Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Cervello e computer: bellezza e segreti dei bit di tutti i giorni

cybersecurity • intelligenza artificiale • realtà virtuale • Android

Ciclo di seminari di divulgazione informatica
in collaborazione con NOVA a.p.s.

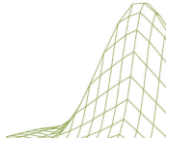
Ferrara, 8-12 Giugno 2020

Programma

Lunedì 8 Giugno: Carlo Giannelli
Cyber security: istruzioni per l'uso – Principi di sicurezza informatica

Martedì 9 Giugno: Guido Sciavicco
Come imparano le macchine – Principi di intelligenza artificiale

Mercoledì 10 Giugno: Marco Alberti
Neuroni di bit – Reti neurali e applicazioni



Giovedì 11 Giugno: Antonino Casile
Informatica e percezione sensoriale – L'ultima frontiera della realtà virtuale

Venerdì 12 Giugno: M. Roma, G. Turri, L. Travaglia – NOVA Ferrara
La nascita di un'app Android – Programmazione in Android

Disclaimer

- La presentazione, il filmato, i materiali e i contenuti in essi inclusi sono di proprietà dell'Università di Ferrara
- Il diritto morale d'autore ("Proprietà Intellettuale") appartiene ai singoli docenti/relatori dell'evento
- L'utilizzo è concesso per uso esclusivo e personale
- Nessun altro utilizzo può essere legittimamente esercitato senza la previa autorizzazione scritta dell'Ateneo e dei proprietari del diritto morale d'autore
- Qualunque abuso verrà perseguito a norma di legge
- Per ulteriori informazioni visitare il sito dmi.unife.it/stageInformatica

Università
degli Studi
di Ferrara

Nel futuro da sempre

NOVA

Progetto Lauree Scientifiche
Unife orienta 2020

Stage di orientamento per le Scuole superiori

12/06/2020

Dipartimento
di Matematica
e Informatica

NOVA

Come nasce un'app Android?

Breve tutorial alla creazione della prima (o seconda) app

Che cosa vedremo

- **Introduzione**
  - ❑ Mobile computing
  - ❑ App ibride o native?
  - ❑ Elementi fondamentali
  - ❑ Funzionamento e risorse
  - ❑ Layout
- **Linguaggio: Kotlin**
  - ❑ Fondamenti
  - ❑ Ciclo di vita
  - ❑ Classe R
- **Laboratorio**
- **Pubblicazione**

Mobile eXperience

- Cos'è secondo voi uno smartphone? Ormai tutto tranne un telefono!
  - Questo perché è in realtà un **agglomerato di sensori e dati** che portiamo continuamente in giro, o che lasciamo in Cloud a cui accedere da ogni luogo.
- Personalizzare l'esperienza con usabilità (User eXperience - UX)
  - Le applicazioni devono tenere conto della **fluidità** e dello **schermo ridotto** prima di tutto! Poi della **banda limitata,** del **consumo di batteria** e altri problemi legati all'elettronica e alle telecomunicazioni (es. disconnessioni)
- Mai dimenticare la **sicurezza** e la ricchezza che generano i dati
  - Solitamente non si percepisce lo stesso pericolo che hanno i virus nel PC, ma i dati sensibili sono troppo spesso a rischio!
- Mobile, Nomadic, ad hoc, Ubiquitous, Spontaneous. Quanti computing!
  - Abbiamo elaborazione e raccolta dati ovunque, negli orologi, auto, spazzolini (vedi Bluetooth della OralB e Apple), nei sensori. Questi possono essere impostati per **comunicare o agire spontaneamente**, senza riconfigurazioni

App ibride vs native

- **App native:**
  - A ogni sistema operativo la sua app
  - Scritte in un linguaggio specifico (es. Kotlin, Swift)
  - Più veloci e reattive anche sui dispositivi più lenti
- **App ibride:**
  - Visivamente simili alle app native
  - Necessitano del WebView in quanto sono scritte come pagine web (HTML, CSS, JavaScript, ecc.)
  - Sono portabili, ovvero lo stesso codice può generare app per dispositivi diversi (es. Android e iOS) semplicemente cambiando il compilatore
  - Sono più economiche da sviluppare e da mantenere
  - Apache Cordova è il framework più comunemente usato per creare app ibride partendo da standard web e creando un "pacchetto" pubblicabile su uno store
  - Da non confondere con le Web App!

Ambiente Android

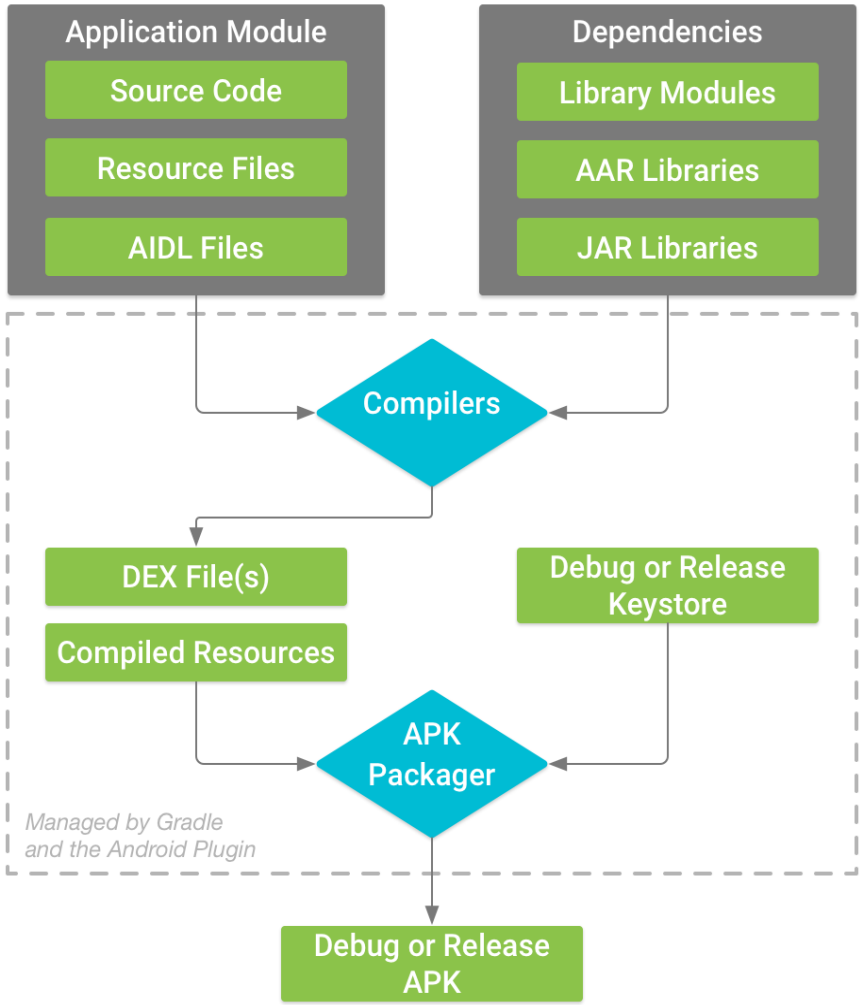
- Prodotto della Google
- Principale SO per mobile al mondo (circa ¾ dei dispositivi)
  - ❑Smartphone e tablet
  - ❑Wear
  - ❑Android TV
  - ❑Cars
  - ❑Things
- Mondo Open Source, figlio di Linux
  - ❑Grande platea di sviluppatori nel bacino dell'open source

Ambiente Android: l'SDK e Android Studio

- L'ambiente di sviluppo Android è noto come **Android SDK**
  - ❑Raccoglie strumenti di sviluppo quali: programmi, emulatori, piattaforme per ciascuna delle versioni di Android, e ha la peculiarità di poter essere gestito e personalizzato attraverso SDK Manager
- Ci si potrebbe interfacciare anche direttamente
- Fortunatamente c'è **Android Studio** come IDE
  - ❑Organizza e gestisce in modo intuitivo e automatizzato l'SDK
  - ❑Nato dalla stessa Google e pensato appositamente per sviluppo Android in tutti gli ambienti in cui il SO si può inserire
  - ❑Accesso e personalizzazione dell'SDK
  - ❑Android Virtual Device integrato
  - ❑Inline debugging

Ambiente Android: Gradle

- Gradle è un tool avanzato di build automation
- Si occupa delle diverse attività che permettono lo sviluppo di un software:
  - ❑Compilazione: da sorgente a binario per la macchina
  - ❑Packaging dei binari (es. APK)
  - ❑Test automatizzati per lo sviluppo
  - ❑Deployment: quando pronto e testato, il sw va configurato sui sistemi per i quali è implementato
  - ❑Generazione della documentazione
  - ❑Gestione automatica e flessibile delle dipendenze e delle varie versioni



https://developer.android.com/studio/build

Aspetti fondamentali

- **Fluidità**

Ciò che l'utente si aspetta è un'interfaccia reattiva. L'esperienza utente senza ritardi è il primo pensiero per il successo dell'app. Questo affronta restrizioni che provengono dalla bassa disponibilità delle risorse (soprattutto della RAM) e si risolvono con una buona gestione delle priorità dei processi (multithreading).

- **Riuso**

Tecnica principale per ottenere la fluidità. Necessità che nasce dalla limitata quantità di risorse tipiche di un ambiente mobile.

- **Stabilità**

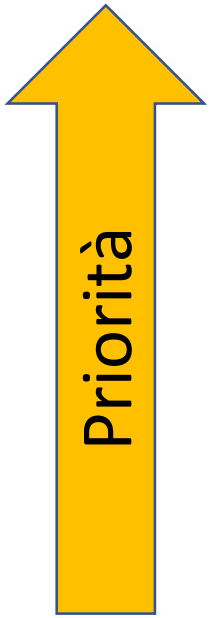
Ogni applicazione lavora in uno spazio di lavoro isolato ed indipendente così da non causare problemi al resto del sistema in caso di crash.

Va evitata qualsiasi pratica che porti un'app ad invadere lo spazio di memoria riservato ad un'altra.

La corretta gestione delle risorse

- Per mantenere la user experience fluida, il sistema operativo può scegliere di terminare alcuni processi non ritenuti necessari in base a **5 livelli di priorità** che definiscono il tipo di processo**:**

  - **foreground**: sono quelli che interagiscono direttamente o indirettamente con l'utente (contengono l'Activity attualmente utilizzata o i Service ad essa collegati)
  - **visibili:** contengono componenti ancora visibili all'utente anche se non vi sono interazioni. Svolgono comunque un ruolo particolarmente critico
  - **service**: contengono dei servizi in esecuzione che generalmente svolgono lavori molto utili all'utente anche se non direttamente collegati con ciò che egli vede nel display
  - **background**: contengono activity non più visibili all'utente. Questa è una categoria solitamente molto affollata composta dal gran numero di applicazioni precedentemente chiuse (buoni candidati all'eliminazione)
  - **empty**: non hanno alcuna componente di sistema attiva. Conservati solo per motivi di cache, per velocizzare la loro riattivazione



Componenti di una app

- **Activity** (UI)
- **Service** (background, reattività)
- **Content Provider** (Sharing tra app)
- **Intent**
- **Broadcast Receiver** (Target, System errors notification)

Activity

- Una Activity rappresenta **l'interfaccia utente** per un particolare task che l'utente può effettuare. Ogni volta che si usa un'app generalmente si interagisce con una o più "pagine" (es. una schermata di login, una di impostazioni, un feed social, ecc...) mediante le quali si consultano dati o si immettono *input*.
- Essendo una singola vista, è formata da una o più **View**, creando insieme la User Interface

Service

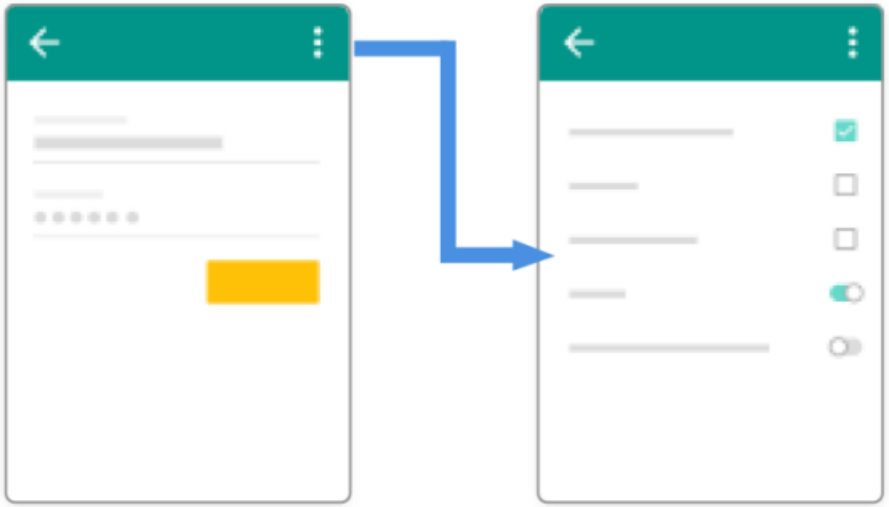
- E’ un processo generalmente in **background**, che lavorerà per molto tempo senza interagire con la UI. Quando invece un service ha una qualsiasi interazione con l'utente allora si troverà in **foreground** (maggiore priorità)
- Esempio: un media player può usare diverse Activity per scegliere il brano musicale da riprodurre e avviare la riproduzione in un **foreground service**, lanciato in un thread separato
- Es. Youtube vs Telegram (riproduzione audio delegata ad Activity vs Service)
- Warning: è buona cosa evitare lunghi processi in background in modo da non consumare troppo la batteria

Content Provider

- Permette di condividere i propri dati (contatti, file archiviati su disco, eventi del Calendario) con il resto del sistema.
- Attraverso di lui si espongono i dati di un'app (es. lista degli SMS ricevuti) ad altre app, nascondendo la struttura e l'implementazione dei dati stessi.
- I Content Provider sono riconoscibili mediante un **URI**, un riferimento univoco.
- Content Provider molto noti del sistema operativo sono:
  - **Contacts**
  - Media store (file multimediali)
  - UserDictionary
  - Calendar

Intent

- E’ un messaggio asincrono che indica un’azione da compiere. Può attivare altri componenti (Activity, Broadcast Receiver o Services) tramite il SO (**gestione eventi**)
- Lancia una nuova activity (che è a sé stante) o un service di terze parti (come la galleria o la fotocamera) o notifiche di specifici eventi
- Questo slega azione da esecutore
- Chiede app di default
- Passa i dati tra le Activities



Broadcast Receiver

- Unità che riceve e gestisce eventi interni (livello batteria basso, perdita connettività, scattare del minuto e quindi cambio dell'orario…) o segnalazioni da parte di altre app. L'evento è un messaggio broadcast ricevuto da uno o più receiver.
- Spesso usati per **l'ascolto di Intent** di tipo broadcast, per discriminare quali tipi di action o dati gestire
- Avvia Activity se succede qualcosa, dando nuove informazioni all'utente nella barra di stato.

Manifest

- File che contiene l'intera descrizione delle componenti (contenuto e comportamento) e i permessi richiesti
- Dice come l'app è strutturata e a che intent o broadcast receiver risponde
- E' un file .xml, unico.

Oltre ai requisiti di sistema (versione minima di SO), ha:

- Singolo <application> con elementi: <activity>, <service>, <receiver> e <provider>
- <uses-permission> ogni qualvolta si vadano ad usare funzionalità di sicurezza
- <supports-screens> per la dimensione di schermo compatibile
- <uses-sdk> per versioni max e min del SDK sul device

Layout

- Slega la parte di logica applicativa da quella di presentazione
- Ognuno è salvato in res/layout in modo univoco
- Match-parent e wrap-content
- **Cascata lineare** (verticale o orizzontale, verificando height e width di ogni elemento) e **relativo** (più flessibile perché si riferisce ai bordi dello schermo, permette innesto di blocchi lineari) (`LinearLayout` vs `RelativeLayout`)
- Ogni elemento è detto **View**
- Tutte le immagini che compongono la grafica vanno catalogate in res/drawable in base ai dpi (dots per inch)
- Stringhe, stili e colori in res/values

Andiamo a programmare!

MVC

separare logica di controllo e presentazione

MVC – Model View Controller

- MODEL
  - Dati
  - Stato delle risorse
  - Business logic
  - In pratica rappresenta il "cervello", o meglio la memoria dell'applicazione, completamente astratto e distaccato dalle altre componenti, e perciò le sue informazioni sono riusabili
- VIEW
  - Rappresentazione del modello
  - Renderizzare la User Interface
  - Comunica col controller per reagire agli eventi generati dall'utente
  - E' la parte più "stupida", meno sa, meglio è.
  - Obiettivo è essere flessibile ai cambiamenti
- CONTROLLER
  - E' il mezzo di comunicazione tra modello e View
  - Quando la View informa di un evento sulla pagina visualizzata, il controller deve informarne il modello astratto (Activity o Fragment)
  - Quando cambia il modello deve decidere cosa fare con la View corrispondente (update?)

MVC – Model View Controller

Fondamenti di Kotlin

Partiamo dai ricordi

Dovremo ricordare cosa sono:

- Variabili, funzioni e Lambda expressions
- CLASSE e il suo COSTRUTTORE ed EREDITARIETÀ (super e this)
- Override
- Try catch

Variabili e Funzioni in Kotlin

- Var vs. val
  - **Var** rappresenta variabili mutabili e non definitive. Una volta inizializzato, siamo liberi di mutare i dati detenuti dalla variabile e il suo tipo di dato.
  - Tutte le variabili **Val** devono essere assegnate alla dichiarazione, o in un costruttore di classe, quindi sono di sola lettura.
- Le funzioni vengono dichiarate con **fun**
- **Lambda expression**: indica un insieme di istruzioni che possono essere salvate come fossero variabili, passate a un programma ed eseguite successivamente.
  - `val sum = { x: Int, y: Int -> x + y }`
  - La lambda expression è contenuta tra {} nella forma *input -> operazioni_output*
- Si inserisce _ davanti ad un campo il cui valore può essere omesso

Classe e Costruttore


- ```
  class Person constructor(val name: String, val
  age: Int? = null)
  ```

  …

  ```
  val person = Person("John")
  val personWithAge = Person("Mark", 22)
  ```

- Usiamo i costruttori per creare oggetti. Assomigliano a dichiarazioni di metodi, ma hanno sempre lo stesso nome della classe e non restituiscono nulla.
- Una classe può avere un costruttore principale e uno o più costruttori secondari aggiuntivi.
  ```

Ereditarietà

- Anche in Kotlin esiste il concetto di ereditarietà, in base al quale possiamo determinare una classe **base** e una **derivata**
- La "madre di tutte le classi" è la classe **Any** che non prevede nessun metodo predefinito
- Una classe per poter essere base di altre deve essere definita espressamente come **open.** In caso contrario è di default **final** e quindi non estendibile
- Per dichiarare che una classe deriva da un'altra si usa il simbolo " **:** "
- **this** è una keyword che permette di richiamare la classe stessa in cui ci si trova
- **super** è una keyword che si riferisce alla classe appena superiore in gerarchia

Override

- Anche nel caso dei metodi di una classe open, per poter essere sovrascritti nella classe derivata devono essere dichiarati anch'essi **open**
- Il tag **override** permette di ridefinire i metodi della superclasse che si ereditano per dar loro una personalizzazione più adeguata al caso in uso

```
open class Opera(titolo:String){
      open fun descrizione():String{
      return "Opera titolo \"$titolo\""; }    }

class Libro(titolo:String): Opera(titolo){
      override fun descrizione():String{
      return "${super.descrizione()} \nAutore: $autore";
} }
```

Eccezioni con Try e Catch

- Certe azioni sono rischiose da compiere, quindi si definisce un blocco che permetta di far andare una serie di comandi in modalità controllata, nel cui caso di errore permetta di identificare e gestire in modo adeguato l'eccezione conseguentemente lanciata
- Quindi il blocco "prova" certe istruzioni, e eventuali eccezioni "lanciate" che vengono "prese" da un altro blocco che si occupa di gestirle

```
try {
    log.append(message)
}
catch (IOException e) {
    // handling
}
```

Novità

- Nullables
- Adapter e List view
- Classe R
- Drawables e String.xml
- Ciclo Activity
- Companion object
- attributo android:onClick per catturare le interazioni dell'utente
- StringBuilder() per salvare dati

Nullables

```
var id: Int? = null
val userId = id ?: -1 //userID is -1
```

I Nullables sono un tipo di dato che viene usato quando il valore di una variabile potrebbe essere nullo (null). Un Nullable rappresenta due possibilità:

1. Il valore è presente
2. Il valore non è presente, quindi null

Se ad esempio tentiamo di convertire una stringa in un numero, non sempre questa operazione ha risultato positivo, assegnando null nel caso non sia valido.

I Nullable sono rappresentati dal tipo di dato che incapsulano, seguiti da un ?

var someValue: Int?

var message: String? // automaticamente impostate a null

Permettono di definire sempre un valore (null)!

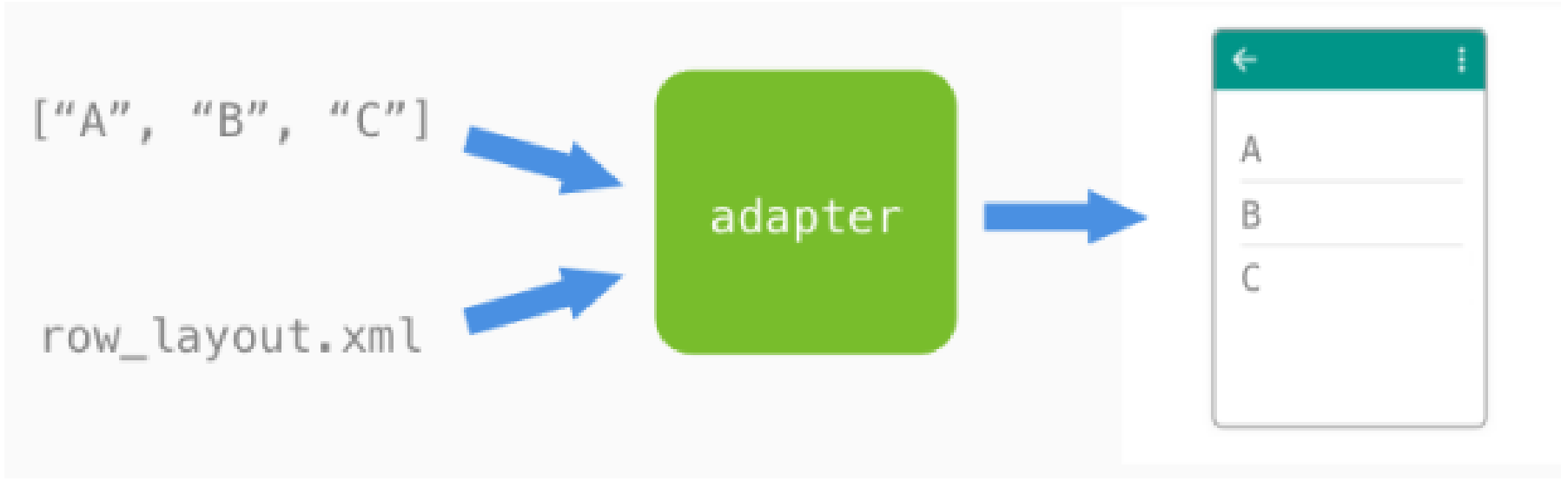
La funzione let di Kotlin, se utilizzata in combinazione con l'operatore di chiamata sicura **?.** , fornisce un modo conciso per gestire le espressioni, una sorta di "if not null, then do…"

```
var someValue: Int? = null
someValue?.let{perform(someValue)}
```

Non darà risultato!

Adapter

- Un oggetto Adapter funge da ponte tra un AdapterView e i dati sottostanti per quella View. L'Adapter fornisce l'accesso ai dati degli elementi ed è responsabile della creazione di una View per ogni elemento nel set di dati.
- Le liste vengono popolate attraverso un oggetto di tipo Adapter. Questo è un componente che si occupa della rappresentazione grafica dei dati e dell'interazione con essi, per ogni elemento della **ListView**.



Classe R

- R è l'abbreviazione di ROOT, ossia la radice dell'albero che contiene tutto l'elenco ordinato delle risorse di un'app. Raccoglie tutti i riferimenti (gli id), in modo che il codice ne abbia visibilità.
- Il primo elemento sarà il layout (R.layout). Altri esempi:
  - R.drawable.logo
  - R.string.hello_world
  - R.color.colorPrimary
  - R.style.AppTheme
- Fanno parte della gerarchia anche le risorse (cartella src), le immagini e le stringhe

Drawables e String.xml

- Le immagini vengono invece raccolti nella cartella drawables, che può assumere un naming semantico che permette di divedere risoluzioni diverse della stessa immagine in base ai dpi (dots per inch):
  - res/drawable/icon.png
  - res/drawablehdpi/icon.png
  - res/drawablexhdpi/icon.png
- Le stringhe di un'app possono essere raccolte in un unico file di risorse, strings.xml:

```
<resources>
    <string name="hello">Hello!</string>
</resources>
```

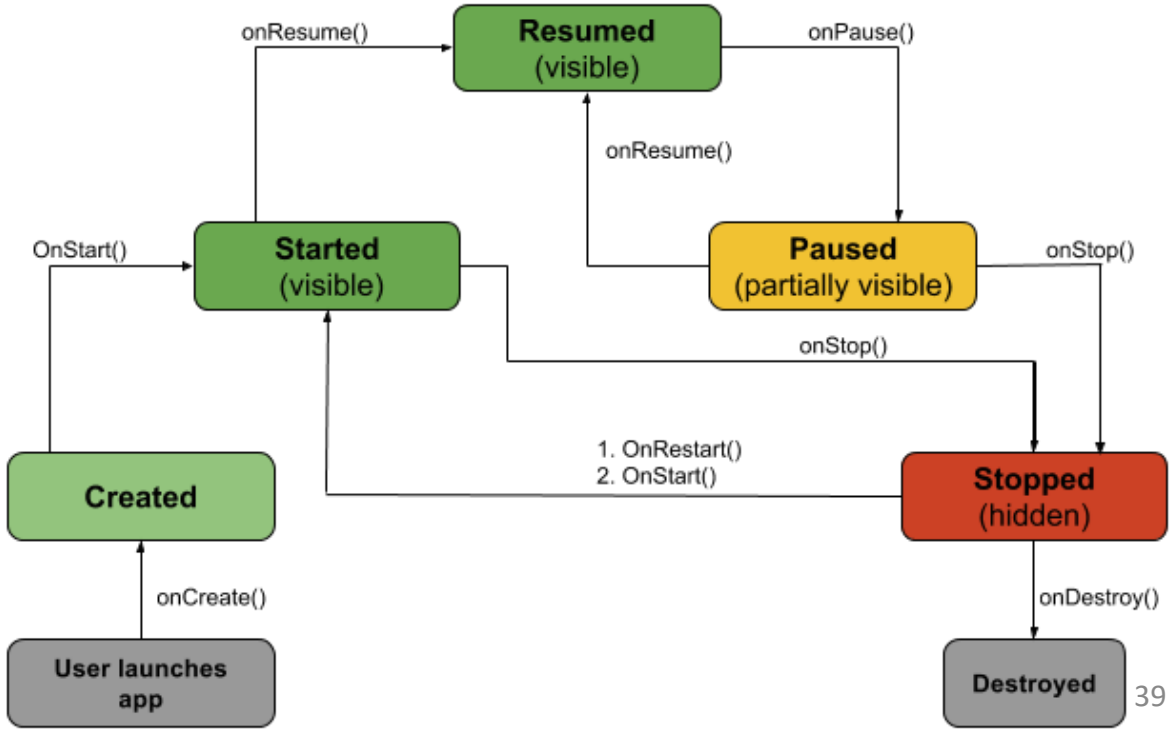
- Questo semplifica la traduzione di un’app in altre lingue, basta infatti fornire un file .xml per ogni lingua supportata:
  - values/strings.xml
  - valuesit/strings.xml
  - valuesde/strings.xml

Ciclo Activity

- Ogni Activity viene implementata come sottoclasse di AppCompatActivity
- Layout viene assegnato ad una Activity con **`setContentView(R.layout.activity_main)`**
- Per cambiare Activity con un Intent:

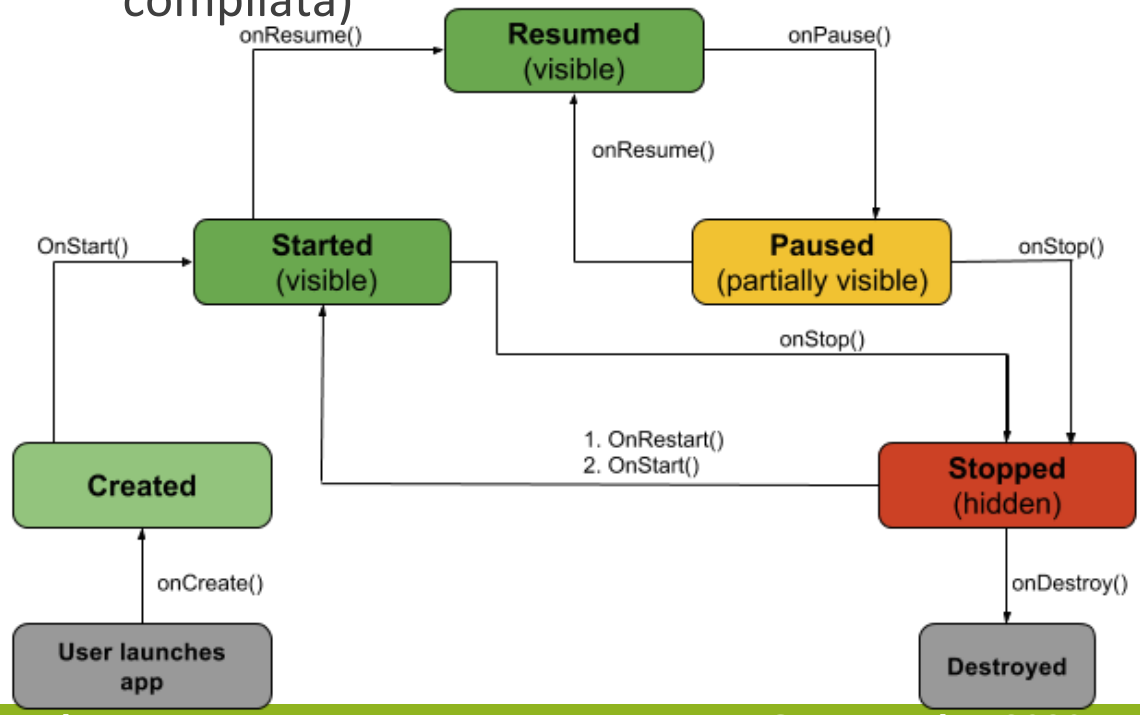
```
val openSettings = Intent(applicationContext,
SettingsActivity::class.java)

startActivity(openSettings)
```



Ciclo Activity

- **onCreate()** è il metodo che avvia i Services di foreground e l'Activity
- **Started** è lo stato in cui l'applicazione è già visibile ma non reagisce
- **Resumed** è lo stato visibile quando l'app si è caricata ed è reattiva
- **Paused** è lo stato in cui l'Activity lascia spazio ad altre viste e fa caricare altro contenuto, quindi è in parte nascosta (es. Activity o popup traslucidi)
- Quando il popup si conclude, scatta il metodo **onResume()**
- Quando si preme il tasto "Back" o si cambia applicazione, l'Activity passa nello stato **Stopped**, in cui è completamente oscurata da altro contenuto
- Quando si torna alla schermata precedente, quella Activity subisce la transizione da Stopped a Started grazie al metodo **onRestart()**
- Quando l'Activity ha concluso il suo scopo e non si può più raggiungere passa allo stato **Destroyed** e viene liberato lo spazio che occupava in memoria
  - (es. sezione in cui inserisci i dati dopo averla compilata)



Companion Object

- Il **Companion Object** è un singleton (una classe che ha solo un'istanza) ai cui membri si può accedere direttamente tramite il nome della classe che lo contiene
- Ricordare che, anche se i membri del companion object sembrano membri `static` di altri linguaggi, in fase di esecuzione sono ancora membri di istanze di oggetti reali e possono, ad esempio, implementare interfacce.

onClick

- Serve ad associare alla View la funzione logica di callback da scatenare nel momento in cui riceve un click (modificando layout.xml)
- Questa si dice **Gestione ad Eventi**
- La callback function associata va definita all'interno dell'Activity (in Kotlin) mentre nel file layout.xml va associata tale funzione alla View corrispondente
- Es. **`android:onClick="metodo_OnClickListener"`**

StringBuilder()

- Gli oggetti stringa sono immutabili e non sono adatti al caso in cui sia necessario aggiungere o inserire caratteri in essi perché le operazioni di stringa su String creano sempre un nuovo oggetto String.
- Per aggiungere e inserire, è meglio usare la classe java.lang.StringBuffer o java.lang.StringBuilder. Una volta terminata la manipolazione della stringa, è possibile convertire un oggetto StringBuffer o StringBuilder in una stringa (creano un buffer interno che contiene la stringa definitiva).

```
fun concatenaConStringBuilder_eConfronta() {
    val builder = StringBuilder()
    builder.append("Hello")
           .append(" ")
           .append("Unife")

    assertEquals("Hello Unife", builder.toString())
}
```

Pubblicazione

- Fare il debug del codice
- Aggiornare la versione del programma
- Gradle è un tool per l'automatizzazione dei task più comuni eseguiti in un progetto. Si occupa principalmente della compilazione, della gestione delle dipendenze e del deploy di un'app.
- Procedimento: **Build** -> **Generate Signed APK**, poi **Generate Signed APK Wizard** su **Key store path** scegliendo **Create new.** Poi **ok** e **Finish**.
- Oppure Build < Build Bundles < Build APK e controlla dove salva il .apk (locate)
- **Versa una tantum 25$ dal tuo profilo da** https://play.google.com/apps/publish
- **Pensa a come monetizzare**
- Premi "Aggiungi nuova applicazione" e trascina il .apk
- Falla scaricare ai tuoi amici!

“You don’t learn to walk by following the rules. You learn by doing and by falling over”

- *Richard Branson*

Grazie a tutti per la partecipazione

Laboratorio

Hands on, e andiamo a sviluppare la nostra prima app

Creiamo un nuovo progetto

- Creiamo un progetto con una sola activity vuota
- Vogliamo ottenere una lista che ci ricordi le cose da fare: To Do List
- Vediamo allo step 0 che cosa ci troviamo davanti
  - **Manifest**, il biglietto da visita della nostra app
  - **Activity** principale (e unica, per ora)
  - **Layout**, pronto per essere modificato

Manifest.xml

- Prima cosa da modificare!!! ☺
- Rappresenta il descrittore dell'intera app. Definisce tutte le componenti dell'app e le **permission** richieste

Step 0 – Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
package="com.example.mytodolistapp">

    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme"
tools:ignore="GoogleAppIndexingWarning">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Step 0 – Main Activity

```
package com.example.<<nome_app>>

import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Step 0 – layout: activity_main.xml

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

Step 0 – layout: activity_main.xml

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

Dimensionamento: wrap_content vs match_parent

Vincoli (relativi o assoluti) per il posizionamento delle View

Step 1 – modifichiamo il layout

Andiamo a modificare il layout della nostra MainActivity, aggiungendo una lista di stringhe e un pulsante

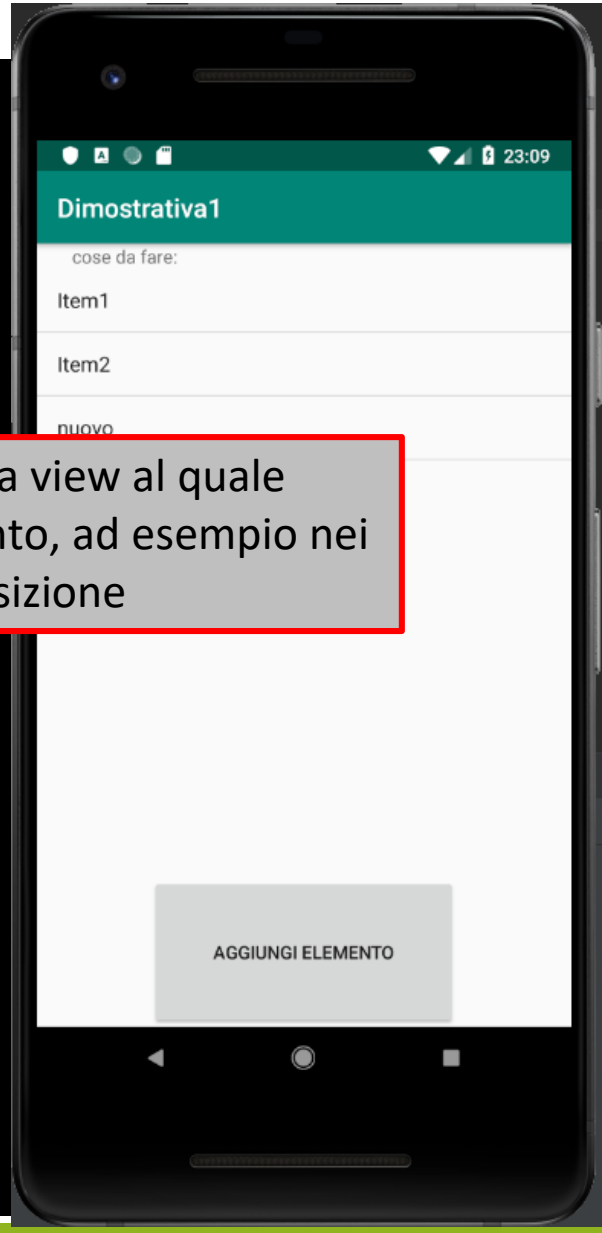
Step 1 – modifichiamo il layout (activity_main.xml)


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

    <TextView
            android:id="@+id/dafare"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="cose da fare:"
            app:layout_constraintBottom_toTopOf="@id/taskListView"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
           app:layout_constraintHorizontal_bias="0.083"
            app:layout_constraintVertical_bias="0.04"/>

 continua…
```

Attribuisco un ID alla view al quale posso fare riferimento, ad esempio nei vincoli relativi di posizione

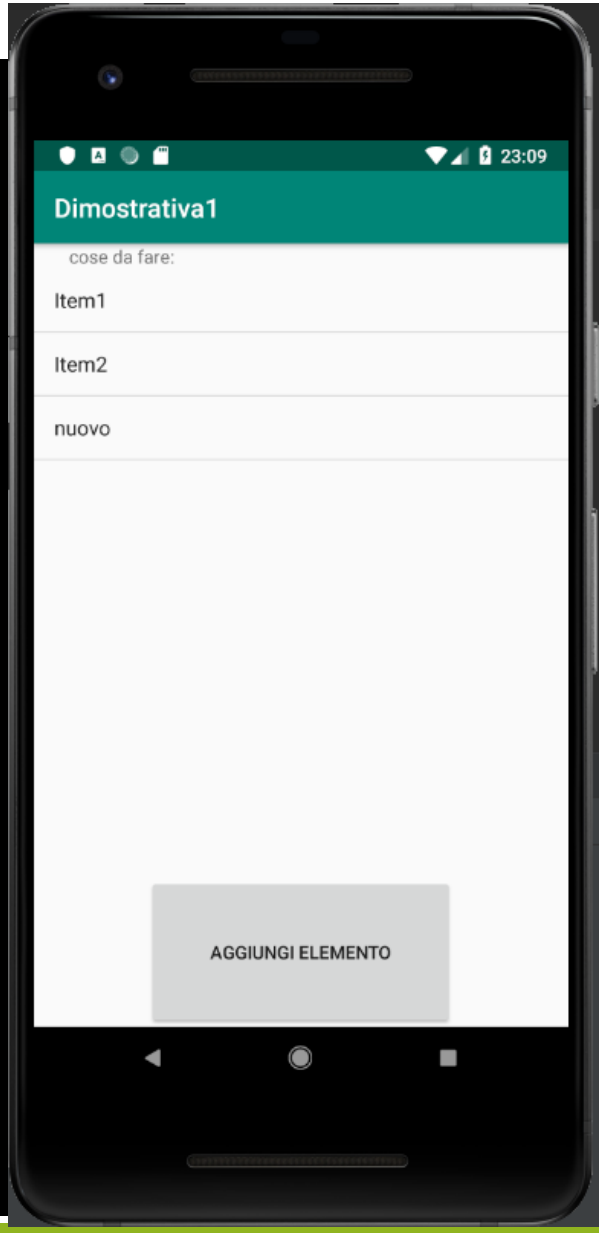


Step 1 – modifichiamo il layout (activity_main.xml)


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

    <TextView
            android:id="@+id/dafare"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="cose da fare:"
            app:layout_constraintBottom_toTopOf="@id/taskListView"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
           app:layout_constraintHorizontal_bias="0.083"
            app:layout_constraintVertical_bias="0.04"/>

 continua…
```



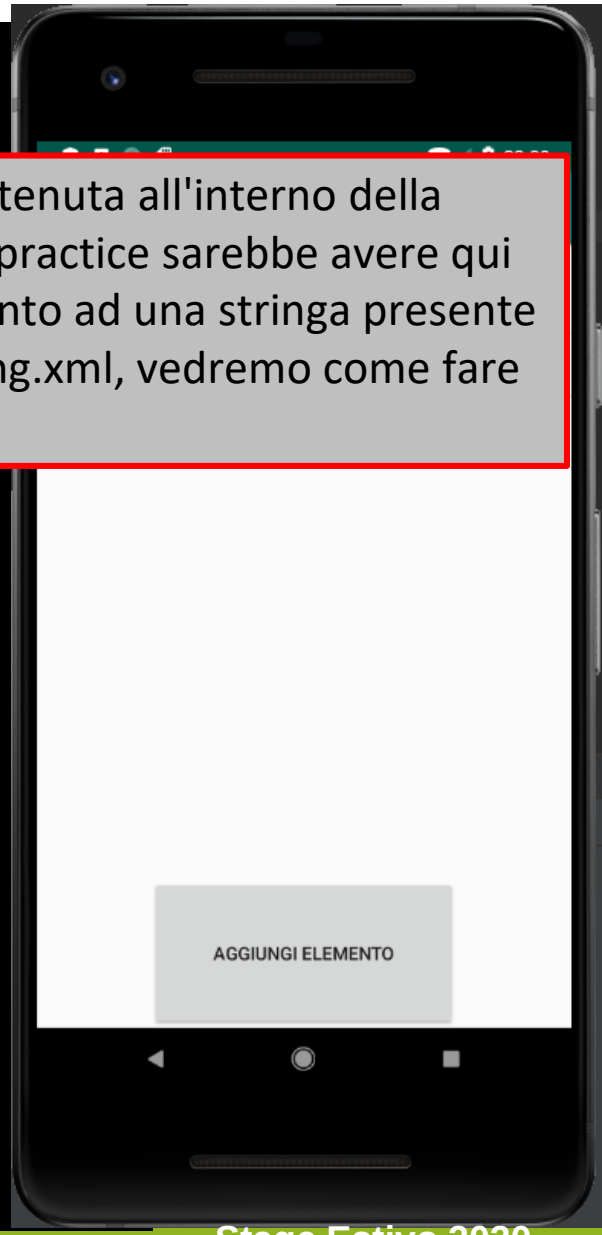
Step 1 – modifichiamo il layout (activity_main.xml)


```
Continua…
<Button
            android:id="@+id/addTaskButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="@android:dimen/app_icon_size"
            android:text="aggiungi elemento"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
           app:layout_constraintTop_toBottomOf="@+id/taskListView"
         app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintVertical_bias="0.0"/>

    <ListView
            android:id="@+id/taskListView"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            app:layout_constraintBottom_toTopOf="@id/addTaskButton"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@id/dafare" />

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

Stringa contenuta all'interno della View; best practice sarebbe avere qui un riferimento ad una stringa presente nel file string.xml, vedremo come fare

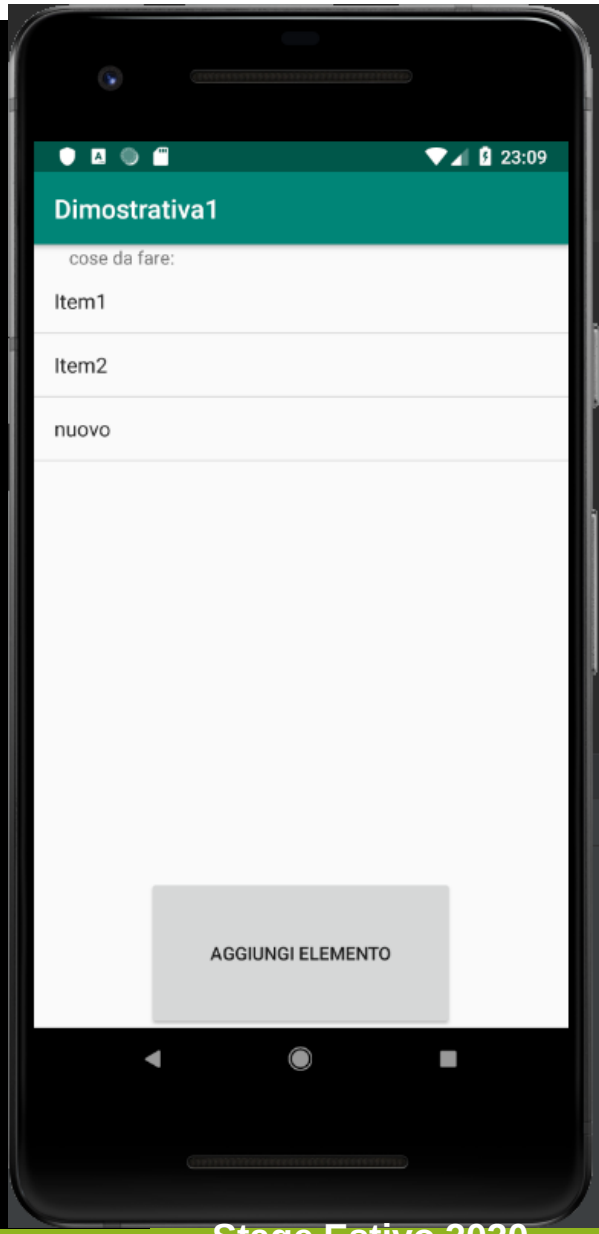


Step 1 – modifichiamo il layout (activity_main.xml)


```
Continua…
<Button
            android:id="@+id/addTaskButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="@android:dimen/app_icon_size"
            android:text="aggiungi elemento"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
           app:layout_constraintTop_toBottomOf="@+id/taskListView"
         app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintVertical_bias="0.0"/>

    <ListView
            android:id="@+id/taskListView"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            app:layout_constraintBottom_toTopOf="@id/addTaskButton"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@id/dafare" />

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



Tips:

- Notate che con **Alt+Invio** si può aprire la tendina dei suggerimenti per risolvere automaticamente eventuali errori o warning segnalati
- Questo "trucco" lo possiamo sfruttare per inserire automaticamente gli *include* necessari, oppure trasformare le stringhe di testo in riferimenti a stringhe, creati automaticamente nell'apposito file *strings.xml* in modo tale da tenere sempre ben in ordine il nostro progetto
- Si tenga conto anche del fatto che quest'ultima cosa è importante in una buona app professionale, in quanto avremo **diverse lingue** da poter implementare, quindi la nostra app dovrebbe recuperare automaticamente la stringa nella lingua in cui l'utente ha settato il telefono.
- ***Alt+Invio > Extract string resource*** per esportare la stringa nel file string.xml, e tenere nel layout solo il riferimento corretto a tale stringa, identificata dall' ID che noi vogliamo attribuirgli

Step 2 – modifichiamo le funzionalità dell'Activity

Andiamo a modificare il comportamento della nostra MainActivity, inizializziamo la lista e prepariamo le funzionalità su di essa e sul pulsante

Step 2 – Modifichiamo le funzionalità dell'Activity


```
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }

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

        taskListView.adapter = adapter
        taskListView.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position,
id -> }
    }
    fun addTaskClicked(view: View) {

    }
    private fun makeAdapter(list: List<String>): ArrayAdapter<String> =
        ArrayAdapter(this, android.R.layout.simple_list_item_1, list)
}
```

Inizializzazione della lista e del relativo adapter, la cui funzione di inizializzazione è esplicitata in fondo

Step 2 – Modifichiamo le funzionalità dell'Activity


```
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }

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

        taskListView.adapter = adapter
        taskListView.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position,
id -> }
    }
    fun addTaskClicked(view: View) {

    }
    private fun makeAdapter(list: List<String>): ArrayAdapter<String> =
        ArrayAdapter(this, android.R.layout.simple_list_item_1, list)
}
```

Prima cosa da fare: overriding onCreate: chiamo la funzione della superclasse, poi associo il layout da visualizzare al file xml

Step 2 – Modifichiamo le funzionalità dell'Activity


```
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }

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

        taskListView.adapter = adapter
        taskListView.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position,
id -> }
    }
    fun addTaskClicked(view: View) {

    }
    private fun makeAdapter(list: List<String>): ArrayAdapter<String> =
        ArrayAdapter(this, android.R.layout.simple_list_item_1, list)
}
```

Assegno l'adapter all'oggetto corrispondente della lista istanziata, e associo anche un listener dell'evento click sulla lista

Step 2 – Modifichiamo le funzionalità dell'Activity


```
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }

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

        taskListView.adapter = adapter
        taskListView.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position,
id -> }
    }
    fun addTaskClicked(view: View) {

    }
    private fun makeAdapter(list: List<String>): ArrayAdapter<String> =
        ArrayAdapter(this, android.R.layout.simple_list_item_1, list)
}
```

Sarà la funzione chiamata al click sul pulsante di "aggiungi elemento"

Step 2 – Modifichiamo le funzionalità dell'Activity


```
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }

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

        taskListView.adapter = adapter
        taskListView.onItemClickListener = AdapterView.OnItemClickListener { parent, view, position,
id -> }
    }
    fun addTaskClicked(view: View) {

    }
    private fun makeAdapter(list: List<String>): ArrayAdapter<String> =
        ArrayAdapter(this, android.R.layout.simple_list_item_1, list)
}
```

Step 2 – Modifichiamo le funzionalità dell'Activity

Aggiungiamo il riferimento alla funzione associata al pulsante nel layout della activity_main.xml:

```
<Button
        android:id="@+id/addTaskButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="addTaskClicked"
        android:padding="@android:dimen/app_icon_size"
        android:text="@string/add_task"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/taskListView"
        app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintVertical_bias="0.0"/>
```

Step 3 – Creiamo una nuova Activity

Prepariamo una seconda Activity che dovrà essere chiamata cliccando sul pulsante "aggiungi elemento"

Step 3 – Chiamare una nuova Activity

```
import android.content.Intent  //aggiunto
```

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1  //aggiunto

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }
...
```

Sarà il codice di riferimento che useremo per la richiesta di aggiungere un elemento alla lista

```
fun addTaskClicked(view: View) {
    val intent = Intent(this, TaskDescriptionActivity::class.java)
//aggiunto: ricorda di creare la activity con questo nome
    startActivityForResult(intent, ADD_TASK_REQUEST)
}
```

Step 3 – Chiamare una nuova Activity

```
import android.content.Intent  //aggiunto
```

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1  //aggiunto

    private val taskList = mutableLis
    private val adapter by lazy { make
...
```

Implementiamo la funzione all'evento click con la creazione di un Intent che sarà incaricato di chiamare la nuova Activity, nominata TaskDescriptionActivity

```
fun addTaskClicked(view: View) {
    val intent = Intent(this, TaskDescriptionActivity::class.java)
//aggiunto: ricorda di creare la activity con questo nome
    startActivityForResult(intent, ADD_TASK_REQUEST)
}
```

Step 3 – Chiamare una nuova Activity

```
import android.content.Intent  //aggiunto
```

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1  //aggiunto

    private val taskList = mutableListOf<String>()
    priv
...
```

```
fun addT
    val intent = Intent(this, TaskDescriptionActivity::class.java)
//aggiunto: ricorda di crea  la activity con questo nome
    startActivityForResult(intent, ADD_TASK_REQUEST)
}
```

La startActivityForResult oltre a chiamare la activity fa sì che alla sua chiusura venga chiamata la funzione onActivityResult, una callback che useremo per l'aggiunta del task.
Inoltre andiamo ad associare un codice di ritorno che ci permetterà di riconoscere in maniera univoca che la callback è stata invocata da quella particolare Activity chiamata dalla MainActivity

Step 3 – Chiamare una nuova Activity

```
import android.content.Intent  //aggiunto
```

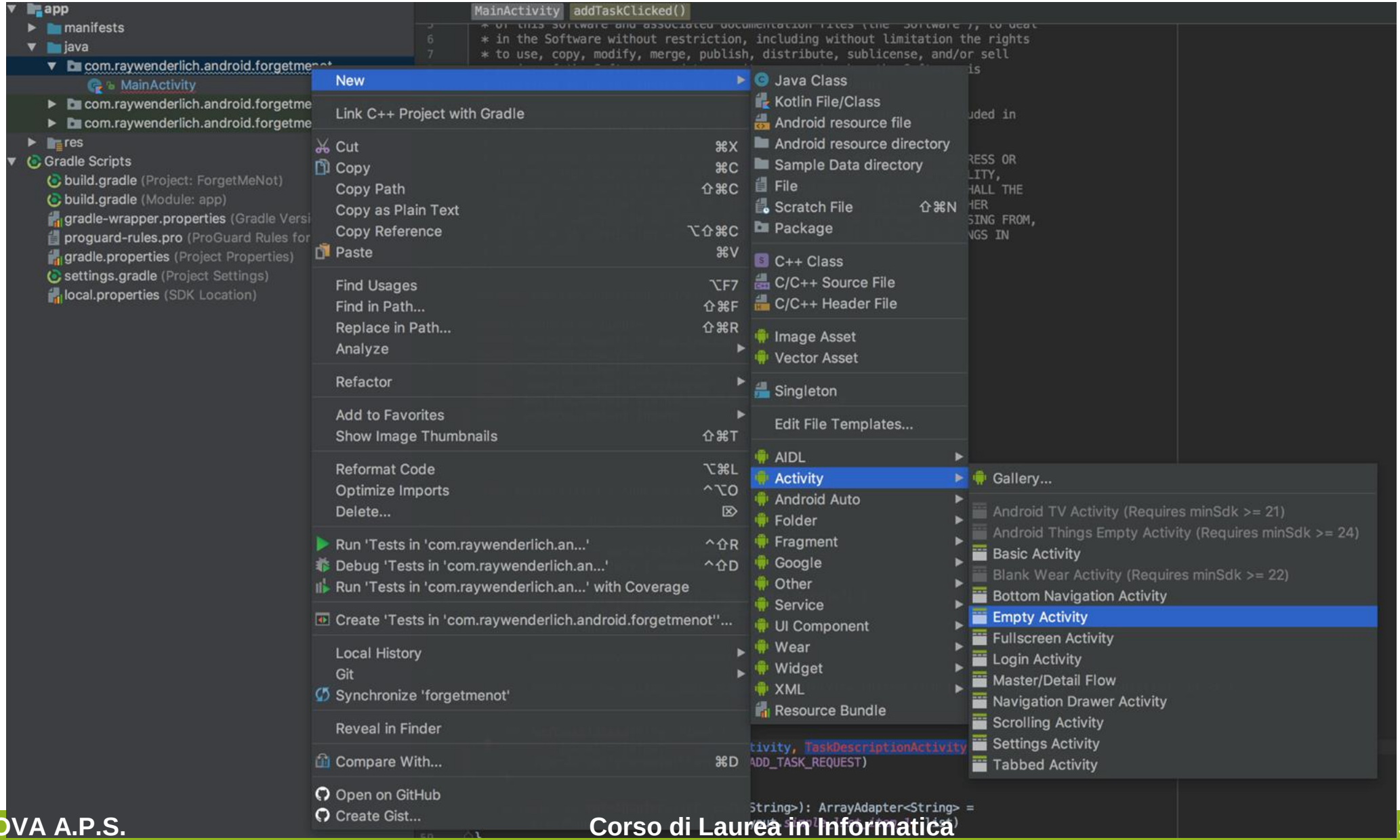
```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1  //aggiunto

    private val taskList = mutableListOf<String>()
    private val adapter by lazy { makeAdapter(taskList) }
...
```

```
fun addTaskClicked(view: View) {
    val intent = Intent(this, TaskDescriptionActivity::class.java)
//aggiunto: ricorda di creare la activity con questo nome
    startActivityForResult(intent, ADD_TASK_REQUEST)
}
```

Step 3 – Creare una nuova Activity



Notiamo che:

- Durante la creazione, automaticamente gli altri campi vengono adattati al nome che diamo alla nostra nuova activity
- Automaticamente troviamo (come nel caso della prima activity "*MainActivity*") gli elementi basilari per l'esistenza dell'activity, sia nel file Kotlin, sia nel layout xml corrispondente
- Automaticamente viene aggiornato anche il **Manifest**
- **Avviando l'app, cliccando sul pulsante verrà chiamata la nuova finestra!**

Step 4 – Implementiamo la nuova Activity

Attribuiamo un layout alla nuova activity e improntiamo le sue funzionalità

Step 4 – Implementiamo la nuova Activity


```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class TaskDescriptionActivity : AppCompatActivity() {

    companion object {
        val EXTRA_TASK_DESCRIPTION = "task" //aggiunto
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_task_description)
    }

    fun doneClicked(view: View) {

    }
}
```

Determiniamo con un companion object che potremo richiamare per definire un attributo nome costante che poi inseriremo nell'intent che passerà il nuovo task alla MainActivity che poi lo inserirà in lista

Step 4 – Implementiamo la nuova Activity


```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class TaskDescriptionActivity : AppCompatActivity() {

    companion object {
        val EXTRA_TASK_DESCRIPTION = "task" //aggiunto4
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_task_description)
    }

    fun doneClicked(view: View) {

    }
}
```

Come avevamo già fatto, implementiamo la onCreate, richiamando quella della superclasse e associando alla activity il relativo layout

Step 4 – Implementiamo la nuova Activity


```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class TaskDescriptionActivity : AppCompatActivity() {

    companion object {
        val EXTRA_TASK_DESCRIPTION = "task" //aggiunto4
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_task_description)
    }

    fun doneClicked(view: View) {

    }
}
```

Questa è la funzione (da implementare) da chiamare quando clicchiamo sul pulsante per aggiungere alla lista il nuovo task inserito

Step 4 – Implementiamo la nuova Activity


```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View

class TaskDescriptionActivity : AppCompatActivity() {

    companion object {
        val EXTRA_TASK_DESCRIPTION = "task" //aggiunto4
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_task_description)
    }

    fun doneClicked(view: View) {

    }
}
```

Step 4 – Modifichiamo il layout della nuova Activity

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

    <TextView
            android:id="@+id/descriptionLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="aggiungi descrizione:"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

continua…
```

Inseriamo un titolo da posizionare in cima alla schermata che riporti il testo "aggiungi descrizione:" con un suo ID

Step 4 – Modifichiamo il layout della nuova Activity


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

    <TextView
            android:id="@+id/descriptionLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="aggiungi descrizione:"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

continua…
```

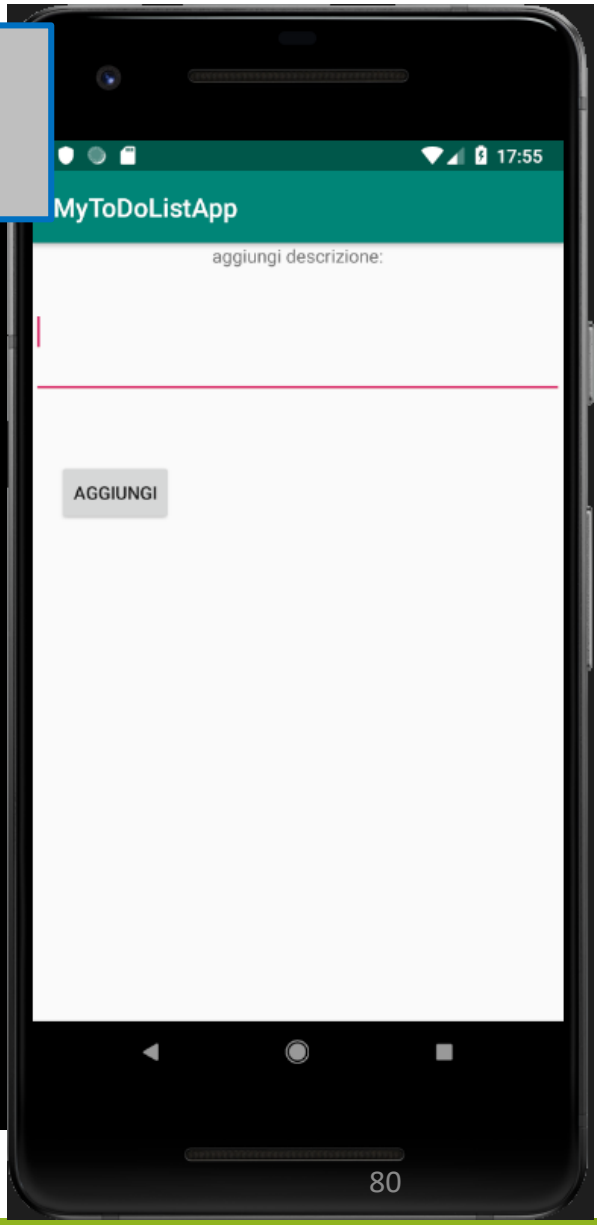
Step 4 – Modifichiamo il layout della nuova Activity


```
Continua…
    <EditText
            android:id="@+id/descriptionText"
            android:layout_width="match_parent"
            android:layout_height="100dp"
            android:inputType="textMultiLine"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionLabel" />

    <Button
            android:id="@+id/doneButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="doneClicked"
            android:text="aggiungi"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionText"
            android:layout_marginTop="50dp"/>

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

EditText è una finestra in cui possiamo inserire il testo, cioè la descrizione del nostro task



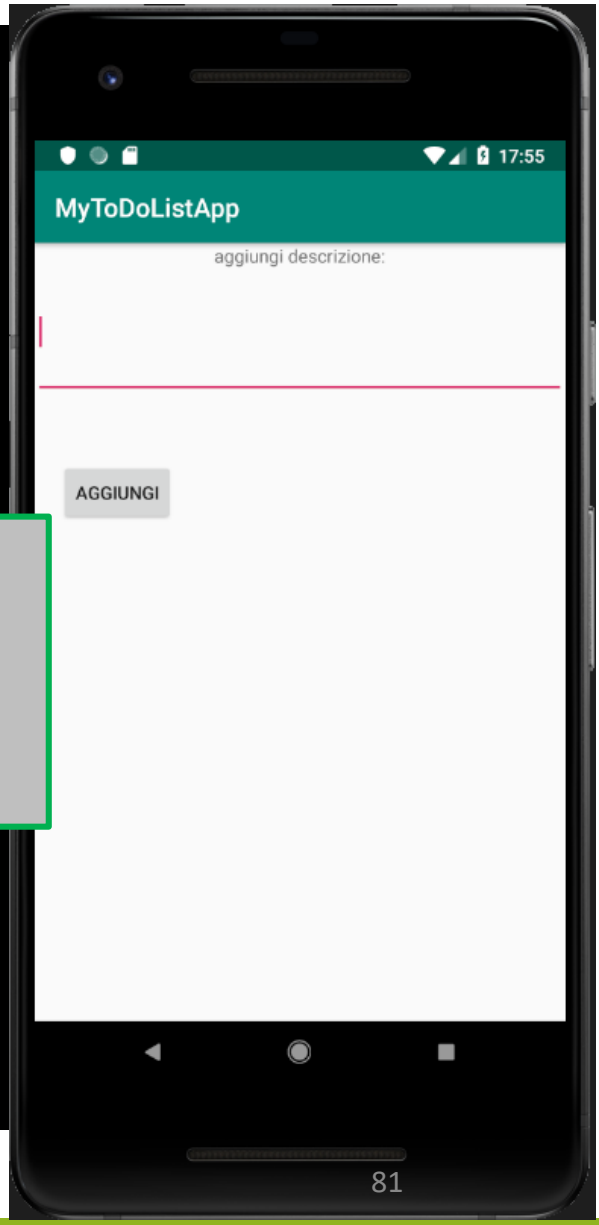
Step 4 – Modifichiamo il layout della nuova Activity

```
Continua…
    <EditText
            android:id="@+id/descriptionText"
            android:layout_width="match_parent"
            android:layout_height="100dp"
            android:inputType="textMultiLine"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionLabel" />

    <Button
            android:id="@+id/doneButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="doneClicked"
            android:text="aggiungi"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionText"
            android:layout_marginTop="50dp"/>

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

Inseriamo anche un pulsante "*doneBotton*", che riporta la scritta "*aggiungi*", e diciamogli quale è la funzione da chiamare al click

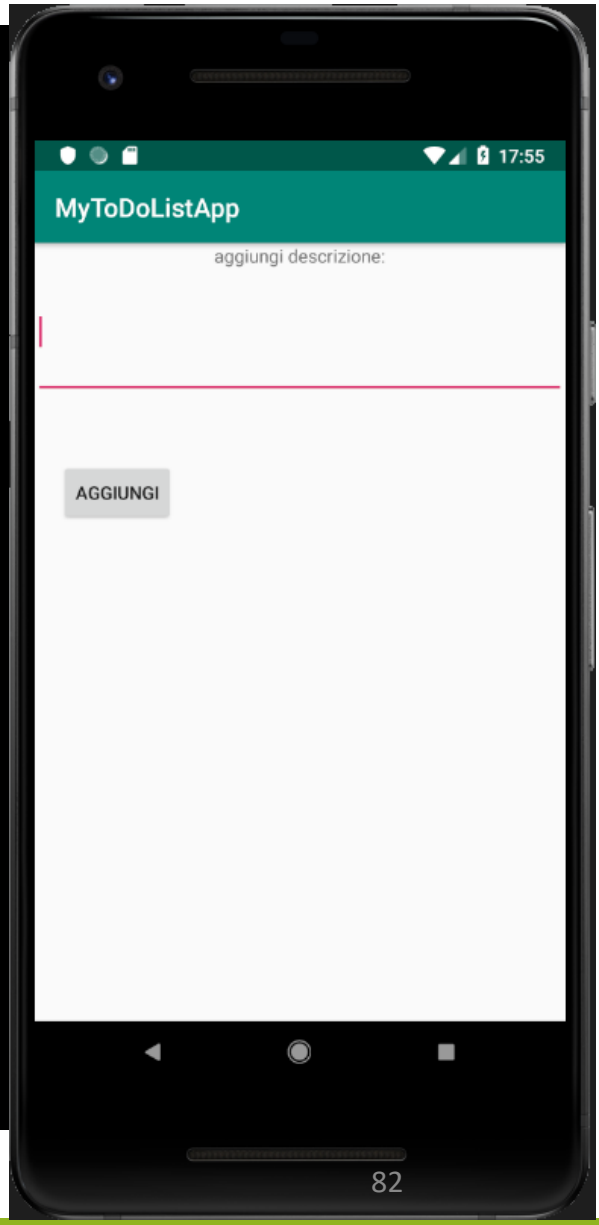


Step 4 – Modifichiamo il layout della nuova Activity


```
Continua…
    <EditText
            android:id="@+id/descriptionText"
            android:layout_width="match_parent"
            android:layout_height="100dp"
            android:inputType="textMultiLine"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionLabel" />

    <Button
            android:id="@+id/doneButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="doneClicked"
            android:text="aggiungi"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toBottomOf="@id/descriptionText"
            android:layout_marginTop="50dp"/>

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



Tips:

- Anche qui abbiamo scritto le stringhe direttamente nel layout come:

```
android:text="aggiungi descrizione:"
```

e ci viene segnalato un warning. Infatti questo funziona, ma non è buona pratica, è meglio sfruttare ***Alt+Invio > Extract string resource*** per esportare la stringa nel file string.xml, e tenere nel layout solo il riferimento corretto a tale stringa, identificata dall' ID che noi vogliamo attribuirgli

Step 5 – Fermare la seconda Activity

Importante tanto quanto chiamare la activity è il modo in cui la si ferma.

Ora andiamo ad implementare la funzione chiamata dopo aver completato l'inserimento della descrizione del task, al click sul pulsante, e in seguito modificare la MainActivity perché sia in grado di inserire l'elemento in lista

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

Valore costante contenente il testo inserito nella *EditText* identificato con ID "*descriptionTest*" nel layout

```
import android.app.Activity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

Creazione dell'Intent con il quale passiamo alla MainActivity il nuovo task.
Inseriamo un nome e il dato nell'intent che ora lo incapsulerà.

```
import android.app.Activity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

```
import android.app.Activity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

`final void`	`setResult(int resultCode, Intent data)` Call this to set the result that your activity will return to its caller.
`final void`	`setResult(int resultCode)` Call this to set the result that your activity will return to its caller.

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

Tramite la setResult, nelle 2 versioni viste, sto passando l'intent che incapsula il task alla MainActivity, insieme ad un codice che distingue il caso positivo o di mancato inserimento

```
import android.app.Activity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

Termino la TaskDescriptionActivity, e a questo punto viene chiamata la onActivityResult nella MainActivity

```
ity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

```
fun doneClicked(view: View) {
        // 1
        val taskDescription = descriptionText.text.toString()

        if (!taskDescription.isEmpty()) {
            // 2
            val result = Intent()
            result.putExtra(EXTRA_TASK_DESCRIPTION, taskDescription)
            setResult(Activity.RESULT_OK, result)
        } else {
            // 3
            setResult(Activity.RESULT_CANCELED)
        }

         // 4
        finish()

    }
```

```
import android.app.Activity
import kotlinx.android.synthetic.main.activity_main.*
import android.content.Intent  //aggiunto
```

Step 5 – Fermare la seconda Activity

A questo punto dobbiamo fare un *override* della funzione onActivityResult all'interno della MainActivity per sfruttarla per leggere cosa è stato inserito nella seconda Activity, e in caso di corretto inserimento di un task andare ad aggiungere l'elemento alla lista

Step 5 – Fermare la seconda Activity

MainActivity.kt

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

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    // 1
    if (requestCode == ADD_TASK_REQUEST) {
        // 2
        if (resultCode == Activity.RESULT_OK) {
            // 3
            val task =
data?.getStringExtra(TaskDescriptionActivity.EXTRA_TASK_DESCRIPTION)
            task?.let {
                taskList.add(task)
                // 4
                adapter.notifyDataSetChanged()
            }
        }
    }
}
```

Controllo il codice di richiesta per verificare che sia proprio la Activity TaskDescriptionActivity invocata dalla MainActivity ad essere terminata causando la chiamata della callback

```
import android.app.Activity
```

Step 5 – Fermare la seconda Activity

MainActivity.kt

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

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    // 1
    if (requestCode == ADD_TASK_REQUEST) {
        // 2
        if (resultCode == Activity.RESULT_OK) {
            // 3
            val task =
data?.getStringExtra(TaskDescriptionActivity.EXTRA_TASK_DESCRIPTION)
            task?.let {
                taskList.add(task)
                // 4
                adapter.notifyDataSetChanged()
            }
        }
    }
}
```

Controllo che sia stato inserito effettivamente un nuovo task

```
import android.app.Activity
```

Step 5 – Fermare la seconda Activity

MainActivity.kt

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

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    // 1
    if (requestCode == ADD_TASK_REQUEST) {
        // 2
        if (resultCode == Activity.RESULT_OK) {
            // 3
            val task =
data?.getStringExtra(TaskDescriptionActivity.EXTRA_TASK_DESCRIPTION)
            task?.let {
                taskList.add(task)
                // 4
                adapter.notifyDataSetChanged()
            }
        }
    }
}
```

Estraggo il dato stringa contenuto nell'intent identificato dal nome attribuitogli, ovvero la stringa definita dal companion object nella seconda activity (ecco perché serviva "*companion*", dovevamo usarlo fuori!)

Step 5 – Fermare la seconda Activity

MainActivity.kt

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

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    // 1
    if (requestCode == ADD_TASK_REQUEST) {
        // 2
        if (resultCode == Activity.RESULT_OK) {
            // 3
            val task =
data?.getStringExtra(TaskDescriptionActivity.EXTRA_TASK_DESCRIPTION)
            task?.let {
                taskList.add(task)
                // 4
                adapter.notifyDataSetChanged()
            }
        }
    }
}
```

Dopo un check di non nullità aggiungo effettivamente il task alla lista e sfruttando l'adapter si notifica un cambiamento alla lista innescando un refresh

Step 5 – Fermare la seconda Activity

MainActivity.kt

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

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    // 1
    if (requestCode == ADD_TASK_REQUEST) {
        // 2
        if (resultCode == Activity.RESULT_OK) {
            // 3
            val task =
data?.getStringExtra(TaskDescriptionActivity.EXTRA_TASK_DESCRIPTION)
            task?.let {
                taskList.add(task)
                // 4
                adapter.notifyDataSetChanged()
            }
        }
    }
}
```

```
import android.app.Activity
```

Wow! L'app sembra funzionare!

- Riscontriamo forse problemi?
  - Beh… proviamo a ruotare il telefono/emulatore
  - Proviamo magari a chiudere e riaprire la nostra app

- Proviamo ora a inserire degli elementi "di default", presenti dalla nascita della nostra app (funzione *onCreate*), in questo modo:

```
override fun onCreate(savedInstanceState: Bundle?) {
…
    taskListView.onItemClickListener =
AdapterView.OnItemClickListener { parent, view, position, id -> }

    taskList.add("Item1")
    taskList.add("Item2")

}
```

Perché succede?

Il problema sta nel fatto che ogni volta che avviene un cambiamento, come la rotazione, o la riapertura della app, nulla è salvato in memoria, viene richiamata la *onCreate*, e tutto riparte dall'inizio.

Non è questo il comportamento che ci aspettiamo.

Infatti, come possiamo vedere dal codice, abbiamo sovrascritto semplicemente la funzione *onCreate*, ma abbiamo visto che nel ciclo di vita della nostra Activity ci sono altre funzioni che giocano un ruolo in chiusura e riapertura.

Quello che dovremo fare sarà salvare le modifiche apportate sfruttando proprio quelle funzioni.

La soluzione in pratica la vedremo tra poco.

ACTION_TIME_TICK

Added in API level 1

```
public static final String ACTION_TIME_TICK
```

Broadcast Action: The current time has changed. Sent every minute. You *cannot* receive this through components declared in manifests, only by explicitly registering for it with `Context#registerReceiver(BroadcastReceiver, IntentFilter)`.

★ This is a protected intent that can only be sent by the system.

Constant Value: "android.intent.action.TIME_TICK"

Step 6 – Data e ora tramite Broadcast Receiver

Impariamo a sfruttare un Broadcast Receiver per catturare un intent broadcast, nel nostro caso ACTION_TIME_TICK, tramite cui il sistema segnala lo scadere del minuto e il cambiamento dell'orario segnato

Step 6 – Data e ora tramite Broadcast Receiver

Inseriamo innanzitutto un nuovo TextView nel layout della MainActivity (activity_main.xml), tale da contenere la stringa contenente data e ora aggiornati, poi passeremo a modificare il comportamento nel file della MainActivity

```
<TextView
android:id="@+id/dateTimeTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="@android:dimen/app_icon_size"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@id/taskListView"
app:layout_constraintBottom_toTopOf="@id/addTaskButton"
android:layout_marginBottom="10dp"/>
```

Questo è l'ID che useremo per riferirci a questo componenete del layout

Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1

companion object {                  //aggiunto fase 6: data
    private const val LOG_TAG = "MainActivityLog"

    private fun getCurrentTimeStamp(): String {
        val simpleDateFormat = java.text.SimpleDateFormat("yyyy-MM-dd HH:mm",
Locale.ITALY)
        val now = Date()
        return simpleDateFormat.format(now)
    }
}
private val taskList = mutableListOf<String>()
private val adapter by lazy { makeAdapter(taskList) }

//aggiunto fase 6: data
private val tickReceiver by lazy { makeBroadcastReceiver() }
```

Inizializziamo "*by lazy*" l'elemento del receiver, tramite una funzione *makeBroadcastReceiver* che dovremo implementare

Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1

companion object {                  //aggiunto fase 6: data
    private const val LOG_TAG = "MainActivityLog"

    private fun getCurrentTimeStamp(): String {
        val simpleDateFormat = java.text.SimpleDateFormat("yyyy-MM-dd HH:mm",
Locale.ITALY)
        val now = Date()
        return simpleDateFormat.format(now)
    }
}
private val taskList = mutableListOf<String>()
private val adapter by lazy { makeAdapter(taskList) }

//aggiunto fase 6: data
private val tickReceiver by lazy { makeBroadcastReceiver() }
```

Creiamo un companion object contenente un valore costante che funzionerà da tag identificativo e una funzione per acquisire nel formato prescelto la data e l'ora attuali

Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val ADD_TASK_REQUEST = 1

companion object {                  //aggiunto fase 6: data
    private const val LOG_TAG = "MainActivityLog"

    private fun getCurrentTimeStamp(): String {
        val simpleDateFormat = java.text.SimpleDateFormat("yyyy-MM-dd HH:mm",
Locale.ITALY)
        val now = Date()
        return simpleDateFormat.format(now)
    }
}
private val taskList = mutableListOf<String>()
private val adapter by lazy { makeAdapter(taskList) }

//aggiunto fase 6: data
private val tickReceiver by lazy { makeBroadcastReceiver() }
```

Step 6 – Data e ora tramite Broadcast Receiver

Aggiungiamo questa funzione in fondo alla classe MainActivity

```
private fun makeBroadcastReceiver(): BroadcastReceiver { //aggiunto fase 6: data
    return object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent?) {
            if (intent?.action == Intent.ACTION_TIME_TICK) {
                dateTimeTextView.text = getCurrentTimeStamp() //dobbiamo aver
creato il textview con questo id nel layout
            }
        }
    }
}
```

Questa funzione ritorna un oggetto *BroadcastReceiver*, nel quale viene già implementata tramite *override* la funzione *onReceive* che si attiva al ricevimento di un intent broadcast, e qui, se è ricevuto un *ACTION_TIME_TICK,* andiamo ad aggiornare l'orario visualizzato. Ricordiamo che questa funzione è chiamata per inizializzare l'elemento che abbiamo chiamato "*tickReceiver*".

Step 6 – Data e ora tramite Broadcast Receiver

Aggiungiamo questa funzione in fondo alla classe MainActivity

```
private fun makeBroadcastReceiver(): BroadcastReceiver { //aggiunto fase 6: data
    return object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent?) {
            if (intent?.action == Intent.ACTION_TIME_TICK) {
                dateTimeTextView.text = getCurrentTimeStamp() //dobbiamo aver
creato il textview con questo id nel layout
            }
        }
    }
}
```

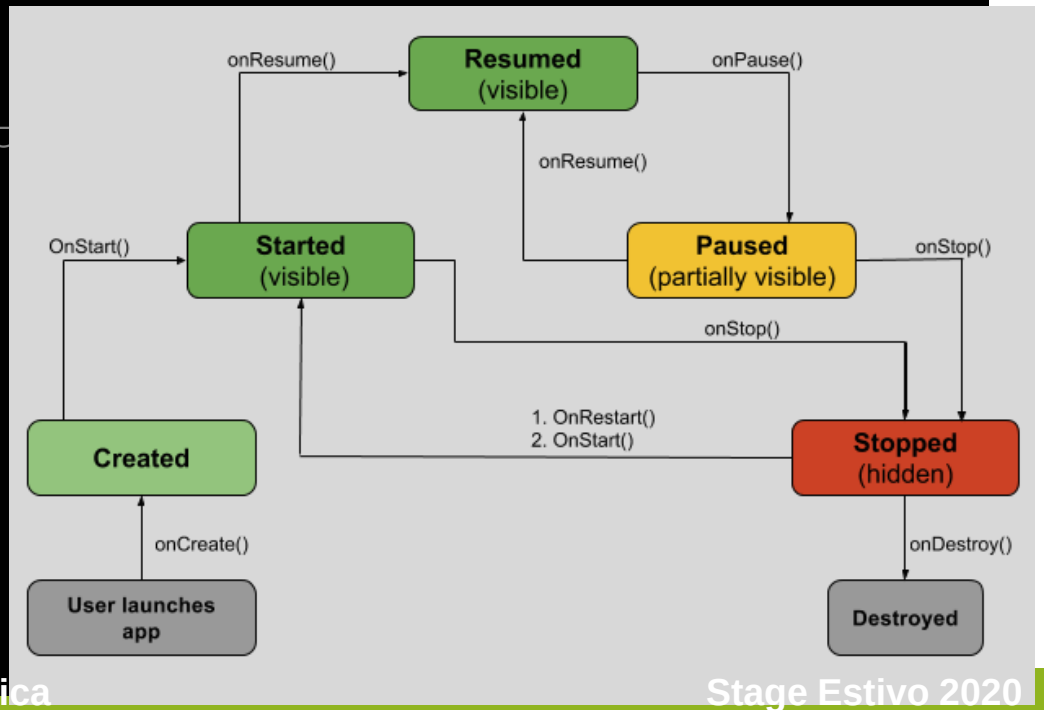
Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
override fun onResume() { //aggiunto fase 6: data
    // 1
    super.onResume()
    // 2
    dateTimeTextView.text = getCurrentTimeStamp()
textview nel layout
    // 3
    registerReceiver(tickReceiver, IntentFilter(Intent.ACTION_TIME_TICK))
}

override fun onPause() { //aggiunto fase 6: dat
    // 4
    super.onPause()
    // 5
    try {
        unregisterReceiver(tickReceiver)
    } catch (e: IllegalArgumentException) {
        Log.e(MainActivity.LOG_TAG, "Time tick
    }
}
```

Qui facciamo l'override della *onResume*, ricordiamo il ciclo di vita dell'activity, quindi nel momento in cui essa torna ad essere attiva e visibile all'utente, alla partenza o dopo una pausa, aggiorniamo subito l'orario, e poi associamo al nostro BroadcastReceiver "*tickReceiver*" l'intent *ACTION_TIME_TICK*



Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
override fun onResume() { //aggiunto fase 6: data
    // 1
    super.onResume()
    // 2
    dateTimeTextView.text = getCurrentTimeStamp() //dovremmo aver creato il
textview nel layout
    // 3
    registerReceiver(tickReceiver, IntentFilter(Intent.ACTION_TIME_TICK))
}

override fun onPause() { //aggiunto fase 6: data
    // 4
    super.onPause()
    // 5
    try {
        unregisterReceiver(tickReceiver)
    } catch (e: IllegalArgumentException) {
        Log.e(MainActivity.LOG_TAG, "Time tick Receiver not registered", e)
    }
}
```

Nell'override della *onPause*, in un blocco try/catch per la gestione di eventuali eccezioni, è buona pratica deregistrare il *tickReceiver* in modo tale che non continui a ricevere segnalazioni ogni minuto svolgendo operazioni mentre l'activity non è attiva

Step 6 – Data e ora tramite Broadcast Receiver

MainActivity.kt

```
override fun onResume() { //aggiunto fase 6: data
    // 1
    super.onResume()
    // 2
    dateTimeTextView.text = getCurrentTimeStamp() //dovremmo aver creato il
textview nel layout
    // 3
    registerReceiver(tickReceiver, IntentFilter(Intent.ACTION_TIME_TICK))
}

override fun onPause() { //aggiunto fase 6: data
    // 4
    super.onPause()
    // 5
    try {
        unregisterReceiver(tickReceiver)
    } catch (e: IllegalArgumentException) {
        Log.e(MainActivity.LOG_TAG, "Time tick Receiver not registered", e)
    }
}
```

Step 7 – Dati persistenti al lancio dell'app

Ora che le funzionalità ci sono, passiamo a risolvere i problemi che ci rimangono, primo fra tutti dobbiamo far sì che la nostra app, al momento della chiusura salvi il suo stato attuale per poi essere ripreso ad un nuovo lancio, in modo tale da non perdere nulla dei nostri loschi piani salvati nella To-Do-list

Step 7 – Dati persistenti al lancio dell'app

Aggiungiamo in cima alla MainActivity queste proprietà:

```
private val PREFS_TASKS = "prefs_tasks"
private val KEY_TASKS_LIST = "tasks_list"
```

Poi dopo la sovrascrittura della *onPause* andiamo a implementare anche la *onStop* per permettere alla Activity di **salvare il suo stato** quando viene terminata:

```
override fun onStop() {
    super.onStop()

    // Salva tutti i dati da rendere persistenti
    val savedList = StringBuilder()
    for (task in taskList) {
        savedList.append(task)
        savedList.append(",")
    }

    getSharedPreferences(PREFS_TASKS, Context.MODE_PRIVATE).edit()
        .putString(KEY_TASKS_LIST, savedList.toString()).apply()
}
```

In questo modo creiamo una stringa contenente tutti i task presenti nella lista separati da una virgola

Step 7 – Dati persistenti al lancio dell'app

Aggiungiamo in cima alla MainActivity queste proprietà:

```
private val PREFS_TASKS = "prefs_tasks"
private val KEY_TASKS_LIST = "tasks_list"
```

Poi dopo la sovrascrittura della *onPause* andiamo a implementare anche la *onStop* per permettere alla Activity di **salvare il suo stato** quando viene terminata:

```
override fun onStop() {
    super.onStop()

    // Salva tutti i dati da rendere persistenti
    val savedList = StringBuilder()
    for (task in taskList) {
        savedList.append(task)
        savedList.append(",")
    }

    getSharedPreferences(PREFS_TASKS, Context.MODE_PRIVATE).edit()
        .putString(KEY_TASKS_LIST, savedList.toString()).apply()
}
```

Step 7 – Dati persistenti al lancio dell'app

getSharedPreferences

Added in API level 1

```
public abstract SharedPreferences getSharedPreferences (String name,
                int mode)
```

Retrieve and hold the contents of the preferences file 'name', returning a SharedPreferences through which you can retrieve and modify its values. Only one instance of the SharedPreferences object is returned to any callers for the same name, meaning they will see each other's edits as soon as they are made. This method is thead-safe.

Parameters	
name	**String**: Desired preferences file. If a preferences file by this name does not exist, it will be created when you retrieve an editor (SharedPreferences.edit()) and then commit changes (Editor.commit()).
mode	**int**: Operating mode. Value is either **0** or a combination of MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE, and MODE_MULTI_PROCESS

Returns	
SharedPreferences	The single SharedPreferences instance that can be used to retrieve and modify the preference values.

Step 7 – Dati persistenti al lancio dell'app

Aggiungiamo in cima alla MainActivity queste proprietà:

```
private val PREFS_TASKS = "prefs_tasks"
private val KEY_TASKS_LIST = "tasks_list"
```

Poi dopo la sovrascrittura della *onPause* andiamo a implementare anche la *onStop* per permettere alla Activity di **salvare il suo stato** quando viene terminata:

```
override fun onStop() {
    super.onStop()

    // Save all data which you want to persist
    val savedList = StringBuilder()
    for (task in taskList) {
        savedList.append(task)
        savedList.append(",")
    }

    getSharedPreferences(PREFS_TASKS, Context.MODE_PRIVATE).edit()
        .putString(KEY_TASKS_LIST, savedList.toString()).apply()
}
```

Richiamaiamo il file di nome "*prefs_tasks*" contenente i dati salvati, e se non esiste viene creato. Con il metodo *edit()* abilitiamo la modifica e con putstring() inseriamo, attribuendogli un nome-chiave, la lista salvata come stringa "*savedList*"

Step 7 – Dati persistenti al lancio dell'app

Aggiungiamo in cima alla MainActivity queste proprietà:

```
private val PREFS_TASKS = "prefs_tasks"
private val KEY_TASKS_LIST = "tasks_list"
```

Poi dopo la sovrascrittura della *onPause* andiamo a implementare anche la *onStop* per permettere alla Activity di **salvare il suo stato** quando viene terminata:

```
override fun onStop() {
    super.onStop()

    // Save all data which you want to persist.
    val savedList = StringBuilder()
    for (task in taskList) {
        savedList.append(task)
        savedList.append(",")
    }

    getSharedPreferences(PREFS_TASKS, Context.MODE_PRIVATE).edit()
        .putString(KEY_TASKS_LIST, savedList.toString()).apply()
}
```

Step 7 – Dati persistenti al lancio dell'app

A questo punto andiamo a modificare la onCreate in modo tale che quando viene chiamata, alla riapertura dell'app, non vada a creare una activity nuova e pulita, ma che vada a leggere l'ultimo stato salvato:

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

  ...
    taskList.add("Item1")
    taskList.add("Item2")

    //aggiunto step 7: Dati persistenti
    val savedList = getSharedPreferences(PREFS_TASKS,
Context.MODE_PRIVATE).getString(KEY_TASKS_LIST, null)

    if (savedList != null) {
        val items = savedList.split(",".toRegex()).dropLastWhile {
it.isEmpty() }.toTypedArray()
        taskList.addAll(items)
    }
}
```

Proviamo i cambiamenti

- Ora la nostra app tiene in memoria la lista così com'è! ☺
- Ci accorgiamo anche che così facendo gli elementi di default che avevamo inserito direttamente nella *onCreate* si riproducono come funghi, ogni volta che chiudiamo e riapriamo, ma anche ogni volta che ruotiamo il telefono!
  - Questo da un certo punto di vista è un bel segno, significa che effettivamente la nostra modifica funziona:
    - alla apertura della app vengono creati e inseriti in lista per la prima volta
    - poi alla chiusura della activity, chiamando la *onStop*, i dati vengono salvati, compresi quegli elementi
    - infine alla riapertura, chimando di nuovo la *onCreate*, abbiamo il risultato spiacevole di vedere gli stessi elementi inseriti nuovamente in testa alla lista, e in seguito viene richiamato il contenuto della lista salvato in precedenza, che già contine quegli stessi elementi
- Teniamo ancora gli elementi di default per vedere i prossimi effetti, ma ora forse è il caso di riuscire ad eliminare qualche elemento dalla nostra lista

Step 8 – Cancellare elementi dalla lista

Aggiungiamo una funzione in fondo alla MainActivity per attivare una finestra di dialogo, e questa verrà chiamata poi dall'handler dell'evento "click" sugli elementi della lista.

```
private fun taskSelected(position: Int) {

    AlertDialog.Builder(this)

        .setTitle(R.string.alert_title)
        .setMessage(taskList[position])
        .setPositiveButton(R.string.delete, { _, _ ->
            taskList.removeAt(position)
            adapter.notifyDataSetChanged()
        })
        .setNegativeButton(R.string.cancel, {
                dialog, _ -> dialog.cancel()
        })
        .create()
        .show()
}
```

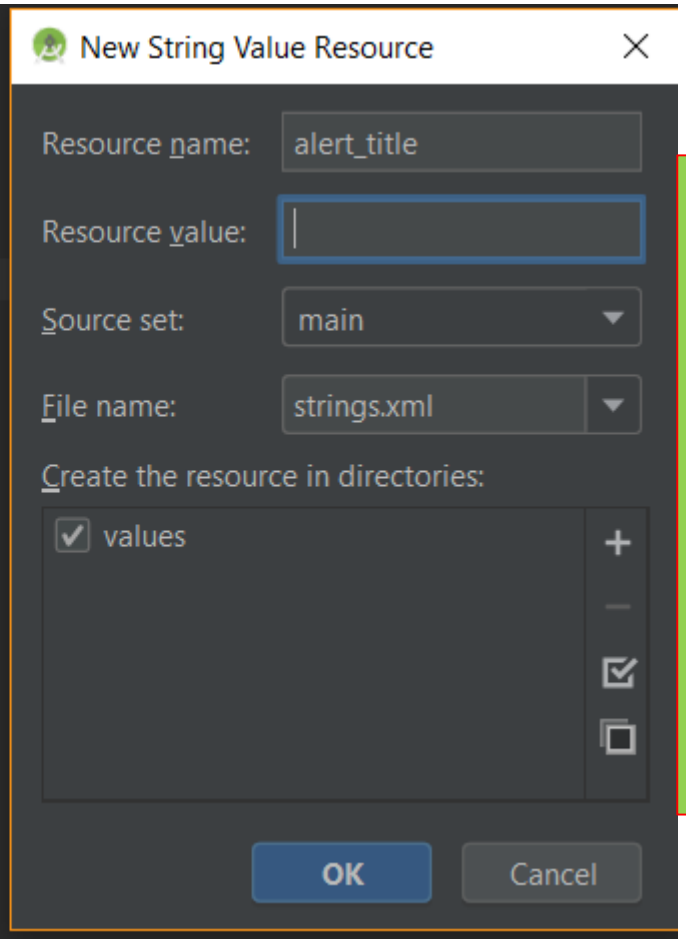
Operazioni chiamate premendo il pulsante di "delete" per eliminare l'elemento, con riferimento alla posizione in lista

Step 8 – Cancellare elementi dalla lista

Aggiungiamo una funzione in fondo alla MainActivity per attivare una finestra di dialogo, e questa verrà chiamata poi dall'handler dell'evento "click" sugli elementi della lista.

```
}

private fun taskSelected(position: Int) {
    // 1
    AlertDialog.Builder( context: this)
        // 2
        .setTitle(R.string.alert_title)
        // 3
        .setMessage(taskList[position])
        .setPositiveButton(R.string.delete, { _, _ ->
            taskList.removeAt(position)
            adapter.notifyDataSetChanged()
        })
        .setNegativeButton(R.string.cancel, {
                dialog, _ -> dialog.cancel()
        })
        // 4
        .create()
        // 5
        .show()
}
```



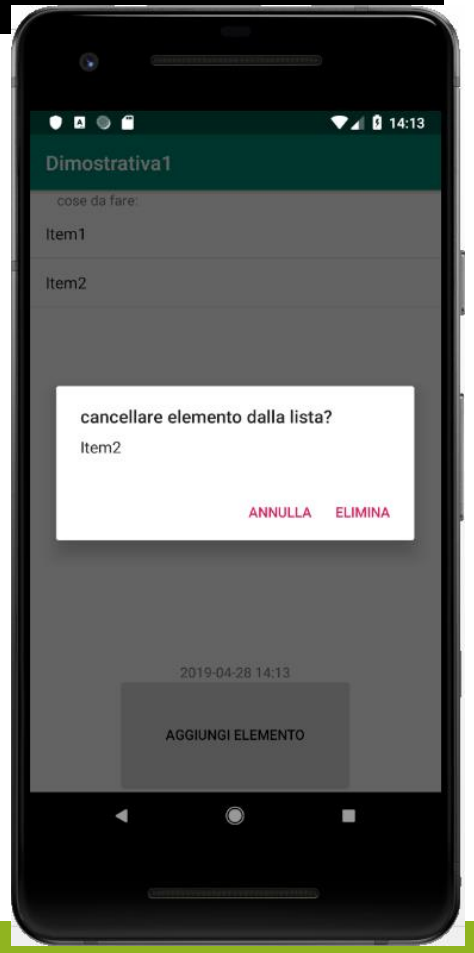
Si può definire l'ID delle stringhe che vogliamo usare nella finestra di dialogo, verranno segnalate in rosso in quanto non esistono nel file *string.xml*, e con Alt+Invio aggiungerle a tale file, dopo aver inserito il valore della stringa nella finestra che si aprirà.
Resource value: "Eliminare?"

Step 8 – Cancellare elementi dalla lista

Andiamo a dire, nella *onCreate,* qual è la funzione da chiamare al momento del "click" su uno degli elementi della lista, aggiornando questo elemento:

```
taskListView.onItemClickListener = AdapterView.OnItemClickListener {
parent, view, position, id -> taskSelected(position)}
```

NOTA: A questo punto sempre con Alt+Invio ci viene anche suggerito di sostituire *parent, view* e *id* con un "_" in quanto non usati, e lo possiamo fare.



Tutto molto bello ma…

- Ancora non possiamo dirci veramente soddisfatti:
  - Sono molti gli eventi che causano la chiusura e ricreazione dell'app
  - Anche la finestra di dialogo sparisce se ruotiamo il telefono e ancora si riproducono come funghi gli elementi di default
  - Non è bene che l'Activity sia chiusa e ricreata ruotando lo schermo:
    - Potrebbe essere in esecuzione qualche evento o operazione su quella Activity
    - Non ci piacerebbe se, per esempio, guardando un video o ascoltando un audio tramite un'Activity di una certa app, tutto riparta da zero ogni volta che ruotiamo l'immagine
    - O peggio che le operazioni lanciate dalla Activity si moltiplichino a dismisura!
- Bisogna trovare il modo di gestire i cambiamenti di configurazione

Step 9 – Gestione cambiamenti di configurazione


```
<activity android:name=".MainActivity">
```

Troviamo il tag della MainActivity nel **Manifest**, e aggiungiamo il codice in questo modo:

```
<activity android:name=".MainActivity" android:configChanges="orientation|screenSize">
```

Non necessario, ma chiarificatore della funzione che viene chiamata nella MainActivity, si potrebbe inserire dopo l'override della *onStop*:

```
override fun onConfigurationChanged(newConfig: Configuration?) {
    super.onConfigurationChanged(newConfig)
}
```

Step 9 – Gestione cambiamenti di configurazione

Ora possiamo anche cancellare i fastidiosi elementi di default, che non è corretto inserire (ed ora sappiamo il perché), dal momento che la nostra app ora è pronta!

```
override fun onCreate(savedInstanceState: Bundle?) {
…
    taskListView.onItemClickListener =
AdapterView.OnItemClickListener { parent, view, position, id ->
taskSelected(position)}

    taskList.add("Item1")
    taskList.add("Item2")

}
```

Pubblicazione

- Fare il debug del codice
- Aggiornare la versione del programma
- Gradle è un tool per l'automatizzazione dei task più comuni eseguiti in un progetto. Si occupa principalmente della compilazione, della gestione delle dipendenze e del deploy di un'app.
- Procedimento: **Build** -> **Generate Signed APK**, poi **Generate Signed APK Wizard** su **Key store path** scegliendo **Create new.** Poi **ok** e **Finish**.
- Oppure Build < Build Bundles < Build APK e controlla dove salva il .apk (locate)
- **Versa una tantum 25$ dal tuo profilo da** https://play.google.com/apps/publish
- **Pensa a come monetizzare**
- Premi "Aggiungi nuova applicazione" e trascina il .apk
- Falla scaricare ai tuoi amici!

You don’t learn to walk by following the rules. You learn by doing and by falling over

Richard Branson

Grazie a tutti per la partecipazione

Bibliografia

- https://koenig-media.raywenderlich.com/uploads/2015/09/activity_lifecycle_pyramid.png
- https://developer.android.com/reference/android/app/Activity
- https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/let.html
- https://kotlinlang.org/docs/reference/basic-syntax.html
- https://www.raywenderlich.com/6754-a-comparison-of-swift-and-kotlin-languages
- https://www.baeldung.com/tag/core-kotlin
- https://developer.android.com/reference/android/content/SharedPreferences.html
- https://medium.com/androiddevelopers/who-lives-and-who-dies-process-priorities-on-android-cb151f39044f
- https://www.html.it/pag/19496/