

SUPSI

Introduzione allo sviluppo Android

Sviluppo di Applicazioni Mobile

Vanni Galli, lecturer and researcher SUPSI

Obbiettivi

- Le lezioni del corso per Android hanno i seguenti obbiettivi:
 - Conoscere la **struttura** delle applicazioni Android
 - Comprendere il **ciclo di vita** delle applicazioni
 - Imparare a disegnare **UI responsive** con l'uso di **fragment**
 - Saper distinguere ed utilizzare gli **explicit** e gli **implicit intent**
 - Familiarizzare con:
 - L'MVC su Android
 - Dialogs e Toolbar
 - SQLite
 - Localizzazione
 - ...

Prerequisiti

- Per seguire la parte Android occorre avere familiarità con Java, in particolare:
 - Classi e oggetti
 - Interfacce
 - Listeners
 - Packages
 - Inner classes
 - Anonymous inner classes
 - Generics

Tools necessari

- Eseguiremo lo sviluppo di applicazioni con Android Studio
- Un'installazione di Android Studio comprende:
 - Android SDK
 - Android SDK tools and platform tools
 - Un'immagine per l'emulatore Android
- Per chi lo dispone è consigliato anche l'utilizzo di un device reale

Versioni di Android

- Le versioni di Android maggiormente usate al giorno d'oggi vanno dalla 7.0 (*Nougat*) alla 11.0 (*R*)
- Il codice che svilupperemo sarà (principalmente) retro-compatibile fino alla versione 4.4 (*KitKat*)
- Consideriamo una versione "moderna" se le API sono ≥ 19

Versioni di Android

- I principali SDK in circolazione (dati di gennaio 2023):

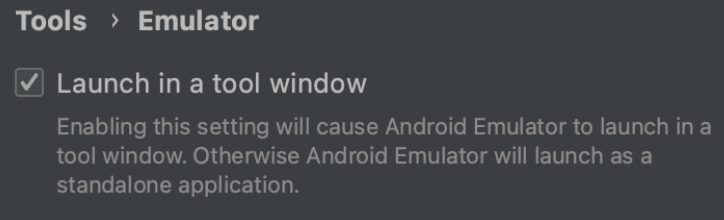
| ANDROID PLATFORM VERSION | | API LEVEL | CUMULATIVE DISTRIBUTION |
|--------------------------|-------------|-----------|-------------------------|
| 4.4 | KitKat | 19 | |
| 5.0 | Lollipop | 21 | 99,3% |
| 5.1 | Lollipop | 22 | 99,0% |
| 6.0 | Marshmallow | 23 | 97,2% |
| 7.0 | Nougat | 24 | 94,4% |
| 7.1 | Nougat | 25 | 92,5% |
| 8.0 | Oreo | 26 | 90,7% |
| 8.1 | Oreo | 27 | 88,1% |
| | | | 81,2% |
| 9.0 | Pie | 28 | |
| | | | 68,0% |
| 10. | Q | 29 | |
| | | | 48,5% |
| 11. | R | 30 | |
| | | | 24,1% |
| 12. | S | 31 | |
| | | | 5,2% |
| 13. | T | 33 | |

Java e/o Kotlin

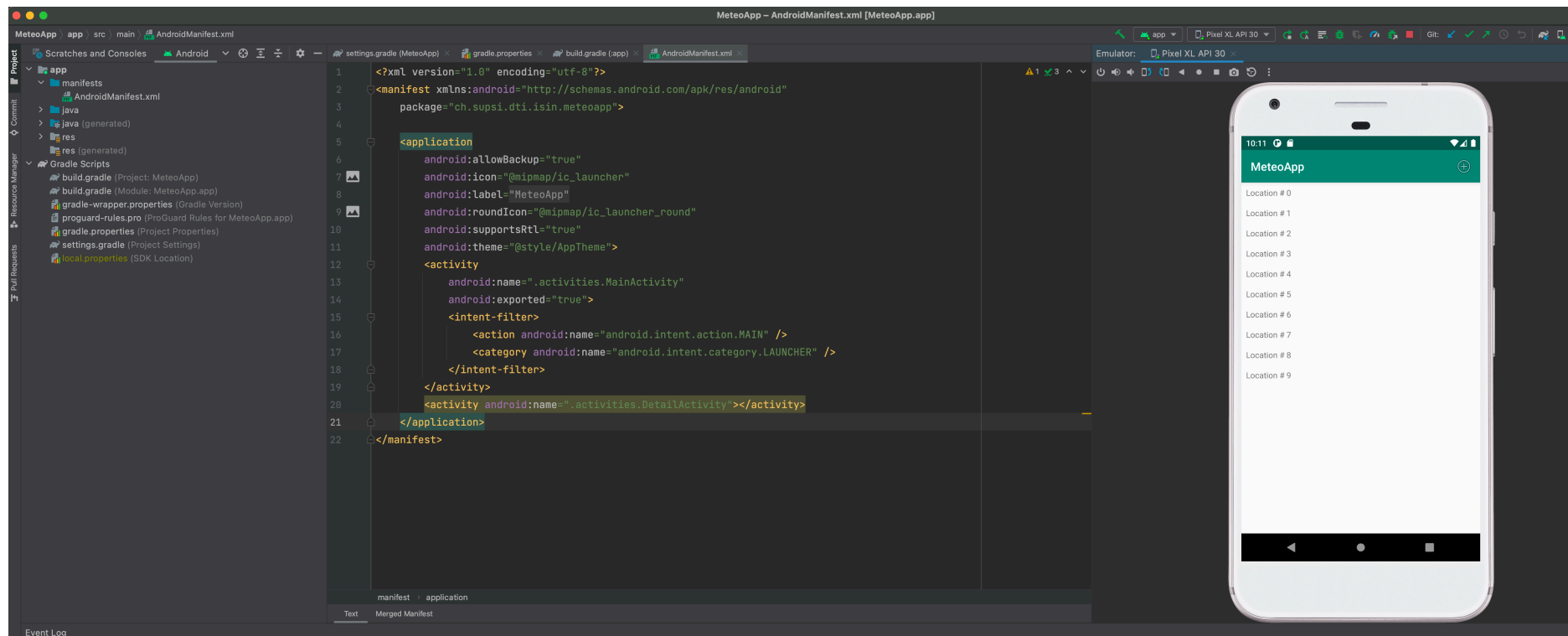
- Kotlin è un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains
- Il linguaggio Kotlin è stato adottato e integrato nell'ambiente di sviluppo Android Studio a partire dalla versione 3.0
- In un progetto, è possibile scrivere codice in Kotlin (file con estensione *.kt*) e in Java assieme
- Non è possibile “mischiare” le sintassi all'interno dello stesso file
- Il corso verrà tenuto in Java, ma gli studenti possono usare Kotlin a piacimento

Lanciare l'applicazione nell'emulatore

- Le applicazioni sviluppate in Android Studio possono essere lanciate su di un device reale oppure su di uno virtuale
- I devices virtuali utilizzano l'emulazione del sistema Android
- È possibile crearli dal menu Tools → AVD Manager
- Dalla versione Bumblebee di Android Studio l'emulatore è di default "integrato" nell'IDE; può però essere "riportato" in una finestra separata nei Settings (Tools > Emulator)



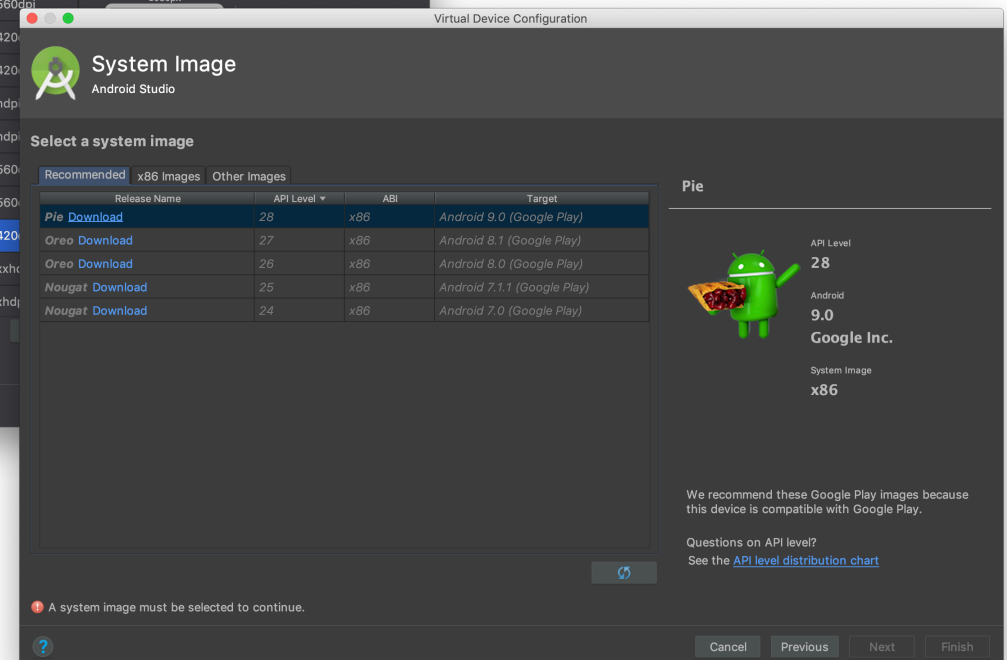
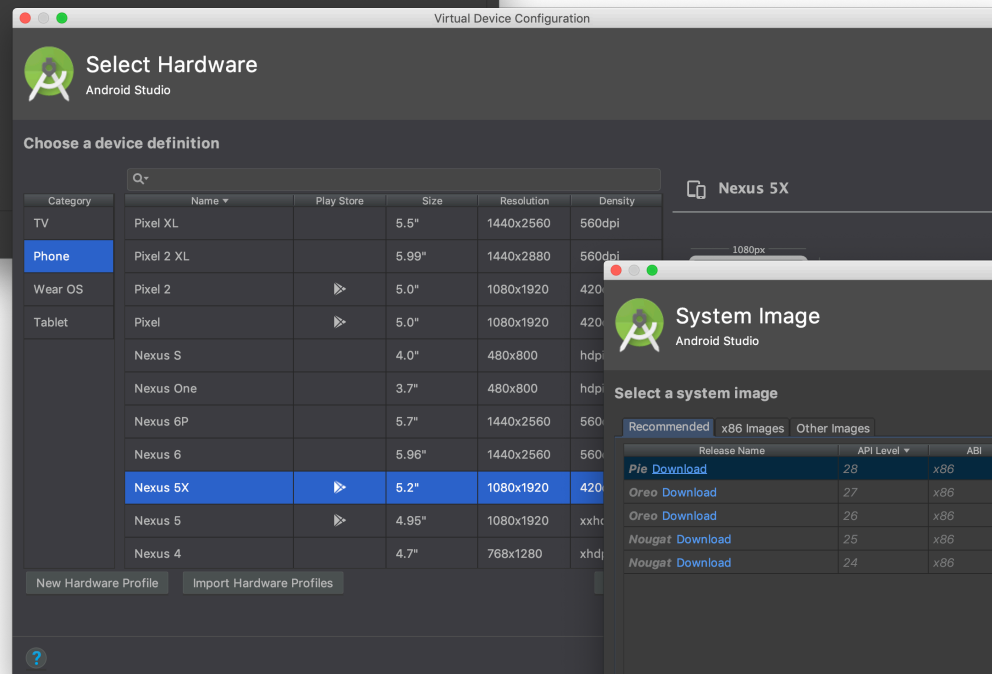
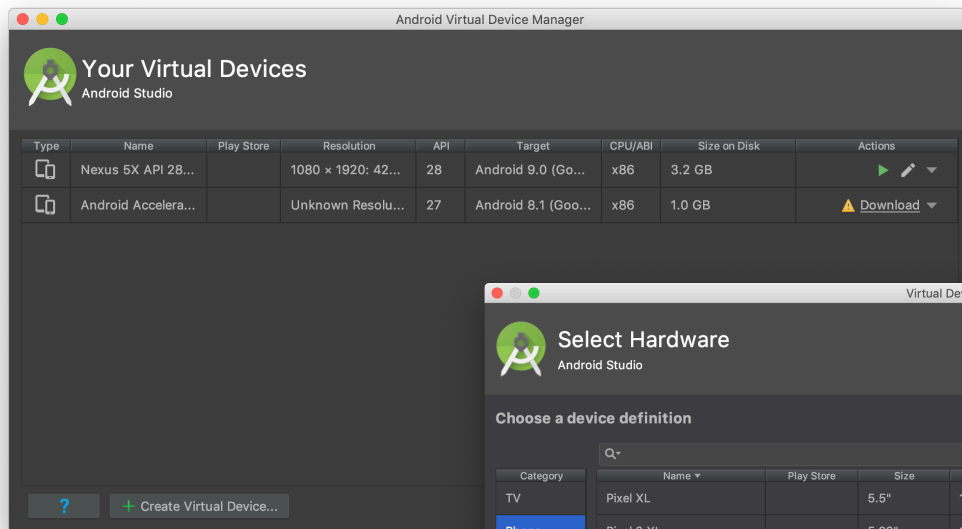
Lanciare l'applicazione nell'emulatore



Lanciare l'applicazione su di un device reale

- È possibile lanciare la propria applicazione su di un device reale
- Su Mac il device viene riconosciuto automaticamente
- Su Windows può essere necessario installare l'*Android Debug Bridge (adb)*
- Prima di poter installare le applicazioni su di un device, occorre attivare il modo di debug (di norma cliccando 7 volte sul *Build Number* nei *Settings*)
- Per le istruzioni si veda developer.android.com/studio/debug/dev-options
- Ulteriori informazioni sono reperibili anche presso developer.android.com/tools/device.html

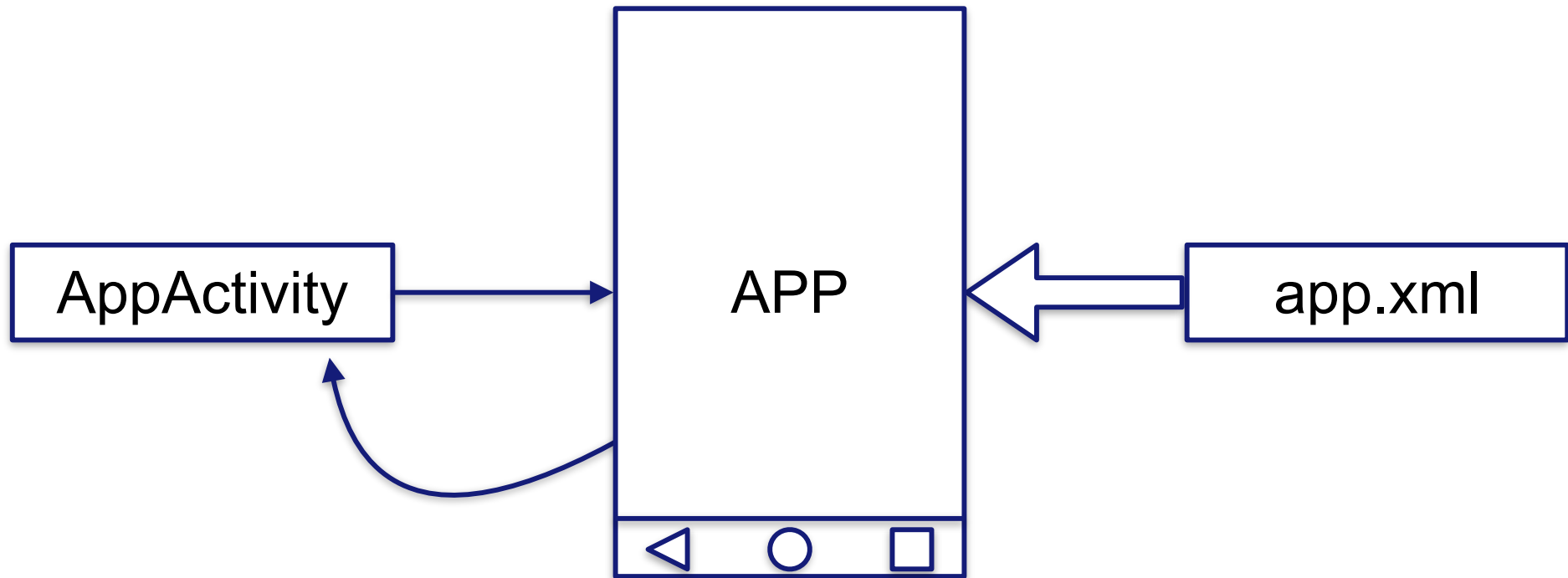
Creazione di un device virtuale



Activities e Layouts

- Ogni applicazione Android è composta da (almeno) un'activity e da un layout
- Un **activity** è un'istanza di Activity (una classe dell'SDK)
 - Un'activity è responsabile di gestire l'interazione utente
 - Scriviamo una sottoclasse di Activity, implementando le funzionalità richieste dall'App
- Un **layout** definisce un set di oggetti UI e la loro posizione sullo schermo
 - La definizione è scritta in XML

Activities e Layouts



I Widgets

- I Widgets sono i “building blocks” su cui si compone la UI.
- Un widget può mostrare del testo o della grafica, può interagire con l’utente, ...
- L’SDK include già molti widget che possono essere configurati a piacimento
- Ogni widget è un’istanza della classe View, o di una delle sue sottoclassi

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <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" />

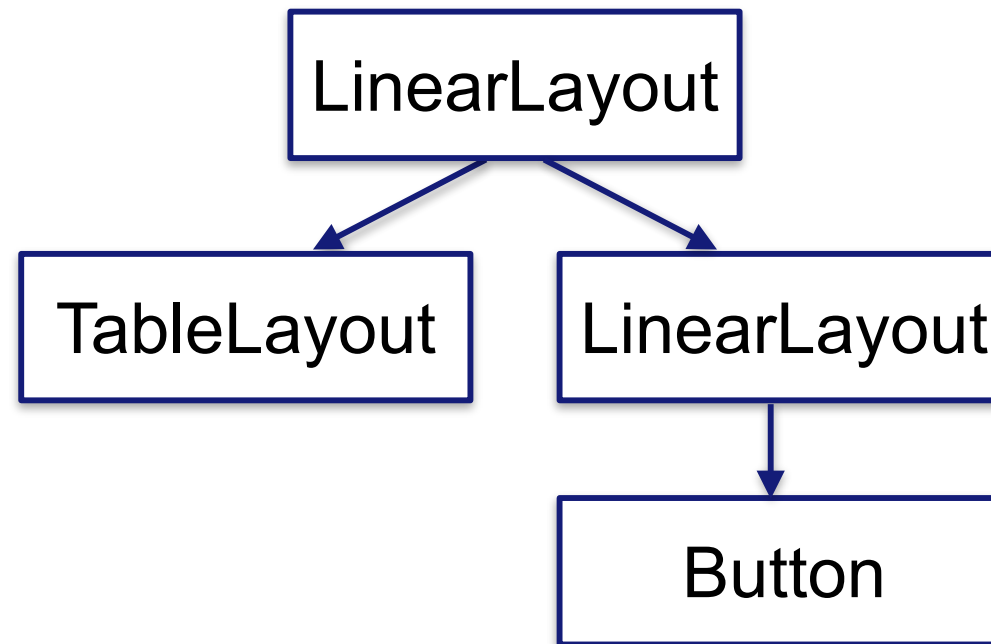
</android.support.constraint.ConstraintLayout>
```

Definizione dei Widgets



Gerarchia della View

- I widget esistono all'interno di una gerarchia (*view hierarchy*)
- Ad esempio:

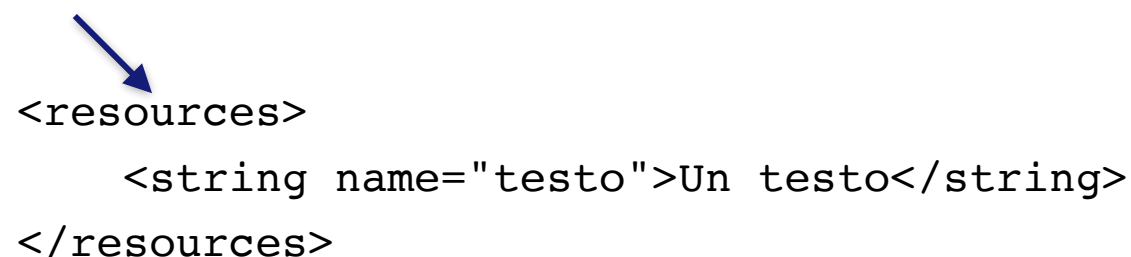


String resources

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="24dp"  
    android:text="@string/testo" />
```

posso definire le stringhe
in un qualsiasi file *.xml* in
/res/values

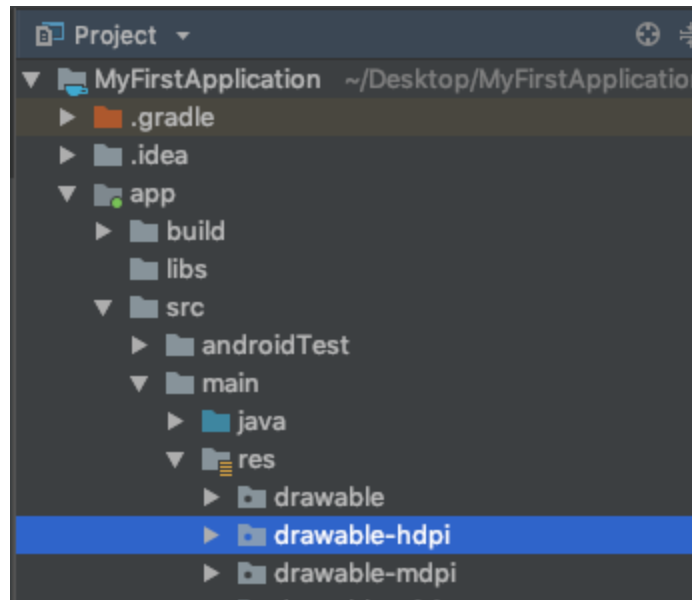
si rifà alla stringa definita
in *strings.xml*



```
<resources>  
    <string name="testo">Un testo</string>  
</resources>
```

Immagini e icone

- Le immagini e le icone usate nella propria applicazione possono essere aggiunte come risorse al progetto
- È possibile copiare immagini o cartelle direttamente da file system; in Android Studio è poi possibile spostarsi nella visualizzazione Project e incollare le risorse appena copiate:



Immagini e icone

- Normalmente, immagini e icone sono presenti in diverse risoluzioni, ad esempio:
 - mdpi: medium-density screens (~160dpi)
 - hdpi: high-density screens (~240dpi)
 - xhdpi: extra-high-density screens (~320dpi)
 - xxhdpi: extra-extra-high-density: screens (~480dpi)
- Quando viene eseguita un'applicazione, il sistema operativo sceglie l'immagine migliore per il device che la sta eseguendo
- Troppe copie della stessa immagine possono ovviamente far crescere la dimensione dell'applicazione
- È quindi possibile fornire all'applicazione solamente le immagini in risoluzione più alta, l'OS si occuperà poi di fare un down scaling per le risoluzioni inferiori

Immagini e icone

- Le immagini e le icone possono venire referenziare direttamente in un layout

```
<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp" />
```

uso la keyword drawable


i nomi dei file devono essere lowercase e senza spazi!

Dal Layout alla UI

- Come fanno gli elementi XML a diventare degli oggetti nella View?
- La risposta è nella Activity:

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

“inflate” del layout:
ogni widget è istanziato in
base ai suoi attributi



nome del layout



Resources e Resources ID

- Una resource è un pezzo della propria applicazione che non è codice (immagini, video, audio, file XML, ...)
- Per accedere ad una risorsa occorre accedervi con il relativo ID
- A livello di layout, gli ID vengono definiti direttamente nei vari widget che lo compongono

Resources e Resources ID

- Android auto-genera degli ID per varie risorse (come le View)
- Non genererà però ID per i singoli widget
- Questi vanno definiti manualmente:

```
<Button  
    android:id="@+id/my_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/false_button" />
```

Referenziare i widgets nel codice dell'Activity

- Una volta definiti gli ID per i vari widgets dell'applicazione, è possibile referenziare i widgets direttamente nel codice dell'applicazione
- Tipicamente i widgets vengono referenziati da un Activity legata ad un Layout:

```
public class TestActivity extends AppCompatActivity {  
  
    private Button mButton;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
        mButton = (Button) findViewById(R.id.my_button);  
    }  
}
```


Listeners

- Le applicazioni Android sono (tipicamente) *event driven*
- L'applicazione parte e rimane in attesa di un evento (i.e. un'azione fatta da utente)
- Se l'applicazione è in attesa di un evento, si dice che è in “listening”
- L'SDK Android mette a già disposizione delle interfacce per molti eventi, ad esempio per un click su di un bottone:

```
mButton.setOnClickListener(new View.OnClickListener()  
{  
    @Override  
    public void onClick(View v) {  
        // Does nothing yet  
    }  
});
```

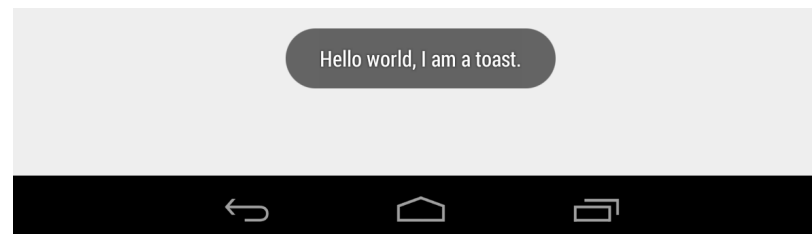
Listeners e anonymous inner classes

- Durante il corso i listeners verranno implementati, in generale, usando delle anonymous inner classes
- In questo modo tutto il codice dei listeners rimane in un unico posto
- Non vi è la necessità di istanziare nuove classi
- Gli studenti possono comunque implementare i listeners usando la tecnica che preferiscono

Toast

- Dalla documentazione Android: *“A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.”*
- I toasts possono essere molto utili per informare l'utente di quello che sta succedendo, senza dover richiedere un'ulteriore azione da parte dell'utente
- Possono essere creati con il seguente metodo statico:

```
public static Toast makeText(Context context, int resId, int duration)
```



Toast - Esempio

```
Toast.makeText(  
    MyActivity.this, // NON this!  
    R.string.my_toast,  
    Toast.LENGTH_SHORT).show();
```

se sono all'interno di un listener, passo l'istanza dell'attività (e NON della inner class)

resource ID della stringa da mostrare

Logging

- La classe *android.util.Log* può essere usata per fare il logging su di un log condiviso di sistema
- La classe implementa metodi diversi per livelli di log diversi, ad esempio:

```
public static int d(String tag, String msg)
```

- Il primo parametro (*tag*) è tipicamente una costante contenente il nome della classe corrente; con il tag si vuole facilitare l'identificazione della source del log

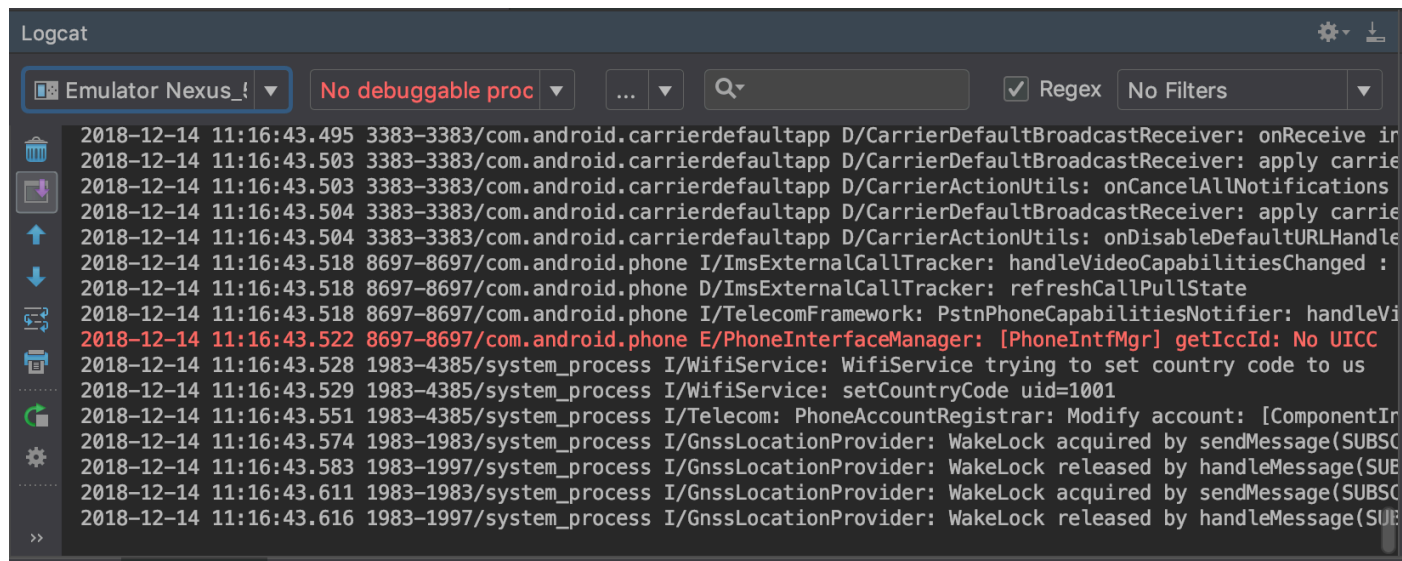
Logging di Exception

- Ogni metodo della classe Log ha 2 signatures: uno riceve il TAG e la stringa da loggare, mentre l'altro riceve anche un'istanza di *Throwable*
- In questo modo è facile loggare le informazioni che sono relative ad una particolare *Exception*

```
try {  
    myValue = myArray[myIndex];  
} catch (ArrayIndexOutOfBoundsException ex) {  
    Log.e(TAG, "Index was out of bounds", ex);  
}
```

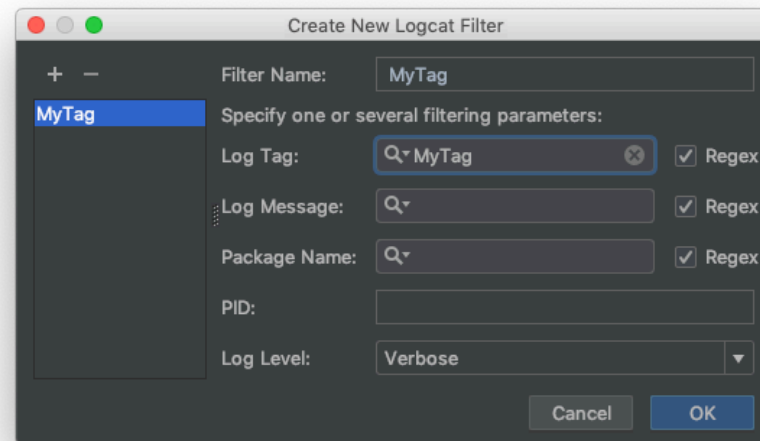
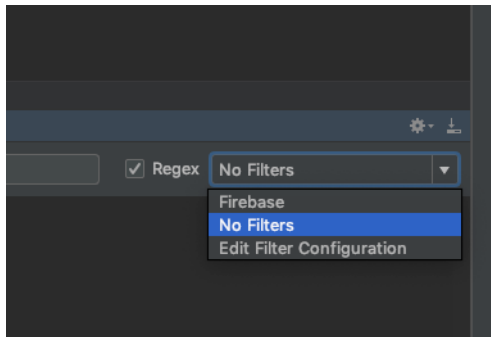
Il Logcat

- Importanti informazioni sull'esecuzione del programma (in particolare in caso di crash) si possono trovare nel Logcat di Android Studio:



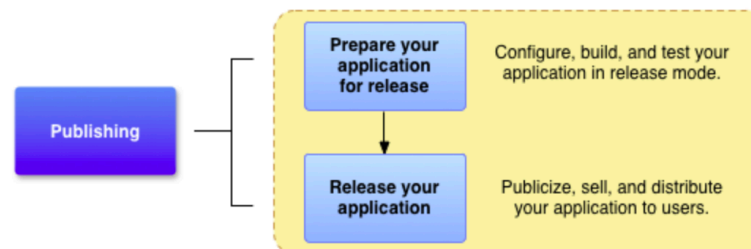
Filtrare il Logcat

- In modo da ritrovare più facilmente i messaggi loggati è creare dei filtri nel Logcat
- Il filtro va creato tramite l'apposito bottone in alto a destra nel Logcat:



Il processo di build

- Durante il processo di build, il compilatore prende le risorse, il codice e il file `AndroidManifest.xml` e li “trasforma” in un file `.apk`
- Il file viene poi segnato con una chiave di debug, per permettere l’esecuzione sull’emulatore (o su un device personale)
- Per rilasciare un’applicazione su larga scala, occorre segnlarla con una chiave per il release
- Maggiori informazioni si trovano presso developer.android.com/tools/publishing/preparing.html



Tools command-line

- È possibile anche compilare le applicazioni in modo manuale
- Gli ambienti di sviluppo moderni (come Android Studio) usano un tool chiamato *Gradle*
- Per usare Gradle da command line occorre spostarsi, con il terminale, nella directory di progetto e lanciare:

```
./gradlew installDebug
```

- Ulteriori informazioni sulle possibilità fornite da Gradle si possono ottenere con:

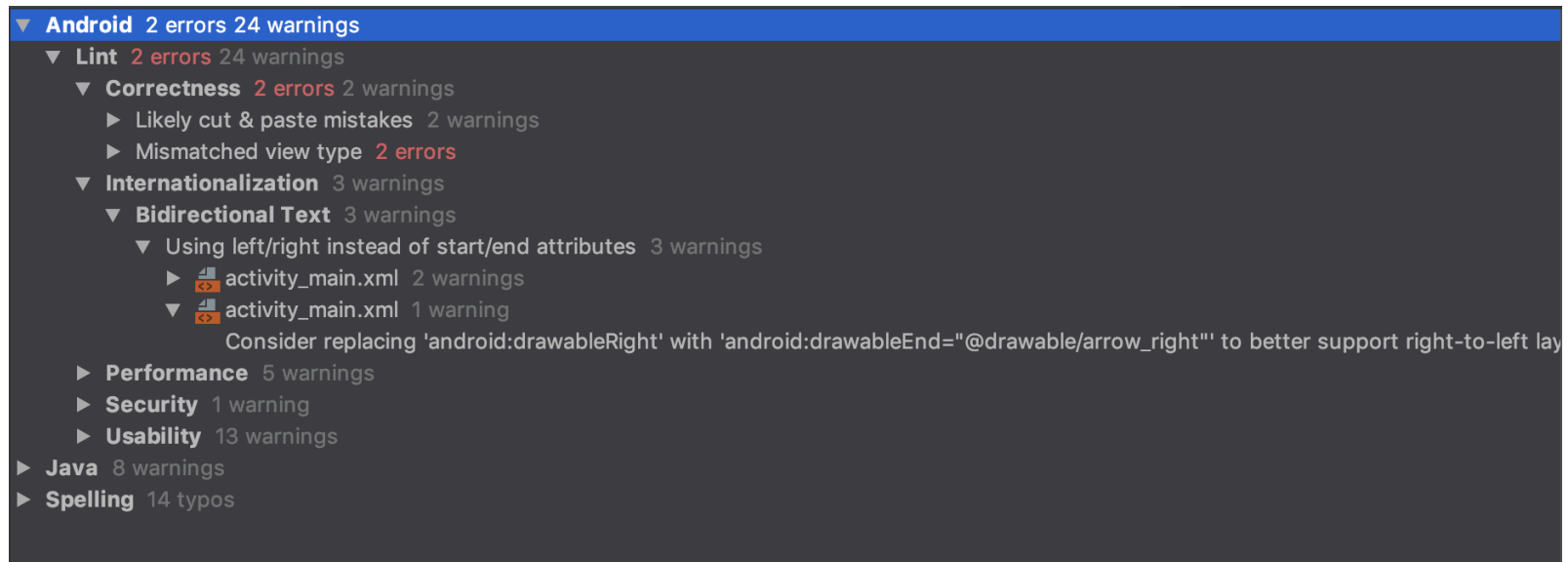
```
./gradlew tasks
```

Il debugging su Android

- Il debugging su Android segue principalmente quello delle applicazioni Java "comuni"
- Ci possono però essere degli errori derivanti da "parti specifiche" di Android (come ad esempio nel caso di una risorsa mancante)
- In questi casi si può usare *Android Lint*
- Lint è un analizzatore statico del codice Android
- Lint può essere invocato dal menu Analyze → Inspect Code...

Android Lint

- Lint usa la sua "conoscenza" del framework di Android per cercare più in profondità nel codice e per scovare potenziali problemi che il compilatore non trova

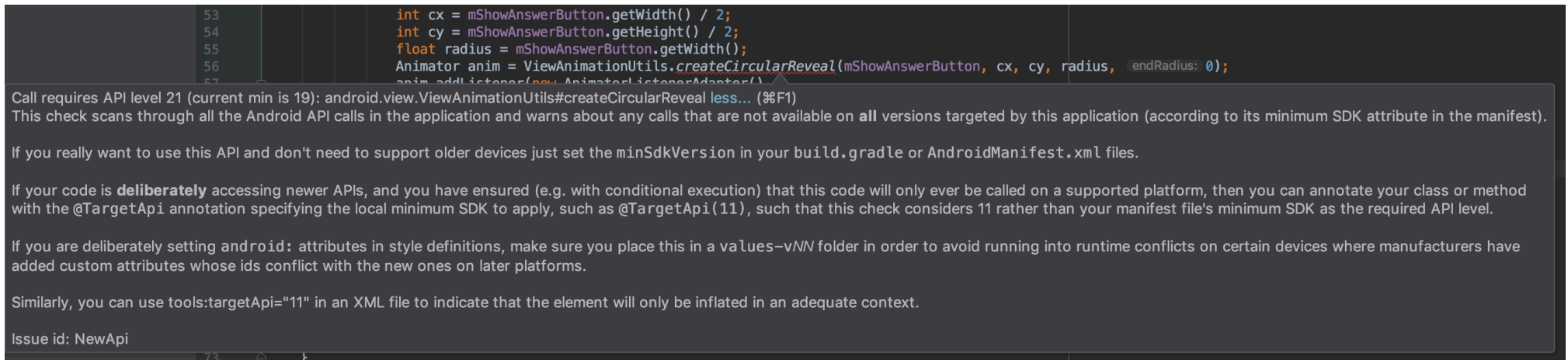


Retro-compatibilità

- Per gestire la (retro-)compatibilità (vedi slides 5-6) è possibile agire su tre parametri (tutti indicati nel file *build.gradle*):
- **minSdkVersion**: definisce il livello di API sotto la quale Android si rifiuta di installare l'applicazione
- **targetSdkVersion**: indica ad Android il livello di API con cui far girare l'applicazione (nuove API possono modificare, ad esempio, l'aspetto dei widgets; un'applicazione può quindi girare con un API più bassa)
- **compileSdkVersion**: indica la versione del compilatore per fare il *build* dell'applicazione (e di solito si usa la più recente)

Retro-compatibilità

- Il compilatore non si accorge di potenziali problemi legati alla compatibilità (l'importante per il compilatore è che la `compileSdkVersion` sia ok)
- Lint viene in aiuto (prima di compilare) anche in queste eventualità:



Retro-compatibilità

- Problemi segnalati da Lint possono essere risolti ad esempio nel seguente modo:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {  
    // codice per le nuove versioni  
} else {  
    // codice per le vecchie versioni  
}
```

- L'*Android Developer Documentation* indica per ogni classe e metodo da che API sono disponibili:

ViewAnimationUtils

added in API level 21

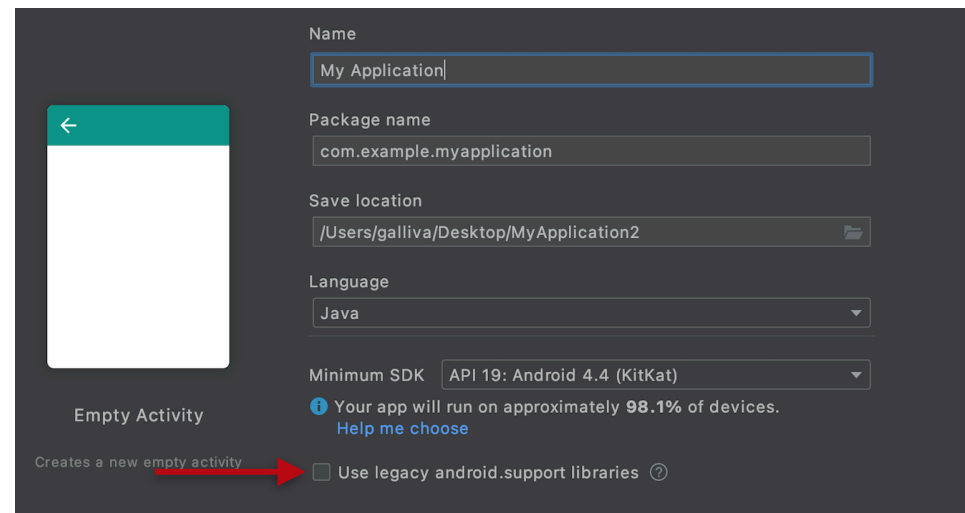
```
public final class ViewAnimationUtils  
extends Object
```

[java.lang.Object](#)

↳ android.view.ViewAnimationUtils

Jetpack e AndroidX

- *Android Jetpack Components*, chiamato *Jetpack* in breve, è un insieme di librerie create da Google per facilitare vari aspetti dello sviluppo Android
- Ciascuna delle librerie Jetpack si trova in un pacchetto che inizia con *androidx*
- Per questo motivo i termini "AndroidX" e "Jetpack" sono intercambiabili
- Rimane possibile (e sconsigliato) usare le vecchie librerie support



Live Templates

- Quali principianti nello sviluppo Android si è spesso confrontati con una sintassi che "non si ricorda"
- Un importante aiuto può essere dato dai *Live Templates* di Android Studio (disponibili sotto *Preferences...*)
- È possibile definire degli "snippet" di codice da richiamare facilmente mentre si scrive il codice
- Una volta nell'editor, basta iniziare a digitare l'abbreviazione dello snippet per poterlo poi inserire

