Mobile Applications Security

Prof. Cristian CIUREA, PhD

Department of Economic Informatics and Cybernetics

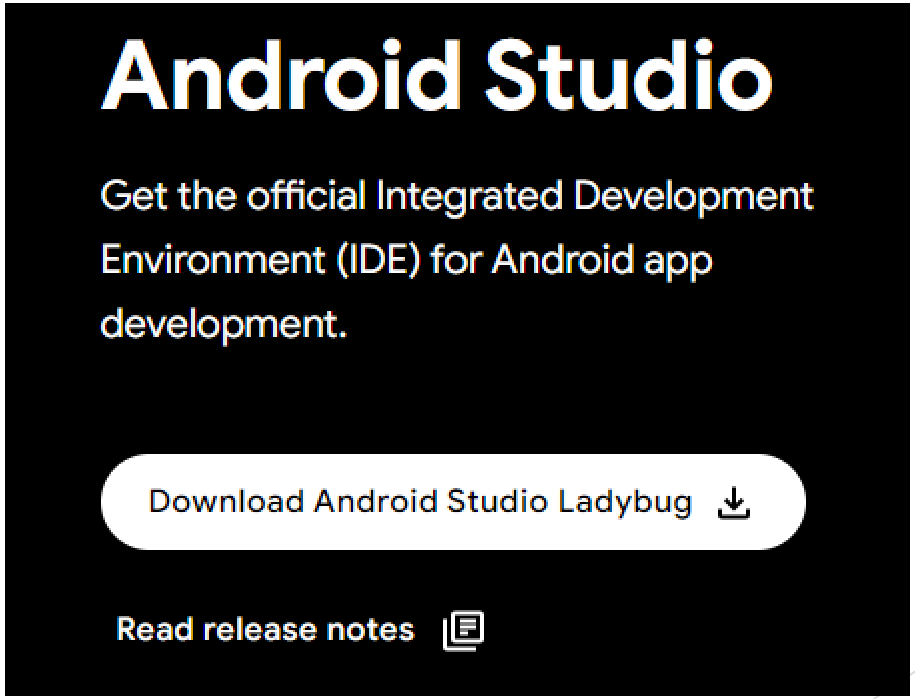
cristian.ciurea@ie.ase.ro

Agenda

- Android Basics
- Android User Interface
- Networking and Threads (*AsyncTask* + *JSON/XML* parsing)
- Persistent storage - Internal File System
- Persistent storage - databases (*SQLite/Room* + *Firebase*)
- Android Advanced Concepts
- Android Security Concepts

Skills requirement

- Have experience in Java (or Kotlin)
- Have experience in using Android Studio (or IntelliJ IDEA, or Eclipse with ADT)



Android Studio

Get the official Integrated Development Environment (IDE) for Android app development.

Android Basics

- Mobile operating systems
- Development of native mobile applications in Android
- Development tools for Android apps

Android Basics

Mobile operating systems:

- Management of hardware and software resources
- Memory management
- Management of processes
- Management of input/output devices
- Functions of mobile application
- Files management
- User interface

Android Basics

Mobile operating systems:

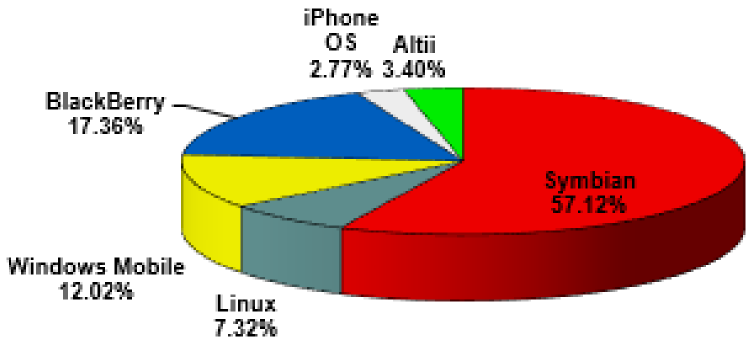
- Own operating systems
  - Feature phones
  - Development platform like Java ME
- Operating systems for smartphones/tablets
  - Ability to develop applications based on a SDK

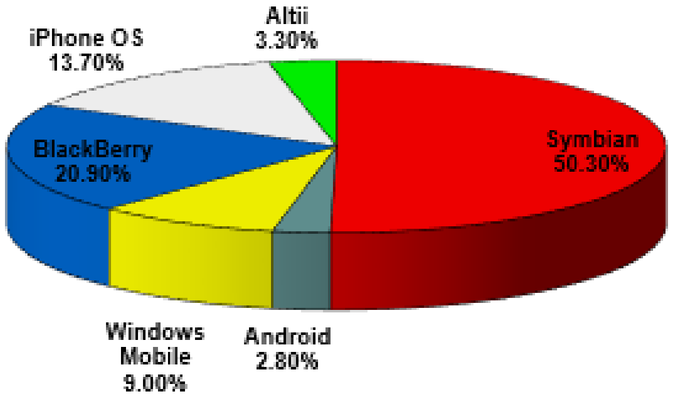
Android Basics

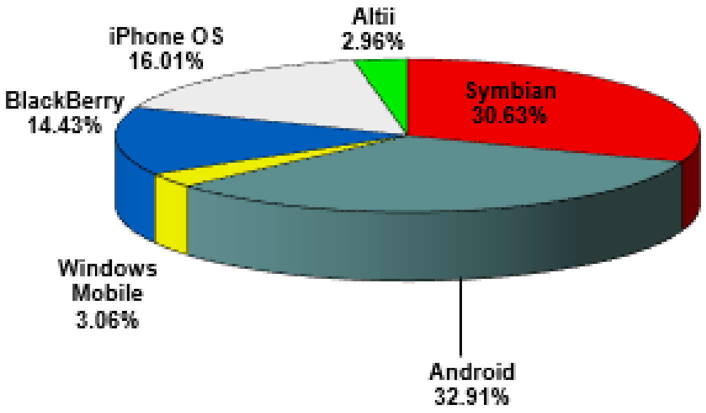
Operating systems for smartphones/tablets:

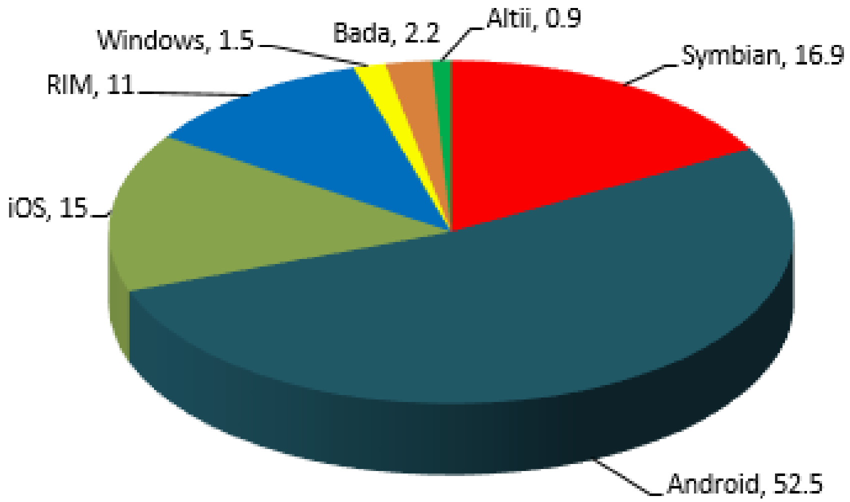
- **Android** (Google)
- Bada (Samsung)
- Tizen (Tizen Association)
- BlackBerry OS (BlackBerry/RIM)
- BREW (Qualcomm)
- Firefox OS (Mozilla)
- iOS (Apple)
- Linux Mobile
- Palm OS/Garnet OS (Palm)
- Symbian (Nokia)
- webOS (HP)
- **Windows Phone**/Windows CE/Windows Mobile (Microsoft)

Android Basics
Smartphone (2008-2011)

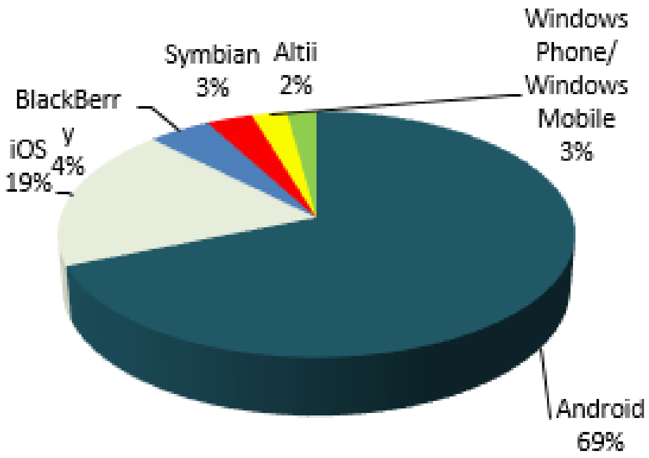


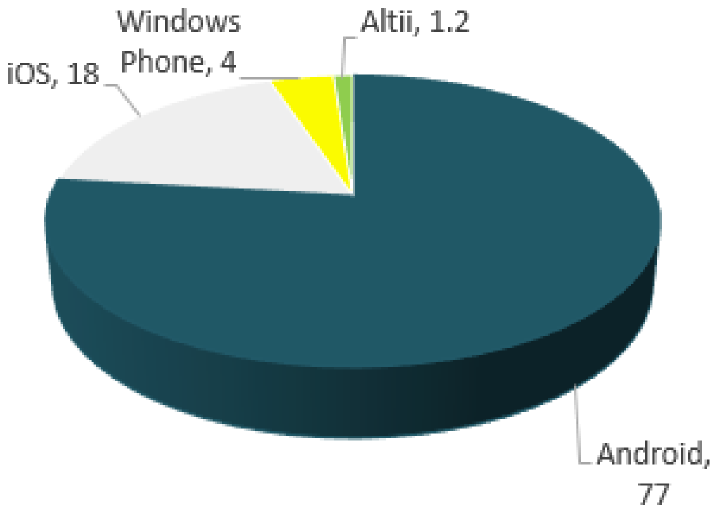


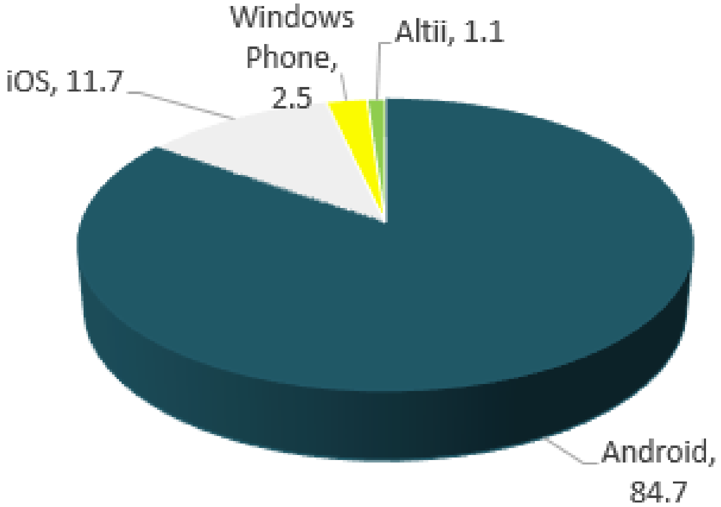




Android Basics
Smartphone (2012 – 2014)



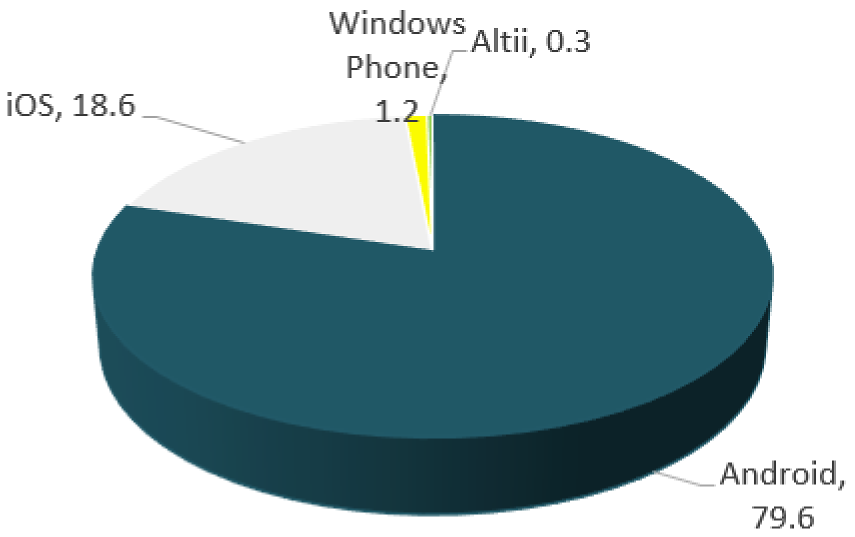


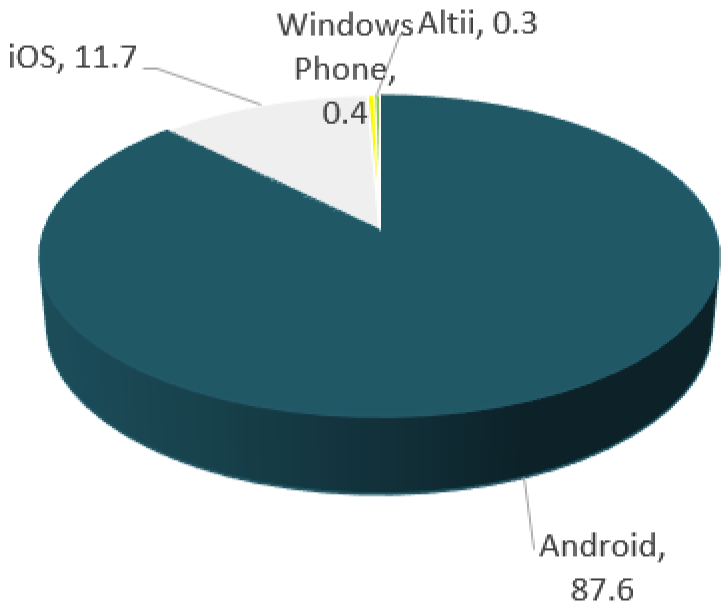


At the end of 2016:

- **Android market share was 88%**
- **99.6% of new smartphones run Android or iOS**

Android Basics
Smartphone (2015 – 2016)





Android Basics
Smartphone (2009 – 2018)

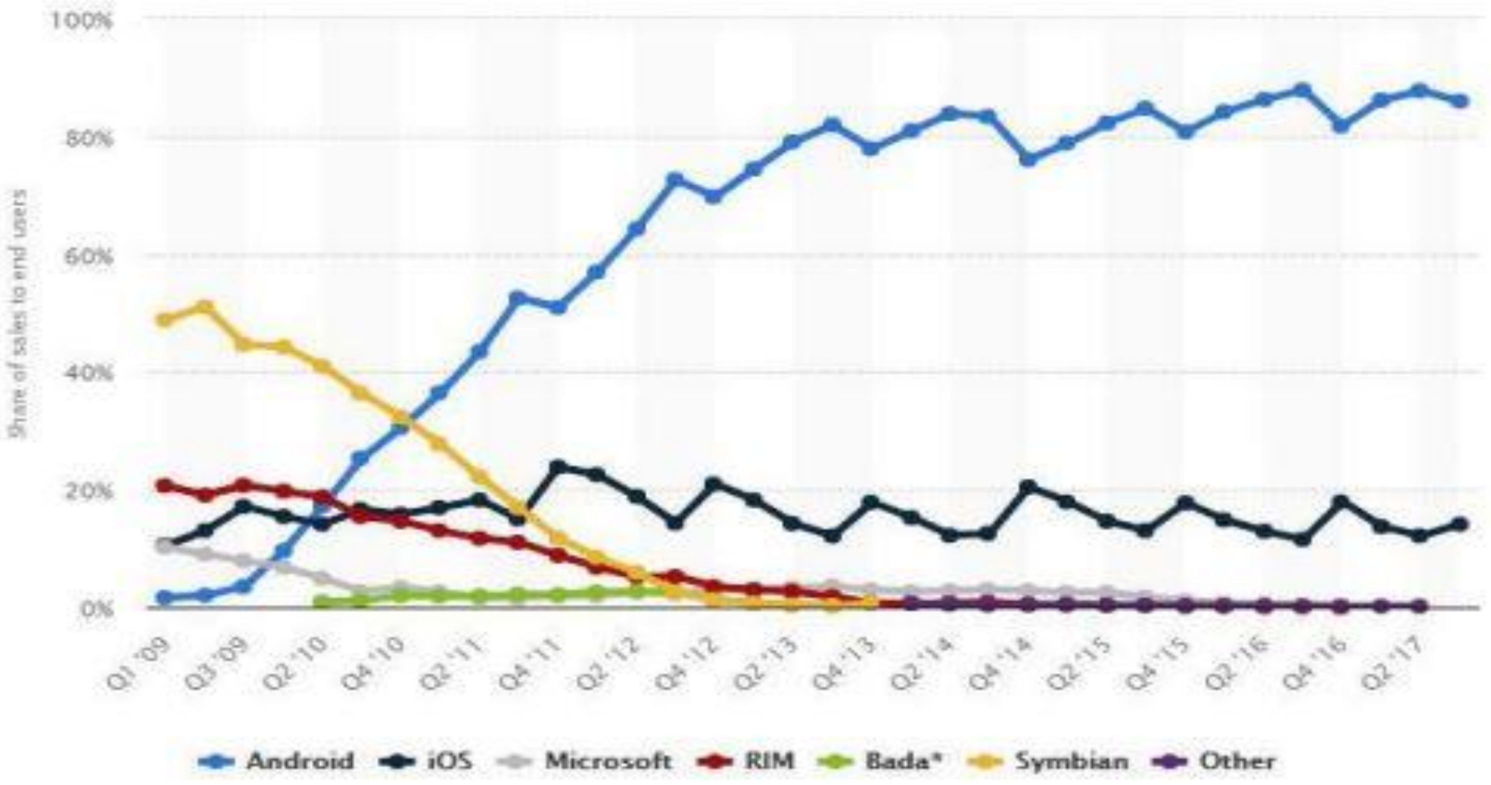
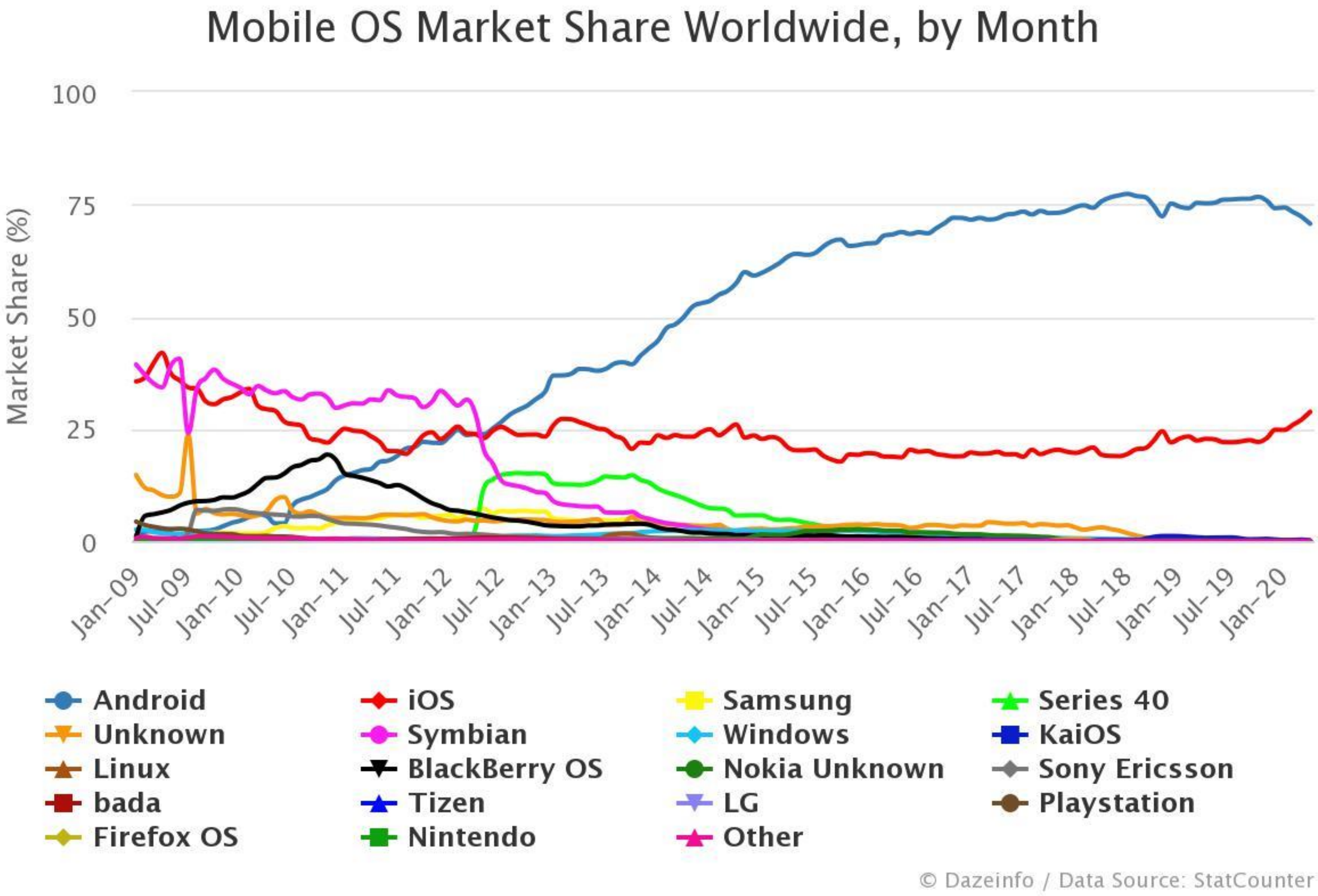


Fig. 1: www.statista.com, Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 1st quarter 2018

Android Basics
Smartphone (2009 – 2020)



© Dazeinfo / Data Source: StatCounter

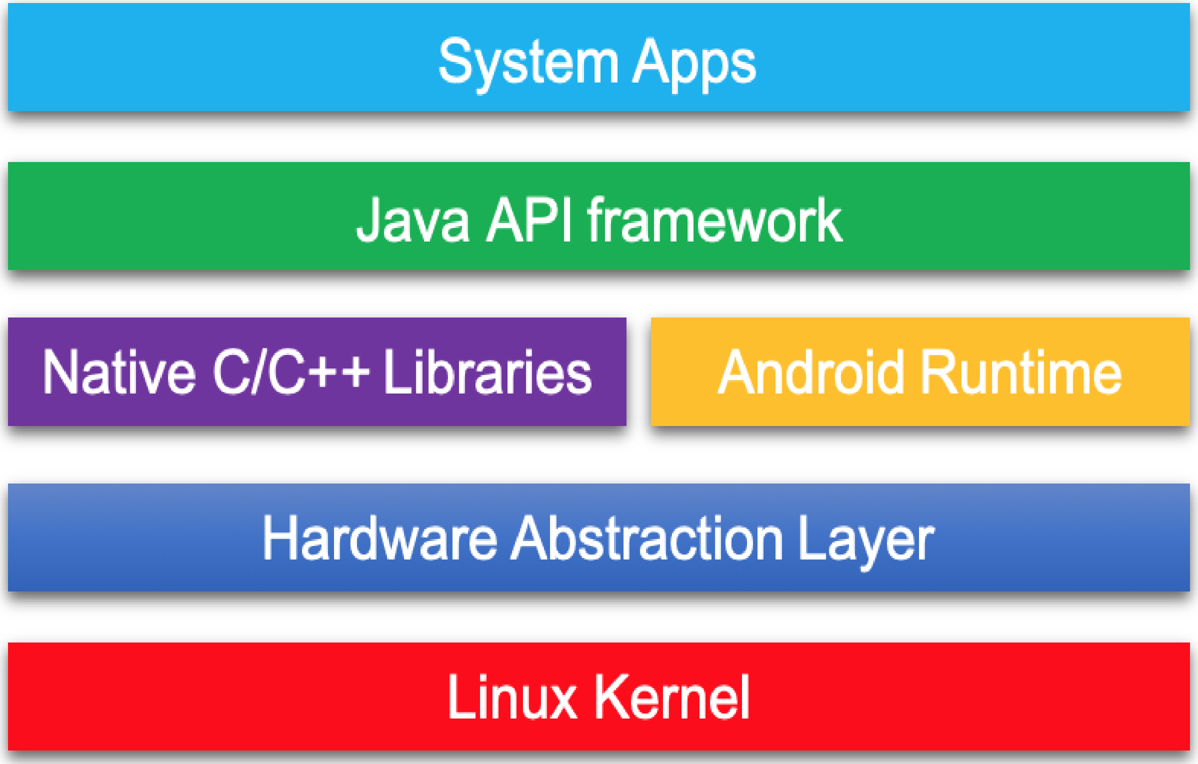
Android Basics

Android:

- Project initiated by Google
- Based on Linux core version 2.6.x/3.x
- Applications are based on Java/Kotlin
- Smartphones:
  - HTC One, Samsung Galaxy, Google Pixel, Huawei
- Tablets:
  - Samsung Galaxy Tab, Google Nexus

Android platform architecture

The Android architecture assumes the existence of the following stack:



System Apps

Java API framework

Native C/C++ Libraries

Android Runtime

Hardware Abstraction Layer

Linux Kernel

Android Basics

Android versions:

Operating system	API level
Android 2.2 (Froyo)	8
Android 2.3.3 – 2.3.7 (Gingerbread)	10
Android 3.x (Honeycomb)	11-13 (tablets)
Android 4.0.x (Ice Cream Sandwich)	14, 15
Android 4.1, 4.2, 4.3 (Jelly Bean)	16, 17, 18
Android 4.4 (KitKat)	19

Android Basics

Android versions:

Operating system	API level
Android 4.4W (Wear)	20
Android 5.0, 5.1.x (Lollipop)	21, 22
Android 6.0 (Marshmallow)	23
Android 7.0 (Nougat)	24, 25
Android 8.0 (Oreo)	26, 27
Android 9.0 (Pie)	28
Android 10 (Queen Cake)	29

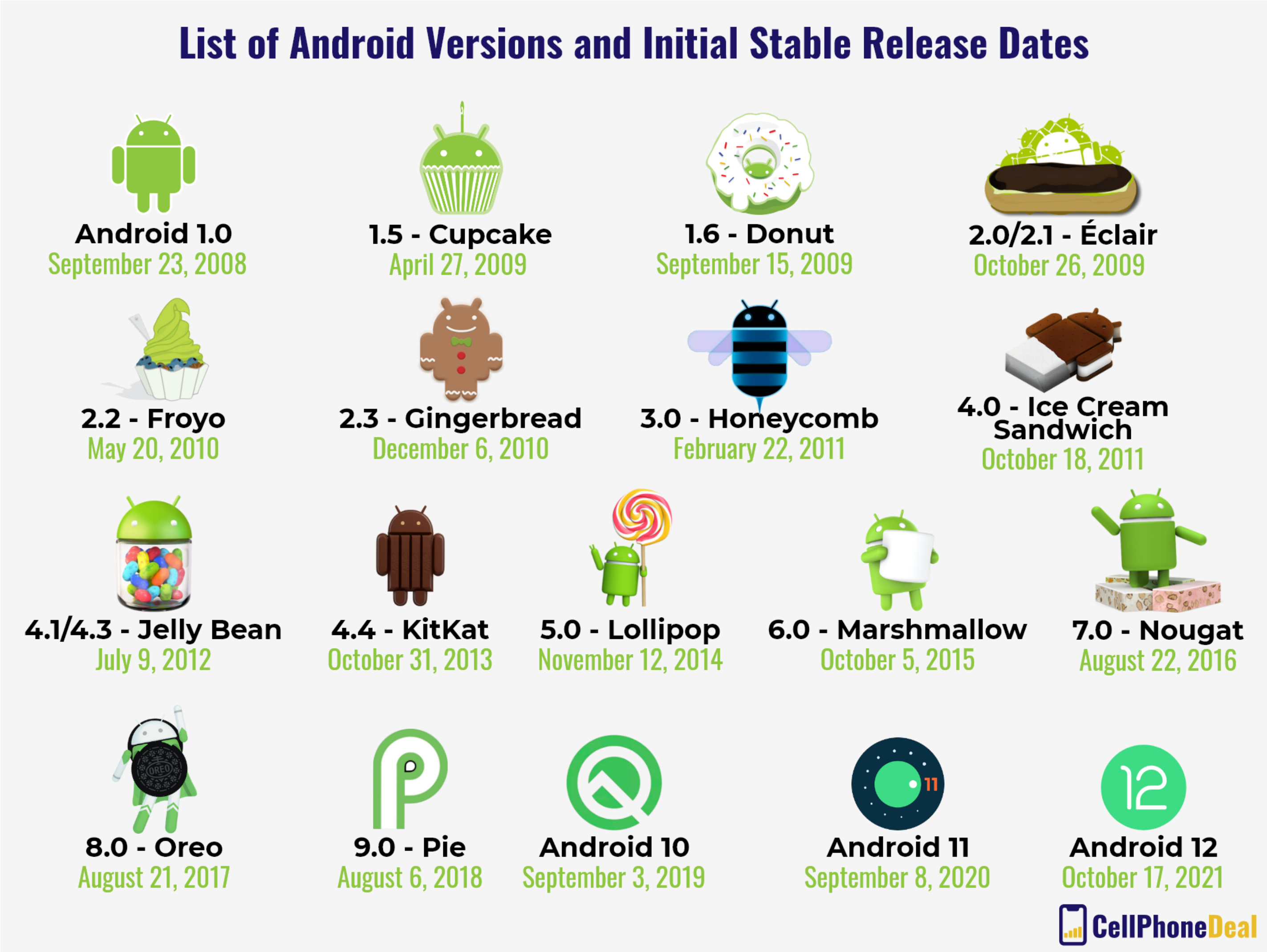
Android Basics

Android versions:

Operating system	API level
Android 11 (Red Velvet Cake)	30
Android 12 (Snow Cone)	31, 32
Android 13 (Tiramisu)	33
Android 14 (Upside Down Cake)	34
Android 15 (Vanilla Ice Cream)	35

Android Basics

Android versions:



List of Android Versions and Initial Stable Release Dates

Android 1.0
September 23, 2008

1.5 - Cupcake
April 27, 2009

1.6 - Donut
September 15, 2009

2.0/2.1 - Éclair
October 26, 2009

2.2 - Froyo
May 20, 2010

2.3 - Gingerbread
December 6, 2010

3.0 - Honeycomb
February 22, 2011

4.0 - Ice Cream Sandwich
October 18, 2011

4.1/4.3 - Jelly Bean
July 9, 2012

4.4 - KitKat
October 31, 2013

5.0 - Lollipop
November 12, 2014

6.0 - Marshmallow
October 5, 2015

7.0 - Nougat
August 22, 2016

8.0 - Oreo
August 21, 2017

9.0 - Pie
August 6, 2018

Android 10
September 3, 2019

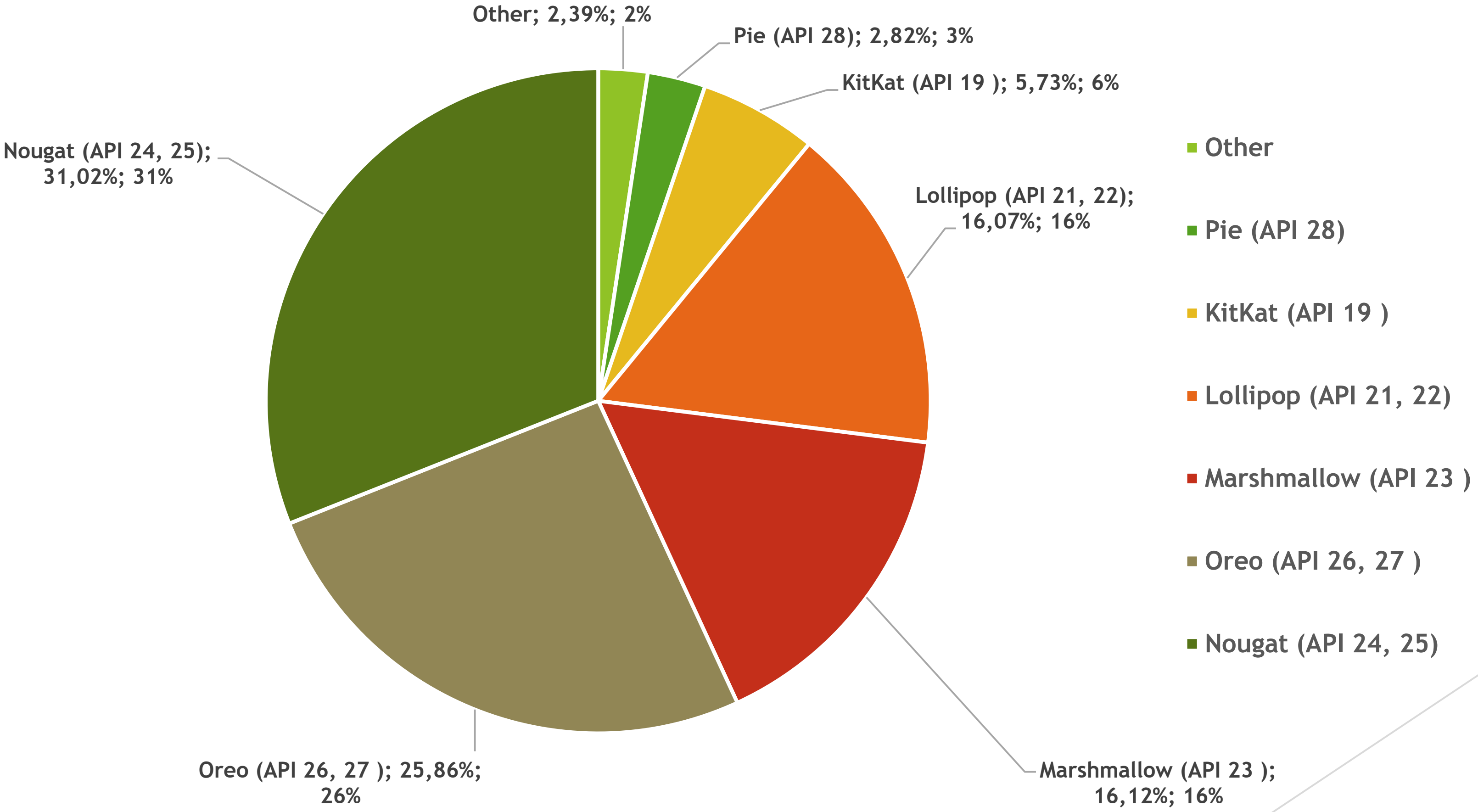
Android 11
September 8, 2020

Android 12
October 17, 2021

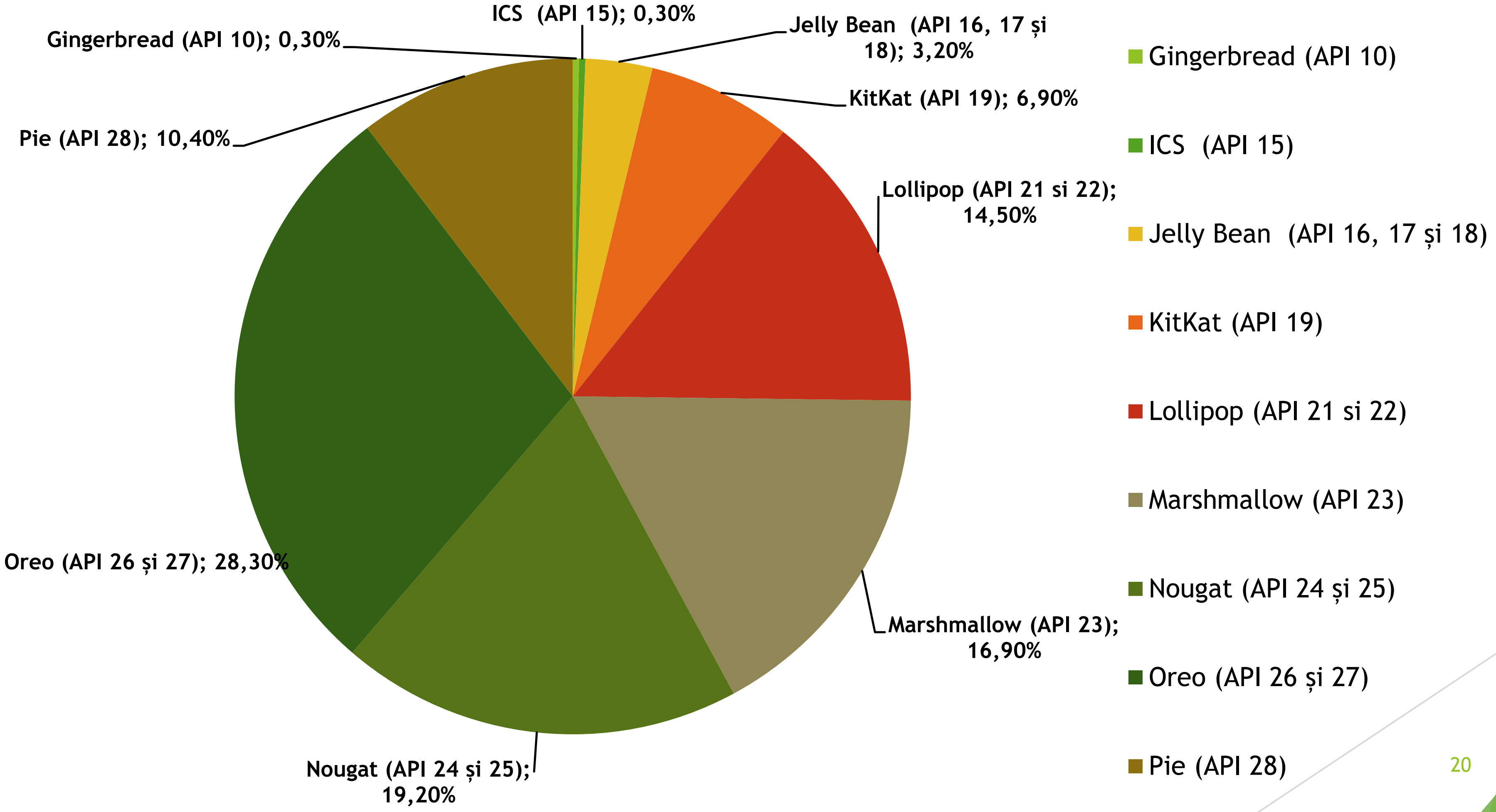
CellPhoneDeal

Android versions in Romania (May 2019)

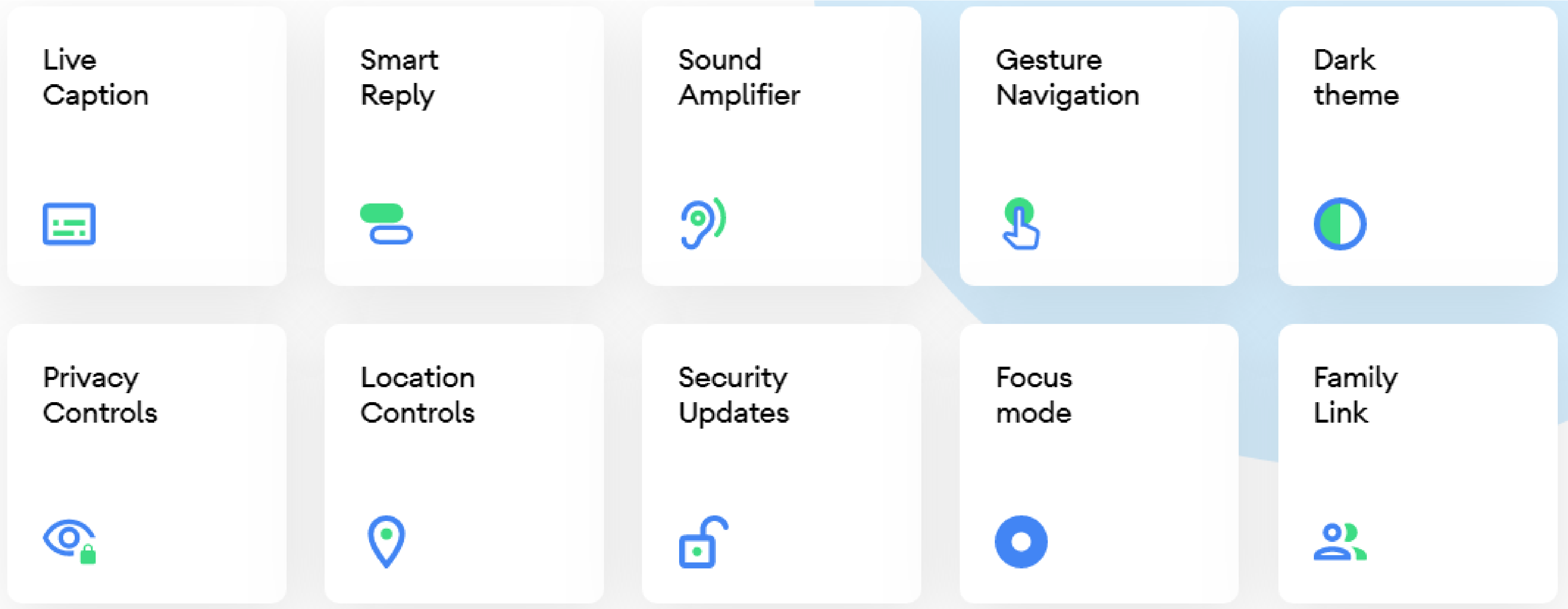
- According to StatCounter, the Android operating system ranks first, from June 2017 to date, in Romania, registering 80.26% in May 2019.



Android versions worldwide (May 2019)



Android 10 Highlights



Live Caption

Smart Reply

Sound Amplifier

Gesture Navigation

Dark theme

Privacy Controls

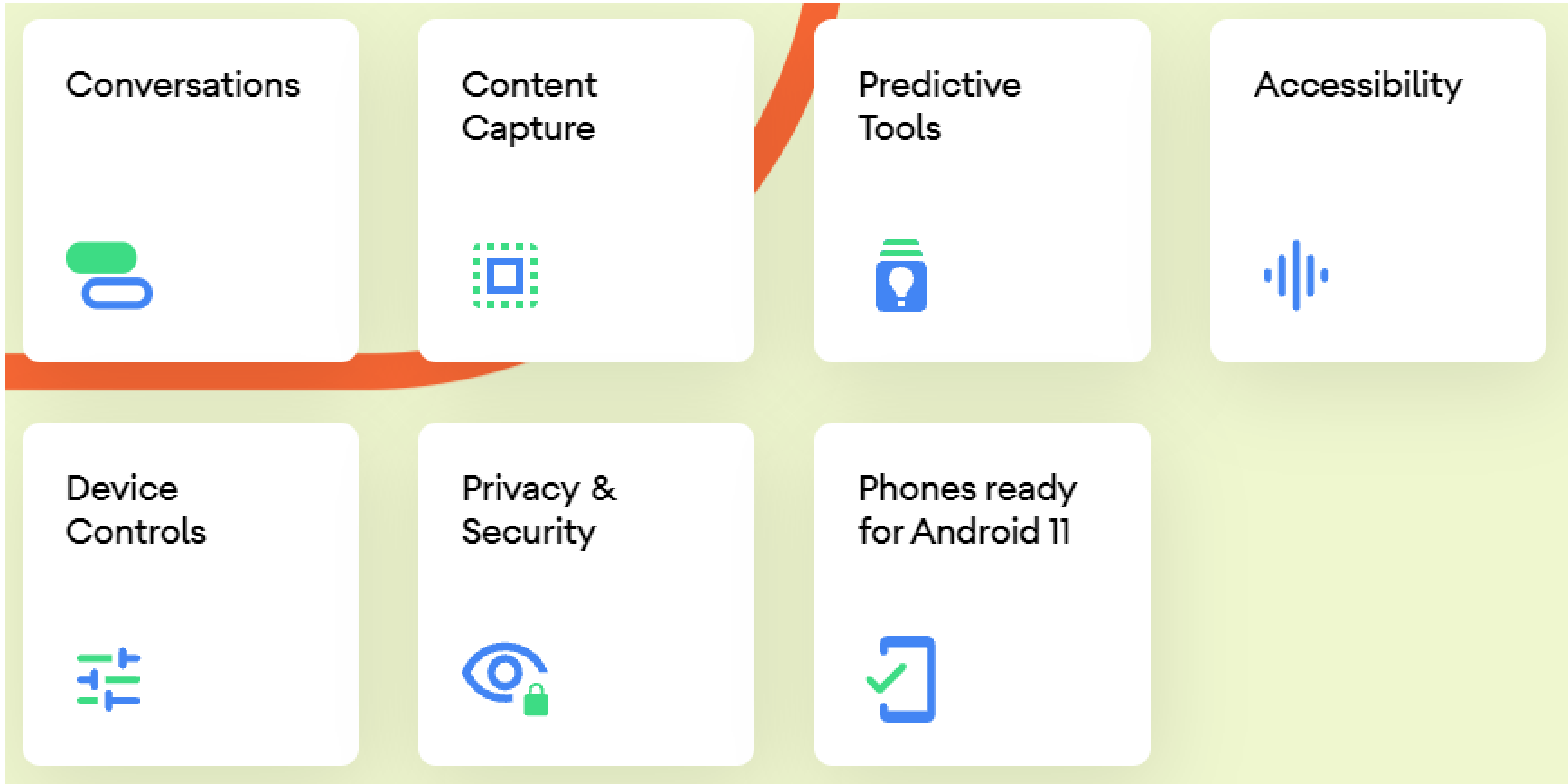
Location Controls

Security Updates

Focus mode

Family Link

Android 11 Highlights



Conversations

Content Capture

Predictive Tools

Accessibility

Device Controls

Privacy & Security

Phones ready for Android 11

Android 12 Highlights

Dynamic Color. Responsive Motion. Conversation Widgets. Accessibility Improvements. Safe. Mic & Camera Indicators and Toggles. Approximate Location Permissions. Privacy Dashboard. Private Compute Core. Effortless. Enhanced Gaming. Scrolling Screenshots.

Android 13 Highlights

Android 13 Highlights

Android 13 features and changes list

- Camera.
- Core functionality.
- Developer productivity and tools.
- Graphics.
- Media.
- Performance and battery.
- Privacy and security.
- Tablets and large screens.

Android 14 Highlights

Android 14: More customization, control and accessibility features

- More ways to express yourself
- More control over your health, security and data
- More features for low-vision and hard-of-hearing users

With Android 14, your health, safety and data is protected. You’ll be better informed of how apps use and share your data, and you’ll be encouraged to better secure your phone from unauthorized access. Plus, even more ways to make your device more you.

Android Basics

Four reasons why Android receives so much attention:

- Android is available for free
- Everyone can get Development Tools, Technical documents and Source code for free
- The training cost is low if you know Java SE
- It is deployed in embedded development field besides mobile phones (e.g. automotive industry)

Android Basics

Mobile applications classification:

- In terms of **implementation**
  - Based on the Web interface
  - Independent/client applications
    - Native (specific API)
    - JIT interpreted or compiled binary code (like Java ME)
- In terms of **network access**
  - Distributed applications
    - Requirement: Network/Internet access
  - Independent (standalone) applications
    - The access to network/Internet is not necessary

Android Basics

- Applications for mobile devices
- Implementation
  - **Native** Applications
  - **Hybrid** Applications
    - Interpretable or JIT compiled binary code
    - Use an intermediate level
  - Mobile **web** applications
- With / without network access

Android Basics

Mobile applications development:

Operating system	Programming language
Android	Java, Kotlin
iOS	Objective-C, C++, Swift
Windows Phone	C#/VB.NET (Silverlight and XNA), C++

Android Basics

Android programming model:

- Linux core
  - C native libraries
- Based on Java
- Native programming interface (C++ code)
- ART – Android RunTime
  - The current execution environment of the Android applications
    - Starting with Android 4.4
  - Compilation before execution
- Own virtual machine (Dalvik VM)
  - Executable binary code is not compatible with Java SE
    - *dex* files
  - Each application runs in a separate process
  - JIT compilation

Android Basics

Programming facilities:

- GUI
- SQLite databases
- Media API: Audio, Video
- Camera
- 2D graphics, Animation
- 3D graphics (OpenGL)
- Personal information management
- Network and communications: Socket, HTTP, Bluetooth
- Telephony
- Sensors: GPS, Accelerometer, Gyroscope, NFC

Android Basics

Development Tools:

- Software needed:
  - Java SE Development Kit (JDK)
  - Android SDK
- Integrated development environments (IDE):
  - Android Studio
    - Official environment (starting with 2013)
  - Eclipse
    - plugin: Android Development Toolkit (ADT)
  - IntelliJ IDEA, NetBeans (plugins required)

Android Basics

Feature	Android Studio	Eclipse ADT
Build system	Gradle	Apache Ant
Maven-based build dependencies	Yes	No
Build variants and multiple-APK generation	Yes	No
Advanced Android code completion and refactoring	Yes	Yes
Graphical layout editor	Yes	Yes
APK signing and keystore management	Yes	Yes
NDK support	Yes	Yes

Source: https://en.wikipedia.org/wiki/Android_Studio

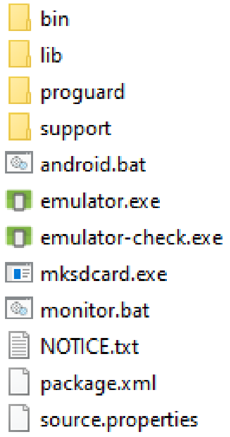
Android Basics

Android SDK:

- Android platform-specific libraries
- Resources and emulators images
- Platform-specific resources
- Tools to compile and generate executable binary content
  - Source code
  - Resources

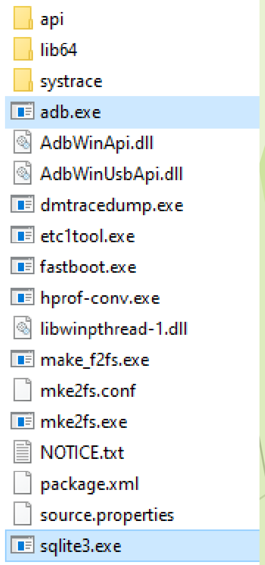
Android Basics

- **Android SDK Tools:** C:\Users\Cristian\AppData\Local\Android\sdk\tools
  - Subfolder: **sdk/tools**
  - Tools:
    - ant scripts to obtain the application binary package
    - monitor (ddms)
    - emulator-arm, emulator-x86



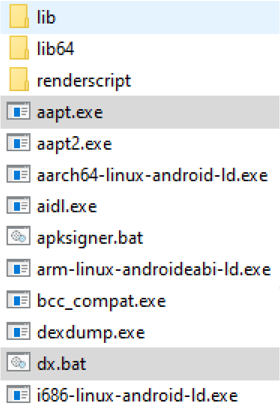
bin
lib
proguard
support
android.bat
emulator.exe
emulator-check.exe
mksdcard.exe
monitor.bat
NOTICE.txt
package.xml
source.properties

- **Android SDK Platform-tools:** C:\Users\Cristian\AppData\Local\Android\sdk\platform-tools
  - Folder: **sdk/platform-tools**
  - Tools:
    - adb – communication with Android devices
    - sqlite3



api
lib64
systrace
adb.exe
AdbWinApi.dll
AdbWinUsbApi.dll
dmtracedump.exe
etc1tool.exe
fastboot.exe
hprof-conv.exe
libwinpthread-1.dll
make_f2fs.exe
mke2fs.conf
mke2fs.exe
NOTICE.txt
package.xml
source.properties
sqlite3.exe

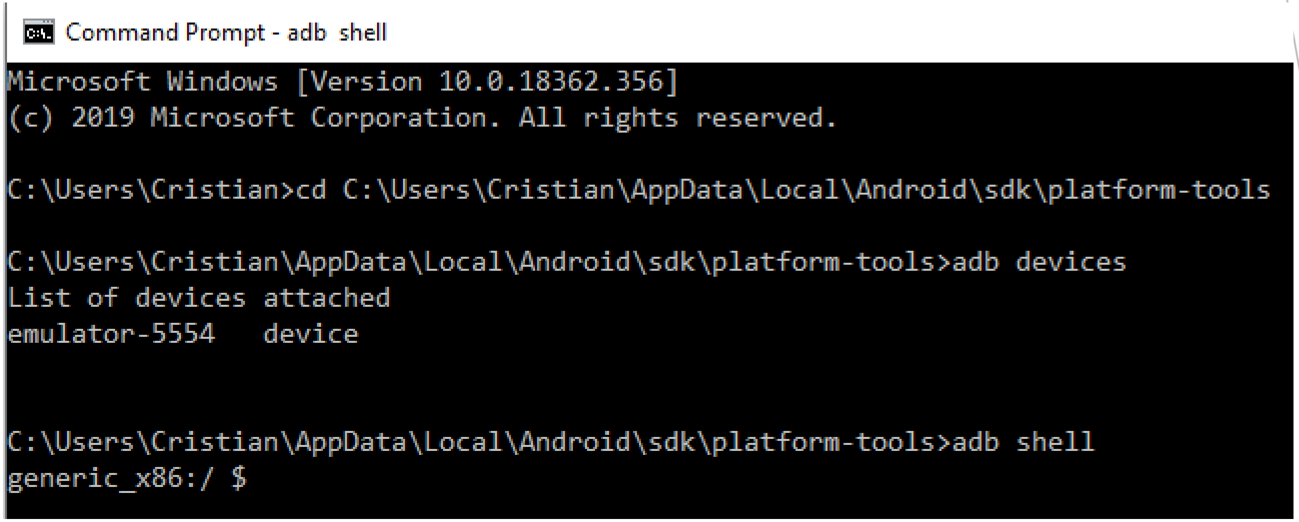
- **Android SDK Build-tools:** C:\Users\Cristian\AppData\Local\Android\sdk\build-tools\25.0.2
  - Folder: **sdk/build-tools/version/**
  - Tools:
    - aapt - compilation of resources, R.java file generation, APK files
    - dx - conversion of Java binary code to Dalvik binary code



lib
lib64
renderscript
aapt.exe
aapt2.exe
aarch64-linux-android-ld.exe
aidl.exe
apksigner.bat
arm-linux-androideabi-ld.exe
bcc_compat.exe
dexdump.exe
dx.bat
i686-linux-android-ld.exe

Android Debug Bridge (ADB)

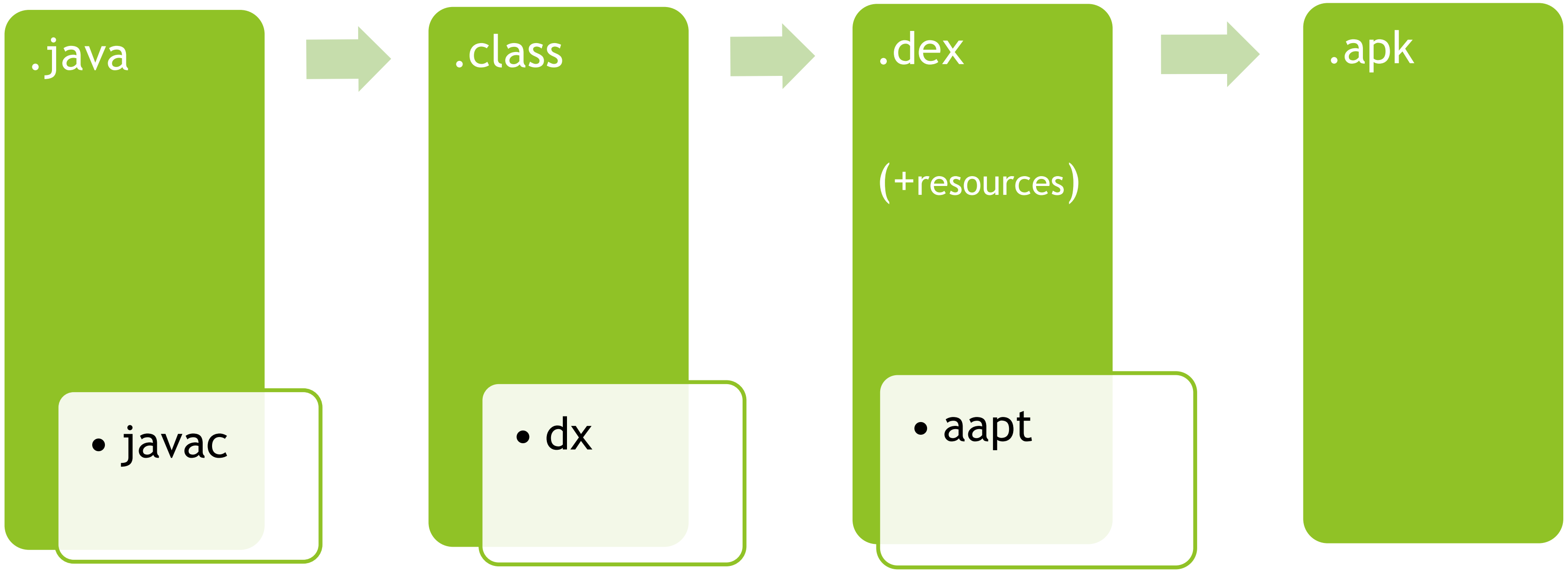
- adb devices
- adb kill-server
- adb install
- adb pull
- adb push
- adb shell



```
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Cristian>cd C:\Users\Cristian\AppData\Local\Android\sdk\platform-tools

C:\Users\Cristian\AppData\Local\Android\sdk\platform-tools>adb devices
List of devices attached
emulator-5554   device


C:\Users\Cristian\AppData\Local\Android\sdk\platform-tools>adb shell
generic_x86:/ $
```

Android Basics

Android binary files:

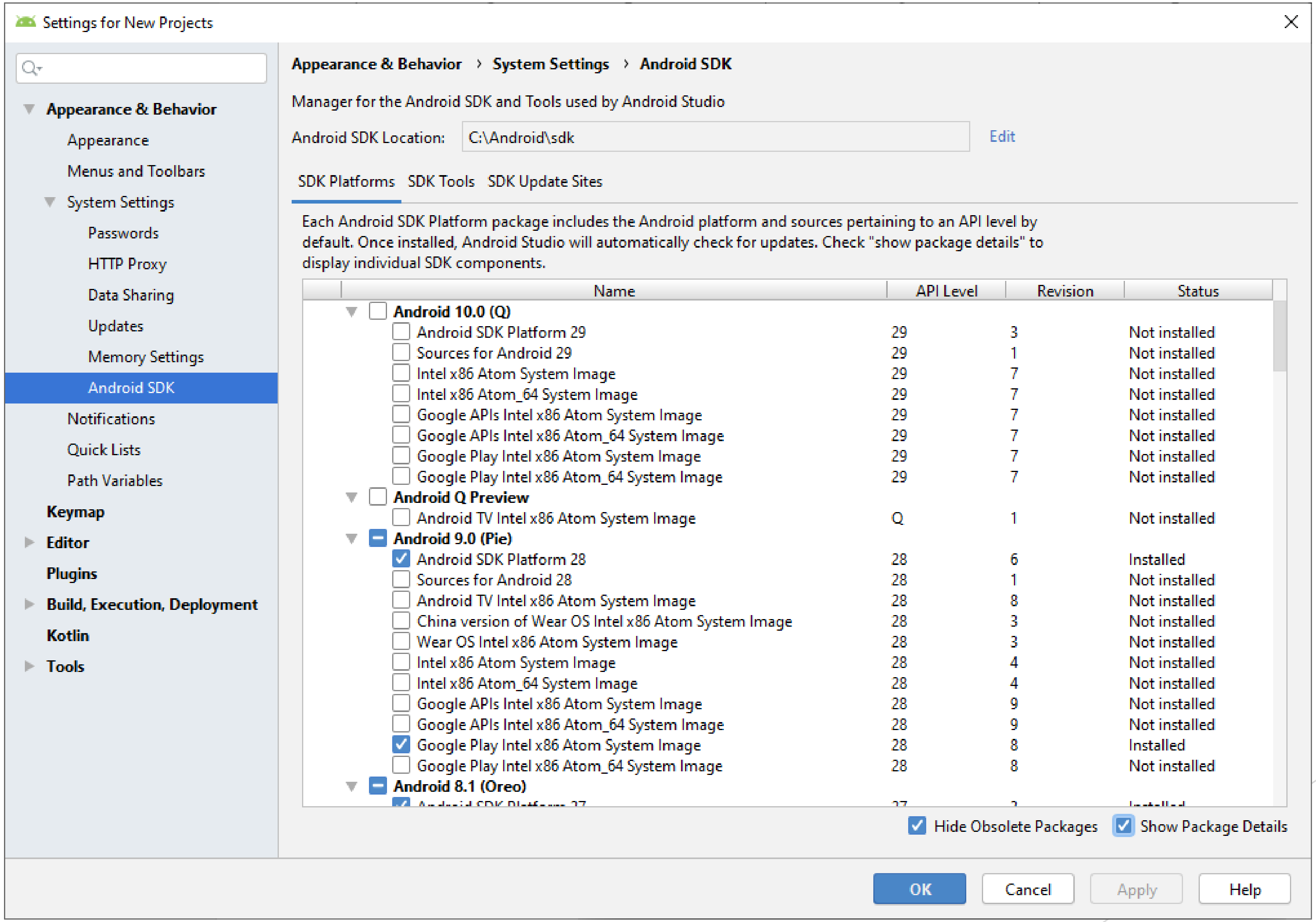


Android Basics

Android SDK Manager:

- Management of platforms and tools needed
- Direct access or from the development environment
- Platform includes:
  - libraries
  - source code
  - documentation
  - emulator images

Android SDK Manager



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Android\sdk Edit

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
☐ **Android 10.0 (Q)**			
☐ Android SDK Platform 29	29	3	Not installed
☐ Sources for Android 29	29	1	Not installed
☐ Intel x86 Atom System Image	29	7	Not installed
☐ Intel x86 Atom_64 System Image	29	7	Not installed
☐ Google APIs Intel x86 Atom System Image	29	7	Not installed
☐ Google APIs Intel x86 Atom_64 System Image	29	7	Not installed
☐ Google Play Intel x86 Atom System Image	29	7	Not installed
☐ Google Play Intel x86 Atom_64 System Image	29	7	Not installed
☐ **Android Q Preview**			
☐ Android TV Intel x86 Atom System Image	Q	1	Not installed
⊟ **Android 9.0 (Pie)**			
☑ Android SDK Platform 28	28	6	Installed
☐ Sources for Android 28	28	1	Not installed
☐ Android TV Intel x86 Atom System Image	28	8	Not installed
☐ China version of Wear OS Intel x86 Atom System Image	28	3	Not installed
☐ Wear OS Intel x86 Atom System Image	28	3	Not installed
☐ Intel x86 Atom System Image	28	4	Not installed
☐ Intel x86 Atom_64 System Image	28	4	Not installed
☐ Google APIs Intel x86 Atom System Image	28	9	Not installed
☐ Google APIs Intel x86 Atom_64 System Image	28	9	Not installed
☑ Google Play Intel x86 Atom System Image	28	8	Installed
☐ Google Play Intel x86 Atom_64 System Image	28	8	Not installed
⊟ **Android 8.1 (Oreo)**			
☑ Android SDK Platform 27	27	3	Installed

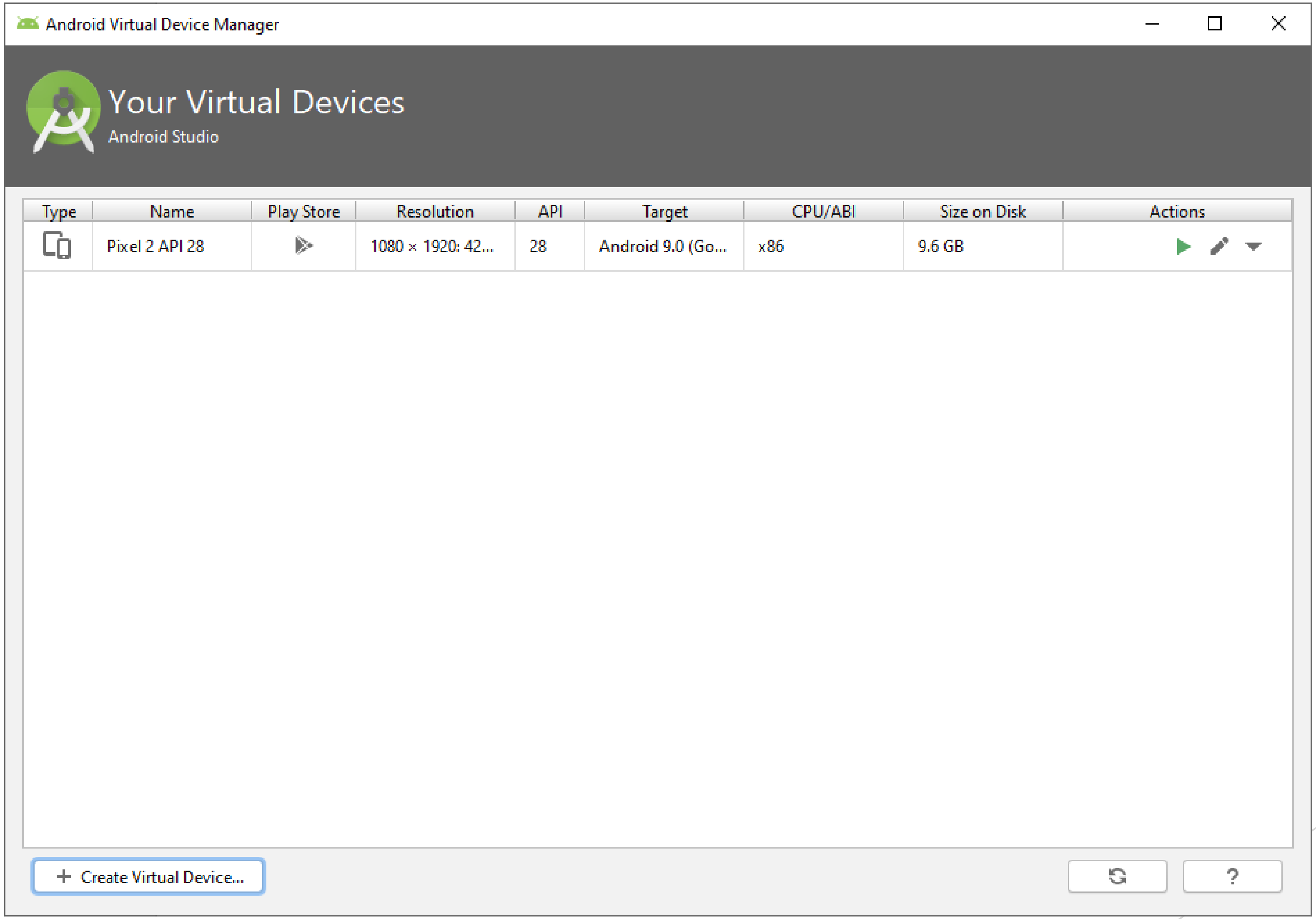
☑ Hide Obsolete Packages ☑ Show Package Details

Android Basics

Android Virtual Device (AVD):

- Android virtual devices:
  - Emulators
- Characteristics:
  - Processor (CPU), display (resolution/size), camera, memory (RAM, internal and persistent, external), API version
- Emulation
  - ARM (Advanced RISC Machine)
  - x86, x64
    - Require Intel HAXM (Hardware Accelerated Execution Manager) and CPU with virtualization support
- Communication through *adb.exe* application

AVD Manager



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 2 API 28	▶	1080 × 1920: 42...	28	Android 9.0 (Go...	x86	9.6 GB	▶ ✎ ▾

Android Basics

Dalvik Debug Monitor Server (DDMS):

- Allow access to:
  - virtual devices
  - physical devices
- Visualization:
  - and possibility to stop processes
  - memory
  - network statistics
  - messages console (LogCat)
- Access to file system

Android Basics

Messages console (LogCat):

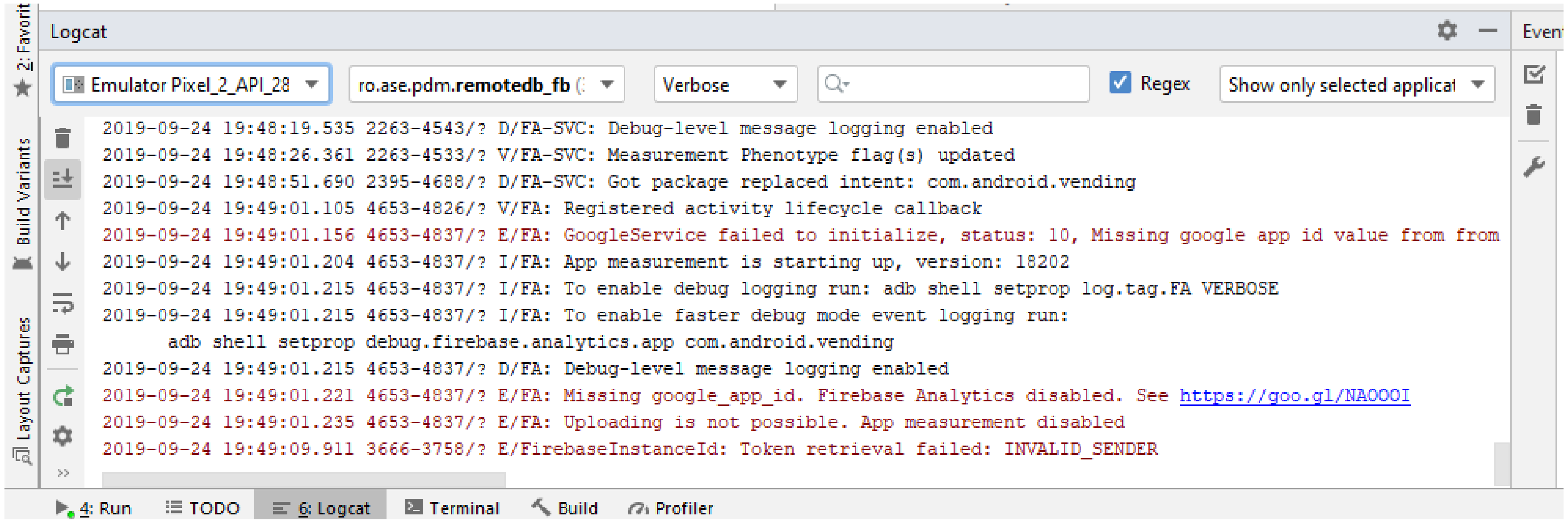
- Displays messages sent from:
  - user applications
  - system applications
- Messages:
  - Warning (w)
  - Debugging (d)
  - Error (e)
  - Information (i)
  - Detailed information (v)
  - Exceptional error (wtf)

Android Basics

Messages console (LogCat):

- Class: android.util.**Log**
- Static methods associated with the types of messages:
  - **e**(), **w**(), **i**(), **d**(), **v**(), **wtf**()
- Parameters:
  - Source identifier message (String)
    - The name of class, application, activity etc.
    - Ability to filter
  - The message that will be displayed (String)
- General static method:
  - **println**()
  - In addition, the first parameter includes the type of message: **Log.ASSERT**, **Log.ERROR**, **Log.INFO** etc.
- Example:
  - `Log.i("Activity1", "Information message");`
  - `println(Log.ASSERT, "Activity 1", "Invalid assertion!");`

LogCat



Emulator Pixel_2_API_28 | ro.ase.pdm.**remotedb_fb** (| Verbose | ☑ Regex | Show only selected applicat

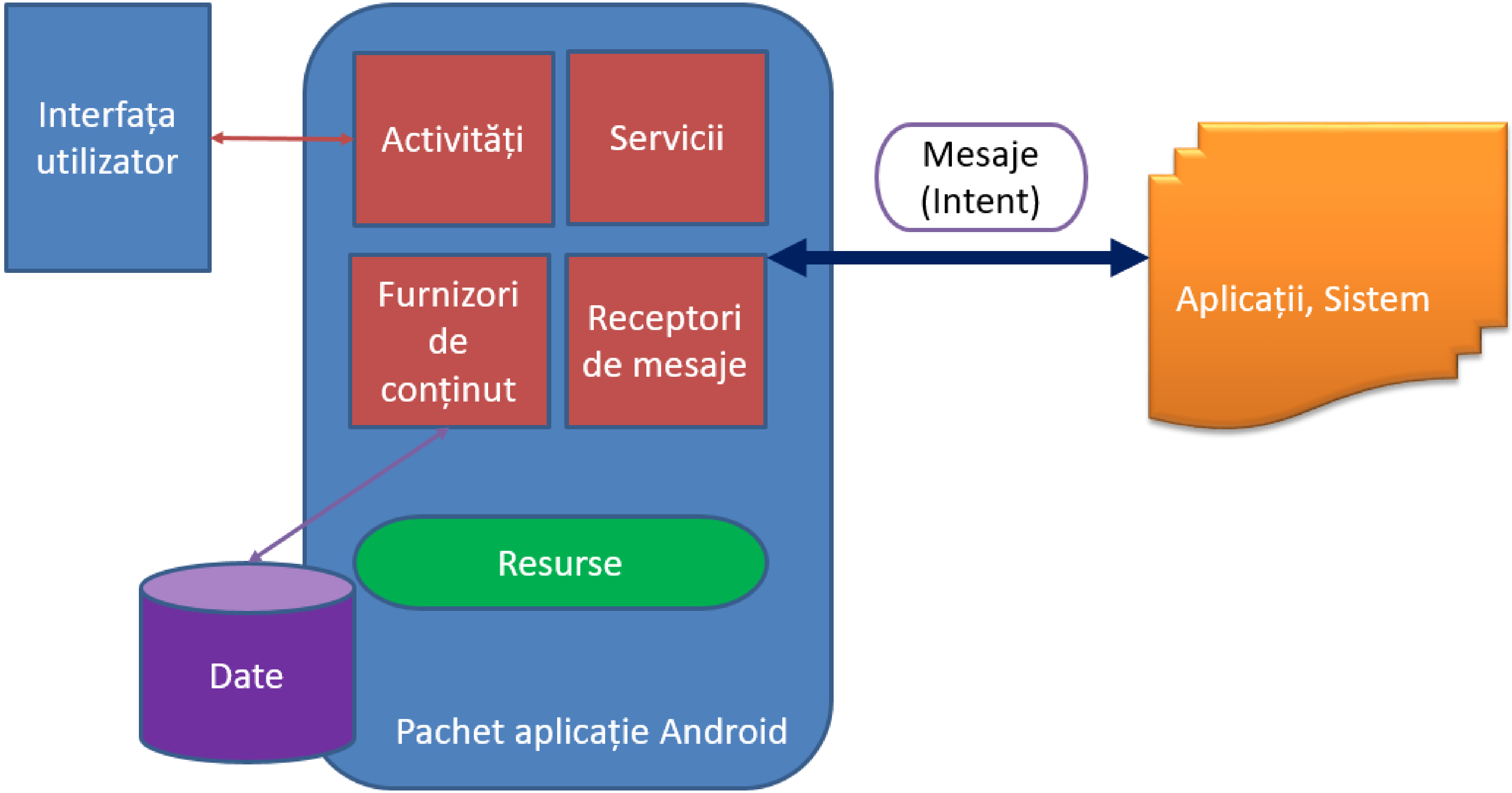
```
2019-09-24 19:48:19.535 2263-4543/? D/FA-SVC: Debug-level message logging enabled
2019-09-24 19:48:26.361 2263-4533/? V/FA-SVC: Measurement Phenotype flag(s) updated
2019-09-24 19:48:51.690 2395-4688/? D/FA-SVC: Got package replaced intent: com.android.vending
2019-09-24 19:49:01.105 4653-4826/? V/FA: Registered activity lifecycle callback
2019-09-24 19:49:01.156 4653-4837/? E/FA: GoogleService failed to initialize, status: 10, Missing google app id value from from
2019-09-24 19:49:01.204 4653-4837/? I/FA: App measurement is starting up, version: 18202
2019-09-24 19:49:01.215 4653-4837/? I/FA: To enable debug logging run: adb shell setprop log.tag.FA VERBOSE
2019-09-24 19:49:01.215 4653-4837/? I/FA: To enable faster debug mode event logging run:
      adb shell setprop debug.firebase.analytics.app com.android.vending
2019-09-24 19:49:01.215 4653-4837/? D/FA: Debug-level message logging enabled
2019-09-24 19:49:01.221 4653-4837/? E/FA: Missing google_app_id. Firebase Analytics disabled. See https://goo.gl/NAOOOI
2019-09-24 19:49:01.235 4653-4837/? E/FA: Uploading is not possible. App measurement disabled
2019-09-24 19:49:09.911 3666-3758/? E/FirebaseInstanceId: Token retrieval failed: INVALID_SENDER
```

Android Basics

Basic components of Android applications:

- Activities
  - The base class is *android.app*.**Activity**
- Services
  - The base class is *android.app*.**Service**
- Content providers
  - The base class is *android.content*.**ContentProvider**
- Broadcast receivers
  - The base class is *android.content*.**BroadcastReceiver**
  - messages
    - The base class is *android.content*.**Intent**

Android Basics



Android Basics

Activities:

- Associated with application windows
- An application can have one or more activities
  - One main activity
- Visual components associated
  - Derived from **View** class

Android Basics

Services:

- Routines running in parallel with the main thread
- Do not have graphical interface
- Allows running actions in the background without blocking:
  - the main thread
  - interaction with applications

Android Basics

Content providers:

- Support for sharing data between applications
- Shared data is stored in different data sources (files, databases, etc.)
- They provide a standard way for data access and updating them
- The access is achieved via a URI like content://

Android Basics

Messages (Intent objects):

- To activate components are used asynchronous messages
  - encapsulated in objects of type **Intent**
- Invocation of components
  - Open browser, initiating phone calls application, display map to a specific geographic location etc.
- Communication between components

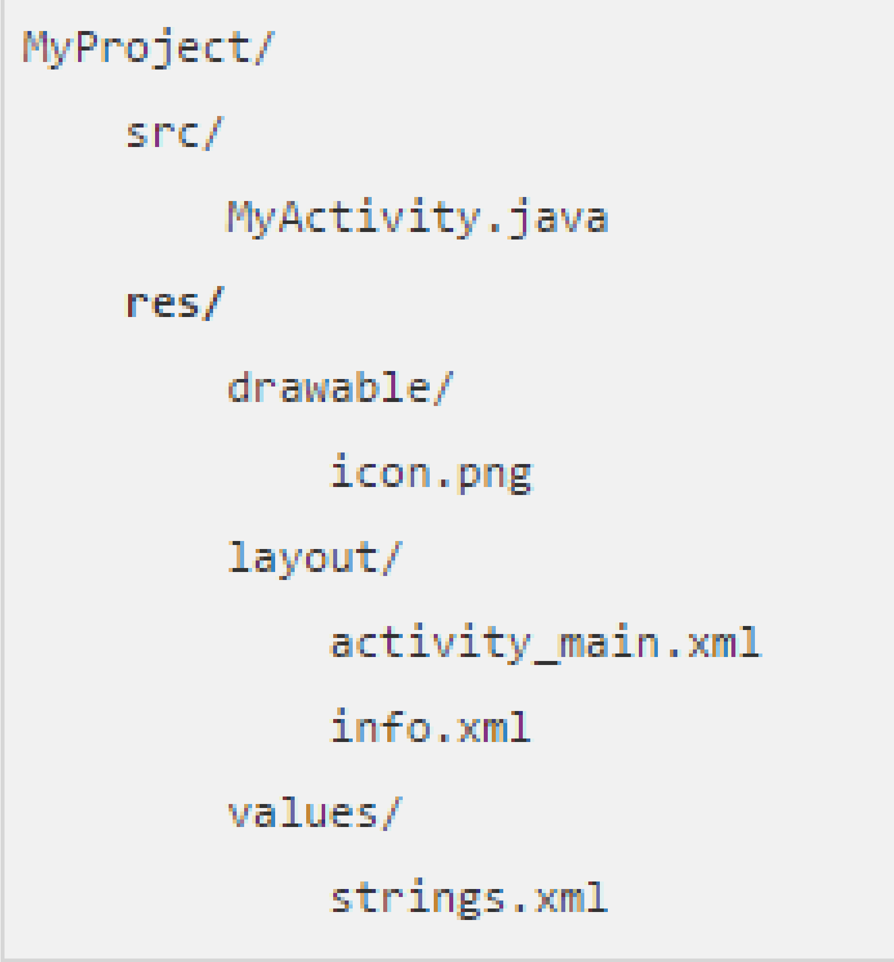
Android Basics

Project structure:

- Source files (src)
- Resources (res)
  - res/drawable
  - res/layout
  - res/values
  - res/menu
  - res/xml
  - res/raw
- Resources taken as streams (assets)
- Configuration file (AndroidManifest.xml)
- Generated files (gen)
  - R.java

Android Basics

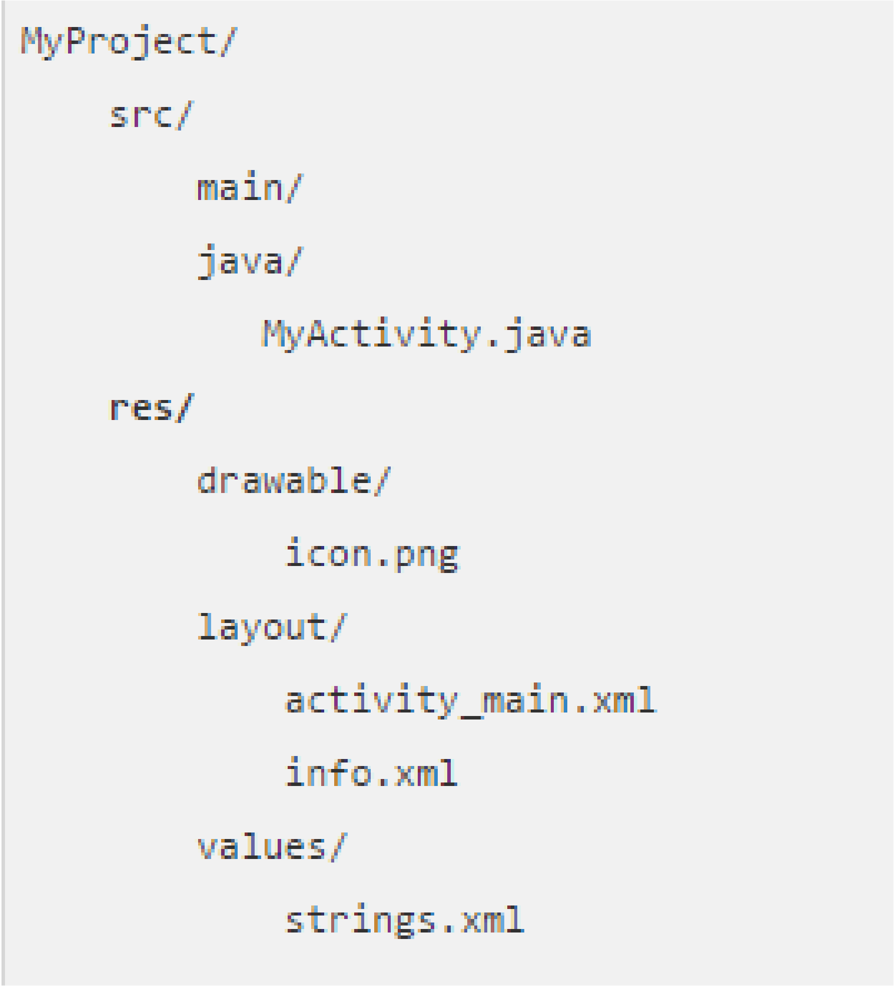
- Organize Resources in **Eclipse**



```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            activity_main.xml
            info.xml
        values/
            strings.xml
```

Android Basics

- Organize Resources in **Android Studio**



```
MyProject/
    src/
        main/
        java/
           MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            activity_main.xml
            info.xml
        values/
            strings.xml
```

Android Basics

AndroidManifest.xml:

- Package information (name, version)
- Application attributes (name, associated icon, theme, memory options, restrictions, permissions etc.)
- Filters for messages, applied inside the application/components
- SDK versions (minimum, maximum, desired)
- Access permissions
  - <**uses-permission** android:name="permission"/>
- Hardware and software requirements
  - <**uses-feature** android:name="requirement" android:required="true/false"/>
- Application components
  - declaration of activities, services, content providers, broadcast receivers
  - names of the classes associated
  - properties

Android Basics

Examples of permissions:

For…	**is needed permission** ***android.permission. …***
Internet/network access	INTERNET
Read and write contacts information	READ_CONTACTS, WRITE_CONTACTS
Read and write Calendar	READ_CALENDAR, WRITE_CALENDAR
Send, read and write SMS	SEND_SMS, READ_SMS, WRITE_SMS
Using telephony	CALL_PHONE
Accessing external storage	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
Identification of geographical position	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION

If you are using **SDK 23** or higher, then you must check run time permissions!

Android permissions

- API 23 (Marshmallow)
- Normal:
  - Access is granted automatically
  - Examples: Internet, Bluetooth, NFC, etc.
- Dangerous:
  - The access is granted individually by the user
  - Applications control the access at execution
  - Examples: Calendar, Camera, Contacts, SMS, Location, Phone, Storage etc.

Android Basics

Hardware and software requirements:

- **android.hardware.camera**
- **android.hardware.camera.autofocus**
- **android.hardware.camera.flash**
- **android.hardware.nfc**
- **android.hardware.sensor.gyroscope**
- **android.hardware.Bluetooth**
- **android.software.live_wallpaper**
- **android.software.home_screen**

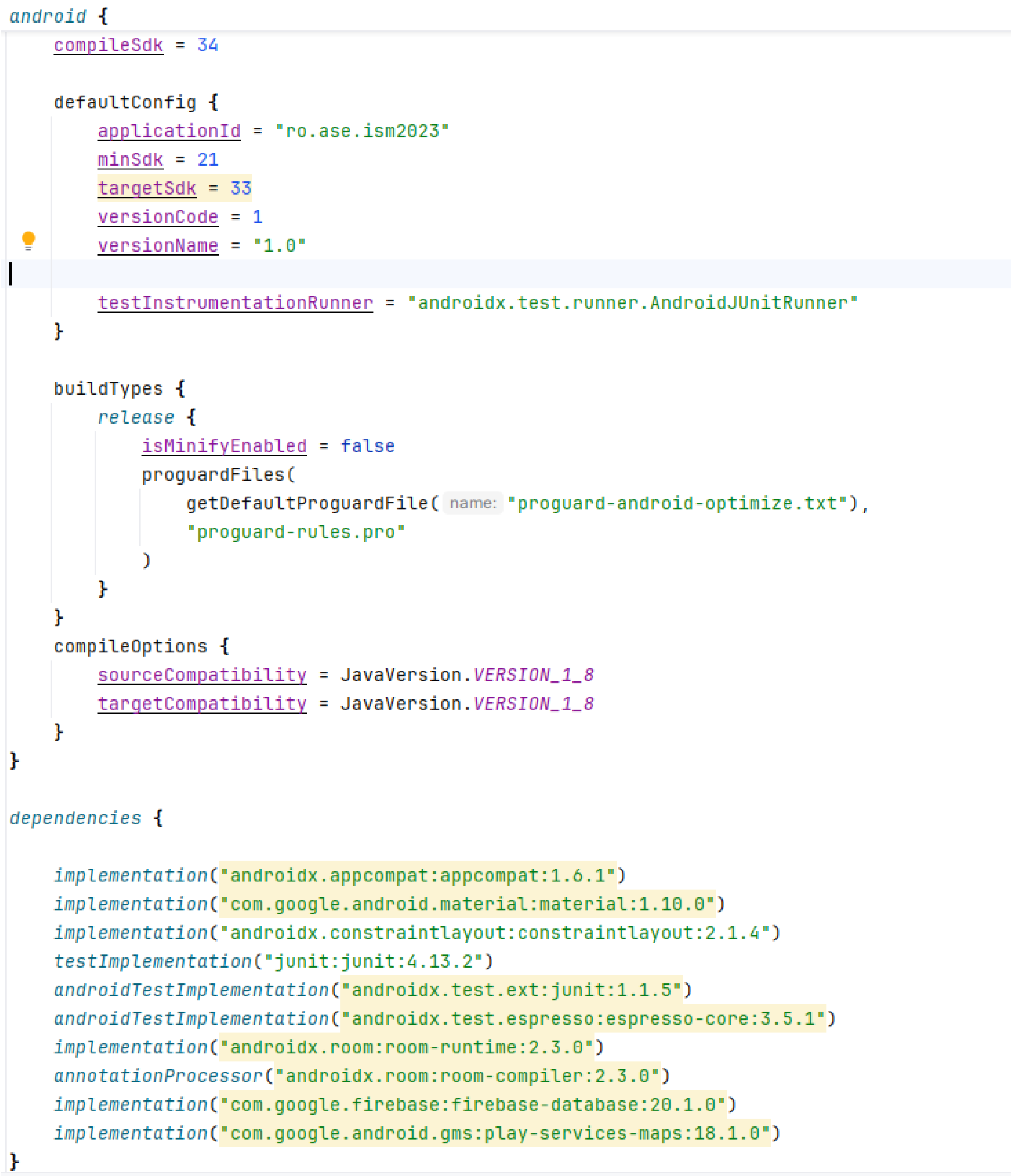
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="ase.pdm.sem1"    android:versionCode="1"  android:versionName="1.0" >
  <uses-permission android:name="android.permission.INTERNET"/>
   <application android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
             android:label="@string/app_name"
      android:theme="@style/AppTheme">
      <activity> ... </activity>
      <service>...</service>
      <provider>...</provider>
      <receiver>...</receiver>
  </application>
</manifest>
```

build.gradle

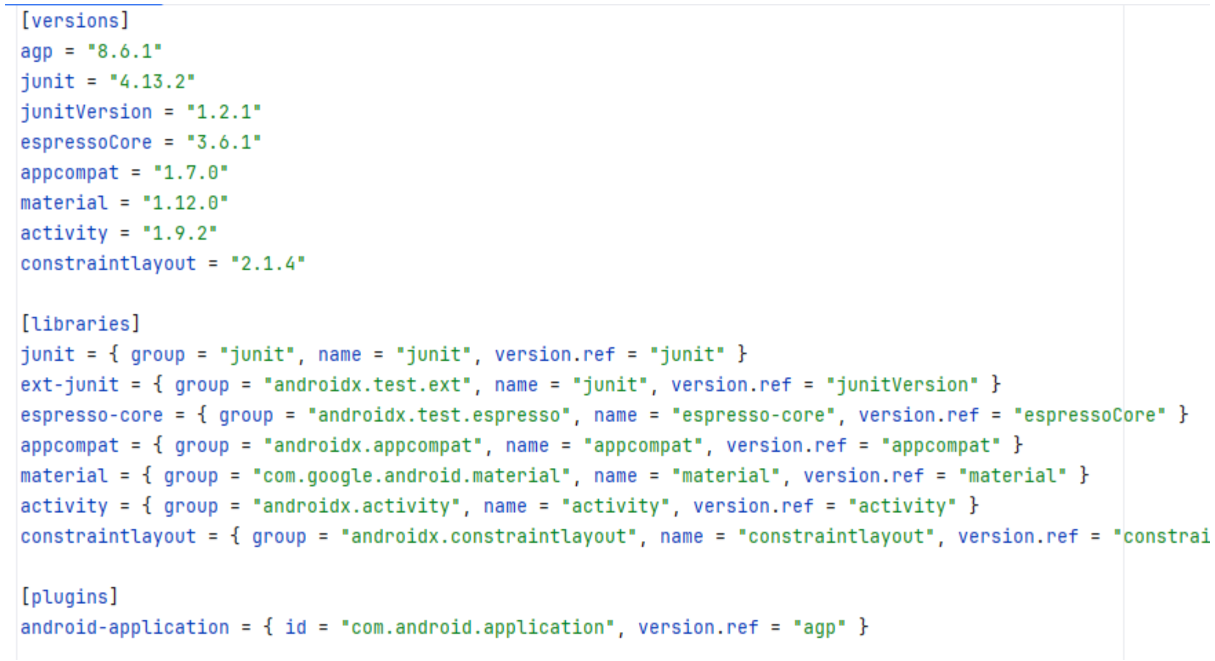
```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.4"

    defaultConfig {
        applicationId "ro.ase.pdm.myapplication"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                                              'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.0.1'
    compile 'com.android.support:design:23.0.1'
}
```

build.gradle

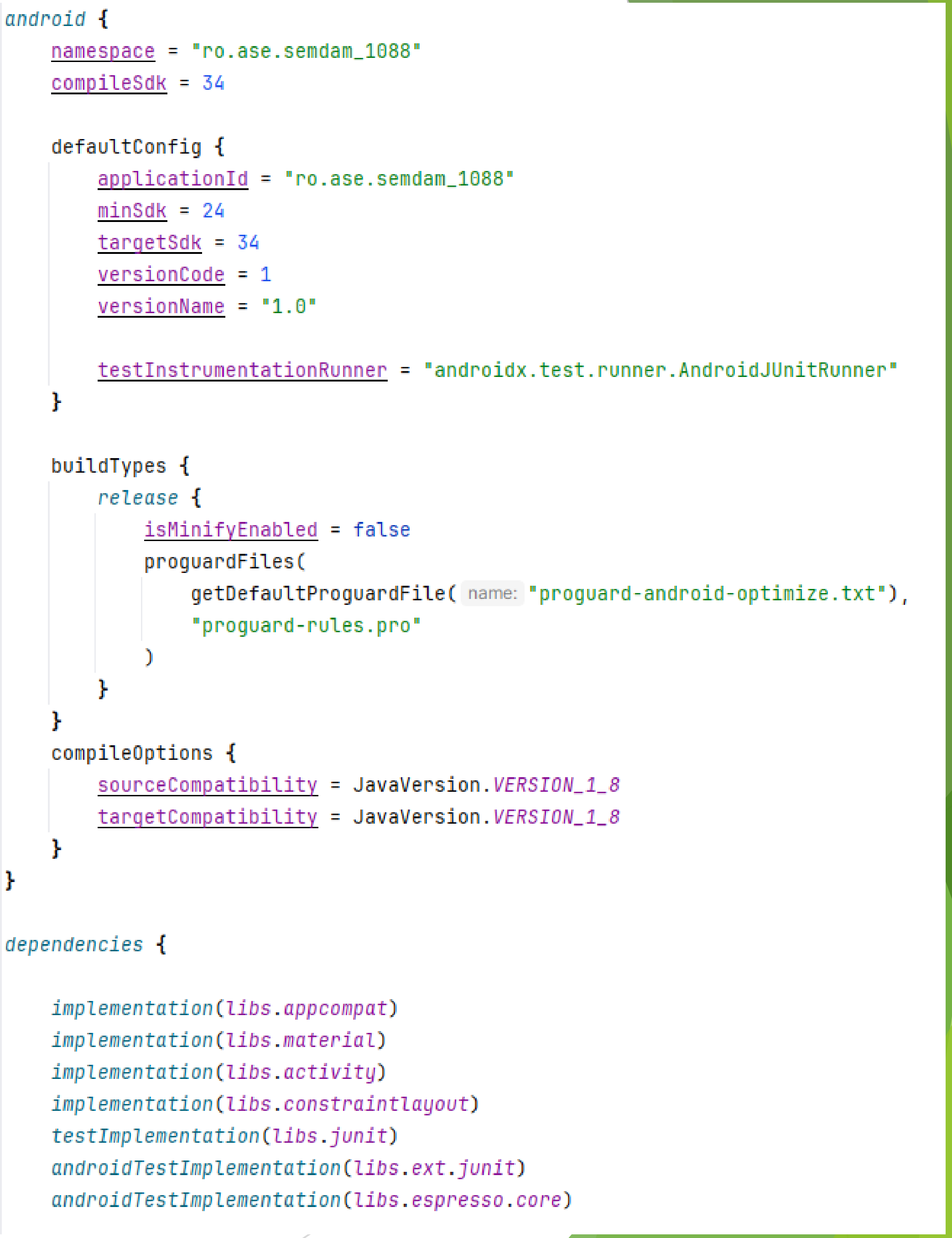


```
android {
    compileSdk = 34

    defaultConfig {
        applicationId = "ro.ase.ism2023"
        minSdk = 21
        targetSdk = 33
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile( name: "proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.10.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
    implementation("androidx.room:room-runtime:2.3.0")
    annotationProcessor("androidx.room:room-compiler:2.3.0")
    implementation("com.google.firebase:firebase-database:20.1.0")
    implementation("com.google.android.gms:play-services-maps:18.1.0")
}
```

build.gradle

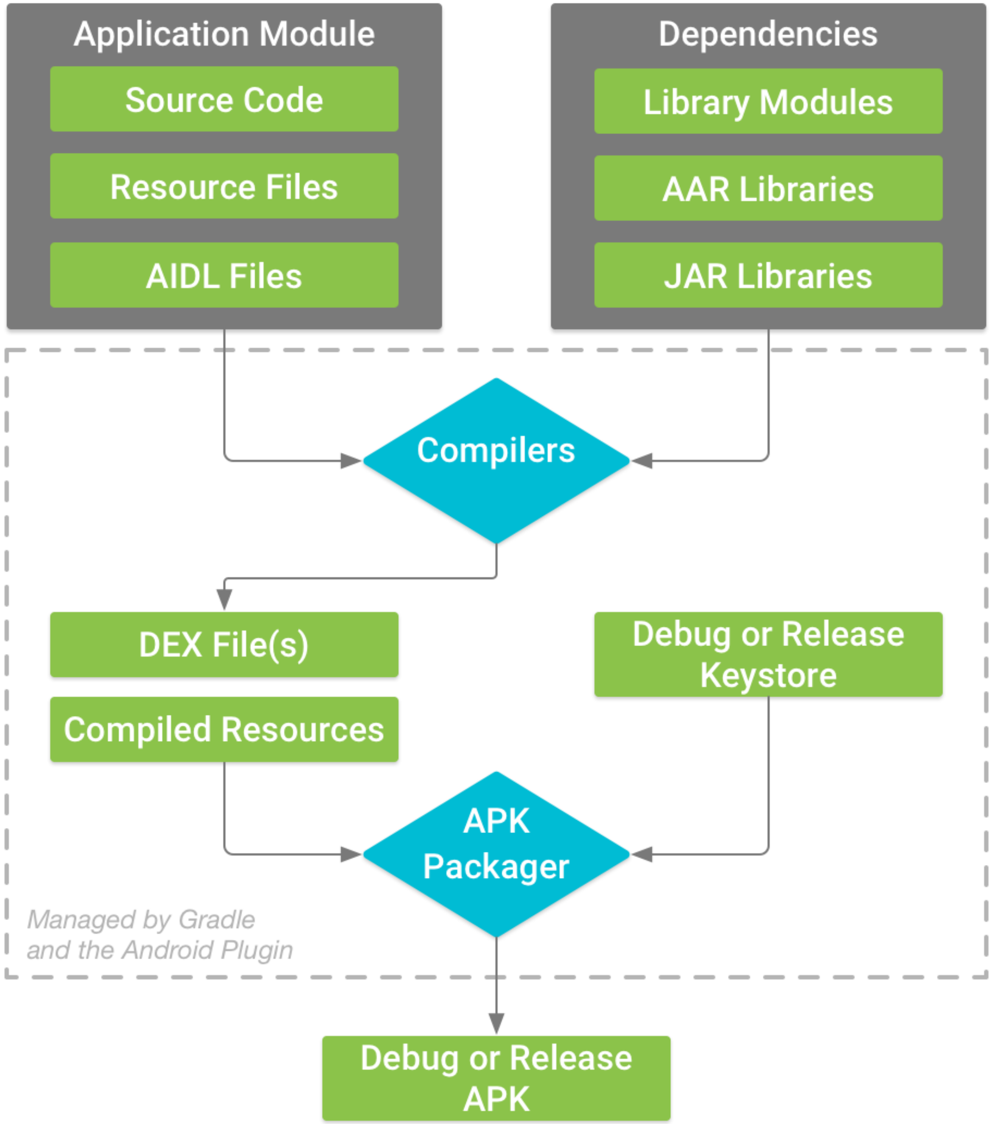


```
[versions]
agp = "8.6.1"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
appcompat = "1.7.0"
material = "1.12.0"
activity = "1.9.2"
constraintlayout = "2.1.4"

[libraries]
junit = { group = "junit", name = "junit", version.ref = "junit" }
ext-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espressoCore" }
appcompat = { group = "androidx.appcompat", name = "appcompat", version.ref = "appcompat" }
material = { group = "com.google.android.material", name = "material", version.ref = "material" }
activity = { group = "androidx.activity", name = "activity", version.ref = "activity" }
constraintlayout = { group = "androidx.constraintlayout", name = "constraintlayout", version.ref = "constrai

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
```



```
android {
    namespace = "ro.ase.semdam_1088"
    compileSdk = 34

    defaultConfig {
        applicationId = "ro.ase.semdam_1088"
        minSdk = 24
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile( name: "proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation(libs.appcompat)
    implementation(libs.material)
    implementation(libs.activity)
    implementation(libs.constraintlayout)
    testImplementation(libs.junit)
    androidTestImplementation(libs.ext.junit)
    androidTestImplementation(libs.espresso.core)
```

The build process



Android Basics

The **Context** class:

- abstract class defined in the package *android.content*
- ensure the access to the application environment
- access to resources
- launching new activities
- access to system services
- access to databases and files

Android Basics

The Application class:

- access to settings and methods of the application
- implicitly generated
- a user class can be created by derivation
- it has an associated context
  - available throughout the application runtime
  - **getApplicationContext()**

Android Basics

Activities:

- Associated to application windows
- An application can have one or more activities
- Stack of activities
  - *Tasks*
- Derived from the base class **android.app.*Activity***
  - Derived from *Context* class (at the top of the hierarchy)
- The context of an activity = this
- Each graphic object refers the context of the belonging activity

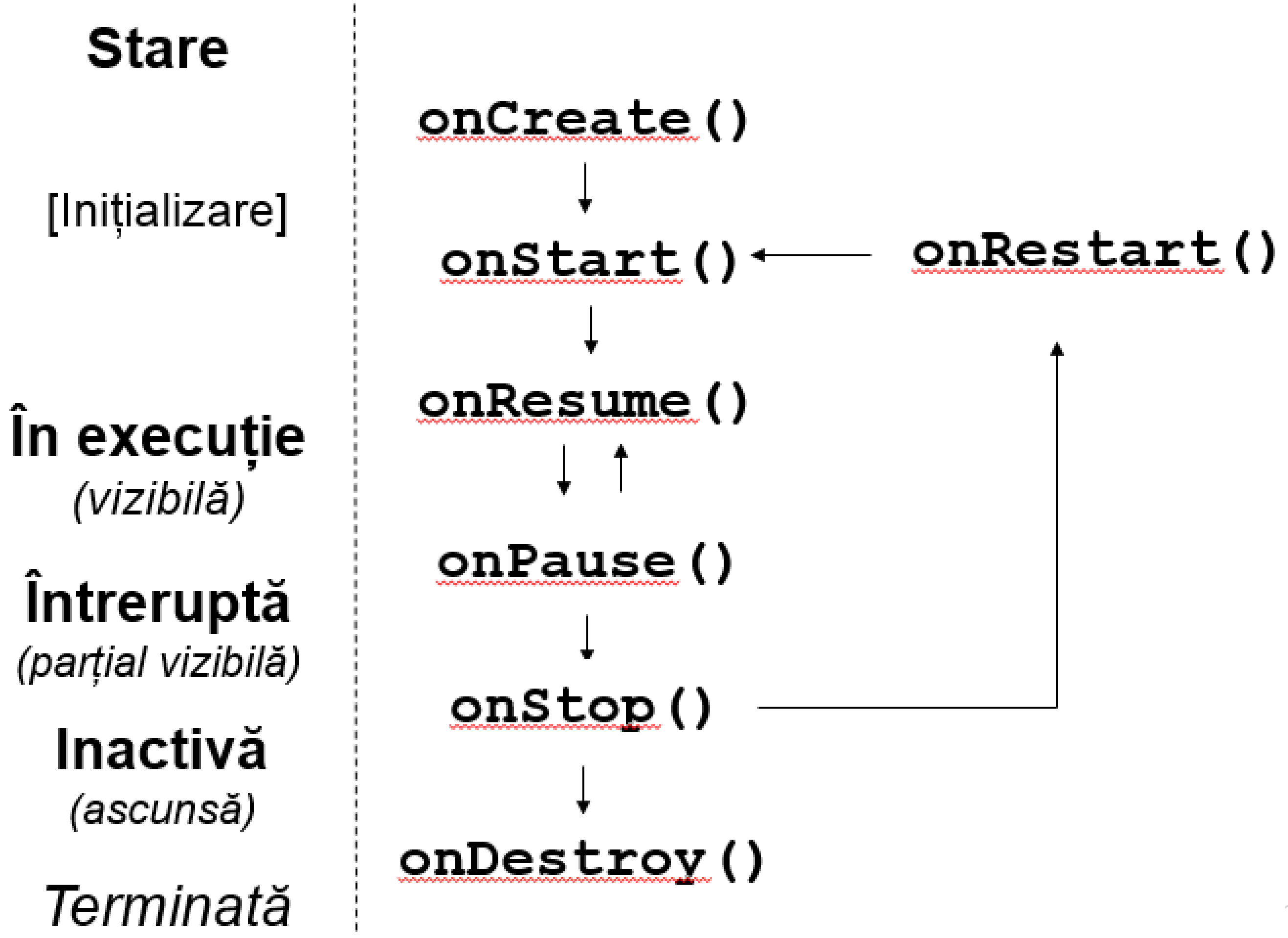
Android Basics

Activities:

- Have an associated windows
  - GUI representation
- They have a life cycle
  - Several states
  - Callback methods
    - Called when passing in a state
    - The possibility of saving the activity state (content, position of visual components, properties etc.)

Android Basics

Activities life cycle:



Stare

[Inițializare]

În execuție
(vizibilă)

Întreruptă
(parțial vizibilă)

Inactivă
(ascunsă)

Terminată

onCreate()

onStart()

onRestart()

onResume()

onPause()

onStop()

onDestroy()

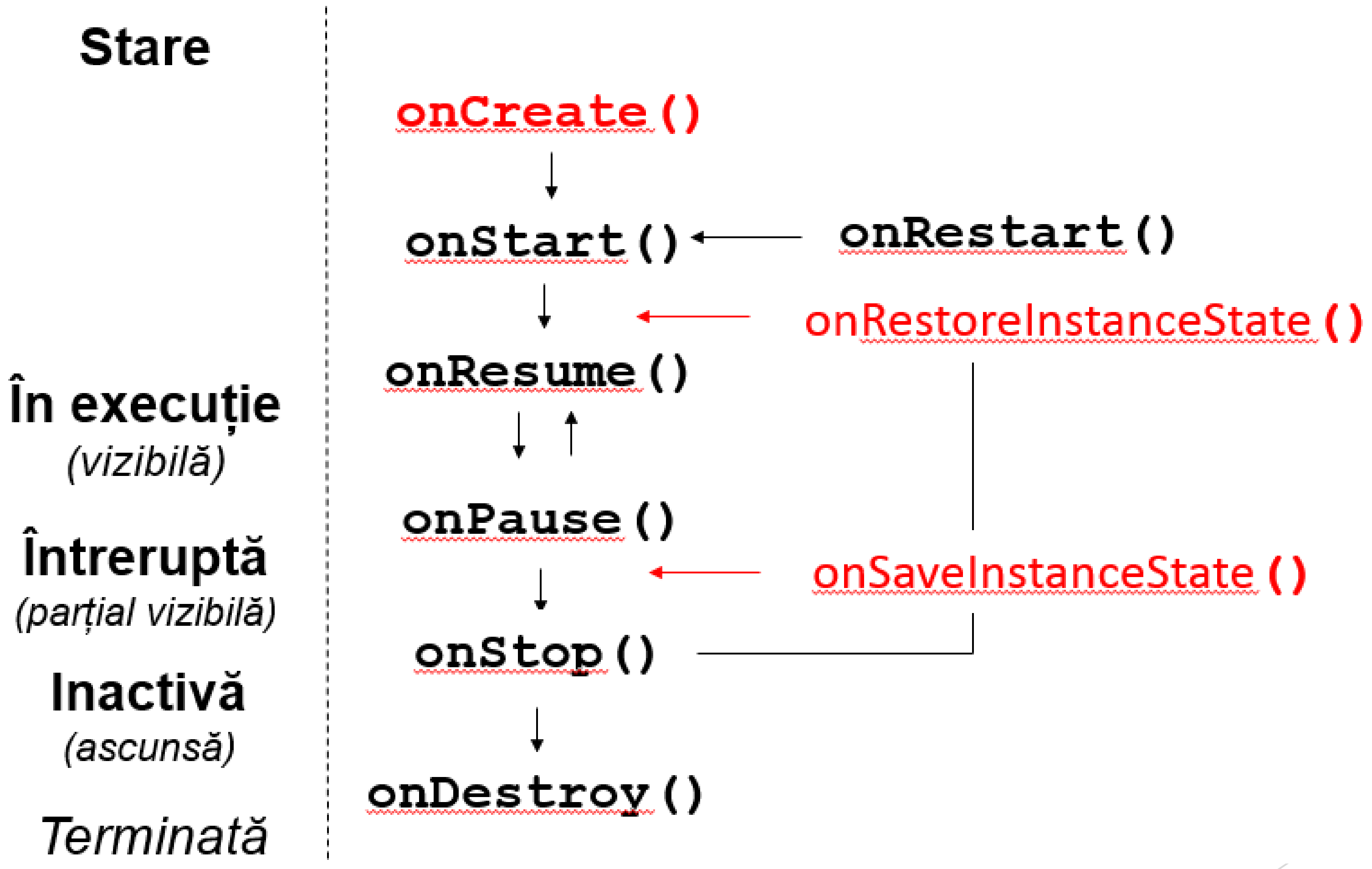
Android Basics

Save/restore the state:

- Save the state
  - onSaveInstanceState(Bundle state)
    - Called at forced exit from the application (when configuration changes)
- Restore the state
  - onRestoreInstanceState(Bundle state) *or*
  - onCreate(Bundle state)
    - *state* can be null!

Android Basics

Activities life cycle:



Stare

onCreate()

onStart()

onRestart()

onRestoreInstanceState()

În execuție
(vizibilă)

onResume()

Întreruptă
(parțial vizibilă)

onPause()

onSaveInstanceState()

Inactivă
(ascunsă)

onStop()

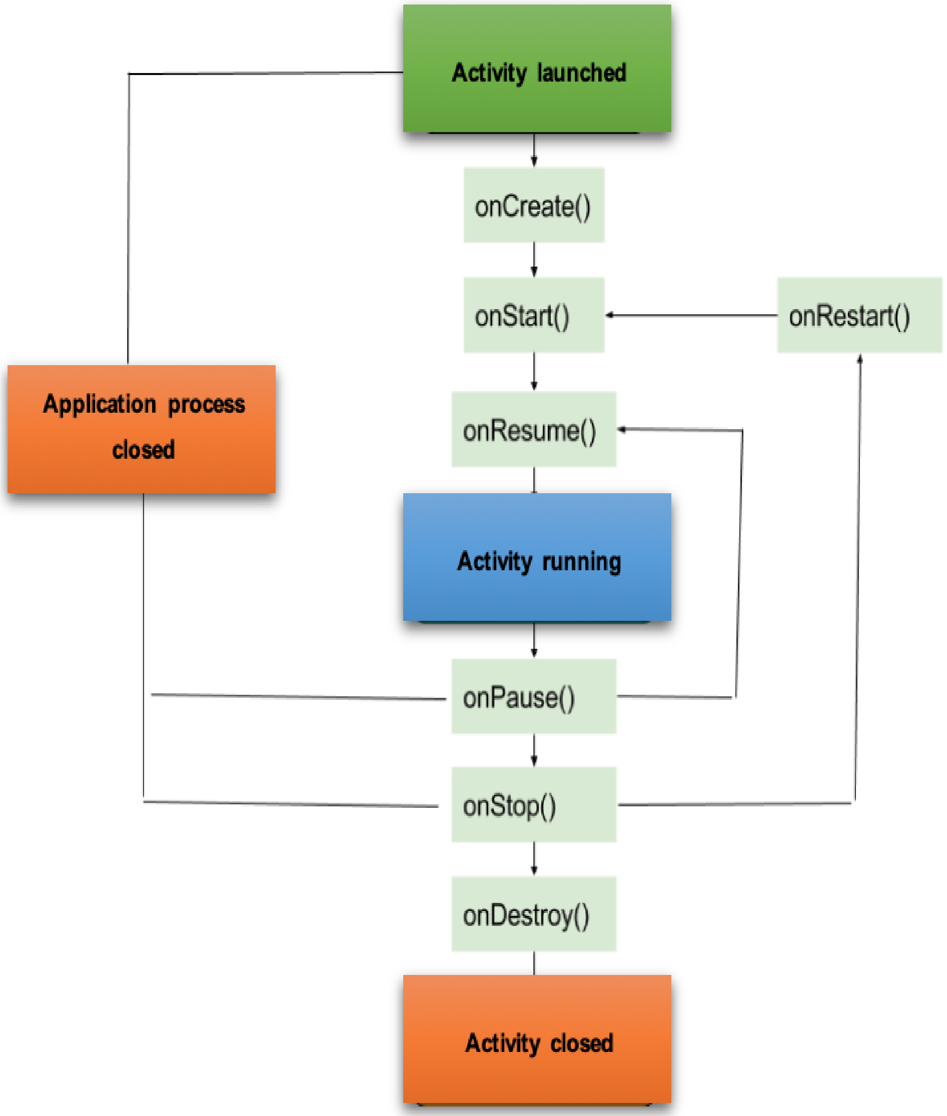
Terminată

onDestroy()

Activities life cycle

- **onCreate()** – is called when the application is created;
- **onStart()** – called before activity is displayed;
- **onResume()** – this is called when the activity becomes visible and the user interacts with it;
- **onPause()** – called when a new activity is brought to the foreground;
- **onStop()** – calling is made when the activity is no longer in use and is not visible;
- **onRestart()** – this is called when the activity returns to the foreground after this function is called onStart ();
- **onDestroy()** – is called when activity is over and destroyed to release memory.

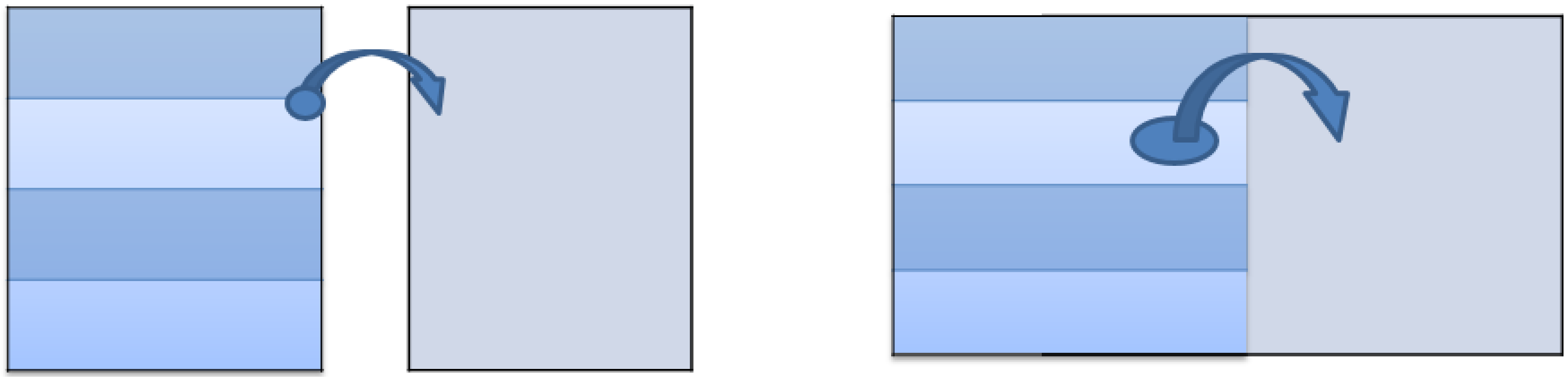
Activities life cycle



Fragments

- **Android Fragment** is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.
- The **FragmentManager** class is responsible to make interaction between fragment objects.
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
- Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.

Fragments



Fragments – lifecycle

Activitate

1. onCreate()
2. onAttachFragment()
3. onStart()
4. onResume()

Fragment

1. onAttach()
2. onCreate()
3. onCreateView()
4. onActivityCreated()
5. onStart()
6. onResume()

Fragments – lifecycle

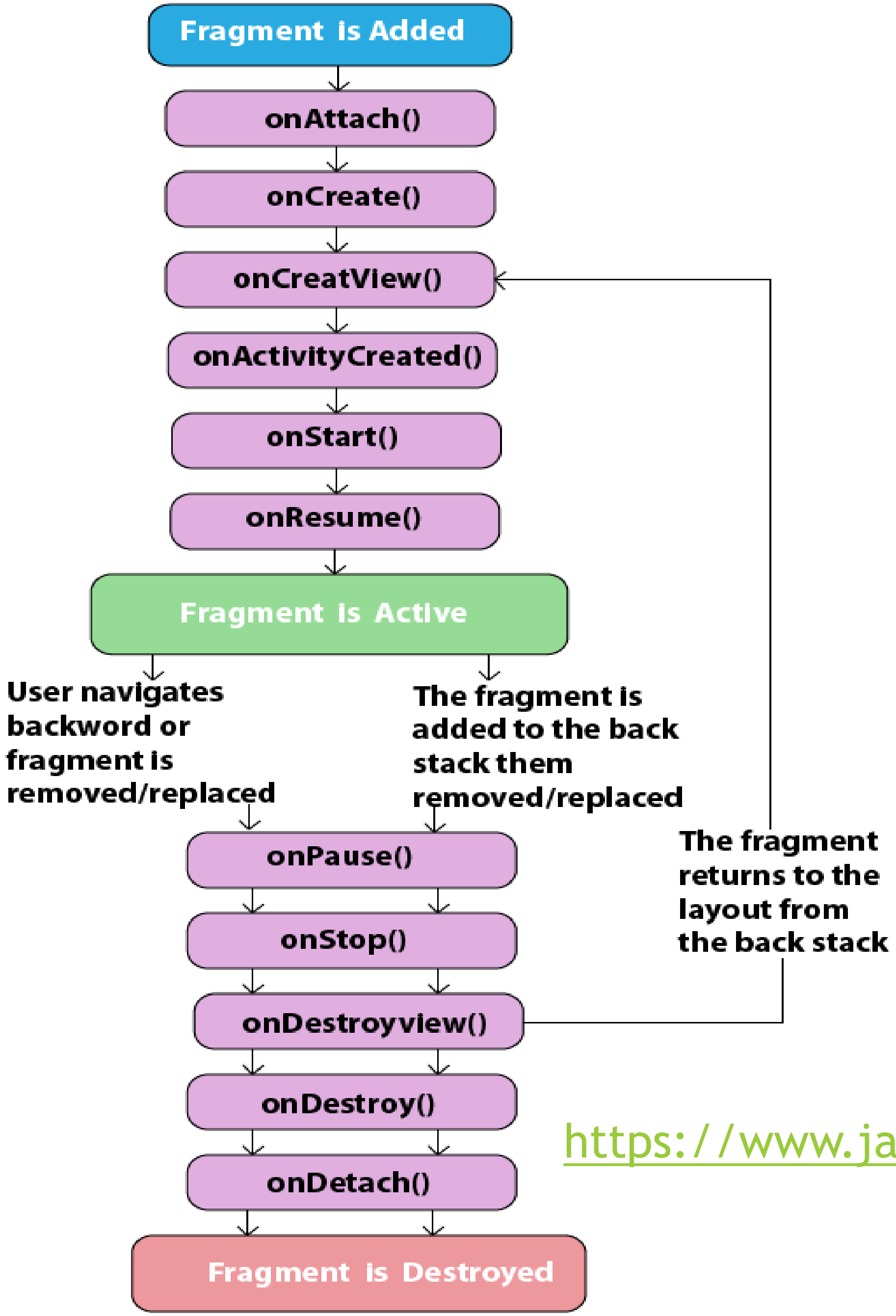
Activitate

1. onPause()
2. onSaveInstanceState()
3. onStop()
4. onDestroy()

Fragment

1. onPause()
2. onSaveInstanceState()
3. onStop()
4. onDestroyView()
5. onDestroy()
6. onDetach()

Fragments

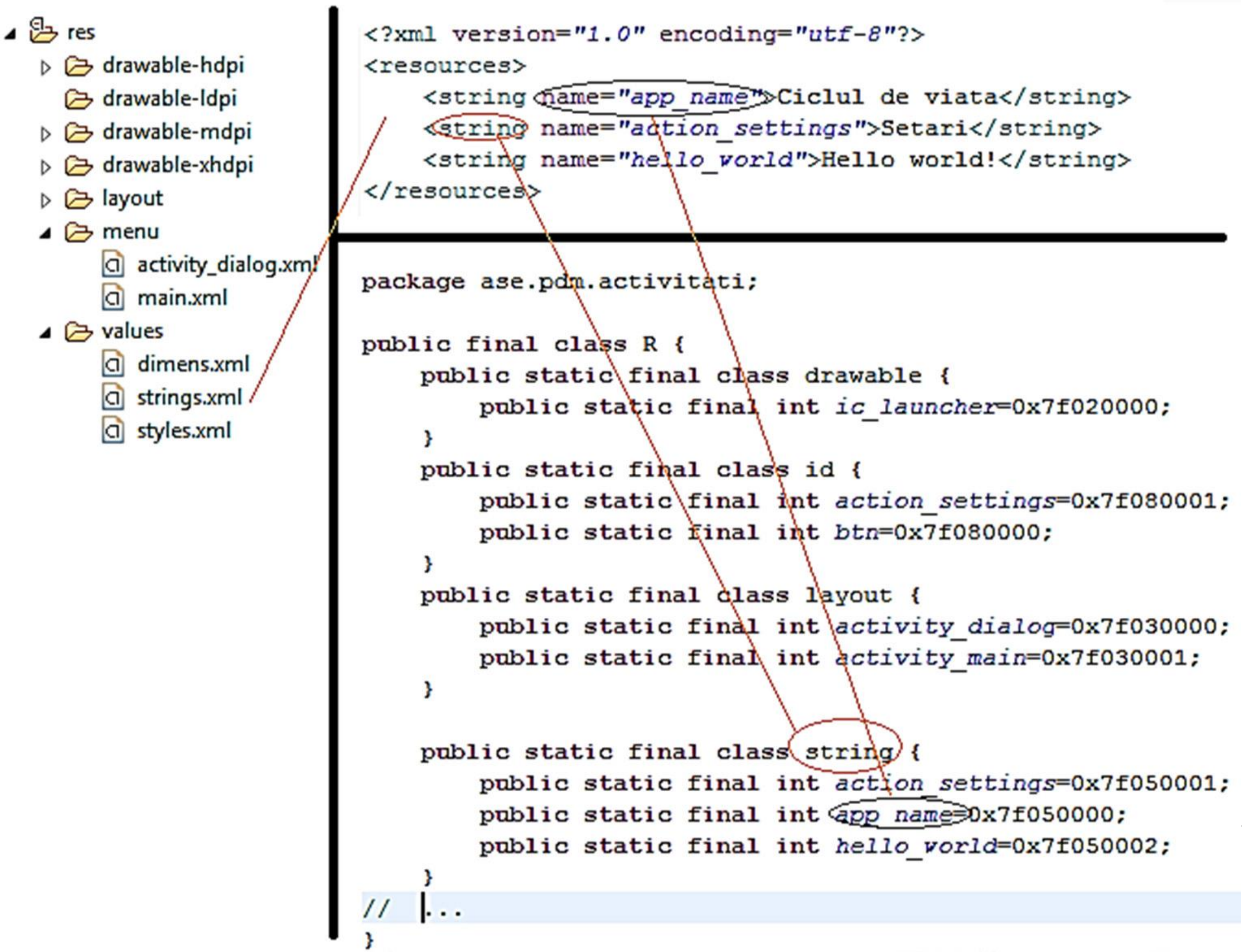


https://www.javatpoint.com/android-fragments

Android User Interface

- The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling.
- The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.
- A typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder

The R class



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Ciclul de viata</string>
    <string name="action_settings">Setari</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

```java
package ase.pdm.activitati;

public final class R {
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080001;
        public static final int btn=0x7f080000;
    }
    public static final class layout {
        public static final int activity_dialog=0x7f030000;
        public static final int activity_main=0x7f030001;
    }

    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050002;
    }
//  ...
}
```

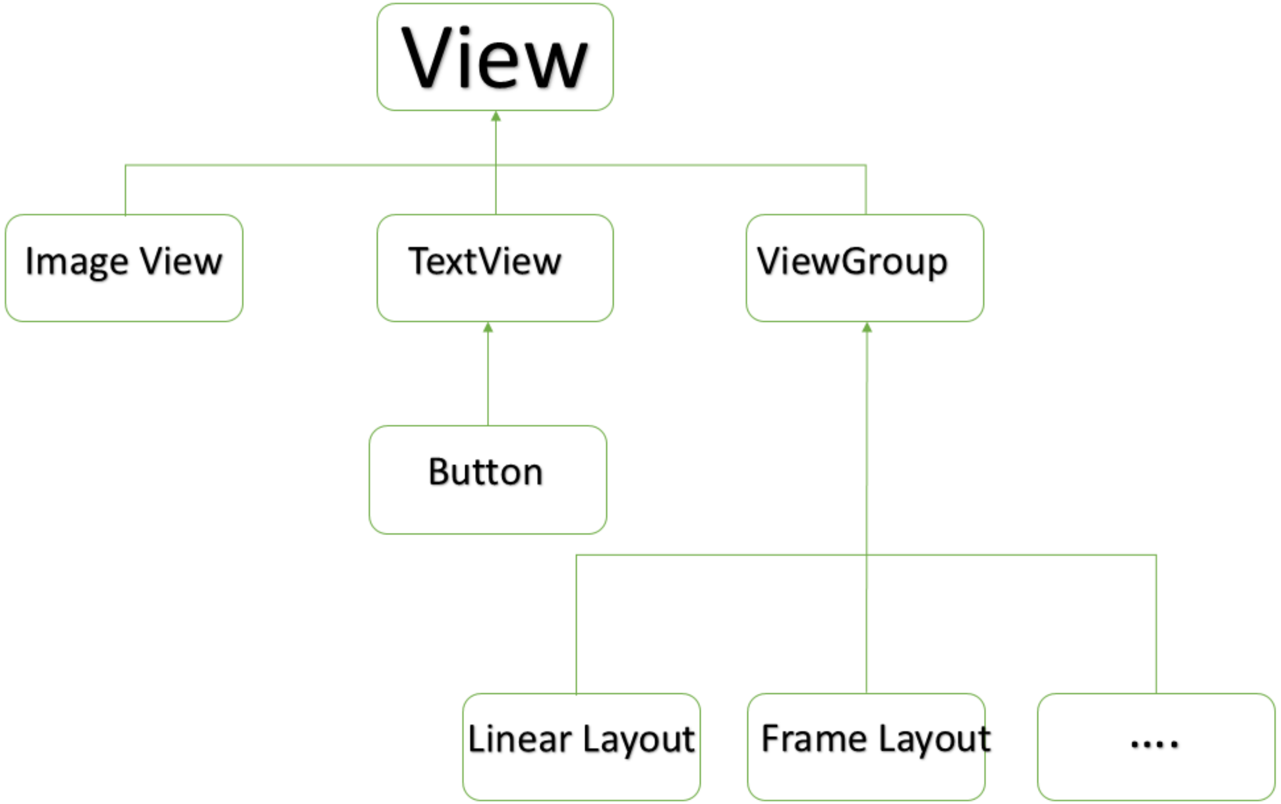
Android User Interface

Android Layout Types:

- **Linear Layout**
- **Relative Layout**
- **Table Layout**
- **Absolute Layout**
- **Frame Layout**
- **List View**
- **Grid View**

Android User Interface

EXAMPLE OF CUSTOM COMPONENTS IN CUSTOM VIEW HIERARCHY



Android User Interface

Creating a simple custom component:

- Instantiate using code inside activity class

```
DateView dateView = new DateView(this);
setContentView(dateView);
```

- Instantiate using Layout XML file

```
<com.example.compoundview.DateView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textColor="#fff" android:textSize="40sp"
android:background="#000"/>
```

Android User Interface

Events are a useful way to collect data about a user's interaction with interactive components of Applications.

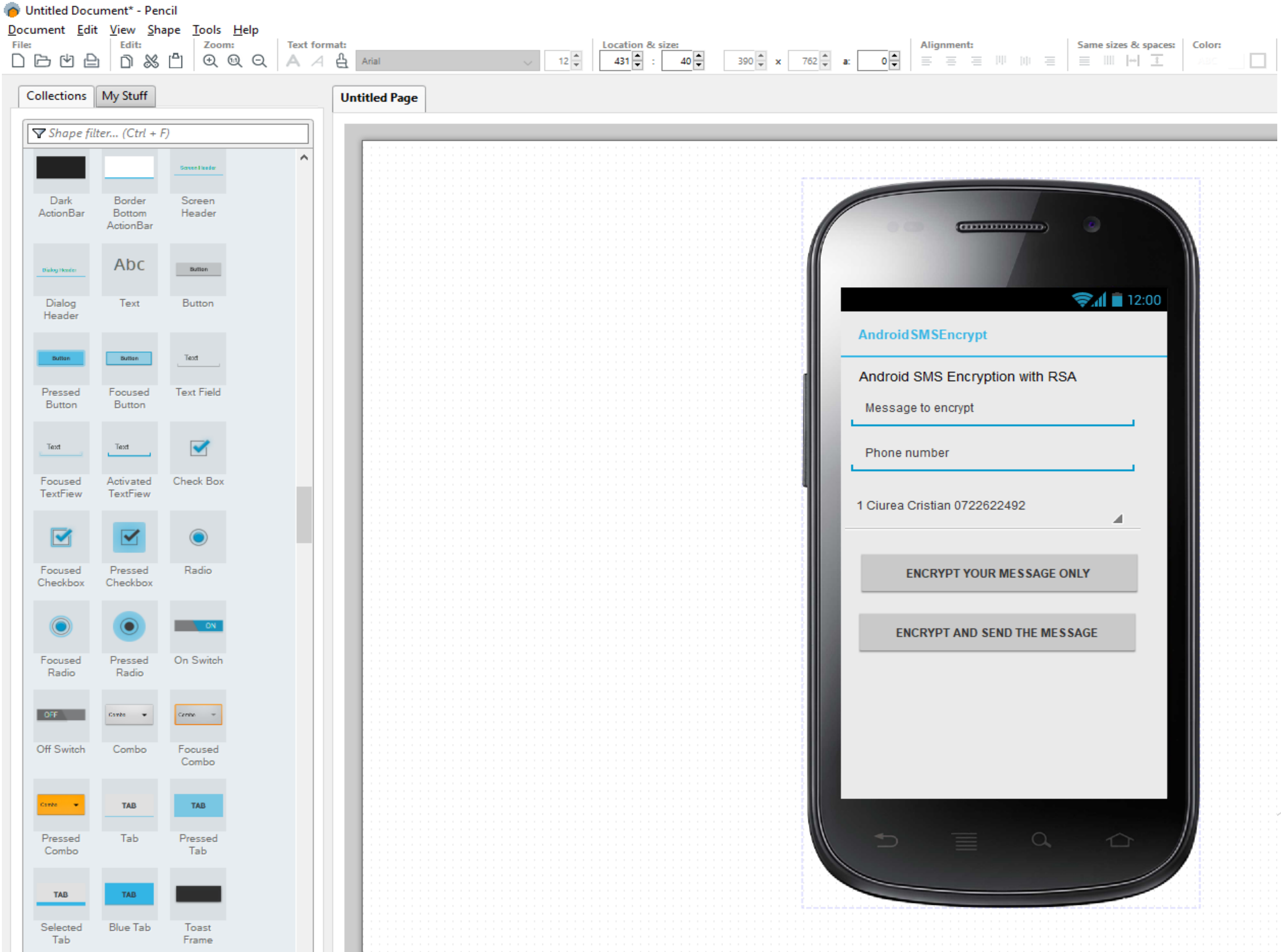
Event Handlers & Event Listeners:

- **onClick()** - OnClickListener()
- **onLongClick()** - OnLongClickListener()
- **onFocusChange()** - OnFocusChangeListener()
- **onKey()** - OnFocusChangeListener()
- **onTouch()** - OnTouchListener()
- **onMenuItemClick()** - OnMenuItemClickListener()
- **onCreateContextMenu()** - onCreateContextMenuItemListener()

Pencil

- Download: https://pencil.evolus.vn/
- Pencil is built for the purpose of providing a free and open-source GUI prototyping tool that people can easily install and use to create mockups in popular desktop platforms.

Pencil


Untitled Document* - Pencil
Document
Edit
View
Shape
Tools
Help
File:
Edit:
Zoom:
Text format:
Arial
12
Location & size:
431
40
390
762
a:
0
Alignment:
Same sizes & spaces:
Color:
Collections
My Stuff
Untitled Page
Shape filter... (Ctrl + F)
Dark ActionBar
Border Bottom ActionBar
Screen Header
Dialog Header
Abc
Text
Button
Pressed Button
Focused Button
Text Field
Focused TextFiew
Activated TextFiew
Check Box
Focused Checkbox
Pressed Checkbox
Radio
Focused Radio
Pressed Radio
ON
On Switch
OFF
Off Switch
Combo
Focused Combo
Pressed Combo
TAB
Tab
Pressed Tab
Selected Tab
Blue Tab
Toast Frame
12:00
AndroidSMSEncrypt
Android SMS Encryption with RSA
Message to encrypt
Phone number
1 Ciurea Cristian 0722622492
ENCRYPT YOUR MESSAGE ONLY
ENCRYPT AND SEND THE MESSAGE

Complex visual components

- **ListView, GridView**
- **Spinner**
- **RecyclerView**
- AutoCompleteTextView
- MultiAutoCompleteTextView
- Gallery
- ListActivity
    - getListView()
    - setListAdapter()
- ListFragment

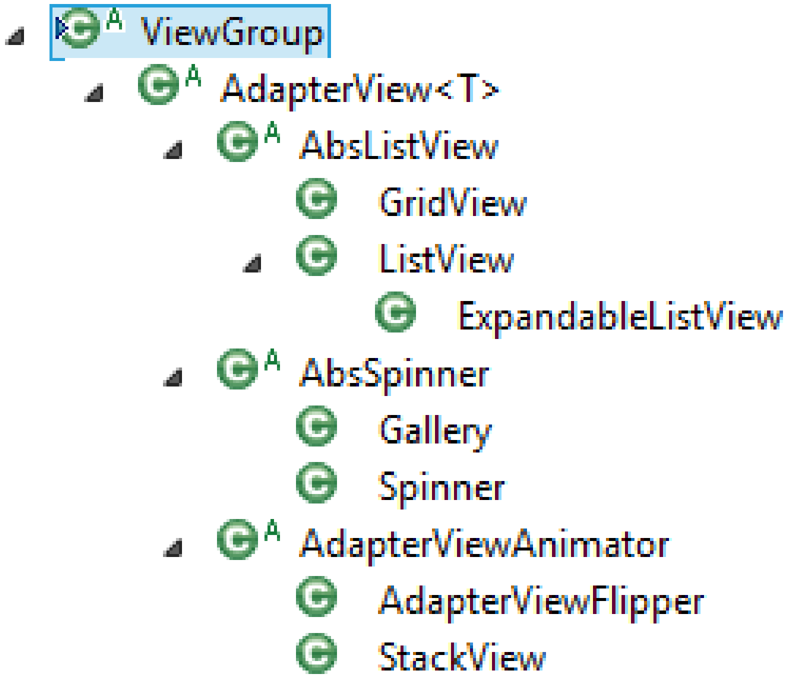
Adapters

- Provides the link between the data source and item list controls
- Access to the data source
- Creates a *View* for each item in the data source

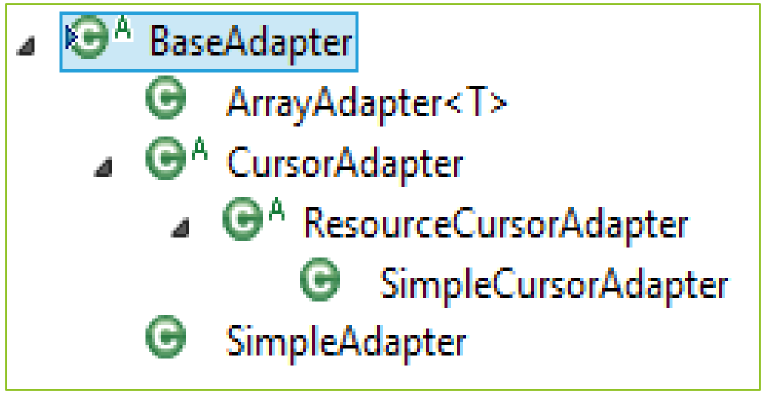
Adapters

- Classes inherited from **BaseAdapter**
  - Implements the interface **Adapter**
- **ArrayAdapter**
- **CursorAdapter**
  - SimpleCursorAdapter
- **SimpleAdapter**

Complex visual components

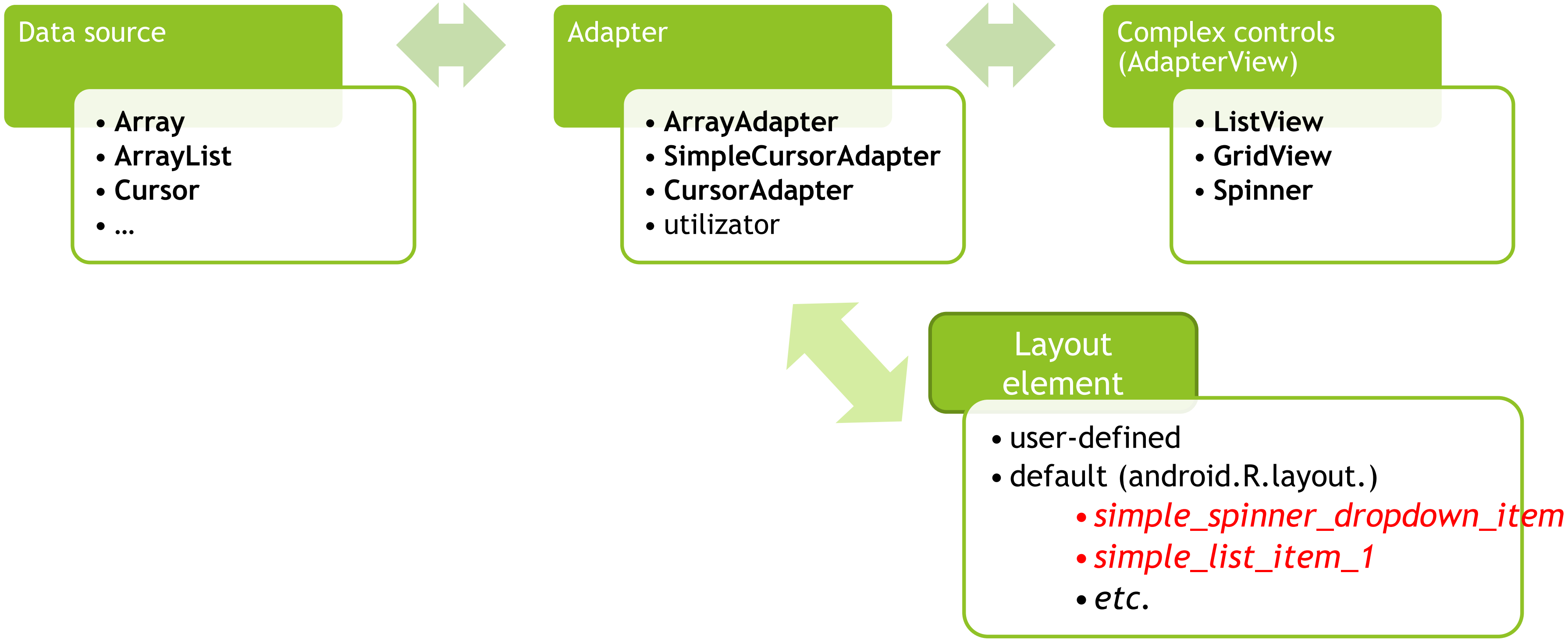


- ViewGroup
  - AdapterView<T>
    - AbsListView
      - GridView
      - ListView
        - ExpandableListView
    - AbsSpinner
      - Gallery
      - Spinner
    - AdapterViewAnimator
      - AdapterViewFlipper
      - StackView



- BaseAdapter
  - ArrayAdapter<T>
  - CursorAdapter
    - ResourceCursorAdapter
      - SimpleCursorAdapter
  - SimpleAdapter

Complex visual components



Data source

- Array
- ArrayList
- Cursor
- …

Adapter

- ArrayAdapter
- SimpleCursorAdapter
- CursorAdapter
- utilizator

Complex controls (AdapterView)

- ListView
- GridView
- Spinner

Layout element

- user-defined
- default (android.R.layout.)
    - *simple_spinner_dropdown_item*
    - *simple_list_item_1*
    - *etc.*

Accessing Network Data

- The Android mechanisms for accessing network data are based on standardized protocols for online resource access.
- An **HttpClient** object is used for remote access by means of POST and GET methods after opening a connection with the **HttpURLConnection** class.
- The main advantage of this type of technique is that it can be used for any type of resources.
- To perform network operations using asynchronous call, the **AsyncTask** class can be used and all the processes implemented in the **doInBackground()** method.

Accessing Network Data

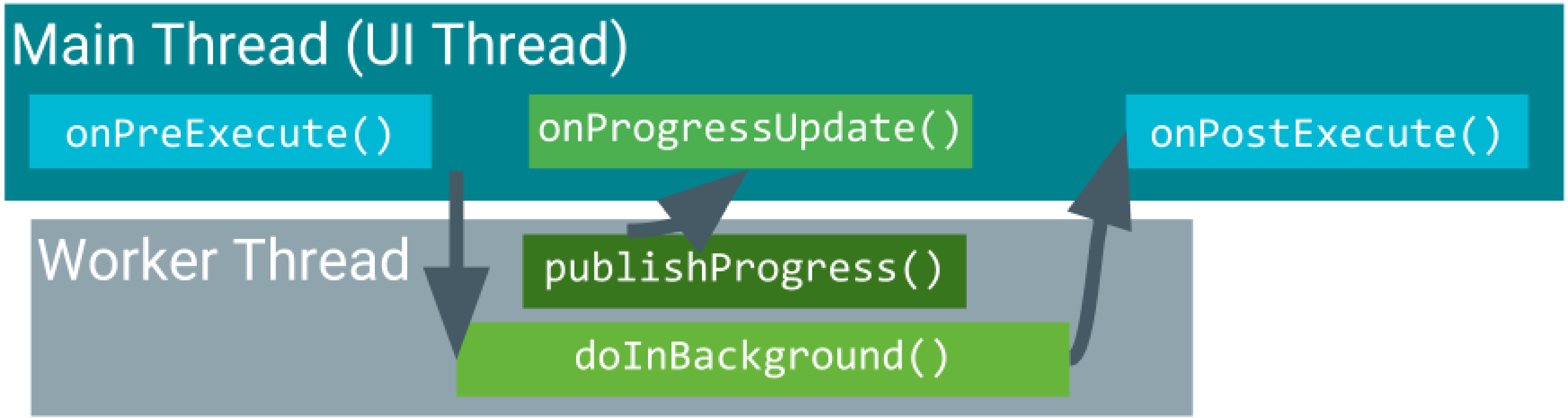
- In order that the Android application can access information about the network in order to make use of remote resources, the **ACCESS_NETWORK_STATE** permission must be added in the **AndroidManifest.xml**.
- <uses-permission android:name="**android.permission.INTERNET**"/>
- <uses-permission android:name="**android.permission.ACCESS_NETWORK_STATE**"/>

Accessing Network Data

In order to access remote resources, the class that extends **AsyncTask** must override three methods:

- onPreExecute() – this method is used to initialize all the variables before executing the request;
- doInBackground() – here is where the actual request for data takes place;
- onPostExecute() – this method implements the application logic after the web request has finished running.

Accessing Network Data



Accessing Network Data

Firul principal

- 1. onPreExecute()
- 4. onProgressUpdate()
- 5. onPostExecute()

Firul secundar

- 2. doInBackground()
  - 3. publishProgress()

Internal File System

In order to save data in an Android application internal file, we need some of the Android API's classes and methods from the **java.io** package:

- **FileOutputStream** class creates an output stream that writes bytes to a file; if the output file exists, it can be replaced or appended to; if it does not exist, a new file will be created;
- **FileInputStream** class creates an input stream used to read bytes from a file;

Internal File System

- **write()** method from the **FileOutputStream** class is equivalent to **write(buffer, 0, buffer.length)** and writes to an internal buffer an array of bytes, starting from the zero offset till the length of the array.
- **getBytes()** method available in all the data wrapper classes returns a new byte array containing the characters of the string encoded using the system's default charset.
- **close()** method closes the stream; implementations of this method should free any resources used by the stream.

Internal File System

The **Context** class provides some methods for interacting with the internal file system:

- **openFileOutput()** method is used to open a private file associated with this context's application package; the method returns a **FileOutputStream** and creates the file if it doesn't already exist.
- **openFileInput()** method is used to obtain a **FileInputStream** and open a file for reading.
- **deleteFile()** method deletes an internal file.
- **fileList()** method generates a String array that contains the names of the application private files.

Android SQLite database

- In order to create a new SQLite database, we must first create a subclass of **SQLiteOpenHelper** and override the **onCreate()** method, where we can execute a SQLite command in order to create tables inside the database.
- **SQLiteOpenHelper** is an abstract class that is used to implement the pattern for creating, opening and upgrading SQLite databases. By implementing the **SQLiteOpenHelper** we hide the logic used to decide if a SQLite database needs to be created or upgraded before it is opened.

Android SQLite database

- In order to write and read from the database, we can call the **getWritableDatabase()** and respectively **getReadableDatabase()** methods.
- We can execute SQLite queries using the **query()** methods of the **SQLiteDatabase**, which accept various query parameters.
- Every SQLite query will return a **Cursor** that points to all the rows found by the query. The **Cursor** is the mechanism with which we can navigate results from a database query and read rows and columns.

Android Room database

Room

- Level of abstraction over SQLite
- ORM (Object Relational Mapping)
- Dependencies
  - implementation "**androidx.room:room-runtime:2.3.0**"
  - annotationProcessor "**androidx.room:room-compiler:2.3.0**"

Android Room database

Objects used:

- Entity
- Data operations
- Database
- Java annotations

Android Room database

Entities

- **@Entity**
  - the Java class associated with a table
  - ***tableName***
    - Custom table name
- **@PrimaryKey** – primary key field
  - **autogenerate**
- **@ColumnInfo**
  - ***name***
    - Custom field name in the table
- **@Ignore** – the field is not included in the table

Android Room database

Relationships between entities

- **foreignKey** property from @Entity
- The **@ForeignKey** value
- Attributes
  - **entity**
    - The class associated with the parent entity
  - **parentColumns**
    - The column names in the parent table
  - **childColumns**
    - The names of the columns in the child table
  - **onDelete** , **onUpdate**
    - CASCADE

Android Room database

Data operations

- **@Dao**
  - Defines the interface for data operations
- Includes methods for selection, insertion, modification
- The methods must be annotated
- Operations must be performed asynchronously

Android Room database

Data operations

- **@Query**
- **@Insert**
- **@Update**
- **@Delete**

Firebase database

- Sign in with a Google Account
- https://**firebase.google.com**/
- Project management console
- https://**console.firebase.google.com**/
- Integration into Android applications
  - Manually
  - Android Studio | Tools | Firebase

Firebase database

Classes and interfaces

- implementation '**com.google.firebase:firebase-database:20.1.0**'
- **FirebaseDatabase**
  - The database
- **DatabaseReference**
  - References to database items
- **DataSnapshot**
  - Copies of data in memory
- **ValueEventListener**
- **ChildEventListener**

Firebase database

Database initialization

- The class **FirebaseDatabase**
- Static method **getInstance()**
- Example
  - **FirebaseDatabase database = FirebaseDatabase.*getInstance*();**

Firebase database

Data retrieval:

- **ValueEventListener**
  - onDataChange(DataSnapshot)
  - onCancelled(DatabaseError)
- Association (DatabaseReference)
  - addValueEventListener()
  - addListenerForSingleValueEvent()
- Remove association
  - removeEventListener()

Android Advanced Concepts

- A **notification** is a message you can display to the user outside of your application's normal UI.
- Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

Android Advanced Concepts

Create and Send Notifications:

- **Step 1 - Create Notification Builder**: create a notification builder using *NotificationCompat.Builder.build()*
- **Step 2 - Setting Notification Properties**: once you have **Builder** object, you can set its Notification properties – **setSmallIcon(), setContentTitle(), setContentText()**
- **Step 3 - Attach Actions**: an action allows users to go directly from the notification to an **Activity** in your application; the action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application
- **Step 4 - Issue the notification**: pass the Notification object to the system by calling **NotificationManager.notify()** to send your notification

Android Advanced Concepts

- To **send an email** from your application, you don’t have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc.
- You will use **ACTION_SEND** action to launch an email client installed on your Android device.
- To send an email you need to specify **mailto:** as URI using **setData()** method and data type will be to **text/plain** using **setType()** method.
- Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client.

Android Advanced Concepts

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);
emailIntent.putExtra(Intent.EXTRA_CC, CC);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");
```

Android Advanced Concepts

To **send an SMS** from your application, you can use SmsManager API or devices Built-in SMS application to send SMS's.

SmsManager API:

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application:

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Of course, both need **SEND_SMS permission.**

Android Security

- Android has security features built into the operating system that significantly reduce the frequency and impact of application security issues.
- Android operating system is based on Linux kernel so that applications isolation, the file system and security rules are Linux specific.
- The system is designed so you can typically build your apps with default system and file permissions and avoid difficult decisions about security.

Android Security

- On Android, encryption is a process through which user data is encoded and hidden on a mobile device with the help of a user chosen key.
- In addition to providing data isolation, supporting full-filesystem encryption, and providing secure communications channels, Android provides a wide array of algorithms for protecting data using cryptography.

Android Security

- Google included in Android from API level 1 a package called **javax.crypto** with all the needed classes and interfaces that can be used to implement encryption and decryption algorithms or key agreements in applications that are cryptographic oriented.
- The package **javax.crypto** includes classes for symmetric key cryptography (AES, DES), public keys encryption (RSA, DH) and message digests (MD5, SHA-1 etc.).

Android Security

Some of the core security features that help you build secure apps include:

- The Android Application Sandbox, which isolates your app data and code execution from other apps.
- An application framework with robust implementations of common security functionality such as cryptography, permissions, and secure inter-process communication (IPC).
- An encrypted filesystem that can be enabled to protect data on lost or stolen devices.
- User-granted permissions to restrict access to system features and user data.
- Application-defined permissions to control application data on a per-app basis.

Android Security

Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

- Protect application and user data;
- Protect system resources (including the network);
- Provide application isolation from the system, other applications, and from the user.

Android Security

To achieve these objectives, Android provides these key security features:

- Robust security at the OS level through the Linux kernel;
- Mandatory application sandbox for all applications;
- Secure inter-process communication;
- Application signing;
- Application-defined and user-granted permissions.

Android Security

- Generally, you should define as few permissions as possible while satisfying your security requirements. Many free applications require full Internet access for advertising.
- Creating a new permission is relatively uncommon for most applications, because the system-defined permissions cover many situations. Where appropriate, perform access checks using existing permissions.
- Malicious applications can install a *BroadcastReceiver* for incoming messages in order to get access to private information. As the messages arrive, they are received by the malicious application and processed.

Android Security

There are 3 specific attack points:

- at the **phone level**:
  - Browser, email: Phishing, Clickjacking, Drive-by Downloading;
  - Calls and messages: Baseband Attacks, SMiShing;
  - Application-level attacks: Stress-based attacks, Attacks based on manipulation of settings, Runtime injection attacks, Permissions-based assassinations;
  - Operating System Attacks: lack of mobile device security, Android rooting, Installed software.
- at **network level**: Packet sniffing, Man-in-the-Middle, Session Pirates, DNS infection, SSL false certificates
- at the **data level**: Faulty Server Configuration, Cross-site Request Forget (CSRF), Poor input validation, SQL Injection, Execution of Operations Command Levels.

Android Security

There are several actions that can be taken at the level of the development of mobile applications:

- correct implementation of file permissions;
- very careful implementation of the intentions;
- checking activities;
- careful use of Broadcast transmission;
- protecting the application-specific services;
- avoiding sniffing intent;
- carefully implement Content Providers;
- tracking the practice guide when using WebView;
- avoid storing cached images as a result of taking a photo with your device's camera;
- avoid storing cached interface elements;
- signing the **.apk** file.

References

- P. Pocatilu, *Programarea dispozitivelor mobile*, Editura ASE, 2012.
- C. Boja, C. Ciurea, M. Doinea – *Android Mobile Applications: A Practical Development Guide*, Editura ASE, 2015, ISBN 978-606-34-0033-9, 418 pg.
- P. Pocatilu, I. Ivan ș.a., *Programarea aplicațiilor Android*, Editura, ASE, 2015.
- S. Komatineni, D. MacLean, *Pro Android 4*, Apress, 2012
- R. Meier, *Professional Android 4 Application Development*, Wiley, 2012
- http://developer.android.com