

# Java & la piattaforma Android

*Corso di Laurea in Ingegneria Informatica*

Anno accademico 2021/2022

**Prof. ENRICO DENTI**

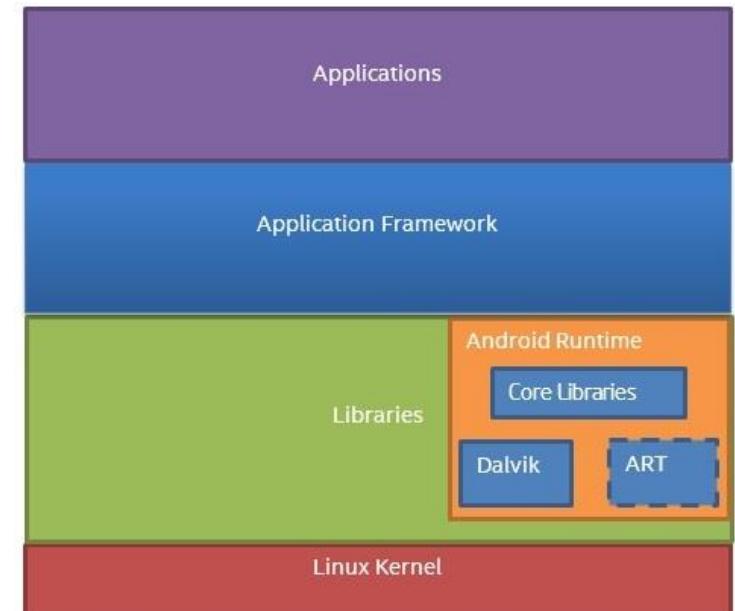
*Dipartimento di Informatica – Scienza e Ingegneria (DISI)*



# ANDROID

Android è un sistema operativo per smartphone & tablet

- basato sul kernel di Linux
- *adotta Java come linguaggio di programmazione*
- *incorpora quasi tutto il framework Java*
- Android ≤ 4.2:  
*macchina virtuale Dalvik*
  - diversa da JVM standard
  - architettura a registri
  - formato ottimizzato  
*Dalvik Executable (.dex)*  
anziché bytecode (.class)
- *Android 5+: macchina virtuale ART*
  - disponibile già in Android 4.4  
a titolo sperimentale





# ANDROID PROGRAMMING

---

- Fin dall'inizio, Android *ha adottato Java* come linguaggio di programmazione
  - stesso linguaggio: lo conoscete già !
  - "quasi" le stesse librerie (tranne la grafica.. simile però a JavaFX)
- Più recentemente, spinta verso la *migrazione a Kotlin*
  - linguaggio più recente, «sponsorizzato» dall'azienda produttrice del principale tool di sviluppo, *Android studio*
  - appropriato perché simile a/interoperabile con Java, ma con
    - niente più tipi primitivi → *everything is an object*
    - enfasi sull'approccio funzionale → *vere lambda expression*



# ANDROID & JAVA

---

- Cosa significa che "*Android adotta(va) Java*"?
  - stesso linguaggio: lo conoscete già
  - "quasi" le stesse librerie (tranne la grafica.. simile però a JavaFX)
  - MA diversa macchina virtuale nell'infrastruttura, quindi *diverso formato del codice oggetto e diverso modello di sviluppo*
    - NON PIÙ bytecode (.class) MA *Dalvik/ELF Executable* (.dex / .elf)
    - NON PIÙ archivi jar (.jar) MA *application packages* (.apk)
    - NON PIÙ solo javac, MA anche altri strumenti
  - organizzazione delle applicazioni come in JavaFX
    - NON PIÙ main classico: il flusso non è più il tuo diretto controllo, è l'infrastruttura a chiamare i tuoi metodi *quando opportuno*
    - NON PIÙ console con stdout, stdin: le app sono *solo grafiche*
    - struttura generale imposta: le classi-app ereditano da una classe base



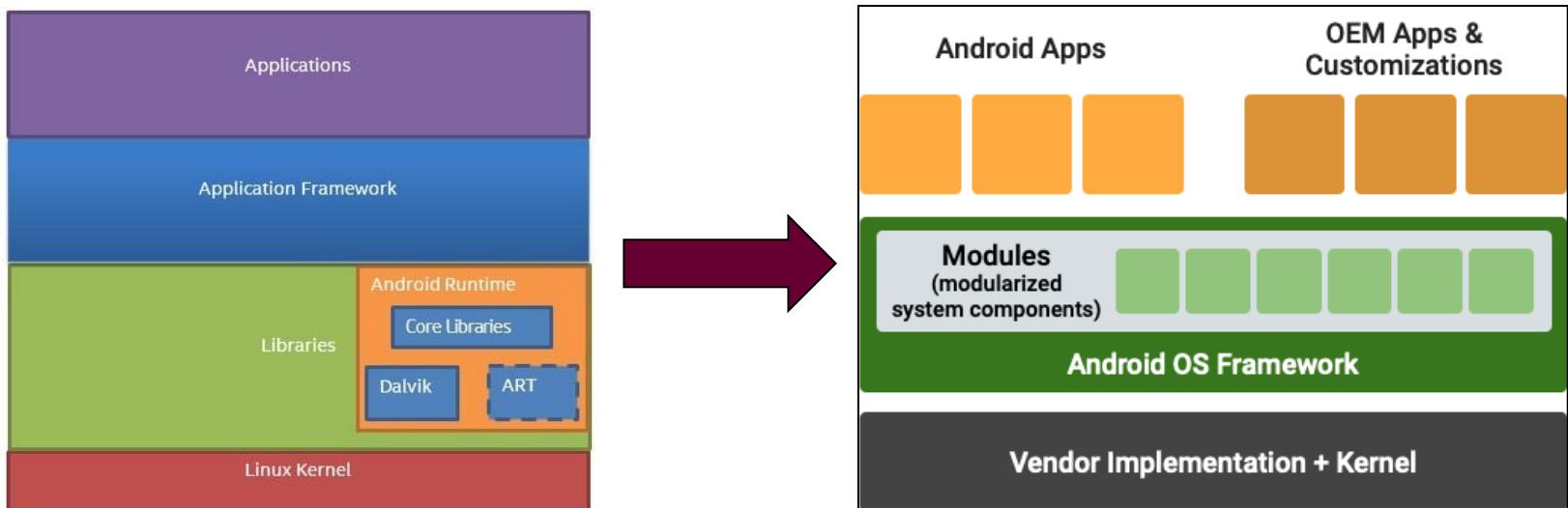
# DA DALVIK AD ART...

---

- **Dalvik** ha accompagnato Android fin dagli esordi
  - architettura a registri, formato ottimizzato..
  - ..ma prestazioni non del tutto soddisfacenti
  - ( e qualche problema di licenza fra Google e Oracle ☺ )
- **ART (Android Run Time)** mirava a prestazioni migliori
  - da Android 5, è la Virtual Machine di default
  - prestazioni migliori: sostituisce il compilatore JIT (just-in-time) di Dalvik con un compilatore AOT (ahead-of-time) che pre-compila tutto il codice durante l'installazione dell'app anziché dopo
    - *at install time, ART compiles apps using the on-device dex2oat tool, which accepts dex files and generates a compiled app executable for the target device*
    - infatti, durante la migrazione da Android 4.4 a 5 diceva «sto ottimizzando le app»: in realtà, le stava convertendo tutte al nuovo formato
  - miglior gestione energia (durata batteria +20% ~) ..
  - ... vs installazione più lenta e maggior occupazione memoria (+10-20%)

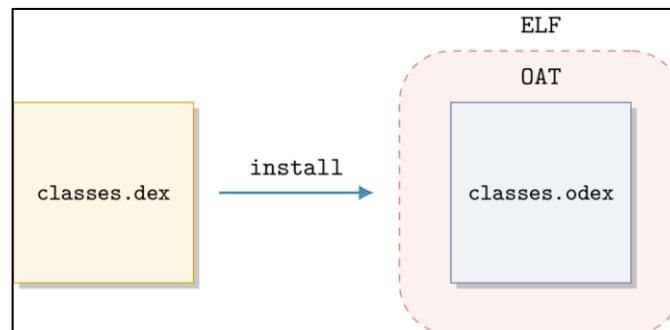
# ... E OLTRE

- Da Android 10 ART supporta la modularizzazione
  - nuovo formato APEX in aggiunta al precedente APK per le app
  - l'architettura modulare consente di aggiornare i componenti del sistema in modo snello, senza influire sulle implementazioni di fornitori di livello inferiore o su app e servizi di livello superiore
- Da Android 12, *ART* è diventato un *modulo*



# ANDROID: VM & BUILD

- La macchina virtuale Android (Dalvik/ART) è *a registri*, anziché a stack come la JVM classica
- Processo di build
  - i sorgenti Java sono compilati normalmente da `javac` → `.class`
  - Android trasforma il bytecode in codice Dalvik → `.dex` files
  - poi, durante l'installazione di un'app (APK) il file DEX è «ottimizzato», ossia *ricompilato in codice nativo* per ART → file `.odex` / `.oat` in formato ELF (è il formato di Linux)





# STRUMENTI

---

*Che strumenti servono per sviluppare in Android?*

- in origine: Eclipse + alcuni strumenti aggiuntivi
  - Android SDK + ADT plugin per Eclipse
  - MA configurazione complicatissima (un incubo..)
- poi: Android Developer Tool (ADT)
  - versione speciale di Eclipse con quei tool già inclusi
  - dal 2014, non più supportata
- da ormai vari anni, **Android Studio**
  - basato su IntelliJ IDEA (uno strumento alternativo a Eclipse)
  - molto completo, moderno (ambiente simile a, ma diverso da, Eclipse)
  - supporta sia **Java** sia **Kotlin**



# ANDROID STUDIO



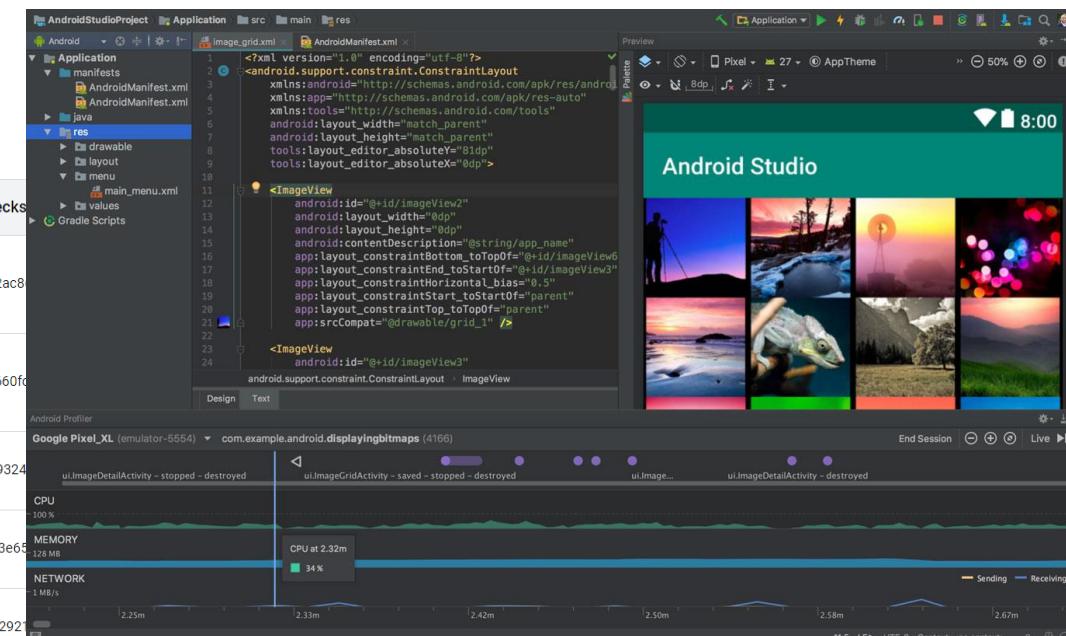
Android Studio provides the fastest tools for building apps on every type of Android device.

[Download Android Studio](#)

Android Studio Chipmunk | 2021.2.1 Patch 1 for Windows 64-bit (929 MB)

Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	<a href="#">android-studio-2021.2.1.15-windows.exe</a> Recommended	929 MB	d99d2b24e232ac8...
	<a href="#">android-studio-2021.2.1.15-windows.zip</a> No .exe installer	940 MB	a992449e546660fd...
Mac (64-bit)	<a href="#">android-studio-2021.2.1.15-mac.dmg</a>	1017 MB	fcaf413951119324...
Mac (64-bit, ARM)	<a href="#">android-studio-2021.2.1.15-mac_arm.dmg</a>	1014 MB	ce1fb8ba48c93e65...
Linux (64-bit)	<a href="#">android-studio-2021.2.1.15-linux.tar.gz</a>	964 MB	0018e0dfc0dd292...
Chrome OS	<a href="#">android-studio-2021.2.1.15-cros.deb</a>	817 MB	1300f2e48734ad57b9598f1f620ab2c72436b7fa60ced2d96266dadaf3078489

See the [Android Studio release notes](#). More downloads are available in the [download archives](#). For Android Emulator downloads, see the [Emulator download archives](#).



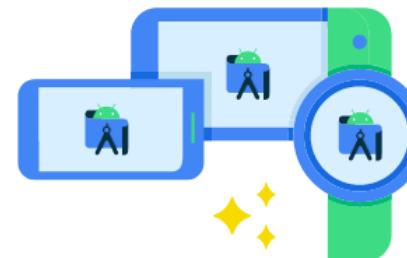


# ANDROID STUDIO

## Everything you need to build on Android

Android Studio is Android's official IDE. It is purpose-built for Android to accelerate your development and help you build the highest-quality apps for every Android device.

[See the release notes](#)



## Code and iterate faster than ever

Based on IntelliJ IDEA, Android Studio provides the fastest possible turnaround on your coding and running workflow.

### Su questa pagina

- [Code and iterate faster than ever](#)
- [Code with confidence](#)
- [Configure builds without limits](#)
- [Create rich and connected apps](#)
- [Eliminate tiresome tasks](#)



# ANDROID STUDIO (1)

## Meet Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on [IntelliJ IDEA](#). On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich [emulator](#)
- A unified environment where you can [develop for all Android devices](#)
- Instant Run to push changes to your running app without building a new APK
- Code templates and [GitHub integration](#) to help you build common app features and import sample code
- [Extensive testing tools and frameworks](#)
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for [Google Cloud Platform](#), making it easy to integrate Google Cloud Messaging and App Engine

This page provides an introduction to basic Android Studio features. For a summary of the latest changes, see [Android Studio Release Notes](#).

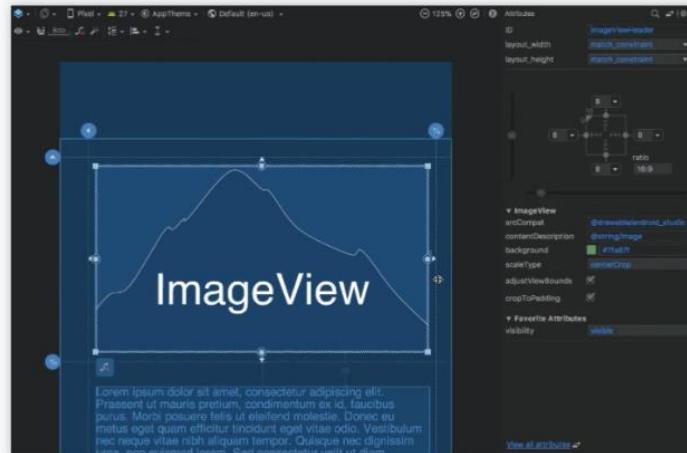


# ANDROID STUDIO (2)

## FEATURE

### Visual layout editor

Create complex layouts with `ConstraintLayout` by adding constraints from each view to other views and guidelines. Then preview your layout on any screen size by selecting one of various device configurations or by simply resizing the preview window.



[MORE ABOUT THE LAYOUT EDITOR](#)

## FEATURE

### Intelligent code editor

Write better code, work faster, and be more productive with an intelligent code editor that provides code completion for Kotlin, Java, and C/C++ languages.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_scrolling)
    val toolbar = findViewById<View>(R.id.toolbar) as Toolbar
    setSupportActionBar(toolbar)

    val fab = findViewById<View>(R.id.fab) as FloatingActionButton
    val emailButton = findViewById<View>(R.id.emailAction) as ImageButton
    val androidButton = findViewById<View>(R.id.androidAction) as ImageButton

    //Icons
    emailButton.setImageResource()
    androidButton.setImageResource(R.drawable.ic_adb_24dp)
    fab.setImageResource(R.drawable.ic_star_24dp)

    //Colors
    val accent = ContextCompat.getColor(context: this, R.color.colorAccent)
    val primary = ContextCompat.getColor(context: this, R.color.colorPrimary)
    val primaryDark = ContextCompat.getColor(context: this, R.color.colorPrimaryDark)

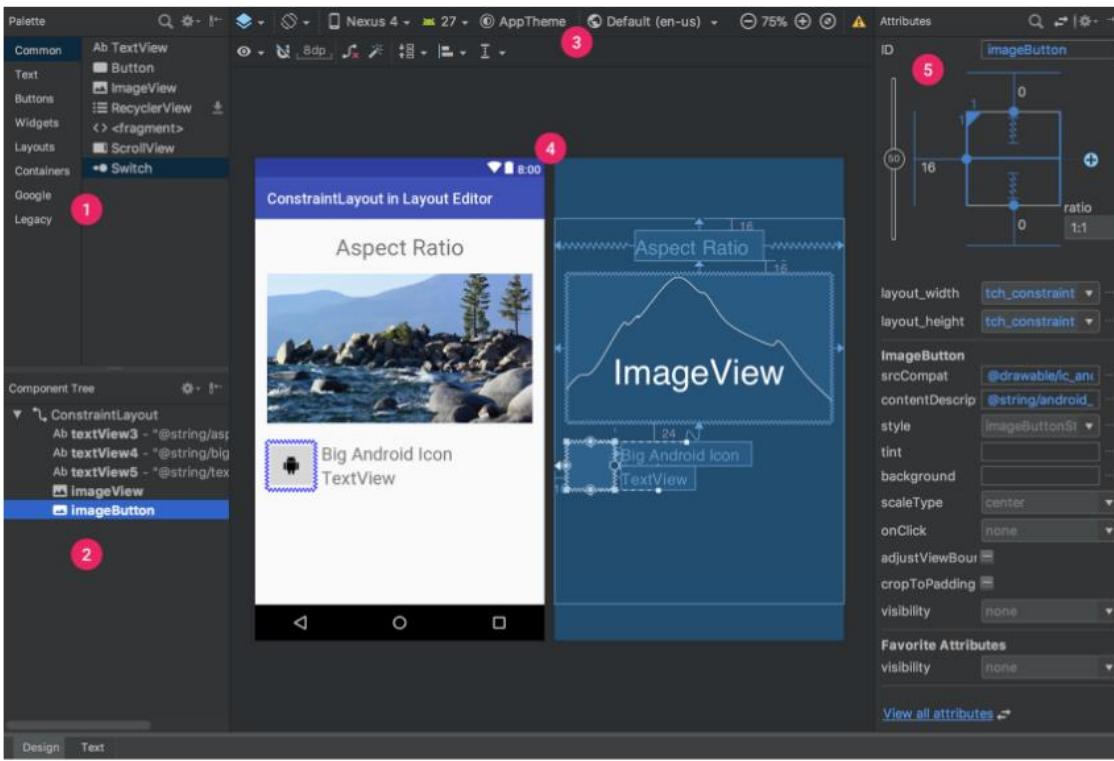
    fab.setOnClickListener { view ->
        Snackbar.make(
            view,
            text: "Email Contact",
            Snackbar.LENGTH_LONG
        )
    }
}
```

[MORE ABOUT THE EDITOR](#)



# ANDROID STUDIO (3)

- 1 **Palette:** List of views and view groups that you can drag into your layout.
- 2 **Component Tree:** View hierarchy for your layout.
- 3 **Toolbar:** Buttons to configure your layout appearance in the editor and to change some layout attributes.
- 4 **Design editor:** Layout in Design or Blueprint view, or both.
- 5 **Attributes:** Controls for the selected view's attributes.



Layout Editor  
Supporto professionale  
per lo sviluppo della GUI

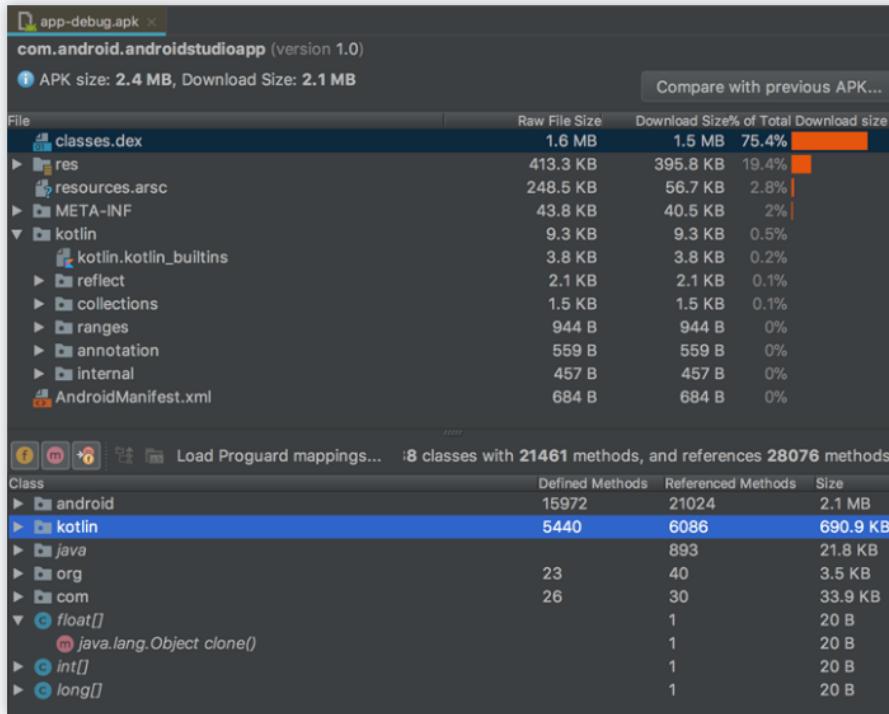


# ANDROID STUDIO (4)

## FEATURE

### APK Analyzer

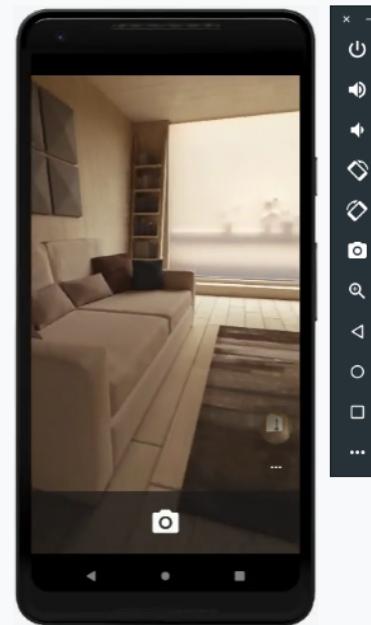
Find opportunities to reduce your Android app size by inspecting the contents of your app APK file, even if it wasn't built with Android Studio. Inspect the manifest file, resources, and DEX files. Compare two APKs to see how your app size changed between app versions.



## FEATURE

### Fast emulator

Install and run your apps faster than with a physical device and simulate different configurations and features, including ARCore, Google's platform for building augmented reality experiences.





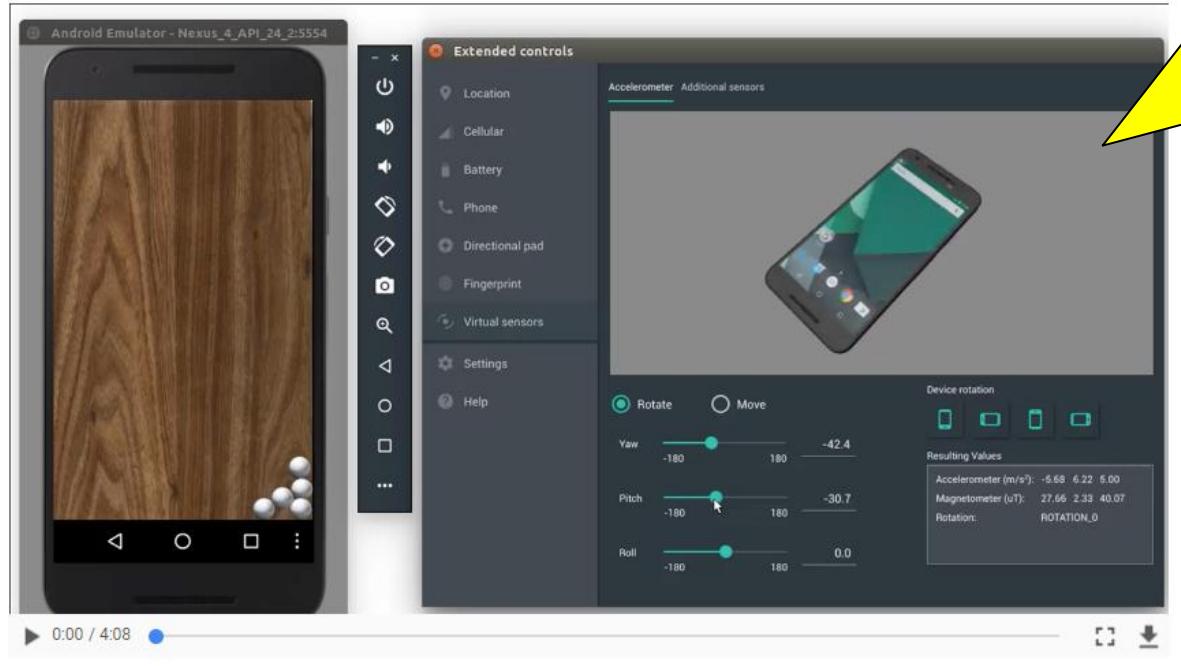
# ANDROID STUDIO (5)

## Run Apps on the Android Emulator

The Android Emulator simulates various Android phone, tablet, Wear OS, and Android TV devices on your computer. It comes with predefined configurations for popular device types and can transfer data faster than a device connected over USB.

The Android Emulator provides almost all the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.

Watch the following video for an overview of some emulator features.



Nuovo scenario:  
si sviluppa su PC un'app  
da eseguire altrove

Serve un EMULATORE  
*altamente configurabile*  
del sistema target, che  
possa simulare *ogni elemento e ogni aspetto!*



# ANDROID STUDIO (6)

## FEATURE

### Flexible build system

Powered by Gradle, Android Studio's build system allows you to customize your build to generate multiple build variants for different devices from a single project.

```
build.gradle
Gradle project sync in progress...
apply plugin: 'com.android.application'
ext.densities = ['mdpi', 'hdpi', 'xhdpi', 'xxhdpi']
ext.abiList = ['armeabi', 'armeabi-v7a', 'x86']

android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.tools

    defaultConfig {
        applicationId "com.google.android.apps.santatracker"
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion
        versionCode 40120000
        versionName "4.0.12"
        vectorDrawables.useSupportLibrary = true
        testInstrumentationRunner 'android.support.test.runner.AndroidJUnitRunner'
    }

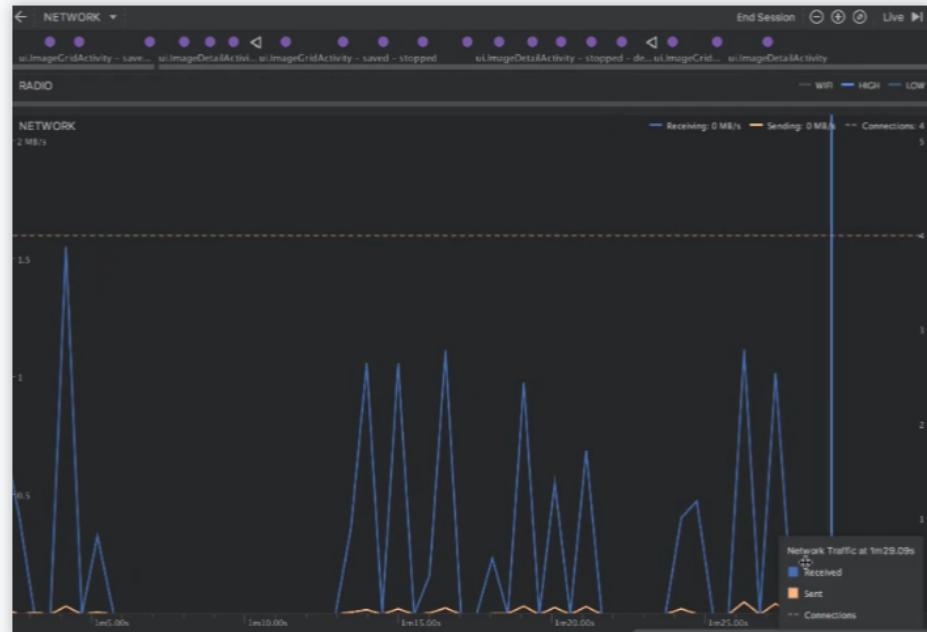
    buildTypes {
        debug {
            applicationIdSuffix ".debug"
            versionNameSuffix "-debug"
            // Enabling multidex support.
            multiDexEnabled true
        }
    }
    release {...}
}
```

MORE ABOUT THE BUILD TOOLS

## FEATURE

### Realtime profilers

The built-in profiling tools provide realtime statistics for your app's CPU, memory, and network activity. Identify performance bottlenecks by recording method traces, inspecting the heap and allocations, and see incoming and outgoing network payloads.



MORE ABOUT THE PROFILERS



# ANDROID STUDIO (7)

## Project structure

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the `AndroidManifest.xml` file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

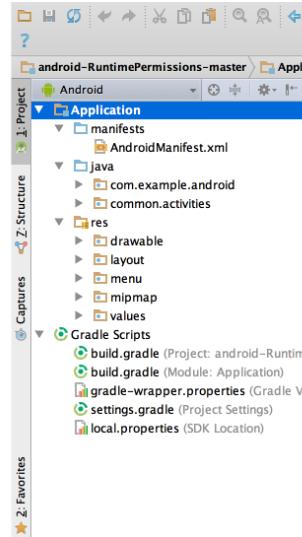


Figure 1. The project files in Android view.

## The user interface

The Android Studio main window is made up of several logical areas identified in figure 3.

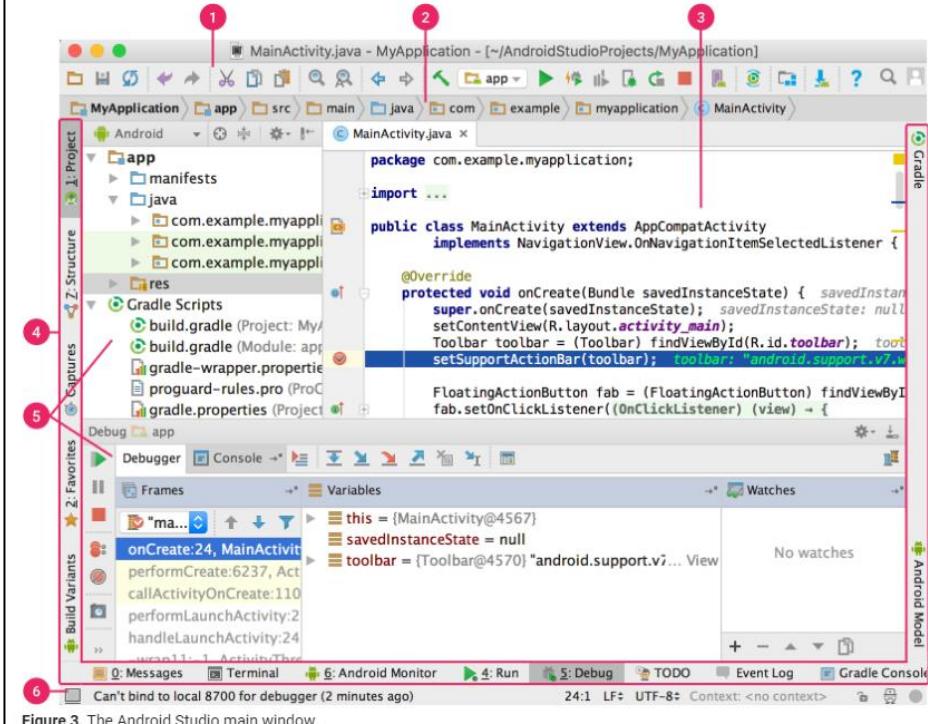


Figure 3. The Android Studio main window.



# ANDROID STUDIO (8)

## Migrate to Android Studio

Migrating your projects to Android Studio requires adapting to a new project structure, build system, and IDE functionality. If you are migrating an Android project from Eclipse, Android Studio provides an import tool so you can quickly move your existing code into Android Studio projects and Gradle-based build files. For more information, see [Migrating from Eclipse](#).

If you are migrating from IntelliJ and your project already uses Gradle, you can simply open your existing project from Android Studio. If you are using IntelliJ, but your project does not already use Gradle, you will need to do a little bit of manual preparation before you can import your project into Android Studio. For more information, see [Migrating from IntelliJ](#).

## Android Studio Basics

Here are some of the key differences you should be aware of as you prepare to migrate to Android Studio.

### Project and module organization

Android Studio is based on the [IntelliJ IDEA](#) IDE. To become familiar with the IDE basics, such as navigation, code completion, and keyboard shortcuts, see [Meet Android Studio](#).

Android Studio does not use workspaces, so separate projects open in separate Android Studio windows. Android Studio organizes code into projects, which contain everything that defines your Android app, from app source code to build configurations and test code. Each project contains one or more modules, which allow you to divide your project into discrete units of functionality. Modules can be independently built, tested, and debugged.



# DEVELOPER.ANDROID.COM

## Documentation for app developers

Whether you're building for Android handsets, Wear OS by Google, Android TV, or Android for Cars, this section provides the guides and API reference you need.

**Get started**

- Build your first app
- Sample code
- API reference
- Design guidelines
- Codelab tutorials
- Training courses

Android devices	
Wear OS	Android for Cars
Android TV	Chrome OS Devices
Best practices	
Dependency Injection	Accessibility
Testing	Security
Performance	Enterprise

**Kotlin**

Android and Kotlin	Kotlin and Jetpack	Kotlin Vocabulary
Android Developers  VISUALIZZA LA PLAYLIST COMPLETA	Android Developers  VISUALIZZA LA PLAYLIST COMPLETA	Android Developers  VISUALIZZA LA PLAYLIST COMPLETA

**Google Play**

Google Play	Play Asset Delivery	PolicyBytes
Google Play  VISUALIZZA LA PLAYLIST COMPLETA	From OBEs to Play Asset Delivery  VISUALIZZA LA PLAYLIST COMPLETA	Play PolicyBytes  VISUALIZZA LA PLAYLIST COMPLETA

**Android Developers** 1,07 Mln di iscritti

[HOME](#) [VIDEO](#) [PLAYLIST](#) [COMMUNITY](#) [CANALI](#) [INFORMAZIONI](#) [Ricerca](#)

**YouTube** IT

**#GoogleI/O** **Androidx Lifecycle** ✓ Consistent  
✓ Nested  
✓ Re-entrant safe

**Fragments: The Good (non-deprecated Parts) have been deprecated in favor of Androidx Lifecycle.**

**What's new in Android**  
60.020 visualizzazioni • 2 settimane fa

To watch this keynote interpreted in American Sign Language (ASL), please click here → <https://goo.gle/3Vkl6i>

Explore new updates on everything happening in the world of Android development including Jetpack, tooling, performance, platform and more!

... [ULTERIORI INFORMAZIONI](#)

**Modern Android Development at Google I/O 2022** **RIPRODUCI TUTTI**

Catch up on the top announcements from Google I/O '22 in Modern Android Development: our suite of libraries, tools and guidance that make it faster and easier to build amazing Android apps...

**Android @ I/O: Modern Android Development** 1:30

**What's new in Jetpack** 18:36

**Jetpack Compose best practices** 21:17

**Lazy layouts in Compose** 24:32

**Updates in Fragments** 17:51

**What's new in Android development tools** 34:28

**Android Studio and Tools** **RIPRODUCI TUTTI**

Get the latest updates on Android Studio tips and tooling with Android Tool Time. #AndroidStudio

Subscribe to Android Developers → <https://goo.gle/AndroidDevs>

**What's new in Android Studio - Bumblebee** 7:04

**What's new in Android Studio Arctic Fox** 9:57

**What's new in Android Studio 4.2** 7:10



---

# Creare applicazioni Android



# ANDROID & JAVA

---

- Sviluppare applicazioni per Android è *molto simile* a svilupparle in Java con JavaFX
  - il linguaggio è invariato: Java è sempre Java!
  - le classi di infrastruttura, pure (Collection, stream, lambda..)
- Cambia ovviamente il framework grafico
  - non più JavaFX (o Swing)..
  - ..ma un framework apposito per i device Android (simile a JavaFX)
- L'impianto diventa decisamente grafico
  - niente più main classico (non c'è neppure il terminale!)
  - come in JavaFX, *le app derivano da una certa classe-base che ne stabilisce i metodi fondamentali (da ridefinire)*
  - necessità di familiarizzare col nuovo ambiente



# SVILUPPARE IN ANDROID

---

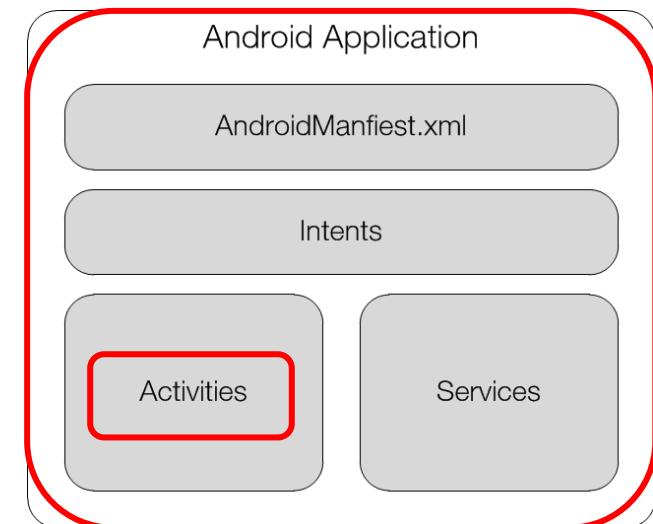
- Operativamente, la *diversa macchina virtuale* implica **qualche passaggio in più**
  - PRIMA bastava generare il bytecode (.class)
  - ORA, in più, occorre generare il «*suo*» formato (.dex / .elf)
- In particolare, *poiché si sviluppa su una piattaforma (PC) un'app destinata a funzionare su un'altra (Android)*, per eseguirla occorre un **emulatore**
  - emula un terminale Android con relativa GUI, tasti, etc
  - possibile emulare vari dispositivi con diverse caratteristiche hw/sw (sensori, trasduttori, display...)



# APPLICAZIONI ANDROID (1)

Una applicazione Android ("app") non ha main perché

- è diverso lo scenario
  - un dispositivo smartphone/tablet è ben diverso da un PC !
- il flusso *non è più sotto il nostro controllo*
  - come in JavaFX, infatti
  - le app devono seguire *un ben preciso schema* per far parte del framework d'esecuzione
  - si sfrutta l'ereditarietà: **la classe-App eredita da una precisa classe base**
  - un'app è fatta di una o più *Activity* ed è sempre descritta da un *manifest* (*se non ha GUI, può essere un Service*)

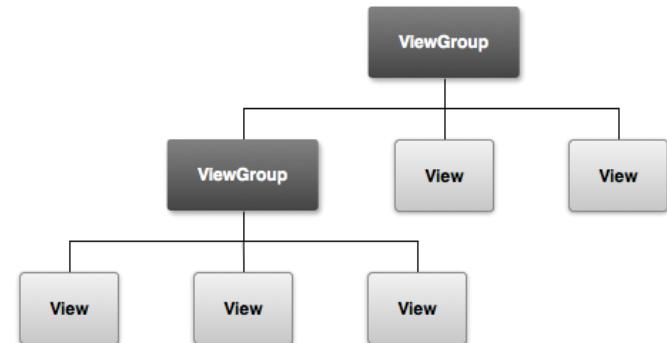




# APPLICAZIONI ANDROID (2)

## Il framework grafico di Android

- Android non usa JavaFX, ha il suo framework specifico
- i componenti grafici sono tutti sottoclassi di **View**
  - sono sottoclassi di View anche text field, bottoni, etc
- strutturalmente, le View sono organizzate in *gruppi*
  - un layout è quindi composto da View e da ViewGroups (invisibili) opportunamente annidati
- Come in JavaFX, solitamente le GUI *non si programmano: si descrivono* tramite un idoneo **vocabolario XML**





# APPLICAZIONI ANDROID (3)

- Le *Activity* rappresentano attività *visibili* (~schermate)
  - approssimativamente, un'activity è "circa come" un Pane di JavaFX
  - esistono anche attività *invisibili* (es. servizi)
- Un'applicazione non banale richiede più attività, ma tipicamente *non tutte* sono parte della nostra applicazione:
  - potrebbero essere parte della dotazione Android (es. maps, email)
  - potrebbe non essere noto *quale* attività vada chiamata (es. email, kmail, gmail...), ma solo il servizio che deve fornire (es: inviare mail)
- L' *Intent* è un *meccanismo flessibile* per connettere attività fra loro, *direttamente* o *indirettamente*





---

**Un primo esempio  
App "Hello World" in Android Studio**

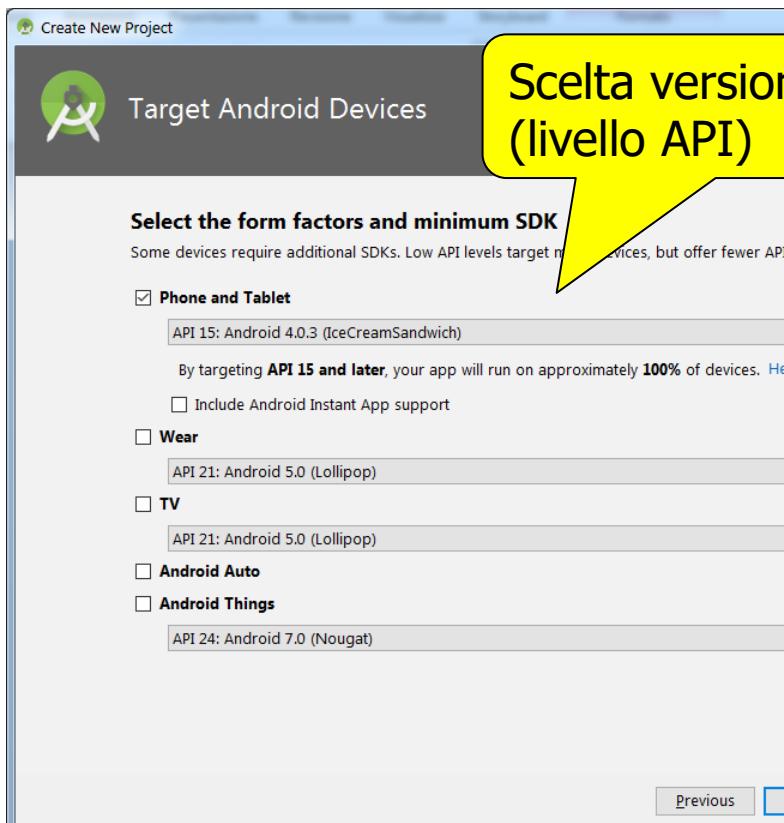


# SVILUPPO (1)

The screenshot shows the Android Studio interface. On the left, the main window has a title bar "Welcome to Android Studio" and displays the Android Studio logo and version "Version 3.1.2". Below the logo, there is a list of options: "Start a new Android Studio project" (highlighted with a blue border), "Open an existing Android Studio project", "Check out project from Version Control", "Profile or debug APK", "Import project (Gradle, Eclipse ADT, etc.)", and "Import an Android code sample". On the right, a modal dialog titled "Create New Project" is open, with the sub-title "Create Android Project". It contains fields for "Application name" (set to "CodFiscApp"), "Company domain" (set to "enricodenti.disi.unibo.it"), "Project location" (set to "C:\Users\Enrico Denti\AndroidStudioProjects\CodFiscApp"), and "Package name" (set to "it.unibo.disi.enricodenti.codfiscapp"). There are also two unchecked checkboxes: "Include C++ support" and "Include Kotlin support". At the bottom of the dialog, a warning message says "⚠️ project location should not contain whitespace, as this can cause problems with the NDK tools." Below the dialog are buttons for "Previous", "Next", "Cancel", and "Finish".

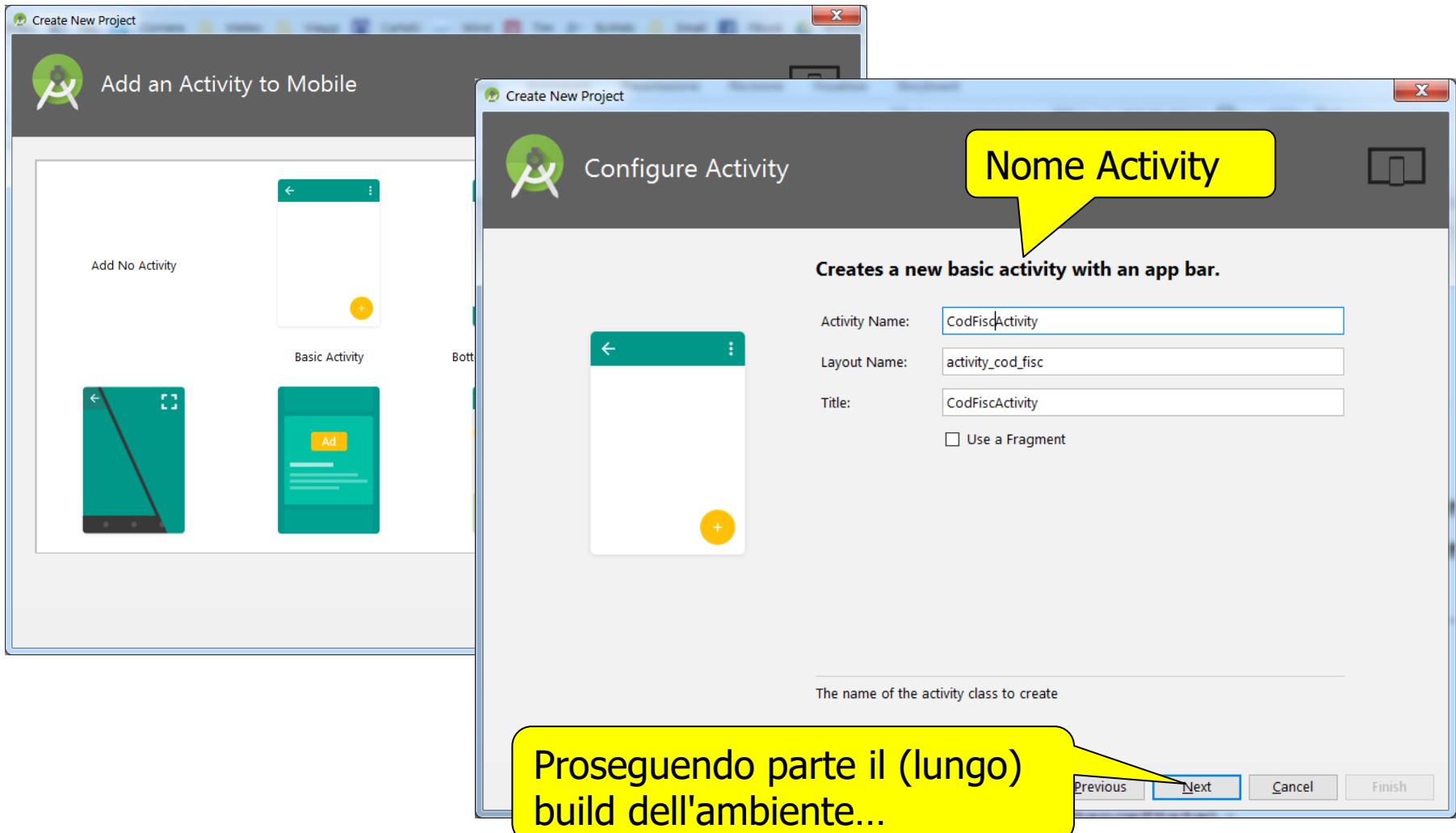


# SVILUPPO (2)



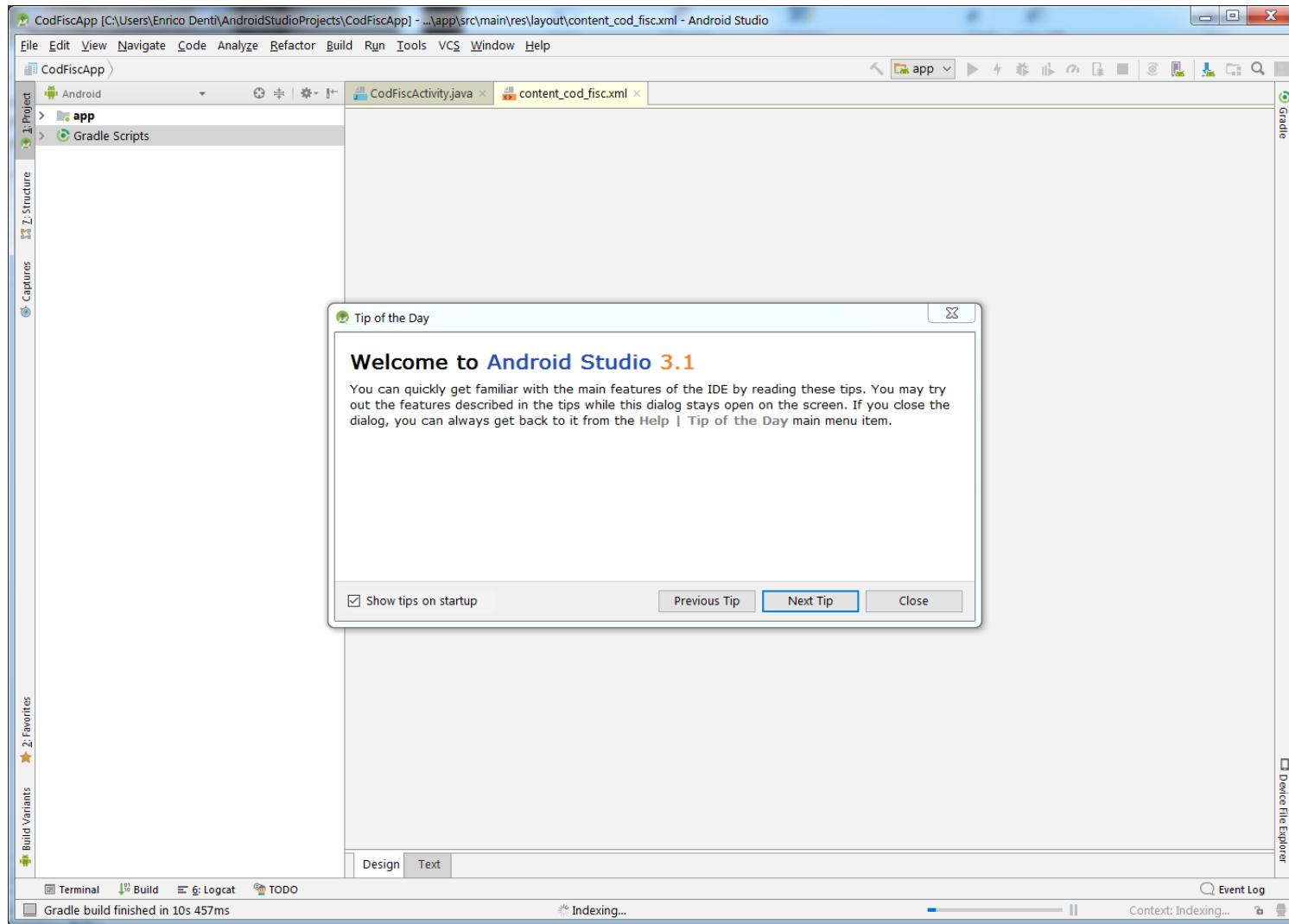


# SVILUPPO (3)





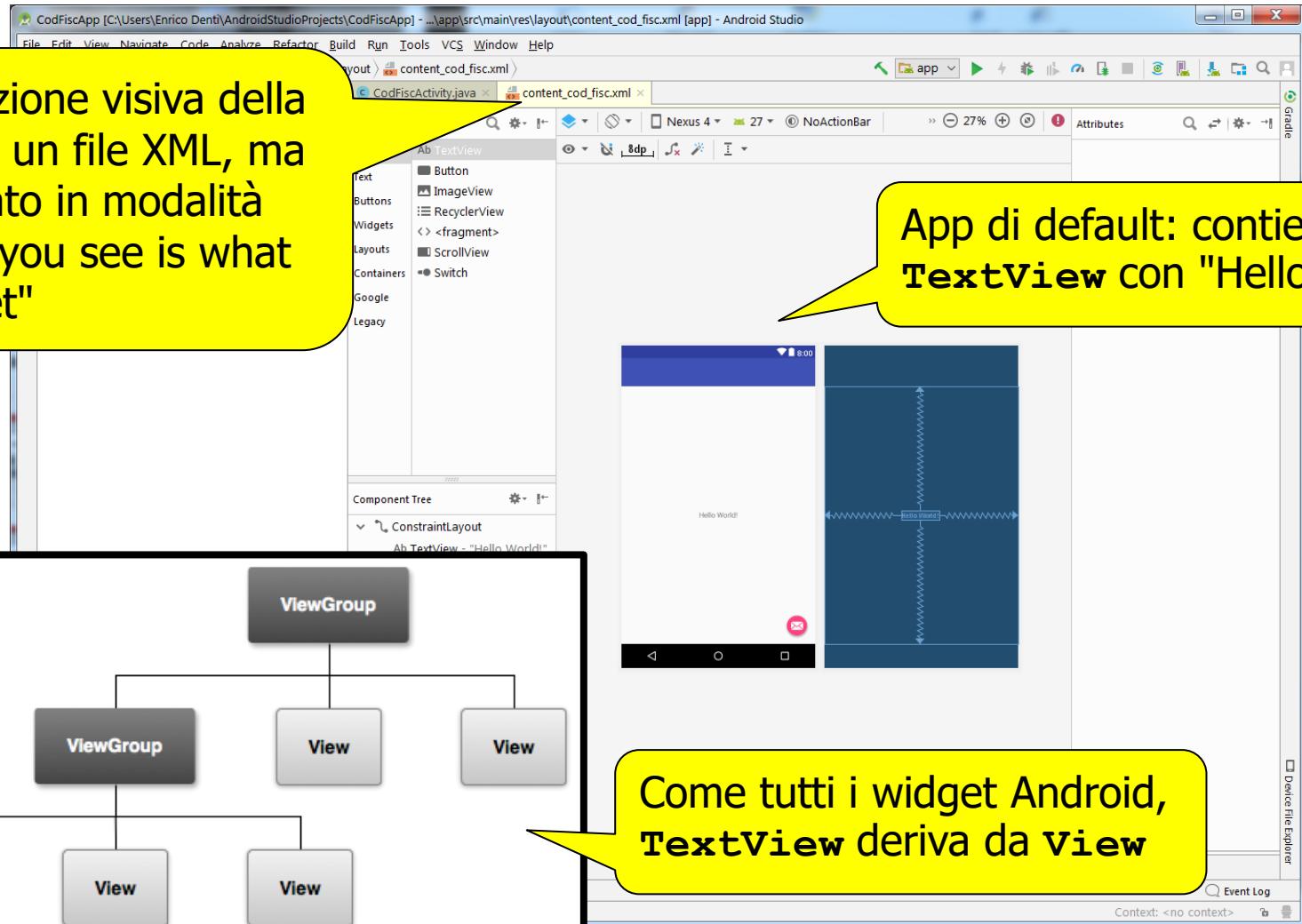
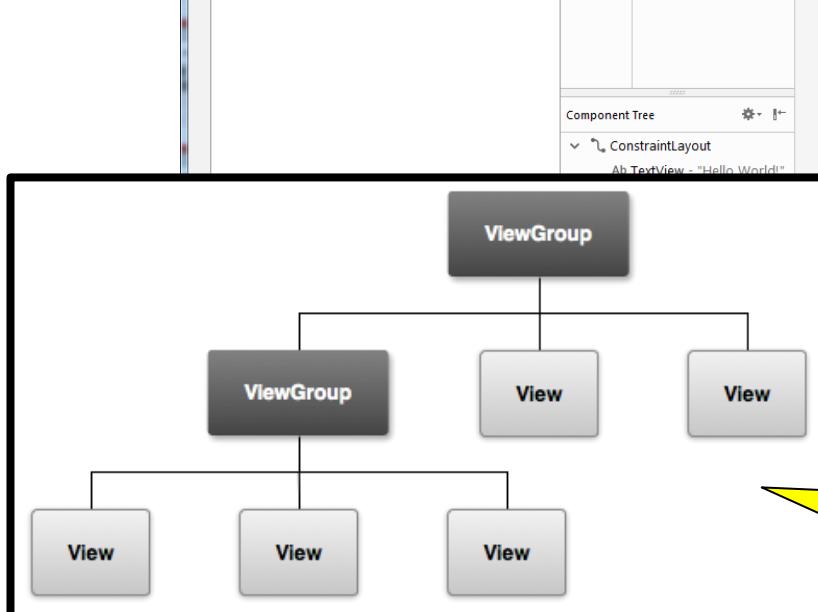
# SVILUPPO (4)





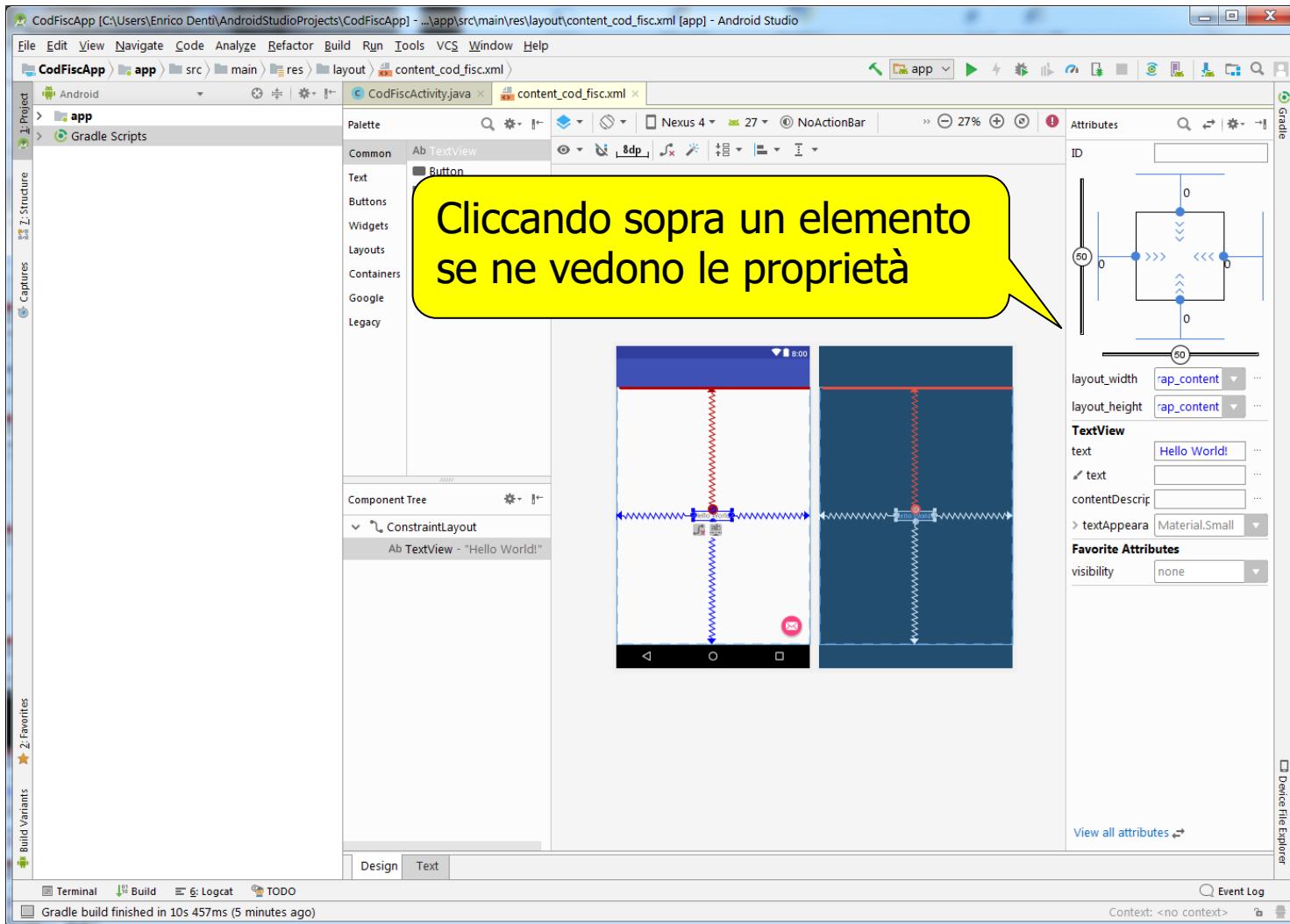
# SVILUPPO (5)

Descrizione visiva della GUI: è un file XML, ma mostrato in modalità "what you see is what you get"





# SVILUPPO (6)





# SVILUPPO (7)

Dietro le quinte, viene generato il codice Java corrispondente

The screenshot shows the Android Studio interface with the file `CodFiscActivity.java` open. The code implements an `AppCompatActivity`, overriding `onCreate` and `onCreateOptionsMenu` methods. It sets the content view to `activity_cod_fisc`, initializes a toolbar, and adds a floating action button (Fab) with a click listener that displays a Snackbar. It also handles options menu items, specifically the action bar's settings item.

```
package it.unibo.disi.enricodeneti.codfiscapp;

import ...

public class CodFiscActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cod_fisc);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(view -> {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_cod_fisc, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

At the bottom of the screen, the status bar indicates: "Gradle build finished in 10s 457ms (3 minutes ago)".



# SVILUPPO (8)

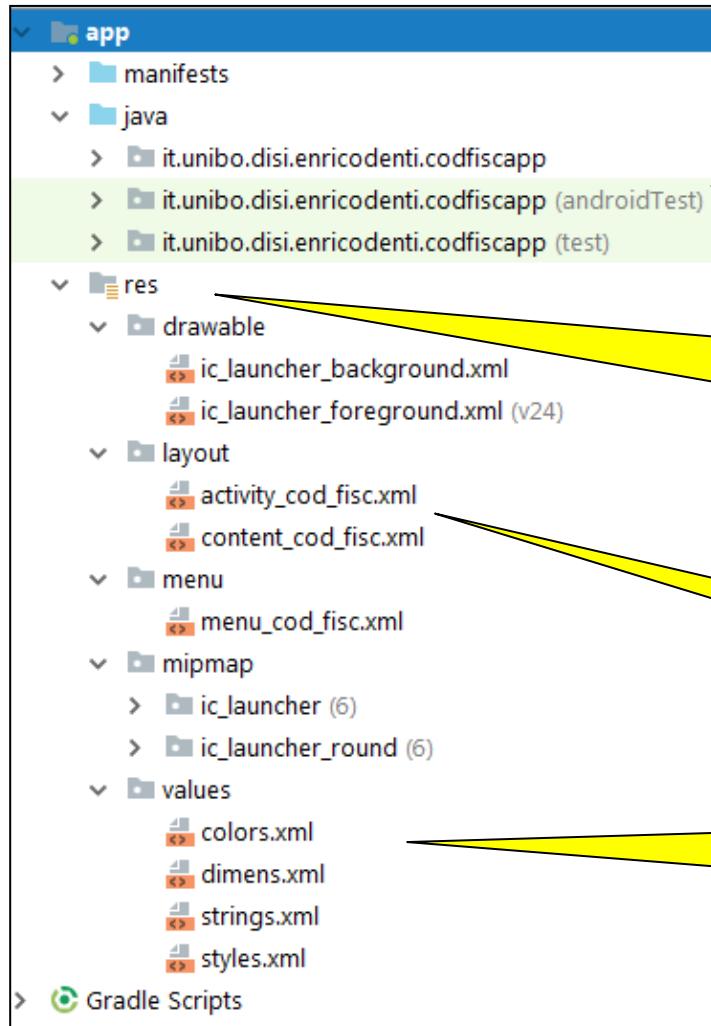
Il metodo onCreate è chiamato al'atto della creazione dell'app

Lo "strano" argomento **R.layout** fa riferimento a un file di risorse (XML) esterno, **activity\_cod\_fisc.xml**

```
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_cod_fisc);  
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
    setSupportActionBar(toolbar);  
  
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);  
    fab.setOnClickListener((view) -> {  
        Snackbar.make(view, text: "Replace with your own action", Snackbar.LENGTH_LONG)  
            .setAction( text: "Action", listener: null).show();  
    });  
}
```



# SVILUPPO (9): LE RISORSE



Vengono generati anche gli scheletri dei test

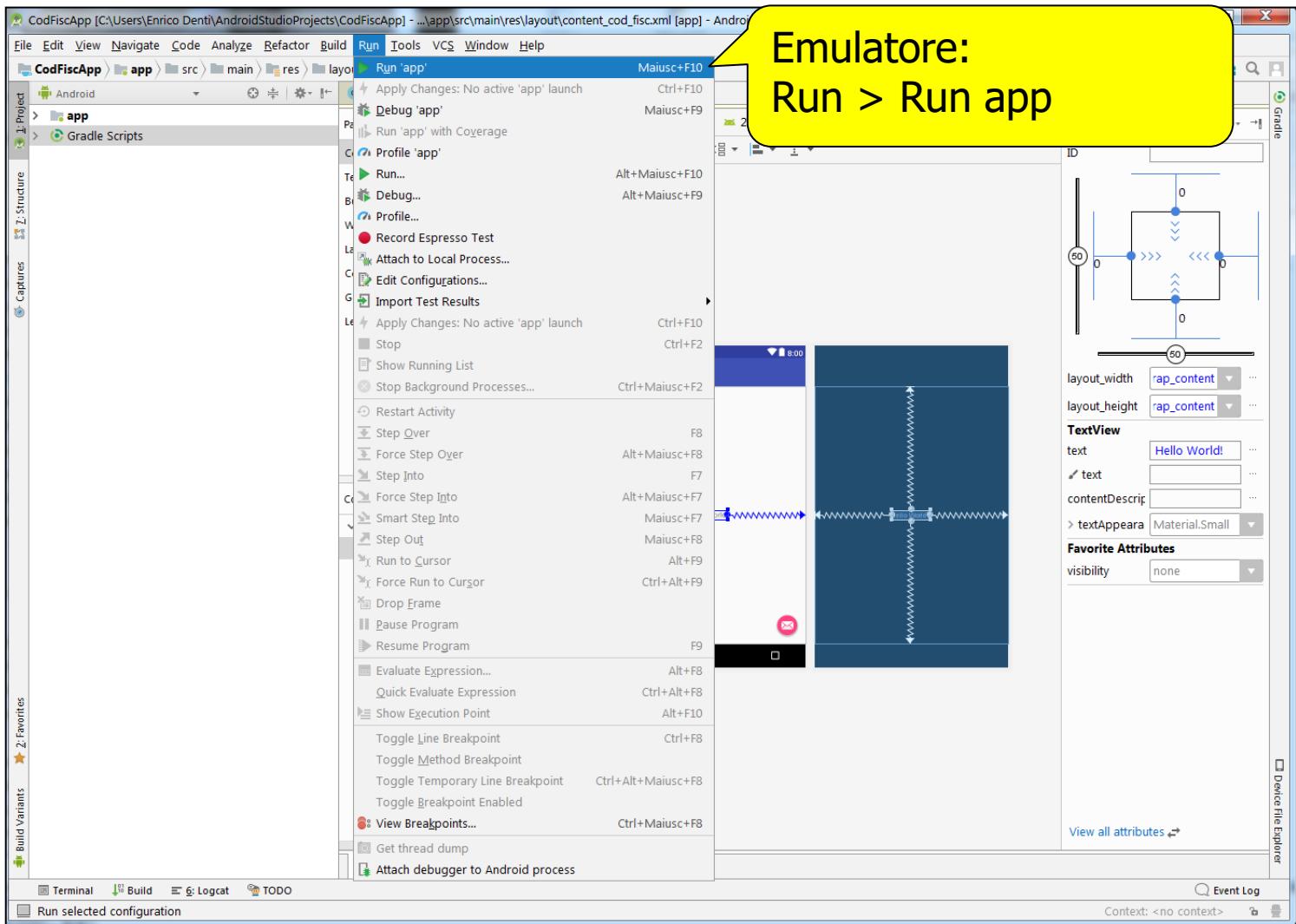
Un'app android usa molte **risorse** definite nel progetto, ma in file esterni al sorgente Java propriamente detto

Layout, menu..

..ma anche stringhe (costanti), stili, colori..

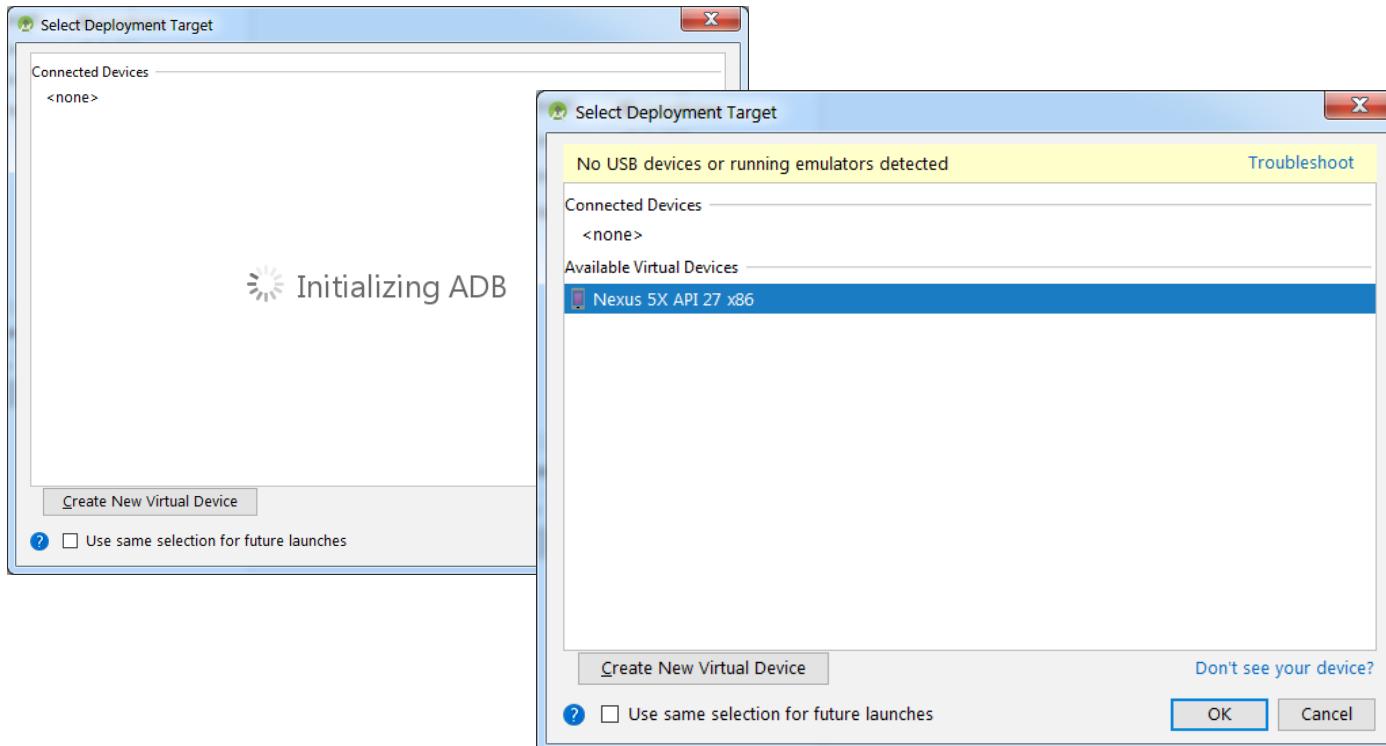


# RUN: device o emulatore?



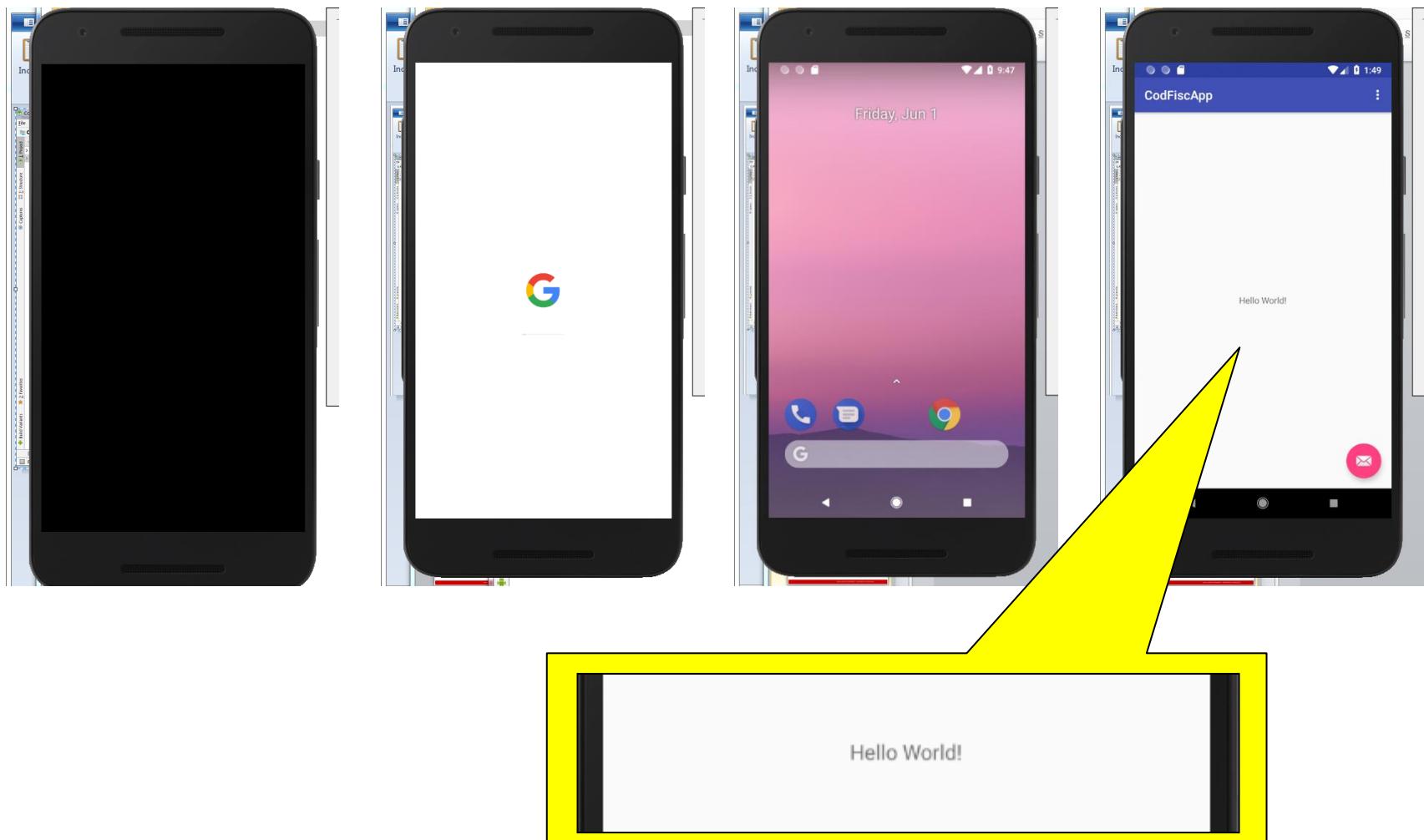


# RUN SU EMULATORE





# BOOT SU EMULATORE..





---

# Un esempio concreto CodFiscApp in Android Studio



# CODICE FISCALE SU ANDROID

- Ricordate il buon vecchio *codice fiscale* da terminale?
- **Facciamolo diventare un'app Android!**
  - la logica dell'applicazione è immutata → **CodFisc** resta com'è (però Android esige un package, mentre **CodFisc** usava il default...)
  - **cambia l'interazione con l'utente:** non più console, ma una GUI
- Per fare la GUI in Android occorre:
  - familiarizzare con il **framework android.widget**
  - struttura: serve un componente di testo che permetta all'utente di inserire i dati (come se fosse la linea di comando) → **EditView**
  - gestione eventi: **EditView** espone il metodo **onEditorAction**, da agganciare all'apposito **onEditorActionListener** in particolare, potremmo voler intercettare il tasto DONE (Fatto)
  - o la programmiamo, o la descriviamo (**editor visuale**)



# CODICE FISCALE SU ANDROID

- In Android solitamente si descrive e si usano i vari tool
- **In questo esempio invece lo programmiamo**
  - non perché sia meglio: semplicemente, per un'app così banale e con poca grafica, *facciamo prima* ☺
  - inoltre, è un cambiamento minimale venendo da JavaFX
- In pratica, occorre **ridefinire il metodo `onCreate`**
  1. creare l'oggetto `EditView`
  2. impostarne la proprietà `singleLine` e il testo di default
  3. agganciare il listener tramite il metodo `onEditorAction`
    - usiamo una lambda con method reference
    - richiede language compliance Java 8 (default: Java 7)
  4. impostare l'oggetto `EditView` come contenuto dell'activity (panel)



# CodFiscAndroid activity

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    EditText txt = new EditText(this);  
    txt.setSingleLine();  
    txt.setText("Immettere i dati della persona. . .");  
    txt.setOnEditorActionListener(this::myHandle);  
    setContentView(txt);  
}
```

Importante perché operi come un textfield

Gestore eventi (lambda)

Anziché il contenuto di default R.layout.xxxx, impostiamo il nostro txt.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    EditText txt = new EditText(context: this);  
    txt.setSingleLine();  
    txt.setText("Immettere i dati della persona: ");  
    txt.setOnEditorActionListener(this::myHandle);  
    setContentView(txt);  
}
```



# CodFiscAndroid eventi

La costante `IME_ACTION_DONE` rappresenta il tasto DONE (Fatto)

Se il tasto premuto non è quello, esce senza fare nulla

```
private boolean myHandle(TextView textView, int actionId, KeyEvent keyEvent) {  
    if (actionId != EditorInfo.IME_ACTION_DONE) return false;  
    String line = textView.getText().toString();  
    String[] args = line.split( regex "\\s+" ); // separa args per calcolaCodice  
    String codice = CodFisc.calcolaCodice(  
        args[1],  
        args[0],  
        Integer.parseInt(args[2]),  
        Integer.parseInt(args[3]),  
        Integer.parseInt(args[4]),  
        args[5].equals("M") ? CodFisc.MASCHIO : CodFisc.FEMMINA, args[6] );  
    textView.setText(codice);  
    return true;  
}
```

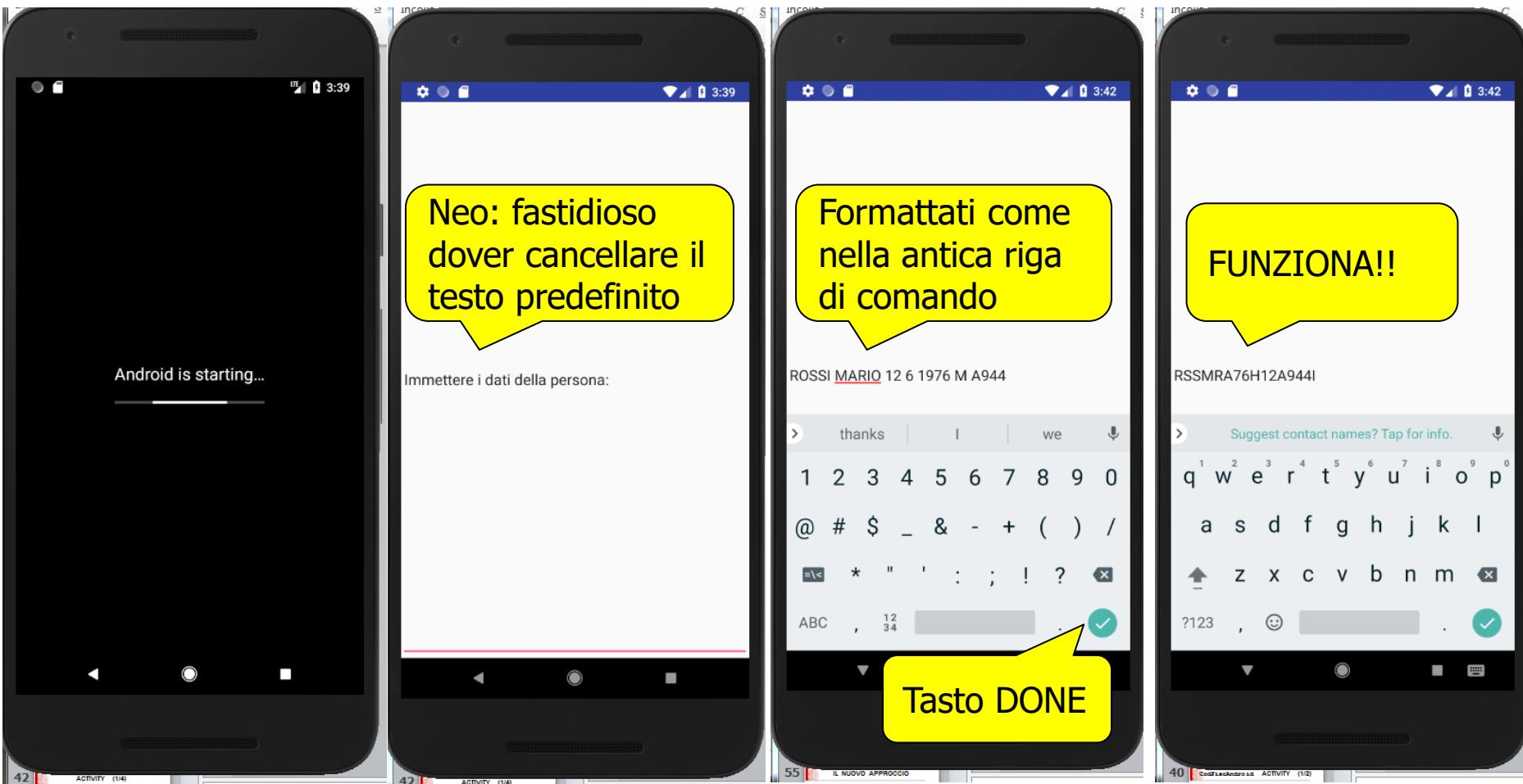
Separa le parti della stringa come se fossero args da riga di comando

Per convenzione, si restituisce `true` se l'evento è stato gestito, `false` altrimenti

**FINITO! Tutto fatto!**  
**Let's run...!**

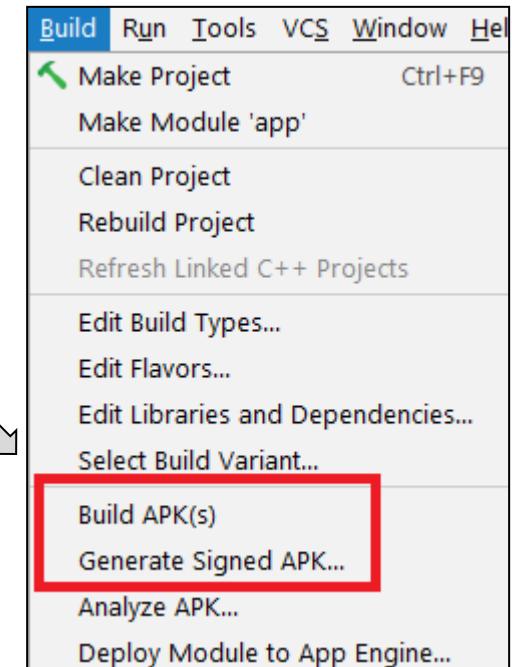
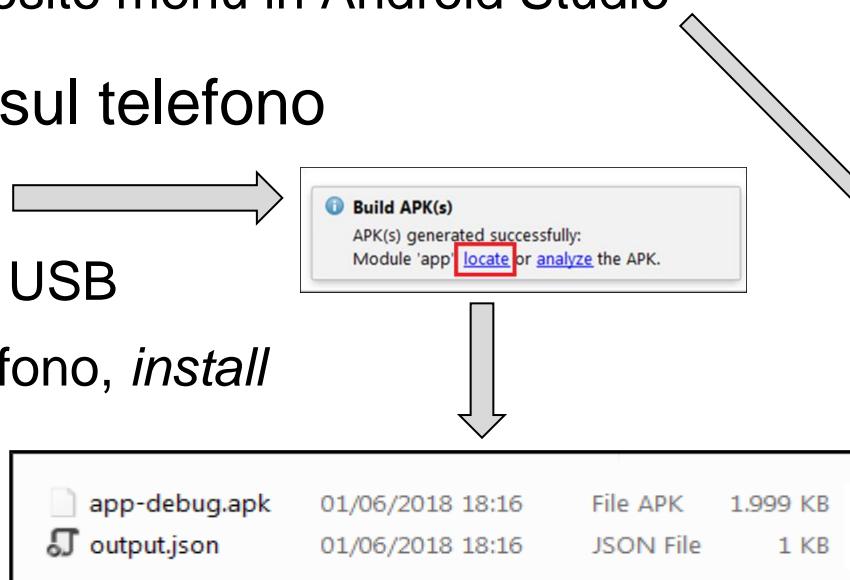


# RUN SU EMULATORE



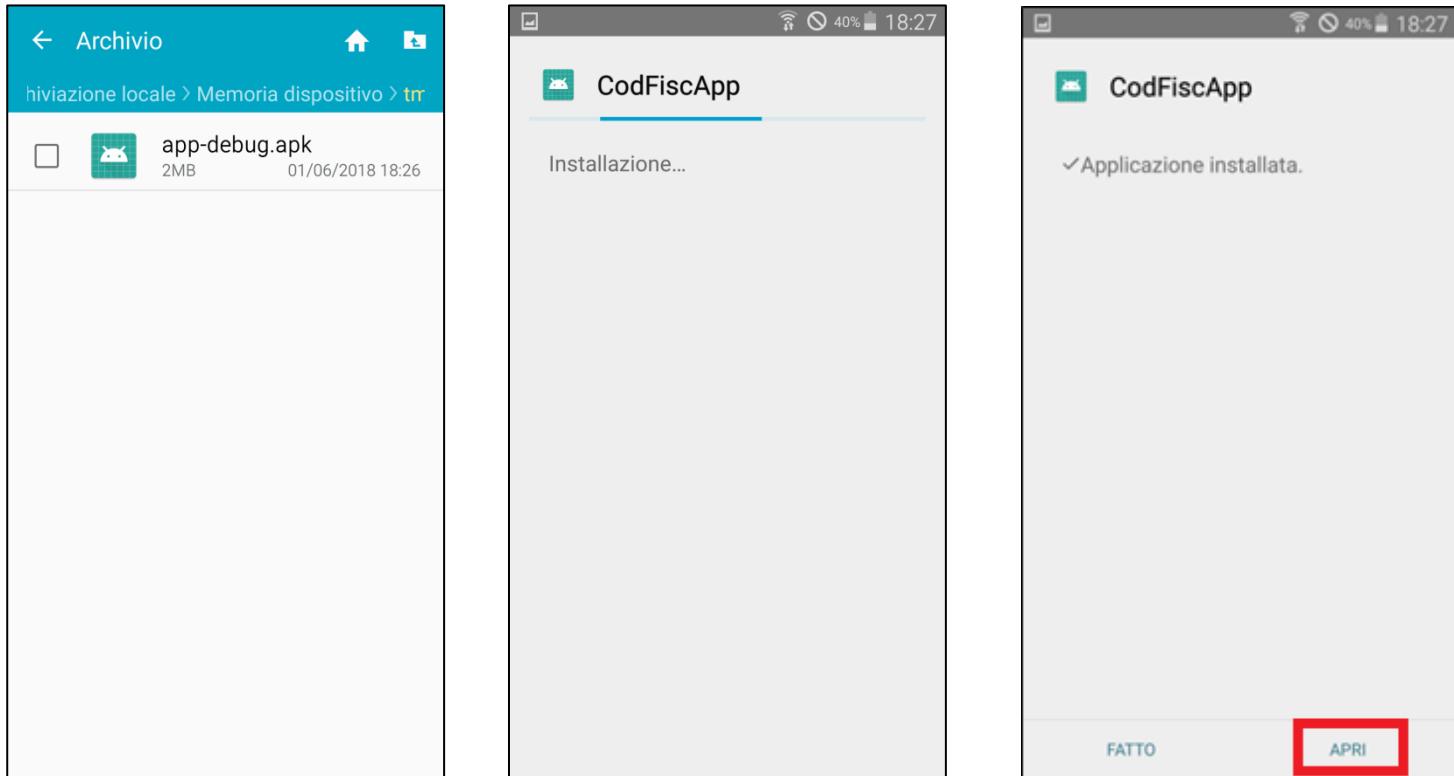
# DEPLOYMENT

- Come installarla su un vero telefono?
  - o si collega il telefono al PC via USB, tramite l'USB driver
  - oppure si copia e installa l'APK manualmente
- Per generare l'APK
  - si usa l'apposito menu in Android Studio
- Per copiarla sul telefono
  - trovarla ☺ →
  - copiarla via USB
  - poi, nel telefono, *install*



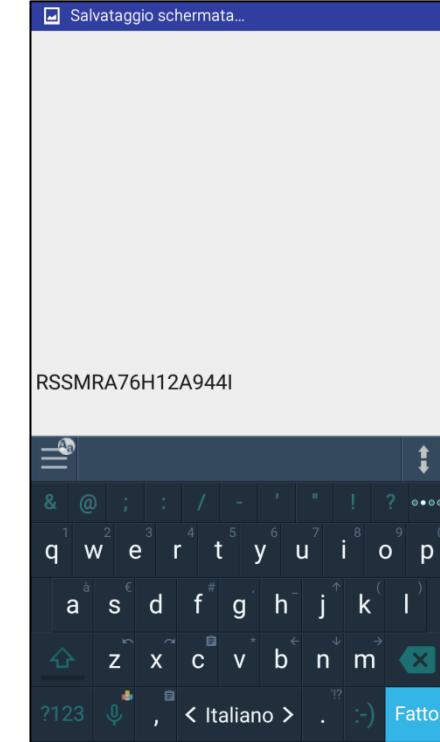
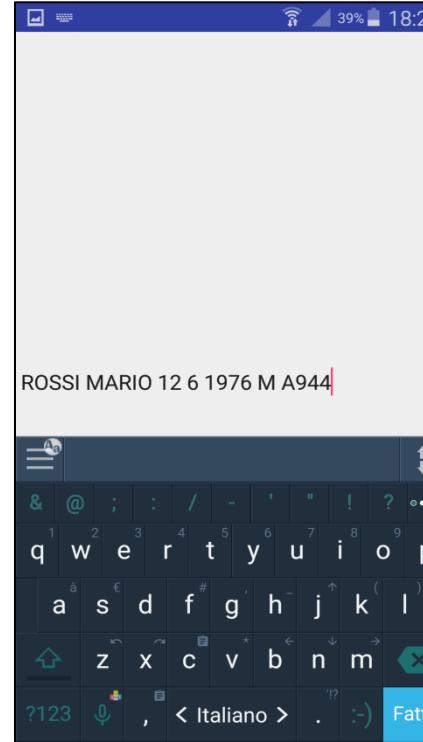
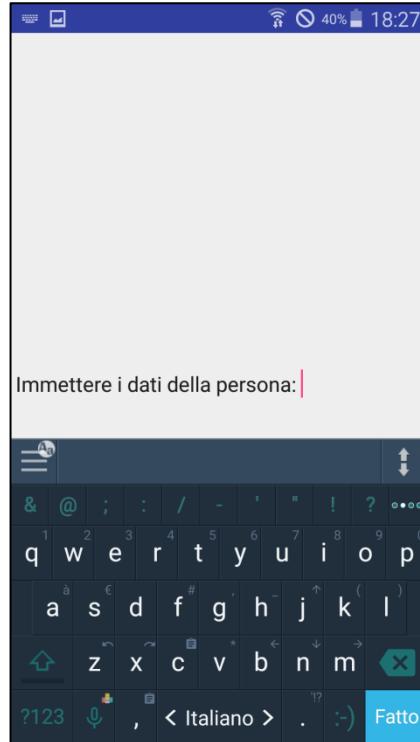
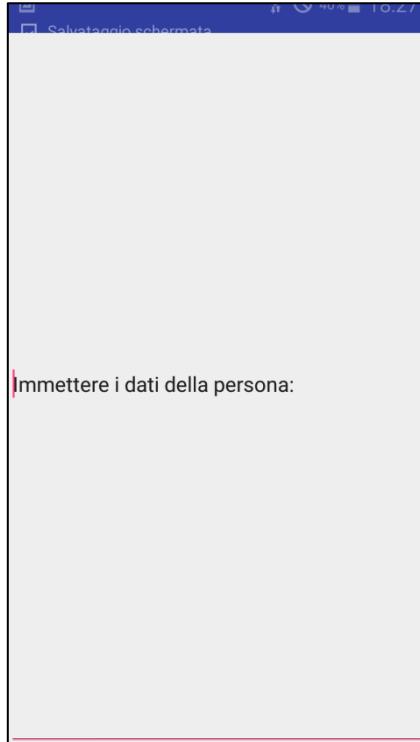


# INSTALLAZIONE





# RUN SU TELEFONO



Tasto DONE



# DISEGNARE LA GUI

---

- In realtà, di solito non si "programma" la GUI: la si *disegna*
  - widget, layout, stringhe immagini diventano *risorse* → file **R**
  - le risorse sono esterne al programma Java, *non più cablate*
  - molto più facili da aggiornare e manutenere
- L'IDE genera la *descrizione* della GUI come *risorse layout*
  - `setContentView` carica sempre e solo i vari file **R.layout.xxx**
  - il programma Java (activity) si occupa di agganciare i listener
  - MA per farlo deve sapere i nomi dei widget descritti nel layout  
→ ogni widget è associato a un *nome simbolico* → file **R.id**
  - da programma occorre recuperare il riferimento all'oggetto,  
a partire dal suo nome simbolico → metodo `findViewById`



# ANDROID STUDIO: MODALITÀ XML

Corrispondente  
vista grafica

Descrizione XML della  
TextView di default

Content XML description corresponding to the visual representation in the preview window.

The screenshot shows the Android Studio interface with the XML file `content_cod_fisc.xml` open. The code defines a `TextView` with various attributes like `layout_width`, `layout_height`, and `text`. The preview window on the right shows a simple white screen with a blue header bar containing the text "Hello World!". The tabs at the bottom of the XML editor are "Design" and "Text", with "Text" being the active tab.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



# CodFiscAndroid REVISED

La struttura della GUI  
*non è più cablata nel programma Java*

```
public class CodFiscAndroidAct...  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_cod_fisc);  
        EditText txt = findViewById(R.id.mytxt);  
        txt.setOnEditorActionListener(...);  
    }  
}
```

È invece *descritta* in un *risorsa layout*  
– qui, *activity\_cod\_fisc*

Il programma Java recupera  
un riferimento all' *EditText* di  
*nome simbolico mytxt*

Tale layout specificherà  
l' *EditText* di nome  
**mytxt** con tutte le  
sue proprietà



# CodFiscAndroid REVISED

The screenshot shows the Android Studio interface with the file `CodFiscActivity.java` open. A red box highlights the `onCreate` method. A yellow callout bubble points to this box with the text "Sorgente Java revisionato". Another yellow callout bubble points to the `myHandle` method with the text "Gestione eventi intoccata!".

```
13 import android.widget.EditText;
14 import android.widget.TextView;
15
16 import it.unibo.disi.enricodenti.codfisc.CodFisc;
17
18 public class CodFiscActivity extends AppCompatActivity {
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_cod_fisc);
24         EditText txt = findViewById(R.id.mytxt);
25         txt.setOnEditorActionListener(this::myHandle);
26     }
27
28     private boolean myHandle(TextView textView, int actionId, KeyEvent keyEvent) {
29         if (actionId != EditorInfo.IME_ACTION_DONE) return false;
30         String line = textView.getText().toString();
31         String[] args = line.split("\\s+"); // separa args per codice
32         String codice = CodFisc.calcolaCodice(
33             args[1],
34             args[0],
35             Integer.parseInt(args[2]),
36             Integer.parseInt(args[3]),
37             Integer.parseInt(args[4]),
38             args[5].equals("M") ? CodFisc.MASCHIO : CodFisc.FEMMINA, args[6] );
39         textView.setText(codice);
40         return true;
41     }
42 }
```

CodFiscActivity > onCreate()



# DISEGNARE LA GUI

Sostituiamo la `TextView` di default col nostro `EditText`, *specificandone le proprietà*

In particolare impostiamo `hint`, il testo di default *che sparirà automaticamente* quando l'utente ci scriverà dentro (molto meglio!) [rosso perché ancora indefinito]

>Create string value resource 'immissione'

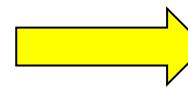
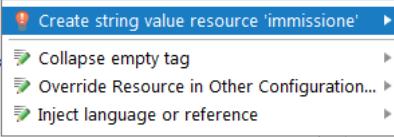
- Resolve dependencies of :app:debugRuntimeClasspath
- Resolve files of :app:debugRuntimeClasspath
- Resolve dependencies of :app:debugCompileClasspath
- Resolve files of :app:debugCompileClasspath
- Resolve files of :app:debugCompileClasspath

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA



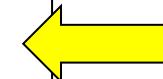
# DISEGNARE LA GUI

```
<EditText  
    android:id="@+id/mytxt"  
    android:singleLine="true"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/immissione">  
</EditText>  
</android.support.constraint.Cons
```

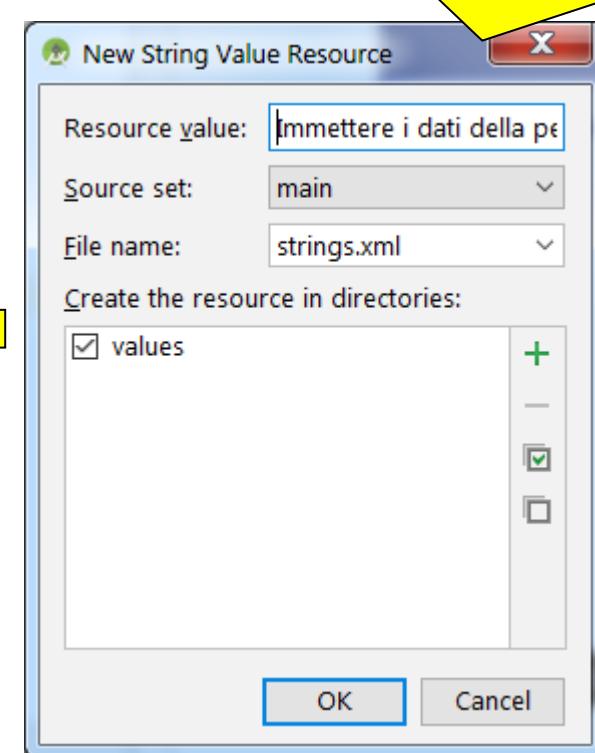


Impostiamo come valore la stringa da mostrare: qui,  
*"Immettere i dati della persona:"*

```
<EditText  
    android:id="@+id/mytxt"  
    android:singleLine="true"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/immissione">  
</EditText>
```



La proprietà **hint** è ora verde, segno che è stata correttamente impostata.





# DISEGNARE LA GUI

The screenshot shows the Android Studio interface. On the left, the project structure is displayed under the 'app' folder, including 'manifests', 'java', 'res' (with 'drawable', 'layout', 'menu', 'mipmap', and 'values' subfolders), and 'Gradle Scripts'. A red box highlights the 'strings.xml' file in the 'values' folder. In the center, three tabs are open: 'CodFiscActivity.java', 'content\_cod\_fisc.xml', and 'strings.xml'. The 'strings.xml' tab shows the following XML code:

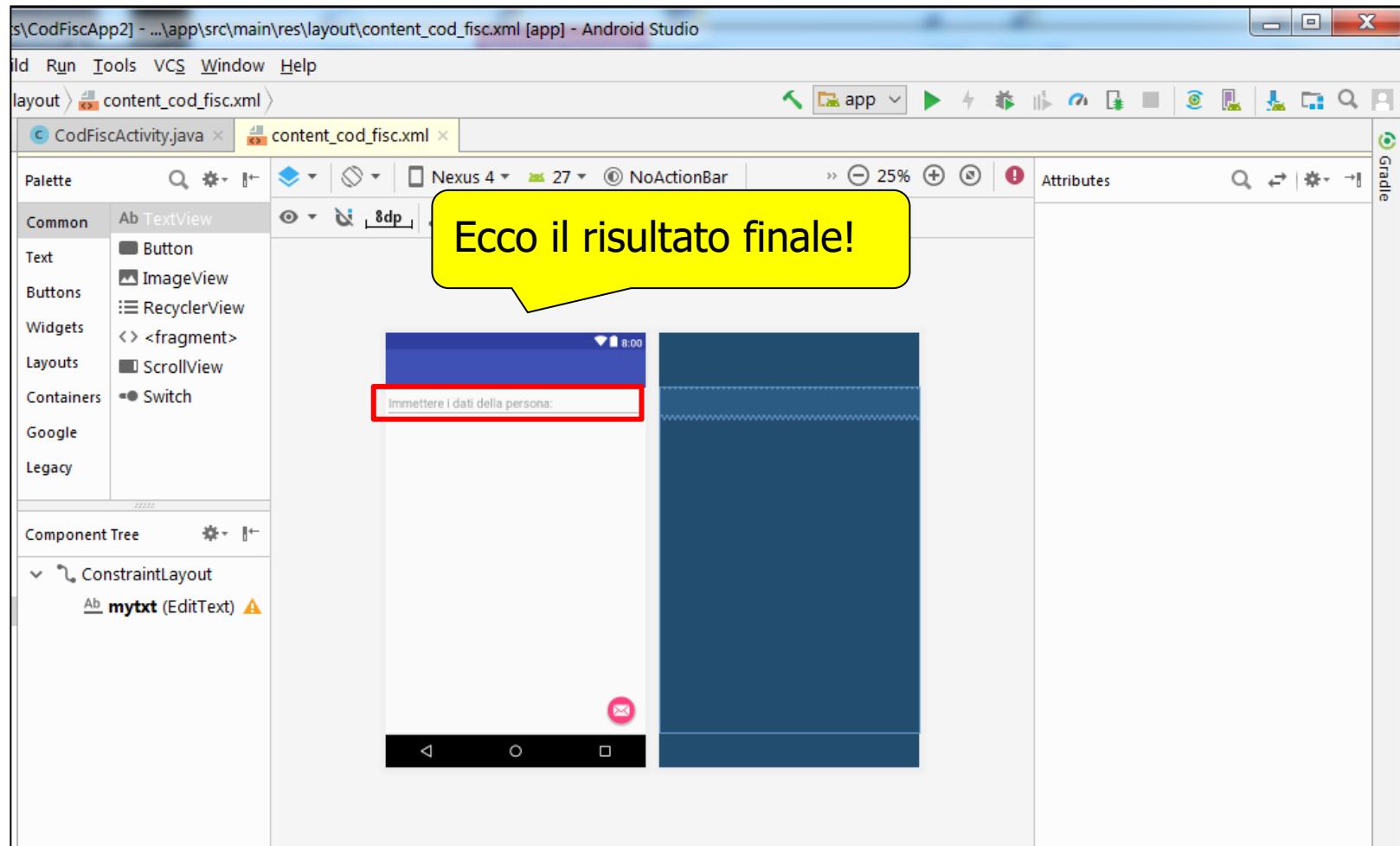
```
<resources>
    <string name="app_name">CodFiscApp2</string>
    <string name="action_settings">Settings</string>
    <string name="immissione">Immettere i dati della persona:</string>
</resources>
```

A yellow callout bubble points to the 'strings.xml' tab with the text: "La risorse stringa sono elencate nel file strings.xml". Another yellow callout bubble points to the code within the 'strings.xml' tab with the text: "Eccola qui!".

Facile tradurre l'app in altre lingue.. ☺

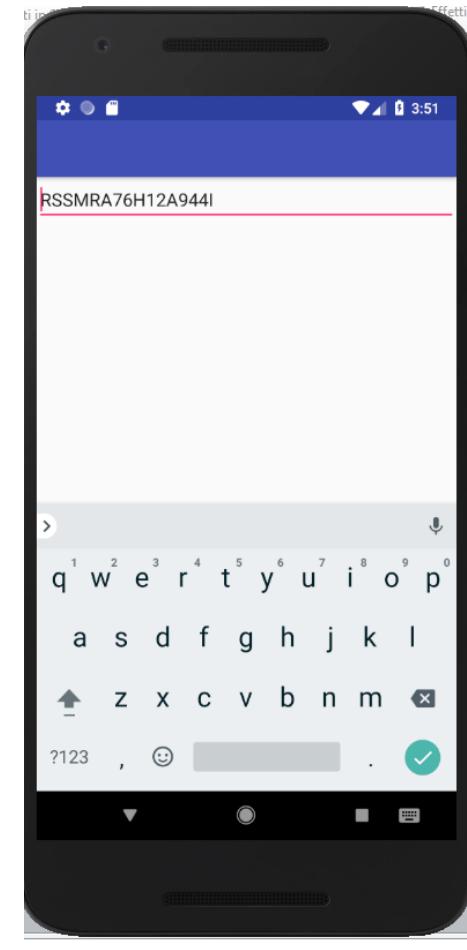
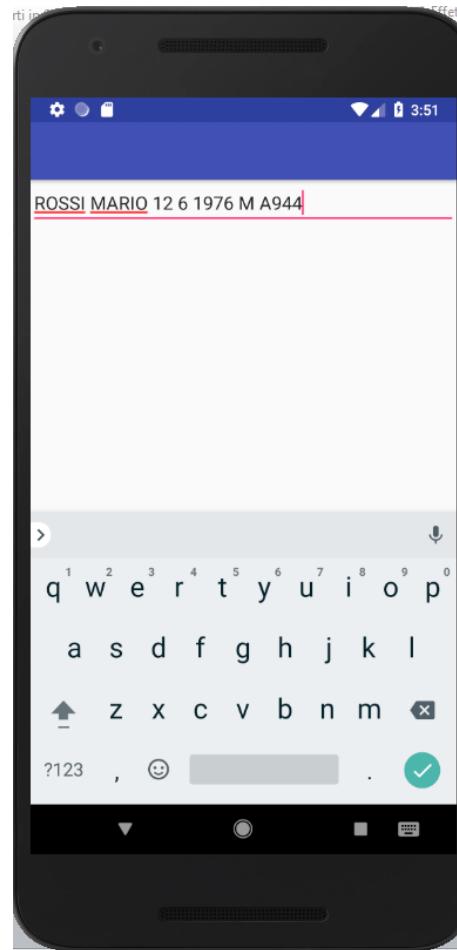
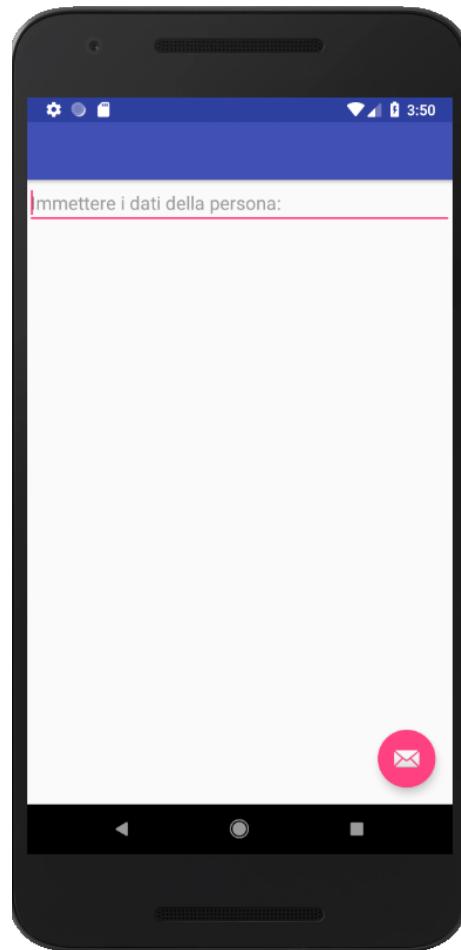


# DISEGNARE LA GUI





# RUN SU EMULATORE





# Intent



# OLTRE LE ACTIVITY: Intent

- L'*Intent* è un meccanismo per connettere più attività fra loro
- È una *descrizione astratta di una operazione da eseguire*
  - esplicito: si indica *la specifica attività* da eseguire
  - implicito: si indica *solo l'azione* da eseguire (non *chi* la eseguirà)
- Idea chiave: scrivere applicazioni i cui pezzi interagiscono fra di loro *senza necessariamente doversi conoscere*

## SCENARIO 1

- App articolata in due attività
  - una activity mostra una lista di URL
  - l'altra **visualizza** l'URL scelto
- L'Intent le connette *direttamente*
  - è creato dalla prima attività
  - contiene il **nome dell'altra attività**

## SCENARIO 2

- App articolata in due attività
  - una activity mostra una lista di URL
  - l'altra **invia** l'URL scelto.
- L'Intent le connette *indirettamente*
  - è creato dalla prima attività
  - contiene il **nome dell'azione da svolgere** (ACTION\_SEND),  
*non chi* la svolgerà



# INTENT

## Un Intent

- è un'istanza di `android.content.Intent`
- può essere definito programmaticamente o come risorsa
- è assimilabile a un "segnale" (evento) asincrono, che viene ricevuto da quei componenti (attività o servizi) registrati per esso
- può contenere dati, utili al ricevente per decidere cosa fare

### ESPLICITO

*l'attività Pippo attiva l'attività Pluto*

```
Intent i = new Intent(  
        this, Pluto.class);
```

### IMPLICITO

*l'attività Pippo attiva una attività  
capace di svolgere l'azione VIEW*

```
Intent i = new Intent(  
        Intent.ACTION_VIEW,  
        Uri.parse("http://...."));
```

*Se ci sono più componenti capaci di  
svolgere l'azione, apparirà un dialog per  
scegliere quale attivare.*

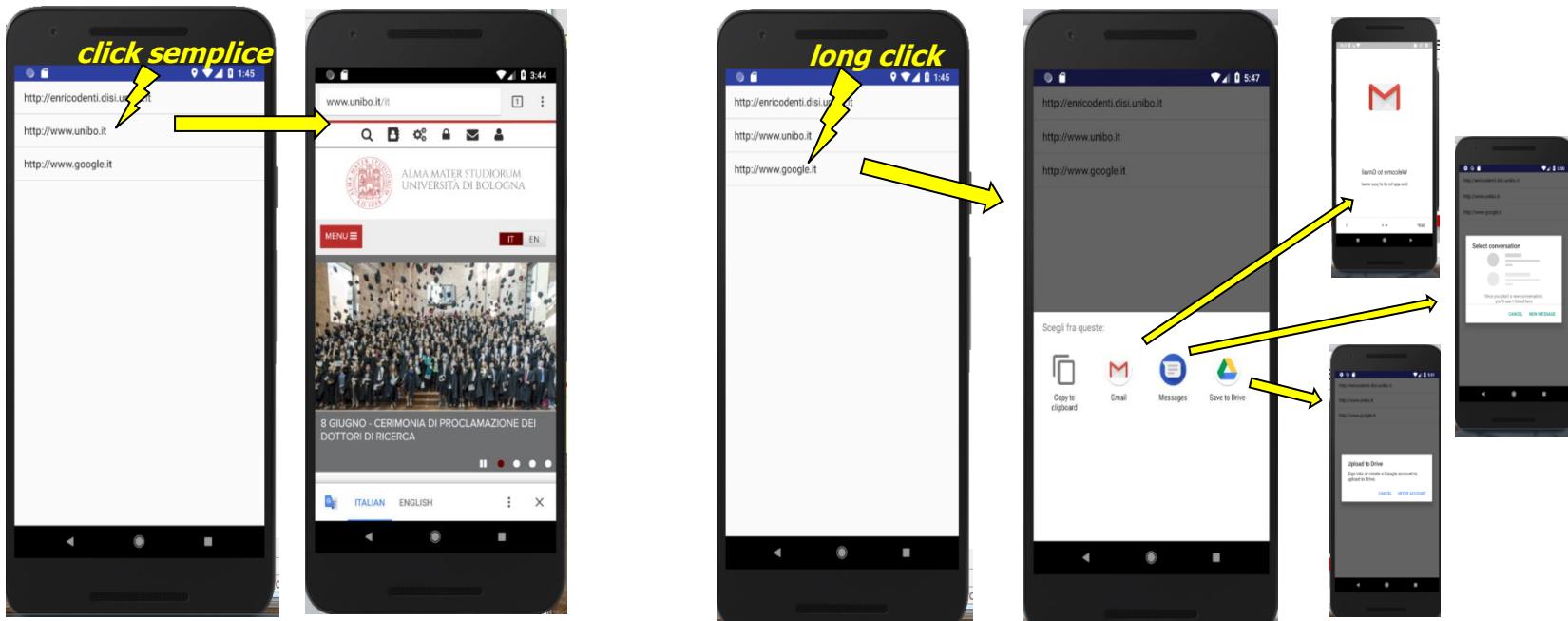


# ESEMPIO (1)

Obiettivo:

un'app che mostri a video una lista di indirizzi web, e

- scegliendone uno con il **click semplice**, *mostri la pagina web*
- scegliendone uno con il **click lungo**, *condivida l'URL*





# ESEMPIO (2)

Obiettivo:

un'app che mostri a video una lista di indirizzi web, e

- scegliendone uno con il **click semplice**, *mostri la pagina web*
- scegliendone uno con il **click lungo**, *condivida l'URL*





# ESEMPIO (3)

- Questa attività mostrerebbe a video una lista di voci...
- ... ma per ora non fa nulla, perché manca la gestione eventi.

```
public class MyActivity extends ListActivity {  
    protected void onResume() {  
        super.onResume();  
        setListAdapter(new ArrayAdapter<String>(  
            this,  
            android.R.layout.simple_list_item_1,  
            android.R.id.text1,  
            new String[] {  
                "http://enricodenti.disi.unibo.it",  
                "http://www.unibo.it",  
                "http://www.google.it"  
            }  
        ));  
    }  
    ... // da completare  
}
```

Estendiamo direttamente ListActivity  
(anziché genericamente Activity)

Cablate nel  
codice??



# ESEMPIO (4)

## Gestione del "click semplice"

- Il listener dell'evento "voce scelta" attiva una *MySimpleClickActivity*
- crea un Intent ESPLICITO contenente l'attività da attivare
- infine, lancia la nuova attività, passandole tale Intent

```
public class MyActivity extends ListActivity {  
    ... // parte precedente  
    protected void onListItemClick(ListView l, View v,  
                                    int pos, long id) {  
        Intent intent = new Intent(this, MySimpleClickActivity.class);  
        intent.putExtra("url", (String) getListAdapter().getItem(pos));  
        startActivity(intent);  
    }  
    ... // da completare  
}
```



# ESEMPIO (5)

La nuova attività *MySimpleClickActivity*

- è chiamata esplicitamente dall'attività principale, *per nome*
- recupera dall' Intent l'URL da aprire
- lo visualizza tramite il componente di sistema **WebView**

```
public class MySimpleClickActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        WebView webview = new WebView(this);  
        setContentView(webview);  
        String url = getIntent().getStringExtra("url");  
        setTitle(url);  
        webview.loadUrl(url);  
    }  
}
```



# ESEMPIO (6)

## Gestione del "click lungo"

- aggiungiamo un altro listener, separato, per l'evento **ItemLongClick**
- occorre ridefinire il "solito" metodo **onCreate** per agganciare alla ListView il suo listener – un **OnItemLongClickListener** a cui vengono passati l'URL e il messaggio da mostrare nel chooser

```
public class MyActivity extends ListActivity {  
    ... // parte precedente  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getListView().setOnItemLongClickListener(...);  
    }  
}
```

Il nostro listener (con gli opportuni argomenti)  
- una lambda o una classe esplicita



# ESEMPIO (7)

## Gestione del "click lungo"

- aggiungiamo un altro listener, separato, per l'evento `ItemLongClick`
- occorre ridefinire il "solito" metodo `onCreate` per agganciare alla `ListView` il suo listener – un `OnItemLongClickListener` a cui vengono passati l'URL e il messaggio da mostrare nel chooser
- il nostro listener si chiamerà **MyLongClickListener**

```
public class MyActivity extends ListActivity {  
    ... // parte precedente  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ListView lv = getListView();  
        lv.setOnItemLongClickListener(  
            new MyLongClickListener(this, getString(R.string.my_msg)));  
    }  
}
```

Recupero (dalle risorse) della stringa da mostrare, chiamata `my_msg`



# ESEMPIO (8)

## Gestione risorse

- aggiungere una stringa
- occorre creare un `StringResource` per ogni stringa che vengono inserite
- il nostro esempio

```
public class MainActivity extends ListActivity {  
    ... /.../  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getListAdapter();  
        new MyLongClickListener();  
    }  
}
```



Risorse stringa: my\_msg

... creato, per l'evento `ItemLongClick` nel `onCreate` per agganciare alla variabile `myLongClickListener` a cui si accede da mostrare nel chooser per `ItemLongClickListener`

```
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        myLongClickListener = new MyLongClickListener();  
    }
```

```
    public void onItemLongClick(AdapterView parent, View view, int position, long id) {  
        String msg = savedInStringList.get(position);  
        Intent intent = new Intent(Intent.ACTION_SEND);  
        intent.setType("text/plain");  
        intent.putExtra(Intent.EXTRA_TEXT, msg);  
        startActivity(intent);  
    }  
}  
  
class MyLongClickListener implements AdapterView.OnItemLongClickListener {  
    @Override  
    public boolean onItemLongClick(AdapterView parent, View view, int position, long id) {  
        String msg = savedInStringList.get(position);  
        Intent intent = new Intent(Intent.ACTION_SEND);  
        intent.setType("text/plain");  
        intent.putExtra(Intent.EXTRA_TEXT, msg);  
        startActivity(intent);  
        return true;  
    }  
}
```

Con un clic, si apre la finestrina per inserire tale nuova risorsa-stringa



# ESEMPIO (9)

---

Cosa deve fare questo listener?

- vogliamo che la reazione al *long touch* sia *condividere l'URL*

Poiché l'azione "condividere" può essere recepita da svariate app, è opportuno *NON cablarne il nome nel codice*

- il listener dovrà perciò costruire un intent IMPLICITO che esprima il desiderio di "*condividere un elemento*"
- l'intent standard **Intent.ACTION\_SEND** fa esattamente questo ☺

Sintassi di `OnItemLongClickListener`

- metodo `onItemLongClick (AdapterView<?> adapterView,  
View view, int pos, long id)`
- in alternativa: una più comoda *lambda con method reference* ☺



# ESEMPIO (10)

Senza lambda:

```
public class MyLongClickListener implements OnItemLongClickListener {  
    private String myMsg;  
    private Activity myMainAct;  
    public MyLongClickListener(Activity myMainAct, String myMsg) {  
        this.myMsg = myMsg;  this.myMainAct = myMainAct;  
    }  
    public boolean onItemLongClick(AdapterView<?> adapterView,  
                                   View view, int pos, long id) {  
        final Intent intent = new Intent(Intent.ACTION_SEND);  
        intent.setType("text/plain");  
        intent.putExtra(Intent.EXTRA_TEXT,  
                      (String) adapterView.getItemAtPosition(pos));  
        myMainAct.startActivity( Intent.createChooser(intent, myMsg) );  
        return true;  
    }  
}
```

Messaggio da mostrare nel chooser

Azione da svolgere

Dati extra per l'intent

Attivazione di una ulteriore attività: un chooser per scegliere fra tutte le attività capaci di gestire l'ACTION\_SEND



# ESEMPIO (11)

## Descrizione XML: dichiara le attività e i permessi

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="it.unibo.disi.enricodentini.mybrowserapp">
4     <uses-permission android:name="android.permission.INTERNET"/> Essenziale
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="MyBrowserApp"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12         <activity
13             android:name=".MyActivity"
14             android:label="MyBrowserApp"
15             android:theme="@style/AppTheme.NoActionBar">
16             <intent-filter>
17                 <action android:name="android.intent.action.MAIN" />
18                 <category android:name="android.intent.category.LAUNCHER" />
19             </intent-filter>
20         </activity>
21         <activity android:name=".MySimpleClickActivity"/> Attività secondaria
22     </application> Attività principale
23
24 </manifest>
```



# ESEMPIO (12)





# ESEMPIO (13)





# RIFERIMENTI

---

Ci sono molti libri, in inglese e in italiano:

- spesso purtroppo strutturati come ultra-prolissi manuali step-by-step
- spesso non pensati *per chi conosce già Java*
- *cercate magari risorse "for Java developers"*

## Fast track for Java developers

- <https://www.slideshare.net/mswolfson/android-for-javascripters-4349928>
- <http://developer.android.com/training/basics/firstapp/index.html>
- <http://android.devapp.it/t003-usiamo-activity-e-intent-nelle-nostre-applicazioni-android>
- <http://www.vogella.com/tutorials/Android/article.html>