoreInstanceState(savedInstanceState: Bundle?) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState)

    // Restore state members from saved instance
    savedInstanceState?.run {
        currentScore = getInt(STATE_SCORE)
        currentLevel = getInt(STATE_LEVEL)
    }
}
```

# Live Example - Android LifeCycle



# Navigating between activities

- To start activity we can use two methods *startActivity()* or *startActivityForResult()*
- This methods require Intent object wich contains information about Activity

Explicit

```
val intent = Intent(this, OtherActivity::class.java)
startActivity(intent)
```

Implicit

```
val intent = Intent(Intent.ACTION_SEND).apply {
    putExtra(Intent.EXTRA_EMAIL, recipientArray)
}
startActivity(intent)
```

# Start activity and waiting for Result [1/2]

- When we need receive result we must use *startActivityForResult()*
- Impement this methods we must add code inside two activity

## First Activity - Fire second Activity

```
companion object {
        const val REQUEST_CODE = 67 //declare request code
    }
    fun activityCall() {
    val intent = Intent(this, OtherActivity::class.java)
    startActivityForResult(intent,REQUEST_CODE)
    }
```

## Implement Receive methods

```
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        // Check which request we're responding to
        if (requestCode == REQUEST_CODE) {
            // Make sure the request was successful
            if (resultCode == Activity.RESULT_OK) {
                // Do something with the data here
            }
        }
    }
```

# Start activity and waiting for Result [2/2]

Send data from Second Activity

```
    fun responseButton(view: View){
        Log.i(TAG, "responseButton")
        val returnIntent = Intent()
        returnIntent.putExtra("result", "data from secondActivity")

        setResult(RESULT_OK, returnIntent)
        finish()
    }
```

# Intent

- Starting an activity
- Starting a service
- Delivering a broadcast
- Explicit Intent - specify which application will satisfy the intent
- Implicit Intent - do not name a specific component, but instead declare a general action to perform

# Building an intent

- Building the Intent object we specify
- **Component name** - only Explicit Intent
- **Action** - A string that specifies the generic action to perform, system Action or own Action
- **Data** - URI object
- **Category** - additional information about the kind of component that should handle the intent
- **Extras** - Key-value pairs that carry additional information required to accomplish the requested action

# Standard System Action

- ACTION_MAIN
- ACTION_VIEW
- ACTION_ATTACH_DATA
- ACTION_EDIT
- ACTION_PICK
- ACTION_CHOOSER
- ACTION_GET_CONTENT
- ACTION_DIAL
- ACTION_CALL
- ACTION_SEND
- ACTION_SENDTO
- ACTION_ANSWER
- ACTION_INSERT
- ACTION_DELETE
- ACTION_RUN
- ACTION_SYNC
- ACTION_PICK_ACTIVITY
- ACTION_SEARCH
- ACTION_WEB_SEARCH
- ACTION_FACTORY_TEST

# Examples with data

- ACTION_VIEW `content://contacts/people/1` – Display information about the person whose identifier is “1”.
- ACTION_DIAL `content://contacts/people/1` – Display the phone dialer with the person filled in.
- ACTION_EDIT `content://contacts/people/1` – Edit information about the person whose identifier is “1”.
- ACTION_VIEW `tel:123` – Display the phone dialer with the given number filled in. Note how the VIEW action does what what is considered the most reasonable thing for a particular URI.
- ACTION_DIAL `tel:123` – Display the phone dialer with the given number filled in.

# Category

- Gives additional information about the action to execute.
- For example, CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application.
- CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.
- That is, if you include the categories CATEGORY_LAUNCHER and CATEGORY_ALTERNATIVE, then you will only resolve to components with an intent that lists both of those categories.
- Activities will very often need to support the CATEGORY_DEFAULT so that they can be found by Context.startActivity().
- DEFAULT category is required for all filters - except for those with the MAIN action and LAUNCHER category.

# Standard Categories

- CATEGORY_DEFAULT
- CATEGORY_BROWSABLE
- CATEGORY_TAB
- CATEGORY_ALTERNATIVE
- CATEGORY_SELECTED_ALTERNATIVE
- CATEGORY_LAUNCHER
- CATEGORY_INFO
- CATEGORY_APP_MARKET

- CATEGORY_HOME
- CATEGORY_PREFERENCE
- CATEGORY_TEST
- CATEGORY_CAR_DOCK
- CATEGORY_DESK_DOCK
- CATEGORY_LE_DESK_DOCK
- CATEGORY_HE_DESK_DOCK
- CATEGORY_CAR_MODE

# Example code

## Explicit Intent

```kotlin
val fileDownloadIntent = Intent(this, FileDownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(fileDownloadIntent)
```

```kotlin
val intent = Intent(this, OtherActivity::class.java)
startActivity(intent)
```

## Implicit Intent

```kotlin
val fileDownloadIntent = Intent(this, FileDownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(fileDownloadIntent)
```

```kotlin

val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}
// Try to invoke the intent.
try {
    startActivity(sendIntent)
} catch (e: ActivityNotFoundException) {
    // Define what your app should do if no activity can handle the intent.
}
```

# Forcing an app chooser

- Using Implicit Intents user can select which app use (if more than one)
- User can seting default app for certain action
- Using *createChooser()* we show the chooser, and e.g. send data to other apps

```
//1. Define Intent
val sendIntent = Intent(Intent.ACTION_SEND)
// Always use string resources for UI text.
// This says something like "Share this photo with"
//2.Create title
val title: String = resources.getString(R.string.chooser_title)
//3. Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)

//4. Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(chooser)
}
```

# Receive Implicit Intnet

- To receive implicit intent we must declare one or more intent filters for each of your app components
- The system delivers an implicit intent to your app component only if the intent can pass through one of your intent filters.

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

# More information

- https://developer.android.com/guide/components/activities/activity-lifecycle
- https://developer.android.com/guide/components/intents-filters

# MOBILE APPLICATION DEVELOPMENT

## Fragment - Lifecycle

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

# Fragment

- Represents a reusable portion of your app's UI
- Fragment defines and manages its own layout
- Has its own lifecycle
- Can handle its own input events
- Fragment must be hosted by an activity or another fragment

# Create a fragment

- Setup environment
- Create a fragment class
- Add a fragment to an activity

Add to project **build.gradle** information about Google Maven repository

```
buildscript {
    ...

    repositories {
        google()
        ...
    }
}

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

# Create a fragment

- Setup environment
- Create a fragment class
- Add a fragment to an activity

Add to app's **build.gradle** information about AndroidX Fragment library

```
dependencies {
    val fragment_version = "1.3.4"

    // Java language implementation
    implementation("androidx.fragment:fragment:$fragment_version")
    // Kotlin
    implementation("androidx.fragment:fragment-ktx:$fragment_version")
}
```

# Create a fragment

- Setup environment
- Create a fragment class
- Add a fragment to an activity

We can use **Fragemnt, DialogFragment, PreferenceFragmentCompat**

```
class ExampleFragment : Fragment(R.layout.example_fragment)
```

# Create a fragment

- Setup environment
- Create a fragment class
- Add a fragment to an activity

Define in XML, *android:name* containing a single class

```
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.example.ExampleFragment" />
```

# Create a fragment

- Setup environment
- Create a fragment class
- Add a fragment to an activity

or (more offten) Define in XML container for fragment

```
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

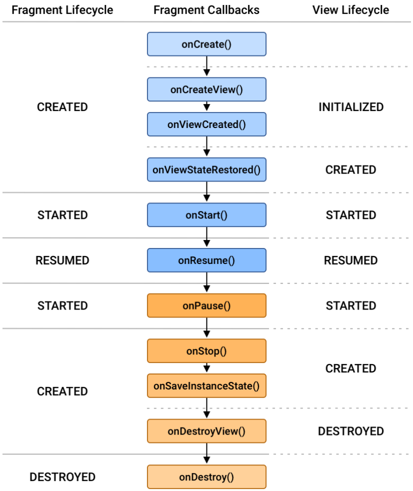
Add code to activity (onCreate())

```
        supportFragmentManager.commit {
                setReorderingAllowed(true)
                add<ExampleFragment>(R.id.fragment_container_view)
```

# Fragment lifecycle

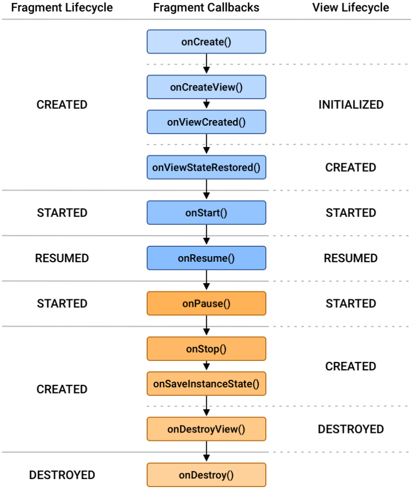
- Each **Fragment** instance has its own lifecycle
- View lifecycle is difrent than Fragment lifecycle
- Fragment state:
  - INITIALIZED
  - CREATED
  - STARTED
  - RESUMED
  - DESTROYED

# Fragment Lifecycle



| Fragment Lifecycle | Fragment Callbacks | View Lifecycle |
|---|---|---|
| CREATED | onCreate() | |
| | onCreateView() | INITIALIZED |
| | onViewCreated() | |
| | onViewStateRestored() | CREATED |
| STARTED | onStart() | STARTED |
| RESUMED | onResume() | RESUMED |
| STARTED | onPause() | STARTED |
| CREATED | onStop() | CREATED |
| | onSaveInstanceState() | |
| | onDestroyView() | DESTROYED |
| DESTROYED | onDestroy() | |

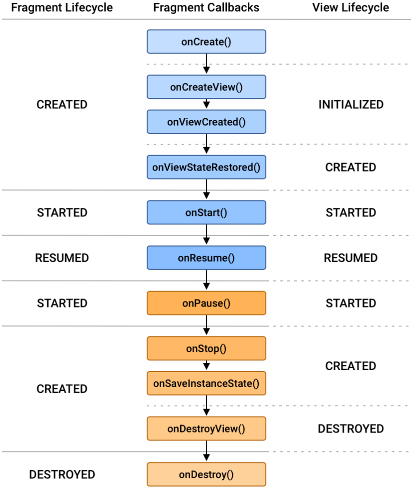
# Fragment Lifecycle



## CREATED

It has been added to a FragmentManager and the onAttach() method has already been called.
The fragment's view Lifecycle is created only when your Fragment provides a valid View instance.
You can also override onCreateView() to programmatically inflate or create your fragment's view.

# Fragment Lifecycle

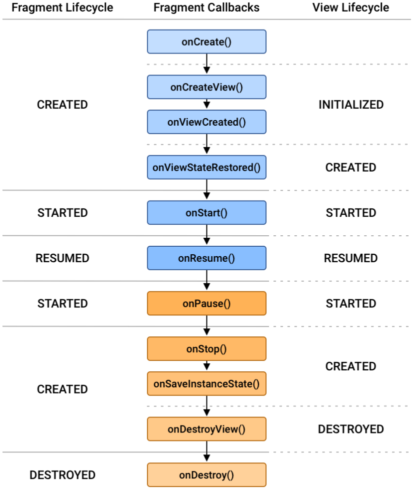


| Fragment Lifecycle | Fragment Callbacks | View Lifecycle |
|---|---|---|
| CREATED | onCreate() | |
| | onCreateView() | INITIALIZED |
| | onViewCreated() | |
| | onViewStateRestored() | CREATED |
| STARTED | onStart() | STARTED |
| RESUMED | onResume() | RESUMED |
| STARTED | onPause() | STARTED |
| CREATED | onStop() | CREATED |
| | onSaveInstanceState() | |
| | onDestroyView() | DESTROYED |
| DESTROYED | onDestroy() | |

## STARTED

This state guarantees that the fragment's view is available, if one was created, and that it is safe to perform a FragmentTransaction on the child FragmentManager of the fragment.

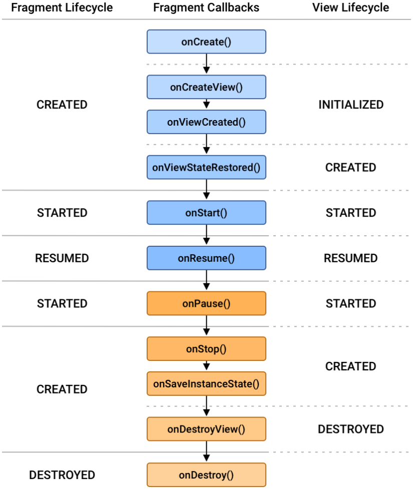
# Fragment Lifecycle



## RESUMED

When the fragment is visible, all Animator and Transition effects have finished, and the fragment is ready for user interaction

# Fragment Lifecycle

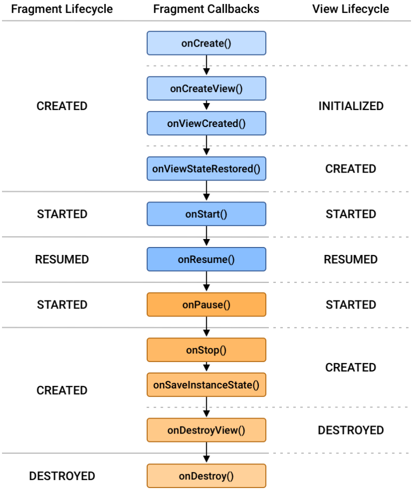


| Fragment Lifecycle | Fragment Callbacks | View Lifecycle |
|---|---|---|
| CREATED | onCreate() | |
| | onCreateView() | INITIALIZED |
| | onViewCreated() | |
| | onViewStateRestored() | CREATED |
| STARTED | onStart() | STARTED |
| RESUMED | onResume() | RESUMED |
| STARTED | onPause() | STARTED |
| CREATED | onStop() | CREATED |
| | onSaveInstanceState() | |
| | onDestroyView() | DESTROYED |
| DESTROYED | onDestroy() | |

## STARTED

As the user begins to leave the fragment, and while the fragment is still visible, the Lifecycles for the fragment and for its view are moved back to the STARTED state

# Fragment Lifecycle

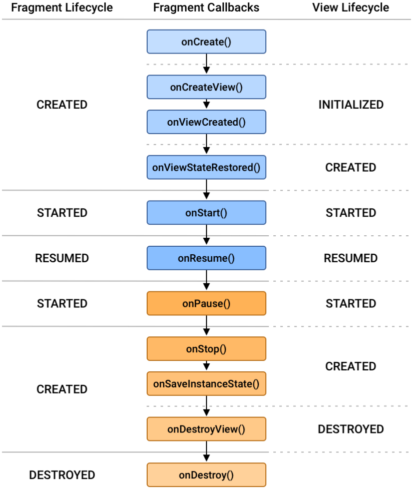


| Fragment Lifecycle | Fragment Callbacks | View Lifecycle |
|---|---|---|
| CREATED | onCreate() | |
| | onCreateView() | INITIALIZED |
| | onViewCreated() | |
| | onViewStateRestored() | CREATED |
| STARTED | onStart() | STARTED |
| RESUMED | onResume() | RESUMED |
| STARTED | onPause() | STARTED |
| CREATED | onStop() | CREATED |
| | onSaveInstanceState() | |
| | onDestroyView() | DESTROYED |
| DESTROYED | onDestroy() | |

## CREATED

Once the fragment is no longer visible, the Lifecycles for the fragment and for its view are moved into the CREATED state Followed either by *onResume()* if the activity returns back to the front, or by *onStop()* if it becomes invisible to the user.

# Fragment Lifecycle



| Fragment Lifecycle | Fragment Callbacks | View Lifecycle |
|---|---|---|
| CREATED | onCreate() | |
| | onCreateView() | INITIALIZED |
| | onViewCreated() | |
| | onViewStateRestored() | CREATED |
| STARTED | onStart() | STARTED |
| RESUMED | onResume() | RESUMED |
| STARTED | onPause() | STARTED |
| CREATED | onStop() | CREATED |
| | onSaveInstanceState() | |
| | onDestroyView() | DESTROYED |
| DESTROYED | onDestroy() | |

## DESTROYED

The fragment is removed, or if the FragmentManager is destroyed

# Lifecycle Methods - to resumed state (interacting with the user)

- *onAttach* - called once the fragment is associated with its activity.
- *onCreate* - called to do initial creation of the fragment.
- *onCreateView* - creates and returns the view hierarchy associated with the fragment.
- *onActivityCreated* - tells the fragment that its activity has completed its own *android.app.Activity-onCreate*.
- *onViewStateRestored* - tells the fragment that all of the saved state of its view hierarchy has been restored.
- *onStart* - makes the fragment visible to the user (based on its containing activity being started).
- *onResume* - makes the fragment begin interacting with the user (based on its containing activity being resumed).

- *onPause* - fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.
- *onStop* - fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.
- *onDestroyView* - allows the fragment to clean up resources associated with its View.
- *onDestroy* - called to do final cleanup of the fragment's state.
- *onDetach* - called immediately prior to the fragment no longer being associated with its activity.

# Service

- An application component that can perform long-running operations in the background
- Not provide a user interface
- Extending the Service class
- You must declare all services in your application's manifest file

# Types of Services

- Foreground
- Background
- Bound

- A foreground service performs some operation that is noticeable to the user
- Foreground services must display a Notification
- This notification cannot be dismissed unless the service is either stopped or removed
- Continue running even when the user isn't interacting with the app

# Types of Services

- Foreground
- Background
- Bound

- A background service performs an operation that isn't directly noticed by the user
- e.g. compact storage
- API 26 or higher - restrictions on running background services when the app itself isn't in the foreground, you shouldn't access location information from the background

# Types of Services

- Foreground
- Background
- Bound

- A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC
- Runs only as long as another application component is bound to it.
- Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

# More information

- https://developer.android.com/guide/fragments
- https://developer.android.com/guide/fragments/lifecycle

# MOBILE APPLICATION DEVELOPMENT

## User Interface

Innovative Open Source courses for Computer Science

30.05.2021

Funded by the European Union

# User Interface

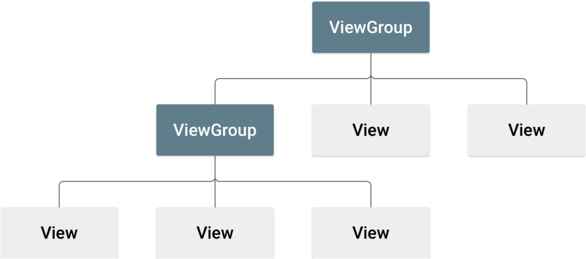
## UI

Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as **structured layout objects** and **UI controls** that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as **dialogs**, **notifications**, and **menus**.

# Layouts

## Layouts

Defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects.



# Declare Layouts

You can declare a layout in two ways:

- **Declare UI elements in XML.** - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.
- **Instantiate layout elements at runtime.** Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

# Define layouts in ***XML*** files

- Declaring your UI in ***XML*** allows you to separate the presentation of your app from the code that controls its behavior.
- ***View*** - usually called ”widgets” and can be one of many subclasses, such as ***Button or TextView***
- ***ViewGroup*** - usually called ”layouts” can be one of many types that provide a different layout structure, such as ***LinearLayout or ConstraintLayout***
- To debug your layout at runtime, use the Layout Inspector tool. `https://developer.android.com/studio/debug/layout-inspector`

# Example XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

- Each layout file must contain exactly one root element, which must be a View or ViewGroup object (ex LinearLayout)
- All layout we store in ***res/layout/***
- Android support different screen sizes

# Diffrent screen sizes

```
res/layout/main_activity.xml                # For handsets
res/layout-land/main_activity.xml           # For handsets in landscape
res/layout-sw600dp/main_activity.xml        # For 7 inch tablets
res/layout-sw600dp-land/main_activity.xml   # For 7 inch tablets in landscape
```

- You can provide screen-specific layouts by creating additional ***res/layout/*** directories—one for each screen configuration that requires a different layout
- Use the available width qualifier (e.g. ***sw600dp*** - screen with 600dp)
- Use orientation qualifiers (e.g. ***land or port*** - layouts for portrait or landscape respectively)

# Type of Layouts I

- Linear Layout - is a view group that aligns all children in a single direction, vertically or horizontally.
- ~~Relative Layout~~ - is a view group that displays child views in relative positions.
- Constraint Layout -is a view to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout
- Table Layout - is a view that groups views into rows and columns.
- Absolute Layout - enables you to specify the exact location of its children.
- Frame Layout - is a placeholder on screen that you can use to display a single view.

# Type of Layouts II

- List View - ListView is a view group that displays a list of scrollable items. (Layouts with an Adapter)
- Grid View - GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. (Layouts with an Adapter)
- `https://www.tutorialspoint.com/android/android_user_interface_layouts.htm`

# Layout Attributes - Common I

- *android* : *id* - This is the ID which uniquely identifies the view.
- *android* : *layout_width* - This is the width of the layout.
- *android* : *layout_height* - This is the height of the layout
- *android* : *layout_marginTop* - This is the extra space on the top side of the layout.
- *android* : *layout_marginBottom* - This is the extra space on the bottom side of the layout.
- *android* : *layout_marginLeft* - This is the extra space on the left side of the layout.
- *android* : *layout_marginRight* - This is the extra space on the right side of the layout.
- *android* : *layout_gravity* - This specifies how child Views are positioned.

# Layout Attributes - Common II

- *android* : *layout_weight* - This specifies how much of the extra space in the layout should be allocated to the View.
- *android* : *layout_x* - This specifies the x-coordinate of the layout.
- *android* : *layout_y* - This specifies the y-coordinate of the layout.
- *android* : *layout_width* - This is the width of the layout.
- *android* : *paddingLeft* - This is the left padding filled for the layout.
- *android* : *paddingRight* - This is the right padding filled for the layout.
- *android* : *paddingTop* - This is the top padding filled for the layout.
- *android* : *paddingBottom* - This is the bottom padding filled for the layout.

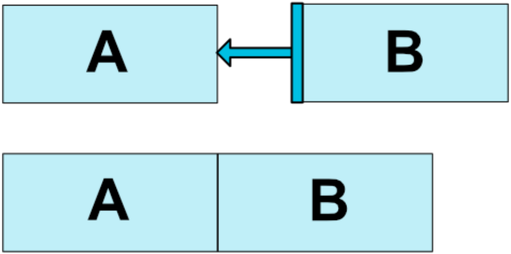
# ConstraintLayout

- A ConstraintLayout is a ***android.view.ViewGroup*** which allows you to position and size widgets in a flexible way.
- There are currently various types of constraints that you can use:
  - Relative positioning
  - Margins
  - Centering positioning
  - Circular positioning
  - Visibility behavior
  - Dimension constraints
  - Chains
  - Virtual Helpers objects
  - Optimizer
- `https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout`

# Relative positioning

- Those constraints allow you to position a given widget relative to another one.
- You can constrain a widget on the horizontal and vertical axis:
  - Horizontal Axis: left, right, start and end sides
  - Vertical Axis: top, bottom sides and text baseline
- The general concept is to constrain a given side of a widget to another side of any other widget.
- They all take a reference id to another widget, or the parent (which will reference the parent container, i.e. the ConstraintLayout)

# Relative positioning - example



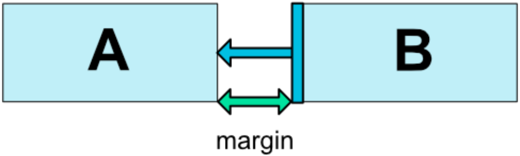
```
<Button android:id="@+id/buttonA" ... />
        <Button android:id="@+id/buttonB" ...
                app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

```
<Button android:id="@+id/buttonB" ...
                app:layout_constraintLeft_toLeftOf="parent" />
```

# Available constraints

- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

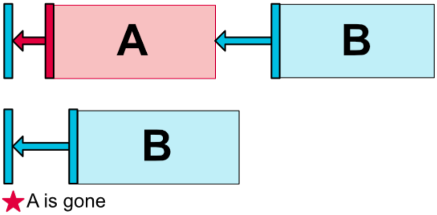
# Margins



If side margins are set, they will be applied to the corresponding constraints, enforcing the margin as a space between the target and the source side.

- android:layout_marginStart
- android:layout_marginEnd
- android:layout_marginLeft
- android:layout_marginTop
- android:layout_marginRight
- android:layout_marginBottom

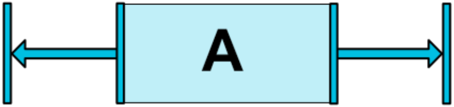
# Visibility behavior



**ConstraintLayout** has a specific handling of widgets being marked as ***View.GONE***. **GONE** widgets, as usual, are not going to be displayed and are not part of the layout itself (i.e. their actual dimensions will not be changed if marked as **GONE**).
But in terms of the layout computations, GONE widgets are still part of it, with an important distinction:

- For the layout pass, their dimension will be considered as zero (basically, they will be resolved to a point)
- If they have constraints to other widgets they will still be respected, but any margins will be as if equals to zero

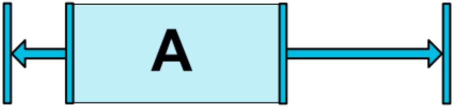
# Centering positioning



```
<androidx.constraintlayout.widget.ConstraintLayout ...>
             <Button android:id="@+id/button" ...
                 app:layout_constraintHorizontal_bias="0.3"
                 app:layout_constraintLeft_toLeftOf="parent"
                 app:layout_constraintRight_toRightOf="parent/>
         </>

```

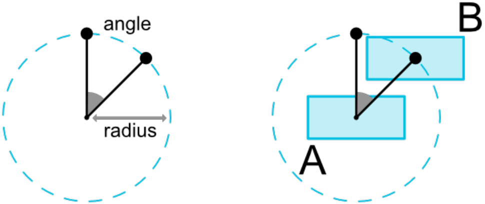
# Centering positioning with bias



```
<androidx.constraintlayout.widget.ConstraintLayout ...>
             <Button android:id="@+id/button" ...
                 app:layout_constraintLeft_toLeftOf="parent"
                 app:layout_constraintRight_toRightOf="parent/>
         </>

```

# Circular positioning



```
<Button android:id="@+id/buttonA" ... />
 <Button android:id="@+id/buttonB" ...
   app:layout_constraintCircle="@+id/buttonA"
   app:layout_constraintCircleRadius="100dp"
   app:layout_constraintCircleAngle="45" />
```

The following attributes can be used:

- layout_constraintCircle : references another widget id
- layout_constraintCircleRadius : the distance to the other widget center
- layout_constraintCircleAngle : which angle the widget should be at (in degrees, from 0 to 360)

# Dimensions constraints

You can define minimum and maximum sizes for the ConstraintLayout itself:

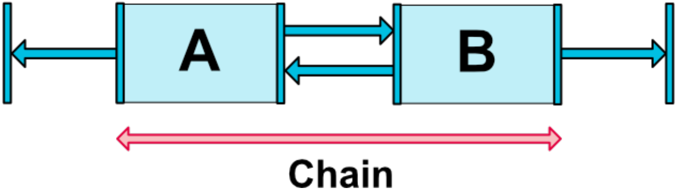
- android:minWidth set the minimum width for the layout
- android:minHeight set the minimum height for the layout
- android:maxWidth set the maximum width for the layout
- android:maxHeight set the maximum height for the layout

The dimension of the widgets can be specified by setting the android:layout_width and android:layout_height attributes in 3 different ways:

- Using a specific dimension (either a literal value such as 123dp or a Dimension reference)
- Using WRAP_CONTENT, which will ask the widget to compute its own size
- Using 0dp, which is the equivalent of "MATCH_CONSTRAINT"

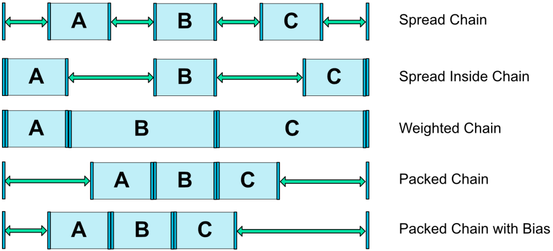
# Chains

Chains provide group-like behavior in a single axis (horizontally or vertically). The other axis can be constrained independently.



A set of widgets are considered a chain if they are linked together via a bi-directional connection (on Figure, showing a minimal chain, with two widgets).

# Chain Style I



- CHAIN_SPREAD – the elements will be spread out (default style)
- Weighted chain – in CHAIN_SPREAD mode, if some widgets are set to MATCH_CONSTRAINT, they will split the available space
- CHAIN_SPREAD_INSIDE – similar, but the endpoints of the chain will not be spread out

# Chain Style II

- CHAIN_PACKED – the elements of the chain will be packed together. The horizontal or vertical bias attribute of the child will then affect the positioning of the packed elements

# More information

https://developer.android.com/training/constraint-layout

https://constraintlayout.com/basics/setup.html

https://www.raywenderlich.com/155-android-listview-tutorial-with-kotlin

# Tools

- https://material.io/resources
- https://romannurik.github.io/AndroidAssetStudio/
- https://material.io/color/
- https://www.img-bak.in/
- https://material.io/resizer/
- https://material.io/devices/

# Material Design - Goal

- Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.
- Develop a single underlying system that allows for a unified experience across platforms and device sizes. Mobile precepts are fundamental, but touch, voice, mouse, and keyboard are all first-class input methods.
- Material is a design system created by Google to help teams build high-quality digital experiences for Android, iOS, Flutter, and the web.

# Material Design - Components

- App bars
- Bunner
- Card
- Floataing Button
- Data Tables
- Dialogs
- List, Image List
- Snackbars
- ToolTip
- ...

# Material Theming

- Material Theming refers to the customization of your Material Design app to better reflect your product's brand.
- You can redefine:
  - Color
  - Typography
  - Shape e.g. change size or button corners.

# MOBILE APPLICATION DEVELOPMENT

## Sensors

Innovative Open Source courses for Computer Science

30.05.2021

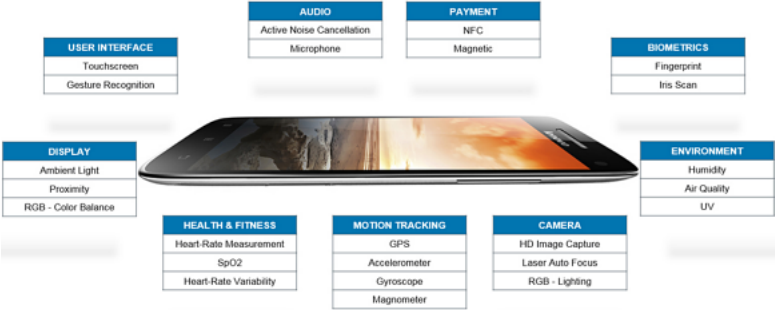
Funded by the European Union

# Sensors in mobile device

- Inertial Sensors: Gyroscope, Accelerometer, Magnetometer (e-Compass)
- Optical Sensors: Proximity, Ambient Light, RGB Color, Image Sensors (Front/Rear)
- Touch Sensors: Multi-Touch, Touchless Hover, Pressure Touch
- Environmental Sensors: Temperature, Humidity, Barometric Pressure, Gas (CO...)*
- Wireless/RF Sensors:GPS, WiFi, Cellular A-GPS, Bluetooth Low Energy, NFC
- Other Sensors:MEMS Microphones, Biometric/Fingerprint*, BioSensors*

MEMS Sensor
* - Future sensors

# Sensors



**USER INTERFACE**
Touchscreen
Gesture Recognition

**AUDIO**
Active Noise Cancellation
Microphone

**PAYMENT**
NFC
Magnetic

**BIOMETRICS**
Fingerprint
Iris Scan

**DISPLAY**
Ambient Light
Proximity
RGB - Color Balance

**ENVIRONMENT**
Humidity
Air Quality
UV

**HEALTH & FITNESS**
Heart-Rate Measurement
SpO2
Heart-Rate Variability

**MOTION TRACKING**
GPS
Accelerometer
Gyroscope
Magnometer

**CAMERA**
HD Image Capture
Laser Auto Focus
RGB - Lighting

`https://www.fierceelectronics.com/components/smartphone-sensor-evolution-rolls-rapidly-forward`

# Single Sensor Use Cases

- Compass Apps
- Tilt Sensing
- Multi-Touch, Touchless Hover
- Ambient Light/Color or Proximity Sensing
- Ambient Temperature and Humidity Sensing
- ...

# Mobile Sensor Fusion Use Cases

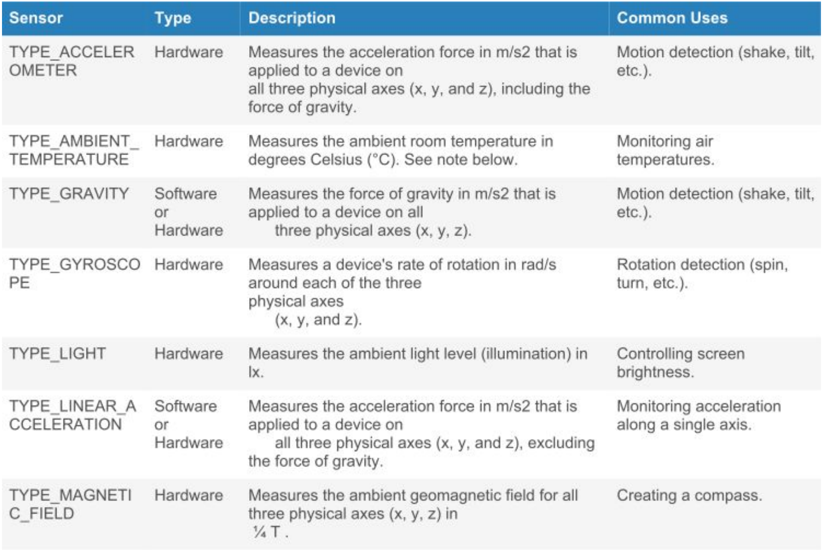
- Gesture UI Control (Motion, Proximity)
- Remote Control App (Motion, Multi-Touch, RF)
- Augmented Reality (Inertial, GPS, Image)
- Indoor Navigation and Positioning (Inertial, Pressure, WiFi)
- Context-Aware Mobile Services(ALL SENSORS !!)
- ...

# Android - sensor

The Android platform supports three broad categories of sensors:

- Motion sensors - These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors - These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers
- Position sensors - These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

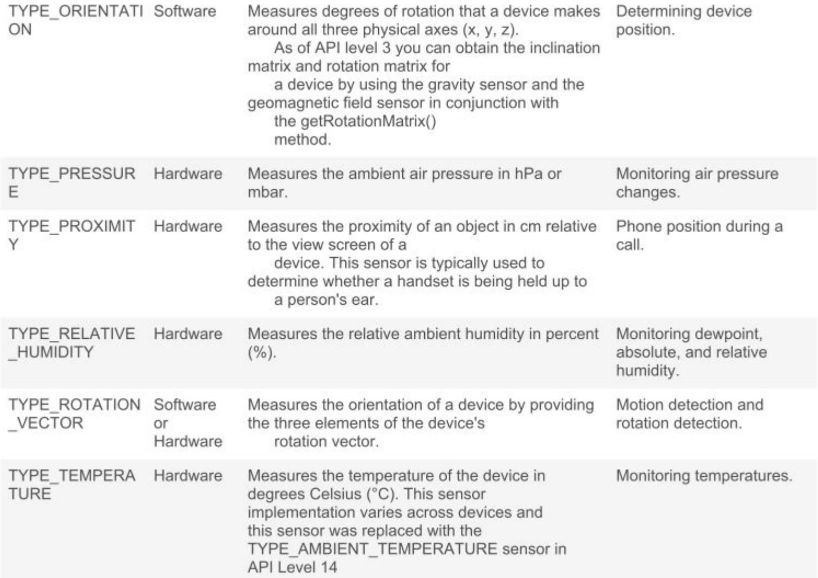
# Android - sensor



| Sensor | Type | Description | Common Uses |
|---|---|---|---|
| TYPE_ACCELEROMETER | Hardware | Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity. | Motion detection (shake, tilt, etc.). |
| TYPE_AMBIENT_TEMPERATURE | Hardware | Measures the ambient room temperature in degrees Celsius (°C). See note below. | Monitoring air temperatures. |
| TYPE_GRAVITY | Software or Hardware | Measures the force of gravity in m/s2 that is applied to a device on all three physical axes (x, y, z). | Motion detection (shake, tilt, etc.). |
| TYPE_GYROSCOPE | Hardware | Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z). | Rotation detection (spin, turn, etc.). |
| TYPE_LIGHT | Hardware | Measures the ambient light level (illumination) in lx. | Controlling screen brightness. |
| TYPE_LINEAR_ACCELERATION | Software or Hardware | Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity. | Monitoring acceleration along a single axis. |
| TYPE_MAGNETIC_FIELD | Hardware | Measures the ambient geomagnetic field for all three physical axes (x, y, z) in ¼ T . | Creating a compass. |

http://developer.android.com/guide/topics/sensors/sensors_overview.html

# Android - sensor



| | | | |
|---|---|---|---|
| TYPE_ORIENTATION | Software | Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method. | Determining device position. |
| TYPE_PRESSURE | Hardware | Measures the ambient air pressure in hPa or mbar. | Monitoring air pressure changes. |
| TYPE_PROXIMITY | Hardware | Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. | Phone position during a call. |
| TYPE_RELATIVE_HUMIDITY | Hardware | Measures the relative ambient humidity in percent (%). | Monitoring dewpoint, absolute, and relative humidity. |
| TYPE_ROTATION_VECTOR | Software or Hardware | Measures the orientation of a device by providing the three elements of the device's rotation vector. | Motion detection and rotation detection. |
| TYPE_TEMPERATURE | Hardware | Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14 | Monitoring temperatures. |

http://developer.android.com/guide/topics/sensors/sensors_overview.html

# Sensor Framework

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

# Sensor Framework

- Android sensor framework - access and acquire raw sensor data by using the Android sensor framework.
- The sensor framework is part of the android.hardware package and includes the following classes and interfaces:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

## SensorManager

We can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

# Sensor Framework

- Android sensor framework - access and acquire raw sensor data by using the Android sensor framework.
- The sensor framework is part of the android.hardware package and includes the following classes and interfaces:
    - SensorManager
    - Sensor
    - SensorEvent
    - SensorEventListener

## Sensor

We can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

# Sensor Framework

- Android sensor framework - access and acquire raw sensor data by using the Android sensor framework.
- The sensor framework is part of the android.hardware package and includes the following classes and interfaces:
    - SensorManager
    - Sensor
    - SensorEvent
    - SensorEventListener

## SensorEvent

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

# Sensor Framework

- Android sensor framework - access and acquire raw sensor data by using the Android sensor framework.
- The sensor framework is part of the android.hardware package and includes the following classes and interfaces:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

### SensorEventListener

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

# Typical steps

- Identifying sensors and sensor capabilities
- Monitor sensor events

## Identifying sensors

Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.

# Typical steps

- Identifying sensors and sensor capabilities
- Monitor sensor events

### Monitoring sensor

Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

# Identifying Sensors and Sensor Capabilities

1. Get a reference to the sensor service

```
^^Iprivate lateinit var sensorManager: SensorManager
...
^^IsensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
^^I
```

2. Get a listing of every sensor on a device

```
val deviceSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_ALL)
^^I
```

2. Or use another constant instead of TYPE_ALL such as TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, or TYPE_GRAVITY.

```
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
    // Success! There's a magnetometer.
} else {
    // Failure! No magnetometer.
}
^^I
```

# Capabilities

```kotlin
private lateinit var sensorManager: SensorManager
private var mSensor: Sensor? = null

...

sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null) {
    val gravSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_GRAVITY)
    // Use the version 3 gravity sensor.
    mSensor = gravSensors.firstOrNull { it.vendor.contains("Google LLC") && it.
            version == 3 }
}
if (mSensor == null) {
    // Use the accelerometer.
    mSensor = if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null)
            {
        sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    } else {
        // Sorry, there are no accelerometers on your device.
        // You can't play this game.
        null
    }
}
```

# Usefull method to get info about Sensor Capabilities

- *getResolution(), getMaximumRange()*
- *getPower()*
- *getMinDelay()*

# Monitoring Sensor Events

To monitor raw sensor data you need to implement two callback methods that are exposed through the *SensorEventListener* interface: onAccuracyChanged() and onSensorChanged()

## Sensor's accuracy changes

```
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
^^I
```

## Sensor reports a new value

```
    override fun onSensorChanged(event: SensorEvent) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        val lux = event.values[0]
        // Do something with this sensor value.
    }
^^I
```

# Monitoring Sensor Events

```
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager
    private var mLight: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }

    override fun onSensorChanged(event: SensorEvent) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        val lux = event.values[0]
        // Do something with this sensor value.
    }
^^I
```

# Monitoring Sensor Events - Register

```

    override fun onResume() {
        super.onResume()
        mLight?.also { light ->
            sensorManager.registerListener(this, light, SensorManager.
                    SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(this)
    }
}
```

# Google Play filters

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
              android:required="true" />
```

# Best Practices for Accessing and Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

## Gather sensor data in the foreground

On devices running Android 9 (API level 28) or higher:

- Sensors that use the continuous reporting mode, such as accelerometers and gyroscopes, don't receive events.
- Sensors that use the on-change or one-shot reporting modes don't receive events.

# Best Practices for Accessing and Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

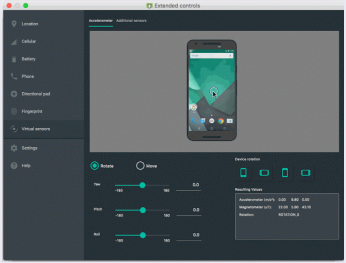
### Unregister sensor listeners

Be sure to unregister a sensor's listener when you are done using the sensor or when the sensor activity pauses. If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor.

# Best Practices for Accessing and Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

## Test with the Android Emulator



● Rotate ○ Move

Yaw -180 180 0.0

Pitch -180 180 0.0

Roll -180 180 0.0

```
Accelerometer (m/s²):  0.00   9.80   0.00
Magnetometer (uT):     22.00  5.90   43.10
Rotation:              ROTATION_0
```

# Best Practices for Accessing and Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

## Don't block the onSensorChanged() method

- Sensor data can change at a high rate - system may call the *onSensorChanged(SensorEvent)* method quite often
- Filtering or reduction of sensor data, you should perform that work outside of the onSensorChanged(SensorEvent) method

# Best Practices for Accessing and Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

## Choose sensor delays carefully

- When you register a sensor with the registerListener() method, be sure you choose a delivery rate that is suitable for your application or use-case.
- Allowing the system to send extra data that you don't need wastes system resources and uses battery power.

# Motion Sensor

Motion sensors are useful for monitoring **device movement, such as tilt, shake, rotation, or swing.**
Sensors' possible architectures vary by sensor type:

- The gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors are either hardware-based or software-based.
- The accelerometer and gyroscope sensors are always hardware-based.

`https://developer.android.com/guide/topics/sensors/sensors_motion`

# Example - Motion Sensor

- Source + Description https://www.raywenderlich.com/10838302-sensors-tutorial-for-android-getting-started

# Position sensors

Position sensors are useful for determining a device's physical position in the world's frame of reference. For example, you can use the geomagnetic field sensor in combination with the accelerometer to determine a device's position relative to the magnetic north pole.

# Compute the device's orientation

```kotlin
private lateinit var sensorManager: SensorManager
...
// Rotation matrix based on current readings from accelerometer and magnetometer.
val rotationMatrix = FloatArray(9)
SensorManager.getRotationMatrix(rotationMatrix, null, accelerometerReading, magnetometerReading)

// Express the updated rotation matrix as three orientation angles.
val orientationAngles = FloatArray(3)
SensorManager.getOrientation(rotationMatrix, orientationAngles)
```

# Compute the device's orientation

The system computes the orientation angles by using a device's geomagnetic field sensor in combination with the device's accelerometer. Using these two hardware sensors, the system provides data for the following three orientation angles:

- **Azimuth (degrees of rotation about the -z axis)** This is the angle between the device's current compass direction and magnetic north. If the top edge of the device faces magnetic north, the azimuth is 0 degrees; if the top edge faces south, the azimuth is 180 degrees. Similarly, if the top edge faces east, the azimuth is 90 degrees, and if the top edge faces west, the azimuth is 270 degrees.

# Compute the device’s orientation

The system computes the orientation angles by using a device’s geomagnetic field sensor in combination with the device’s accelerometer. Using these two hardware sensors, the system provides data for the following three orientation angles:

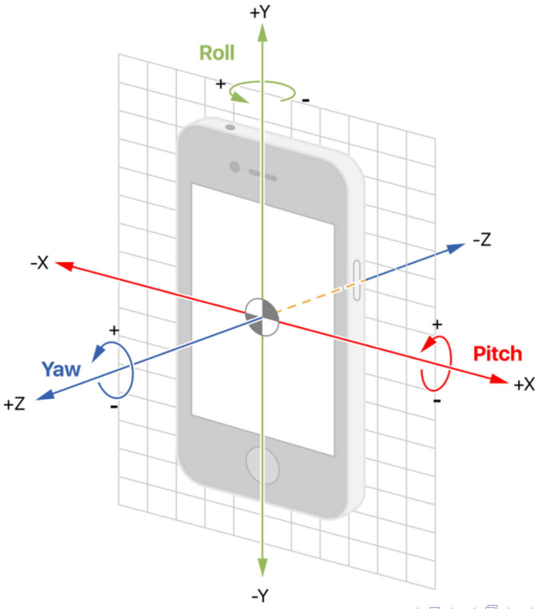
- **Pitch (degrees of rotation about the x axis)** This is the angle between a plane parallel to the device’s screen and a plane parallel to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the top edge of the device toward the ground, the pitch angle becomes positive. Tilting in the opposite direction— moving the top edge of the device away from the ground—causes the pitch angle to become negative. The range of values is -180 degrees to 180 degrees.

# Compute the device's orientation

The system computes the orientation angles by using a device's geomagnetic field sensor in combination with the device's accelerometer. Using these two hardware sensors, the system provides data for the following three orientation angles:

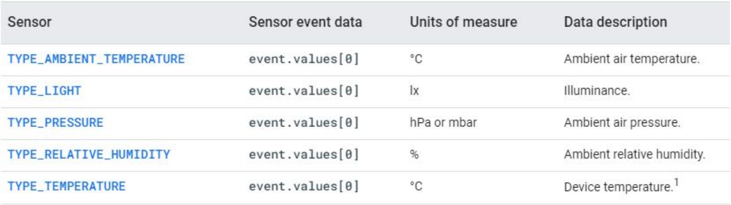
- **Roll (degrees of rotation about the y axis)** This is the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. If you hold the device parallel to the ground with the bottom edge closest to you and tilt the left edge of the device toward the ground, the roll angle becomes positive. Tilting in the opposite direction—moving the right edge of the device toward the ground— causes the roll angle to become negative. The range of values is -90 degrees to 90 degrees.

# Device Orientation



# Environment sensors

- You can use these sensors to monitor relative ambient humidity, illuminance, ambient pressure, and ambient temperature near an Android-powered device
- All four environment sensors are hardware-based and are available only if a device manufacturer has built them into a device.
- environment sensors return a single sensor value for each data event



| Sensor | Sensor event data | Units of measure | Data description |
|---|---|---|---|
| TYPE_AMBIENT_TEMPERATURE | event.values[0] | °C | Ambient air temperature. |
| TYPE_LIGHT | event.values[0] | lx | Illuminance. |
| TYPE_PRESSURE | event.values[0] | hPa or mbar | Ambient air pressure. |
| TYPE_RELATIVE_HUMIDITY | event.values[0] | % | Ambient relative humidity. |
| TYPE_TEMPERATURE | event.values[0] | °C | Device temperature.[1] |

# Example

- AndroidManifest
- Service
- Callback methods
- Emulator

https://www.raywenderlich.com/10838302-sensors-tutorial-for-android-getting-started

# MOBILE APPLICATION DEVELOPMENT

## Location

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

# Location

## location

A service to determine the position of the device and, indirectly, the user. In mobile systems, location is one of the unique features to create location-aware applications.

- Allows the location of the device to be determined
- General location
- Precise location
- Google Play services - recommended location method in Android

# Examples of applications

- Location-specific information (local weather forecast, local news/messages, allergen concentration)
- Information on nearby resources (bank, pharmacy, pub)
- Interactive maps and tourist information
- Location-dependent mobile advertising
- Management of mobile workers

# Google Play services

- Mechanism for determining position based on data from different providers, e.g. GNSS (GPS) module, WiFi module or Bluetooth.
- Fused Location Provider
- Faster position determination
- Reduced energy consumption
- Additional capabilities e.g. geofencing, activity detection

# Adding the Google Play services library

```
apply plugin: ’com.android.application’

...

dependencies{
    implementation ’com.google.android.gms:play-services-location:21.0.0’
}
```

# Steps required to define a location

- Location type definition
- Requesting permission for the device location
- Downloading location (last known, cyclic downloading of location)
- Using the location, e.g. to show a point on the map

# Definition permission

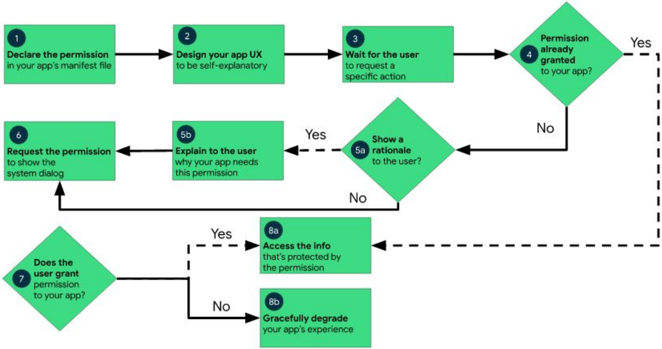
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
...
  <!-- Always include this permission -->
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

  <!-- Include only if your app benefits from precise location access. -->
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <!-- Recommended for Android 9 (API level 28) and lower. -->
  <!-- Required for Android 10 (API level 29) and higher. -->
...
  <!-- Required only when requesting background location access on
       Android 10 (API level 29) and higher. -->
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"
       />
...
  <application...>
    <service
      android:name="MyNavigationService"
      android:foregroundServiceType="location" ... >
    </service>
  </application>
...
</manifest>
```

# Types of locations

- General/Coarse location
- Precise location
- Foreground location
- Background location

# Request for permissions - workflow


1
Declare the permission in your app's manifest file
2
Design your app UX to be self-explanatory
3
Wait for the user to request a specific action
4
Permission already granted to your app?
Yes
No
5a
Show a rationale to the user?
Yes
No
5b
Explain to the user why your app needs this permission
6
Request the permission to show the system dialog
7
Does the user grant permission to your app?
Yes
No
8a
Access the info that's protected by the permission
8b
Gracefully degrade your app's experience

# Example request permission

```
 private fun checkPermission()
{
     if(ContextCompat.checkSelfPermission(this,
             Manifest.permission.ACCESS_FINE_LOCATION
         ) != PackageManager.PERMISSION_GRANTED)
         requestPermissionLauncher.launch(Manifest.permission.
              ACCESS_FINE_LOCATION)
 }
 private val requestPermissionLauncher =
     registerForActivityResult(
         ActivityResultContracts.RequestPermission()
     ){
             isGranted: Boolean ->
         if (isGranted){
             Log.i("Permission:␣", "Granted")
         } else{
             Log.i("Permission:␣", "Denied")
         }
     }
```

# Using Fused Location Provider - last known position

## Object definition

```
private lateinit var fusedLocationClient: FusedLocationProviderClient
```

## Initialization

```
override fun onCreate(savedInstanceState: Bundle?){
...
        fusedLocationClient = LocationServices.getFusedLocationProviderClient(
             this)

}
```

## Retrieval of last known location

```
        checkPermission()
        fusedLocationClient.lastLocation
            .addOnSuccessListener{ location : Location? ->
                val myPosition = location?.let{
                    LatLng(it.latitude,it.longitude)
                }
                myPosition?.let{
                    mMap.addMarker(MarkerOptions().position(myPosition).title("
                         My␣position"))
                    mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))
                }
            }
```

# Continuous location - I

- Object definition
- Initialisation
- Defining return methods
- Enabling and disabling location update information
- Selecting the refresh rate

# Continuous Location II

## Definition of objects

```
    private lateinit var locationRequest: LocationRequest
    private lateinit var locationCallback: LocationCallback
```

## Initialization

```
        fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
        locationRequest = LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY,
            500)
            .build()

        locationCallback = object : LocationCallback(){
            override fun onLocationResult(locationResult: LocationResult){
                if (locationResult != null){
                    super.onLocationResult(locationResult)
                    locationResult.lastLocation?.let{

                    //own code

                    }

                }
            }
        }
```

# Continuous location III

## Activation of location update information

```
            val addTask= fusedLocationClient.requestLocationUpdates(locationRequest, locationCallback, Looper.myLooper())
            addTask.addOnCompleteListener{task->
                if (task.isSuccessful){
                    Log.d("startStopRequestLocation", "Start␣loop␣Location␣Callback.")
                } else{
                    Log.d("startStopRequestLocation", "Failed␣start␣␣Location␣Callback.")
                }
            }
```

## Deactivation of location update information

```
            val removeTask = fusedLocationClient.removeLocationUpdates(locationCallback)
            removeTask.addOnCompleteListener{ task ->
                if (task.isSuccessful){
                    Log.d("startStopRequestLocation", "Location␣Callback␣removed.")
                } else{
                    Log.d("startStopRequestLocation", "Failed␣to␣remove␣Location␣Callback.")
                }
            }
```

# MOBILE APPLICATION DEVELOPMENT

## MVVM

Innovative Open Source courses for Computer Science

30.05.2021

Funded by the European Union

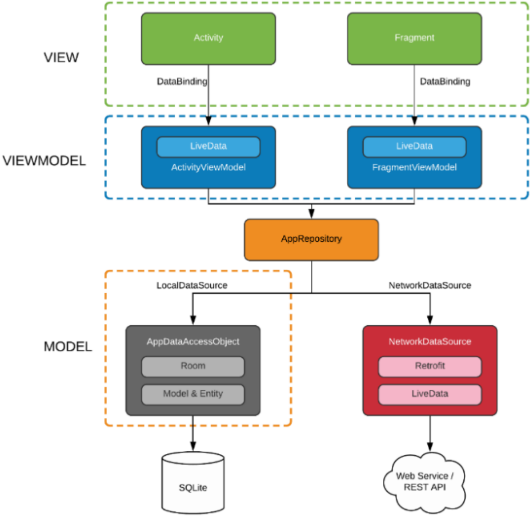
# MVVM

## MVVM

Model — View — ViewModel is the industry-recognized software architecture pattern that overcomes all drawbacks of MVP and MVC design patterns. MVVM suggests separating the data presentation logic(Views or UI) from the core business logic part of the application.

- **Model** - This holds the data of the application. It cannot directly talk to the View. Generally, it's recommended to expose the data to the ViewModel through Observables.
- **View** - It represents the UI of the application devoid of any Application Logic. It observes the ViewModel.
- **ViewModel** - It acts as a link between the Model and the View. It's responsible for transforming the data from the Model. It provides data streams to the View. It also uses hooks or callbacks to update the View. It'll ask for the data from the Model.

# MVVM -2


VIEW
Activity
Fragment
DataBinding
DataBinding
VIEWMODEL
LiveData
ActivityViewModel
LiveData
FragmentViewModel
AppRepository
LocalDataSource
NetworkDataSource
MODEL
AppDataAccessObject
Room
Model & Entity
NetworkDataSource
Retrofit
LiveData
SQLite
Web Service / REST API

# Architecture components

- Collection of libraries that help you design robust, testable, and maintainable apps
- Serve as the main guideline in building the backbone of our project architecture
- Using Android architecture components we don't have to worry much about managing the app's lifecycle or loading data into our UI

Components

- ViewModel
- LiveData
- Lifecycle
- Extend LiveData

# ViewModel

- The ViewModel class is designed to hold and manage UI-related data in a life-cycle conscious way.
- This allows data to survive configuration changes such as screen rotations.
- Architecture Components provides ViewModel helper class for the UI controller that is responsible for preparing data for the UI.
- ViewModel objects are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance.

# ViewModel benefits

- It allows you to persist UI state
- It provides access to business logic.

## Persistence

ViewModel allows the survival through both the state that a ViewModel holds, and operations that a ViewModel trigger. This caching means that you don't have to fetch data again through common configuration changes, such as a screen rotation.

## Access to business logic

ViewModel is the right place to handle business logic in the UI layer. The ViewModel is also in charge of handling events and delegating them to other layers of the hierarchy when business logic needs to be applied to modify application data.

# ViewModel - example

## Define VieModel

```
class MyViewModel : ViewModel() {
    private val users: MutableLiveData<List<User>> by lazy {
        MutableLiveData<List<User>>().also {
            loadUsers()
        }
    }

    fun getUsers(): LiveData<List<User>> {
        return users
    }

    private fun loadUsers() {
        // Do an asynchronous operation to fetch users.
    }
}
```

## Access the list from an activity as follows

```
override fun onCreate(savedInstanceState: Bundle?) {

        // Use the ’by viewModels()’ Kotlin property delegate
        // from the activity-ktx artifact
        val model: MyViewModel by viewModels()
        model.getUsers().observe(this, Observer<List<User>>{ users ->
            // update UI
        })
```

# LiveData

- LiveData is an observable data holder class.
- Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services.
- This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.
- LiveData follows the observer pattern. LiveData notifies Observer objects when the lifecycle state changes
- LiveData also automatically pushes an existing value, if there is one, to any new registered Observer objects.
- When coupled with the automatic-removal feature, this makes LiveData very convenient for dealing with configuration changes.

# Types in LiveData

- LiveData - is immutable by default. By using LiveData we can only observe the data and cannot set the data.
- MutableLiveData - is mutable and is a subclass of LiveData. In MutableLiveData we can observe and set the values using postValue() and setValue() methods
- MediatorLiveData - can observe other LiveData objects such as sources and react to their onChange() events.

# Extend LiveData

- LiveData is a lifecycle-aware component and thus it performs its functions according to the lifecycle state of other application components.
- If the observer's lifecycle state is active i.e., either STARTED or RESUMED, only then LiveData updates the app component.

```
class StockLiveData(symbol: String) : LiveData<BigDecimal>() {
    private val stockManager = StockManager(symbol)

    private val listener = { price: BigDecimal ->
        value = price
    }

    override fun onActive() {
        stockManager.requestPriceUpdates(listener)
    }

    override fun onInactive() {
        stockManager.removeUpdates(listener)
    }
}
```

# Extend LiveData - 2

- **onActive()** - method is called when the LiveData object has an active observer. This means you need to start observing the stock price updates from this method.
- **onInactive()** - method is called when the LiveData object doesn't have any active observers. Since no observers are listening, there is no reason to stay connected.

# Steps to build application with MVVM design patern

- Adding DataBinding and Implementations in your Gradle File
- Create a new class for the Model
- Create a new class for the ViewModel
- Improve View class
- Change layout

# Configure project

build.gradle(app)

```
android {
    compileSdk 31

    dataBinding {
        enabled true
    }


    ...

dependencies {
    //ViewModel
    implementation ’androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1’
    implementation ’androidx.activity:activity-ktx:1.4.0’
    //Lifecycle
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
...
}
}
```

# Model

```
data class SensorData(
    var accX: Float,
    var accY: Float,
    var accZ: Float,
    var gyroX: Float,
    var gyroY: Float,
    var gyroZ: Float,
    var light: Float
)
```

# ViewModel

```kotlin
class SensorViewModel(application: Application): AndroidViewModel(application) {
  private val _sensor = SensorDataLiveData(application)
  private var _pauseReading = MutableLiveData<Boolean>()

  val sensor: LiveData<SensorData>
    get() = _sensor

  fun getPauseReading(): MutableLiveData<Boolean> {
    return _pauseReading
  }

  fun changeButtonStatus()
  {
    if(_pauseReading.value==true) _sensor.registerListeners()
    else _sensor.unregisterListeners()
    _pauseReading.value?.let {
      _pauseReading.value = !it
    }
  }
  init {
      _pauseReading = MutableLiveData(false)
  }
}
```

# Change View code

```
    private lateinit var binding: ActivityMainBinding
    private val sensorViewModel: SensorViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.sensorViewModel = sensorViewModel
        binding.lifecycleOwner = this
    }
```

# Change layout

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable
            name="sensorViewModel"
            type="edu.zut.erasmus_plus.sensors.viewmodel.SensorViewModel" />
    </data>
```

# MOBILE APPLICATION DEVELOPMENT

## Storage

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

# Android storage options

- App-specific storage
- Shared storage
- Preferences
- Databases

### App-specific storage

Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.

# Android storage options

- App-specific storage
- Shared storage
- Preferences
- Databases

**Shared storage**

Store files that your app intends to share with other apps, including media, documents, and other files.

# Android storage options

- App-specific storage
- Shared storage
- Preferences
- Databases

**Preferences**

Store private, primitive data in key-value pairs.

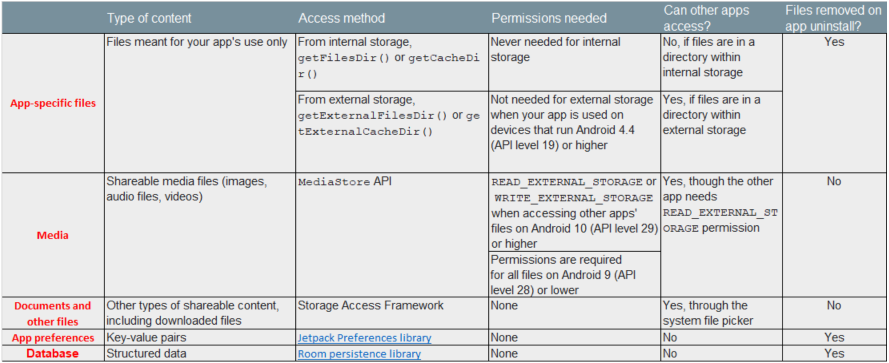
# Android storage options

- App-specific storage
- Shared storage
- Preferences
- Databases

**Databases**

Store structured data in a private database using the Room persistence library.

# Storage overview



| | Type of content | Access method | Permissions needed | Can other apps access? | Files removed on app uninstall? |
|---|---|---|---|---|---|
| App-specific files | Files meant for your app's use only | From internal storage, `getFilesDir()` or `getCacheDir()` | Never needed for internal storage | No, if files are in a directory within internal storage | Yes |
| | | From external storage, `getExternalFilesDir()` or `getExternalCacheDir()` | Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher | Yes, if files are in a directory within external storage | |
| Media | Shareable media files (images, audio files, videos) | `MediaStore` API | `READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_STORAGE` when accessing other apps' files on Android 10 (API level 29) or higher | Yes, though the other app needs `READ_EXTERNAL_STORAGE` permission | No |
| | | | Permissions are required for all files on Android 9 (API level 28) or lower | | |
| Documents and other files | Other types of shareable content, including downloaded files | Storage Access Framework | None | Yes, through the system file picker | No |
| App preferences | Key-value pairs | Jetpack Preferences library | None | No | Yes |
| Database | Structured data | Room persistence library | None | No | Yes |

# Which one to choose?

- How much space does your data require?
- How reliable does data access need to be?
- What kind of data do you need to store?
- Should the data be private to your app?

# Which one to choose?

- How much space does your data require?
- How reliable does data access need to be?
- What kind of data do you need to store?
- Should the data be private to your app?

### How much space does your data require?

Internal storage has limited space for app-specific data. Use other types of storage if you need to save a substantial amount of data.

# Which one to choose?

- How much space does your data require?
- How reliable does data access need to be?
- What kind of data do you need to store?
- Should the data be private to your app?

### How reliable does data access need to be?

If your app's basic functionality requires certain data, such as when your app is starting up, place the data within internal storage directory or a database. App-specific files that are stored in external storage aren't always accessible because some devices allow users to remove a physical device that corresponds to external storage.

# Which one to choose?

- How much space does your data require?
- How reliable does data access need to be?
- What kind of data do you need to store?
- Should the data be private to your app?

### What kind of data do you need to store?

If you have data that's only meaningful for your app, use app-specific storage. For shareable media content, use shared storage so that other apps can access the content. For structured data, use either preferences (for key-value data) or a database (for data that contains more than 2 columns).

# Which one to choose?

- How much space does your data require?
- How reliable does data access need to be?
- What kind of data do you need to store?
- Should the data be private to your app?

### Should the data be private to your app?

When storing sensitive data—data that shouldn't be accessible from any other app—use internal storage, preferences, or a database. Internal storage has the added benefit of the data being hidden from users.

# Access app-specific files

- Internal storage directories - These directories include both a dedicated location for storing persistent files, and another location for storing cache data. The system prevents other apps from accessing these locations, and on Android 10 (API level 29) and higher, these locations are encrypted. These characteristics make these locations a good place to store sensitive data that only your app itself can access.
- External storage directories - These directories include both a dedicated location for storing persistent files, and another location for storing cache data. Although it's possible for another app to access these directories if that app has the proper permissions, the files stored in these directories are meant for use only by your app. If you specifically intend to create files that other apps should be able to access, your app should store these files in the shared storage part of external storage instead.

## Example

- `https://www.journaldev.com/9383/android-internal-storage-example-tutorial`
- `https://developer.android.com/training/data-storage/app-specific`
- `https://github.com/android/storage-samples`
- To further protect app-specific files, use the Security library that's part of Android Jetpack to encrypt these files at rest. The encryption key is specific to your app.

# Access media files from shared storage

- To provide a more enriched user experience, many apps allow users to contribute and access media that's available on an external storage volume.
- The framework provides an optimized index into media collections, called the media store, that allows for retrieving and updating these media files more easily.
- Even after your app is uninstalled, these files remain on the user's device.

# ContentResolver

To interact with the media store abstraction, use a ContentResolver object that you retrieve from your app's context:

```
val projection = arrayOf(media-database-columns-to-retrieve)
val selection = sql-where-clause-with-placeholder-variables
val selectionArgs = values-of-placeholder-variables
val sortOrder = sql-order-by-clause

applicationContext.contentResolver.query(
    MediaStore.media-type.Media.EXTERNAL_CONTENT_URI,
    projection,
    selection,
    selectionArgs,
    sortOrder
)?.use { cursor ->
    while (cursor.moveToNext()) {
        // Use an ID column from the projection to get
        // a URI representing the media item itself.
    }
}
```

# Media files

The system automatically scans an external storage volume and adds media files to the following well-defined collections:

- **Images**, including photographs and screenshots, which are stored in the *DCIM/* and *Pictures/* directories. The system adds these files to the *MediaStore.Images* table.
- **Videos**, which are stored in the *DCIM/*, *Movies/*, and *Pictures/*directories. The system adds these files to the *MediaStore.Video* table.
- **Audio files**, which are stored in the *Alarms/*, *Audiobooks/*, *Music/*, *Notifications/*, *Podcasts/*, *and Ringtones/* directories, as well as audio playlists that are in the *Music/ or Movies/* directories. The system adds these files to the *MediaStore.Audio* table.
- **Downloaded files**, which are stored in the *Download/* directory. On devices that run Android 10 (API level 29) and higher, these files are stored in the *MediaStore.Downloads* table. This table isn't available on Android 9 (API level 28)

# App preferences

- If you have a relatively small collection of key-values that you'd like to save, you should use the SharedPreferences
- A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them
- Each SharedPreferences file is managed by the framework and can be private or shared.

# SharedPreferences

- This class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types.
- You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings.
- This data will persist across user sessions (even if your application is killed).
- *"SharedPreferences"* are saved as XML files in the *shared_prefs* folder

# How using SharePreferences ?

- 1. Get preferences from a specified file

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

- 2. Read

```
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)
```

- 3. Write and Apply (or Commit)changes

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

# Settings

- Applications often include settings that allow users to modify app features and behaviors.
- Settings is a place in your app where users indicate their preferences for how your app should behave.
- Settings is given low prominence in the UI because it's not frequently needed.

# How use settings

- 1. Create an Android XML resource called e.g. *preferences.xml* of the PreferenceScreen type.
- 2. Add to the file PreferencesCategory, and one of *CheckBoxPreference, ListPreference, EditTextPreference*
- 3. Create the class *MyPreferencesActivity* which extends *PreferenceActivity* This activity loads the *preference.xml* file and allows the user to change the values.
- 4. Add to the method *onOptionsItemSelected()* code for running PreferencesActivity

# RoomDatabase

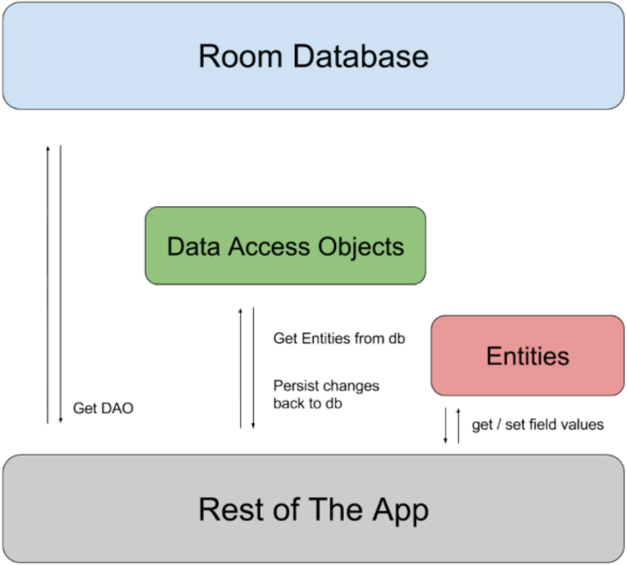
- Room provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.
- Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally.
- Room takes care of caching data when device is offile
- This future causes that room is recommend to use instead of SQLite

# RoomDatabase

Major components in Room

- Database - Contains the database holder and serves as the main access point for the underlying connection to your app's persisted, relational data
- Entity - Represents a table within the database
- DAO - Contains the methods used for accessing the database

# Room architecture diagram



# Example code Entity

```kotlin
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

# Example code DAO

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

# Example code instance Database

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

# Live Example

Room Database Example