



# Lezioni 4-5



# Programmazione Android



- Componenti di un'applicazione
- AndroidManifest.xml
- Le Activity
  - Definizione
  - Ciclo di vita
  - Layout & View e Interazione
    - cenni
  - Un esempio completo



# Componenti di un'applicazione Android



# Componenti di un'applicazione



- Abbiamo già visto che le applicazioni Android non sono un blocco monolitico, ma un **insieme di componenti cooperanti**
- **Activity**
  - Un'attività atomica dell'utente
  - Concretizzata da una “schermata”
  - Può essere composta da vari **Fragment**
- **Service**
  - Un'attività del sistema o dell'app, invisibile all'utente
  - Eseguita “in background”
  - Non interagisce con l'utente
  - Può interagire con le applicazioni (in vari modi)



# Componenti di un'applicazione



- Abbiamo già visto che le applicazioni Android non sono un blocco monolitico, ma un **insieme di componenti cooperanti**
- **Content Provider**
  - Un componente che pubblica “contenuti”
  - Offre un'interfaccia programmatica
  - Utilizzato da altre applicazioni
- **Broadcast Receiver**
  - Un componente che “ascolta” messaggi globali
  - Quando riceve un messaggio di suo interesse, esegue del codice specifico (per esempio, lancia un'attività o dialoga con un servizio)



# Componenti di un'applicazione



- I componenti di un'applicazione dialogano attraverso un **sistema di messaggistica** che è alla base di Android
- **Intent**
  - Un “messaggio” che esprime *un'intenzione* dell'utente o di una applicazione affinché qualcosa avvenga
  - Numerosissimi Intent di sistema, ogni app può definirne altri
  - Una parte della struttura del messaggio è fissata, ma possono includere dati “extra” a piacere
  - Gli Intent possono essere indirizzati a uno specifico componente, oppure emessi in broadcast
  - Ogni applicazione può definire un filtro che dichiara a quali Intent è interessata (e può eventualmente rispondere)

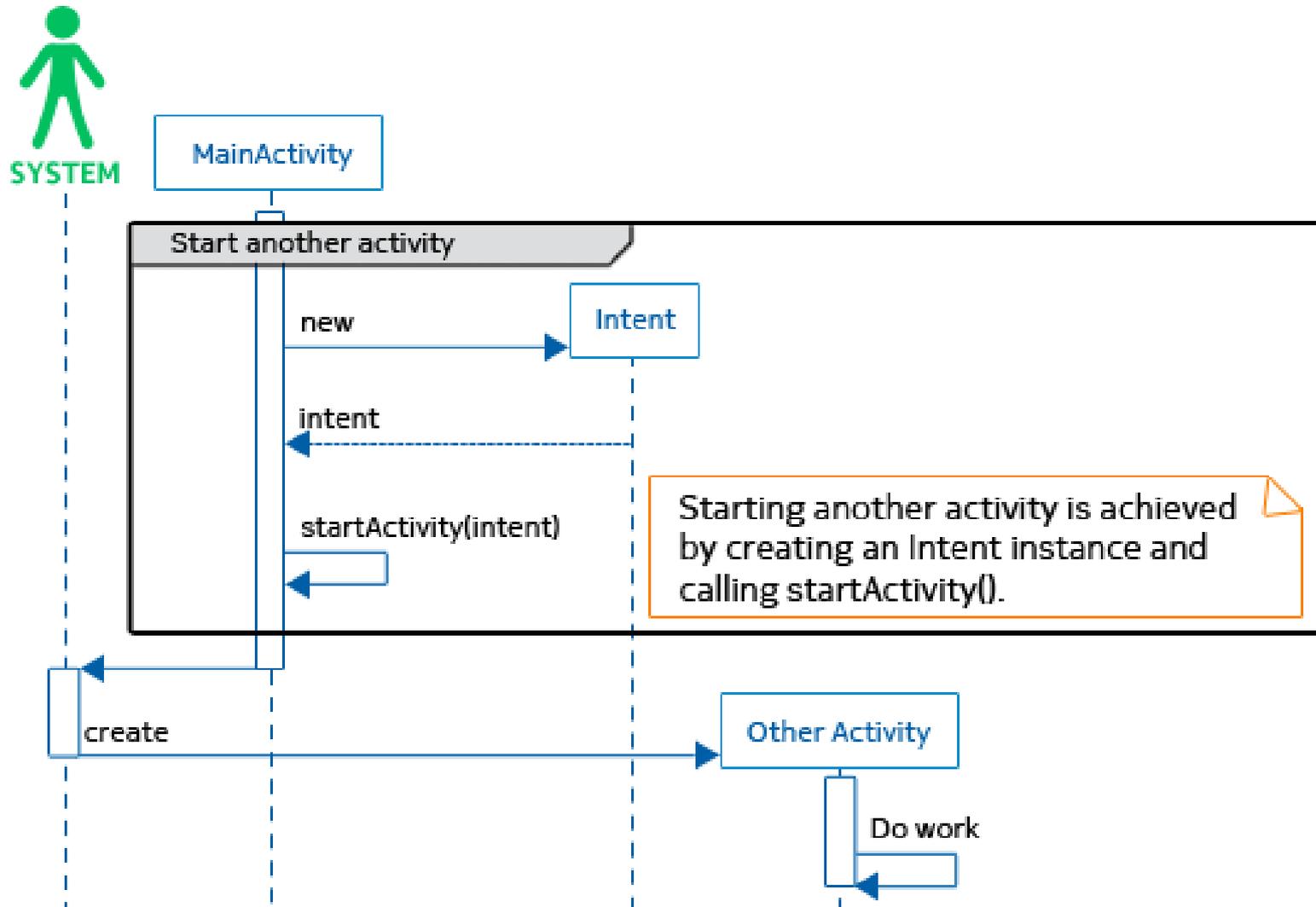


# Componenti di un'applicazione



- Una tipica applicazione Android esegue così:
  - Il Launcher (che è esso stesso un **Activity**) avvia la prima **Activity** dell'App, inviandole un **Intent** che indica l'intenzione di lanciarla
  - L'**Activity** chiama `setLayout()` per impostare la sua UI
  - Il sistema chiama certi metodi dell'**Activity** (*callback*) in risposta alle azioni dell'utente
  - A seconda dei casi, questi metodi potranno
    - lanciare altre **Activity** (inviando loro opportuni **Intent**), sia dell'applicazione sia di altre applicazioni
    - Inviare **Intent** ad **Activity** già in esecuzione o, in broadcast, a tutti gli interessati
    - Interagire con **Services** in background
    - Recuperare o salvare dati tramite **Content Provider**
    - Terminare l'**Activity** (tornando alla precedente)

# L'avvio di una Activity



# L'avvio di una Activity



## Explicit Intent

- Possiamo creare un Intent che chiede **una Activity**
  - Se l'Activity in questione non è in esecuzione, viene lanciata
  - Se è già in esecuzione, viene “svegliata”
  - Viene recapitato l'Intent e l'Activity destinataria diventa “attiva”

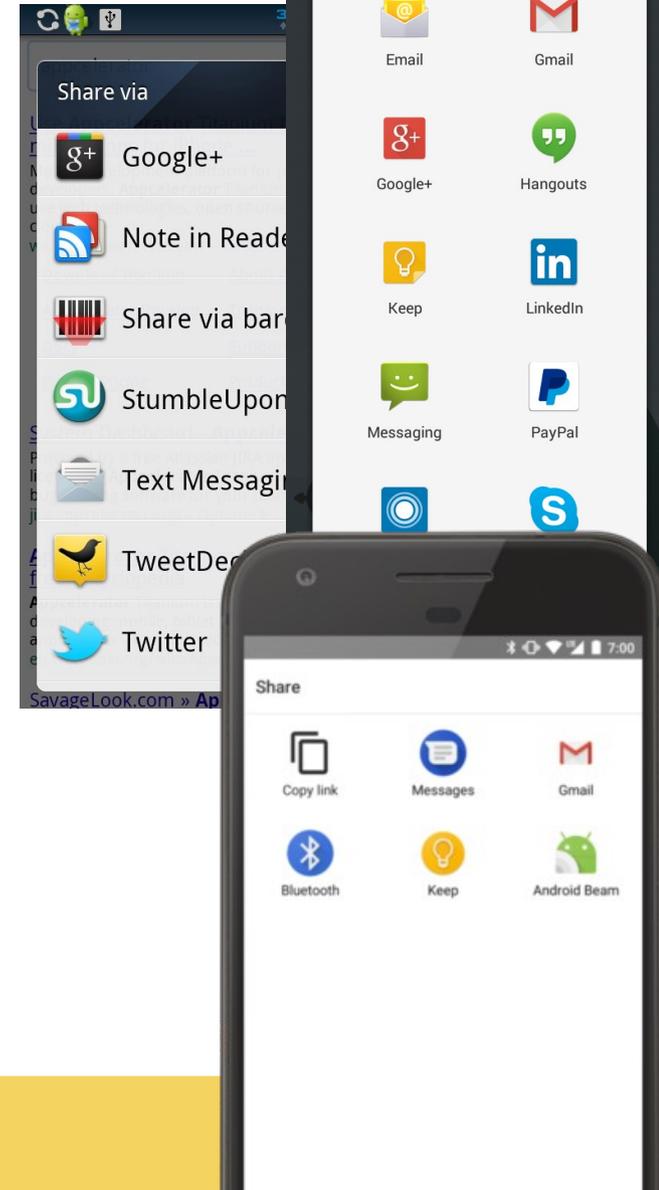
## Implicit Intent

- Possiamo creare un Intent che chiede **una funzione**
  - Il sistema cerca quale Activity può rispondere
  - Se ne trova una, la attiva
  - Se ne trova più di una, chiede all'utente quale lanciare
  - Se non ne trova nessuna, l'avvio fallisce

# Esempio di Implicit Intent



- Per esempio, l'Intent predefinito **ACTION\_SEND** viene usato per indicare l'intenzione di inviare qualcosa a qualcuno
  - L'App che vuole inviare dati crea un Intent implicito con **ACTION\_SEND**, ci mette dentro i dati da inviare, e lo affida al sistema
  - Il sistema cerca tutte le app installate che supportano **ACTION\_SEND**, e chiede all'utente di scegliere quale usare
  - L'utente ne seleziona una, il sistema avvia l'Activity corrispondente, e questa invia il dato





# Implicit Intent di sistema (Android 4.0)



ACTION_AIRPLANE_MODE_CHANGED	AILABLE	ACTION_MEDIA_SCANNER_SCAN_FILE	ACTION_RUN
ACTION_ALL_APPS	ACTION_EXTERNAL_APPLICATIONS_UNAVAILABLE	ACTION_MEDIA_SCANNER_STARTED	ACTION_SCREEN_OFF
ACTION_ANSWER	ACTION_FACTORY_TEST	ACTION_MEDIA_SHARED	ACTION_SCREEN_ON
ACTION_APP_ERROR	ACTION_GET_CONTENT	ACTION_MEDIA_UNMOUNTABLE	ACTION_SEARCH
ACTION_ATTACH_DATA	ACTION_GTALK_SERVICE_CONNECTED	ACTION_MEDIA_UNMOUNTED	ACTION_SEARCH_LONG_PRESS
ACTION_BATTERY_CHANGED	ACTION_GTALK_SERVICE_DISCONNECTED	ACTION_MY_PACKAGE_REPLACED	ACTION_SEND
ACTION_BATTERY_LOW	ACTION_HEADSET_PLUG	ACTION_NEW_OUTGOING_CALL	ACTION_SEND_MULTIPLE
ACTION_BATTERY_OKAY	ACTION_INPUT_METHOD_CHANGED	ACTION_PACKAGE_ADDED	ACTION_SENDTO
ACTION_BOOT_COMPLETED	ACTION_INSERT	ACTION_PACKAGE_CHANGED	ACTION_SET_WALLPAPER
ACTION_BUG_REPORT	ACTION_INSERT_OR_EDIT	ACTION_PACKAGE_DATA_CLEARED	ACTION_SHUTDOWN
ACTION_CALL	ACTION_INSTALL_PACKAGE	ACTION_PACKAGE_FIRST_LAUNCH	ACTION_SYNC
ACTION_CALL_BUTTON	ACTION_LOCALE_CHANGED	ACTION_PACKAGE_FULLY_REMOVED	ACTION_SYSTEM_TUTORIAL
ACTION_CAMERA_BUTTON	ACTION_MAIN	ACTION_PACKAGE_INSTALL	ACTION_TIME_CHANGED
ACTION_CHOOSER	ACTION_MANAGE_NETWORK_USAGE	ACTION_PACKAGE_NEEDS_VERIFICATION	ACTION_TIME_TICK
ACTION_CLOSE_SYSTEM_DIALOGS	ACTION_MANAGE_PACKAGE_STORAGE	ACTION_PACKAGE_REMOVED	ACTION_TIMEZONE_CHANGED
ACTION_CONFIGURATION_CHANGED	ACTION_MEDIA_BAD_REMOVAL	ACTION_PACKAGE_REPLACED	ACTION_UID_REMOVED
ACTION_CREATE_SHORTCUT	ACTION_MEDIA_BUTTON	ACTION_PACKAGE_RESTARTED	ACTION_UMS_CONNECTED
ACTION_DATE_CHANGED	ACTION_MEDIA_CHECKING	ACTION_PASTE	ACTION_UMS_DISCONNECTED
ACTION_DEFAULT	ACTION_MEDIA_EJECT	ACTION_PICK	ACTION_UNINSTALL_PACKAGE
ACTION_DELETE	ACTION_MEDIA_MOUNTED	ACTION_PICK_ACTIVITY	ACTION_USER_PRESENT
ACTION_DEVICE_STORAGE_LOW	ACTION_MEDIA_NOFS	ACTION_POWER_CONNECTED	ACTION_VIEW
ACTION_DEVICE_STORAGE_OK	ACTION_MEDIA_REMOVED	ACTION_POWER_DISCONNECTED	ACTION_VOICE_COMMAND
ACTION_DIAL	ACTION_MEDIA_SCANNER_FINISHED	ACTION_POWER_USAGE_SUMMARY	ACTION_WALLPAPER_CHANGED
ACTION_DOCK_EVENT		ACTION_PROVIDER_CHANGED	ACTION_WEB_SEARCH
ACTION_EDIT		ACTION_REBOOT	
ACTION_EXTERNAL_APPLICATIONS_AVAILABLE			



# Implicit Intent di sistema (Android 4.0)



## Le più frequenti

ACTION\_AIRPLANE\_MODE\_CHANGED  
ACTION\_ALL\_APPS  
ACTION\_ANSWER  
ACTION\_APP\_ERROR  
ACTION\_ATTACH\_DATA  
ACTION\_BATTERY\_CHANGED  
ACTION\_BATTERY\_LOW  
ACTION\_BATTERY\_OKAY  
ACTION\_BOOT\_COMPLETED  
ACTION\_BUG\_REPORT  
ACTION\_CALL  
ACTION\_CALL\_BUTTON  
ACTION\_CAMERA\_BUTTON  
ACTION\_CHOOSER  
ACTION\_CLOSE\_SYSTEM\_DIALOGS  
ACTION\_CONFIGURATION\_CHANGED  
ACTION\_CREATE\_SHORTCUT  
ACTION\_DATE\_CHANGED  
ACTION\_DEFAULT  
ACTION\_DELETE  
ACTION\_DEVICE\_STORAGE\_LOW  
ACTION\_DEVICE\_STORAGE\_OK  
ACTION\_DIAL  
ACTION\_DOCK\_EVENT  
ACTION\_EDIT  
ACTION\_EXTERNAL\_APPLICATIONS\_AV

AILABLE  
ACTION\_EXTERNAL\_APPLICATIONS\_U  
NAVAILABLE  
ACTION\_FACTORY\_TEST  
ACTION\_GET\_CONTENT  
ACTION\_GTALK\_SERVICE\_CONNECTE  
D  
ACTION\_GTALK\_SERVICE\_DISCONNECTE  
D  
ACTION\_HEADSET\_PLUG  
ACTION\_INPUT\_METHOD\_CHANGED  
ACTION\_INSERT  
ACTION\_INSERT\_OR\_EDIT  
ACTION\_INSTALL\_PACKAGE  
ACTION\_LOCALE\_CHANGED  
ACTION\_MAIN  
ACTION\_MANAGE\_NETWORK\_USAGE  
ACTION\_MANAGE\_PACKAGE\_STORAG  
E  
ACTION\_MEDIA\_BAD\_REMOVAL  
ACTION\_MEDIA\_BUTTON  
ACTION\_MEDIA\_CHECKING  
ACTION\_MEDIA\_EJECT  
ACTION\_MEDIA\_MOUNTED  
ACTION\_MEDIA\_NOFS  
ACTION\_MEDIA\_REMOVED  
ACTION\_MEDIA\_SCANNER\_FINISHED

ACTION\_MAIN  
ACTION\_VIEW  
ACTION\_ATTACH\_DATA  
ACTION\_EDIT  
ACTION\_PICK  
ACTION\_CHOOSER  
ACTION\_GET\_CONTENT  
ACTION\_DIAL  
ACTION\_CALL  
ACTION\_SEND  
ACTION\_SENDTO  
ACTION\_ANSWER  
ACTION\_INSERT  
ACTION\_DELETE  
ACTION\_RUN  
ACTION\_SYNC  
ACTION\_PICK\_ACTIVITY  
ACTION\_SEARCH  
ACTION\_WEB\_SEARCH  
ACTION\_FACTORY\_TEST

RUN  
SCREEN\_OFF  
SCREEN\_ON  
SEARCH  
SEARCH\_LONG\_PRESS  
SEND  
SEND\_MULTIPLE  
SENDTO  
SET\_WALLPAPER  
SHUTDOWN  
SYNC  
SYSTEM\_TUTORIAL  
TIME\_CHANGED  
TIME\_TICK  
TIMEZONE\_CHANGED  
UID\_REMOVED  
JMS\_CONNECTED  
JMS\_DISCONNECTED  
UNINSTALL\_PACKAGE  
USER\_PRESENT  
VIEW  
VOICE\_COMMAND  
WALLPAPER\_CHANGED  
WEB\_SEARCH



# Contenuti di un Intent



- **Action** – quale azione si vuole ottenere (String)
- **Data** – su quali dati operare (URI)
- **Category** – categoria dell'azione (String)
- **Type** – tipo MIME di Data (String)
  - viene ricavato da Data se non fornito; se è fornito fa override su quello di Data
- **Component** – componente a cui è indirizzato il messaggio
  - Solo per intent espliciti
- **Extras** – un Bundle (mappa chiavi → valori) di ulteriori campi
- **Flag** – al solito, quando hai dimenticato qualcosa...



# Intent & Intent Filter

- Abbiamo visto che il S.O. ha bisogno di sapere a quali Intent una nostra Activity è interessata o può rispondere, per inoltrare gli Implicit Intent...
  - ... anche se l'Activity non è in esecuzione!
- Ogni Activity dichiara dunque una serie di **Intent Filter** che indicano quali messaggi vuole ricevere
  - Attenzione: qualcuno potrebbe sempre mandarci Explicit Intent che non rispettano i filtri!



# Intent & Intent Filter

- Un Intent Filter consente di specificare quali Intent vogliamo ricevere, in base a
  - Action (indichiamo il nome dell'azione)
  - Category (indichiamo il nome della categoria)
  - Data (indichiamo il tipo MIME e/o lo schema dell'URI)
- Ogni componente (Activity o altro) dichiara i suoi Intent Filter fra i suoi metadati
  - AndroidManifest.xml → esempio fra poco

# Esempi



- Le due chiavi più frequentemente usate sono *action* e *data* (che è espressa come una URI)
  - action = **ACTION\_DIAL**  
data = **tel:0502212773**
    - parte la chiamata a un numero di telefono dato
  - action = **ACTION\_VIEW**  
data = **http://www.di.unipi.it**
    - parte il browser web sulla pagina indicata



# Esempi



- In codice:

- Chiamare un numero di telefono:

```
Intent i = new Intent(Intent.ACTION_CALL);  
i.setData(Uri.parse("tel:0502212773"));  
startActivity(i);
```

- Vedere una pagina web

```
Intent i = new Intent(Intent.ACTION_VIEW);  
i.setData(Uri.parse("http://www.di.unipi.it"));  
startActivity(i);
```

# Esempi



- Intent più complessi richiedono quasi sempre un certo numero di *extra*:
  - action = **ACTION\_SENDTO**  
mime = text/plain  
extras = {
    - EXTRA\_EMAIL: [ “gervasi@di.unipi.it” ]
    - EXTRA\_CC: [ “cisterni@di.unipi.it” ]
    - EXTRA\_SUBJECT: “Riunione di domani”
    - EXTRA\_TEXT: “Carissimi, ...”



# Esempi



- In codice:

```
Intent i = new Intent(Intent.ACTION_SENDTO);  
i.setType("text/plain");  
i.putExtra(Intent.EXTRA_EMAIL, new String[] { "gervasi@di.unipi.it" } );  
i.putExtra(Intent.EXTRA_CC, new String[] {"cisterni@di.unipi.it"});  
i.putExtra(Intent.EXTRA_SUBJECT, "Riunione di domani");  
i.putExtra(Intent.EXTRA_TEXT, "Carissimi, ...");  
startActivity(i);
```

# Esempi



- Spesso si usa il pattern *method chaining*
  - un metodo ritorna come risultato il suo *this*
  - si possono quindi concatenare le chiamate di metodi
- Esempio: impostare una sveglia

```
Intent i = new Intent(Intent.ACTION_SET_ALARM)
    .putExtra(Intent.EXTRA_HOUR, 16)
    .putExtra(Intent.EXTRA_MINUTES, 00)
    .putExtra(Intent.EXTRA_MESSAGE, "Lezione SAM")
    .putExtra(Intent.EXTRA_VIBRATE, true);
startActivity(i);
```



# AndroidManifest.xml

(primo sguardo)



# Contenuti di AndroidManifest.xml



- Si tratta del manifesto dell'Applicazione, che include:
  - **Configurazione** dell'app (nome, icona, package Java, ...)
  - Informazioni sui **permessi** necessari e definiti
  - Elenco dei **componenti** dell'applicazione
    - Configurazione di ciascun componente, inclusa classe Java
    - Dettagli sugli Intent dei vari componenti
  - Altri **metadati**
    - Librerie necessarie, strumenti di profiling, ecc.





# Esempio: Hello Android



```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unipi.di.masterapp.hello"
    android:versionCode="1" android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity android:label="@string/app_name"
            android:name=".HelloAndroidActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Java  
Risorse  
Letterali  
XML



# Esempio più complesso (struttura generale dei tag)



```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <uses-permission />
  <permission />
  <permission-tree />
  <permission-group />

  <instrumentation />
  <uses-sdk />
  <uses-configuration />
  <uses-feature />
  <supports-screens />
  <compatible-screens />
  <supports-gl-texture />

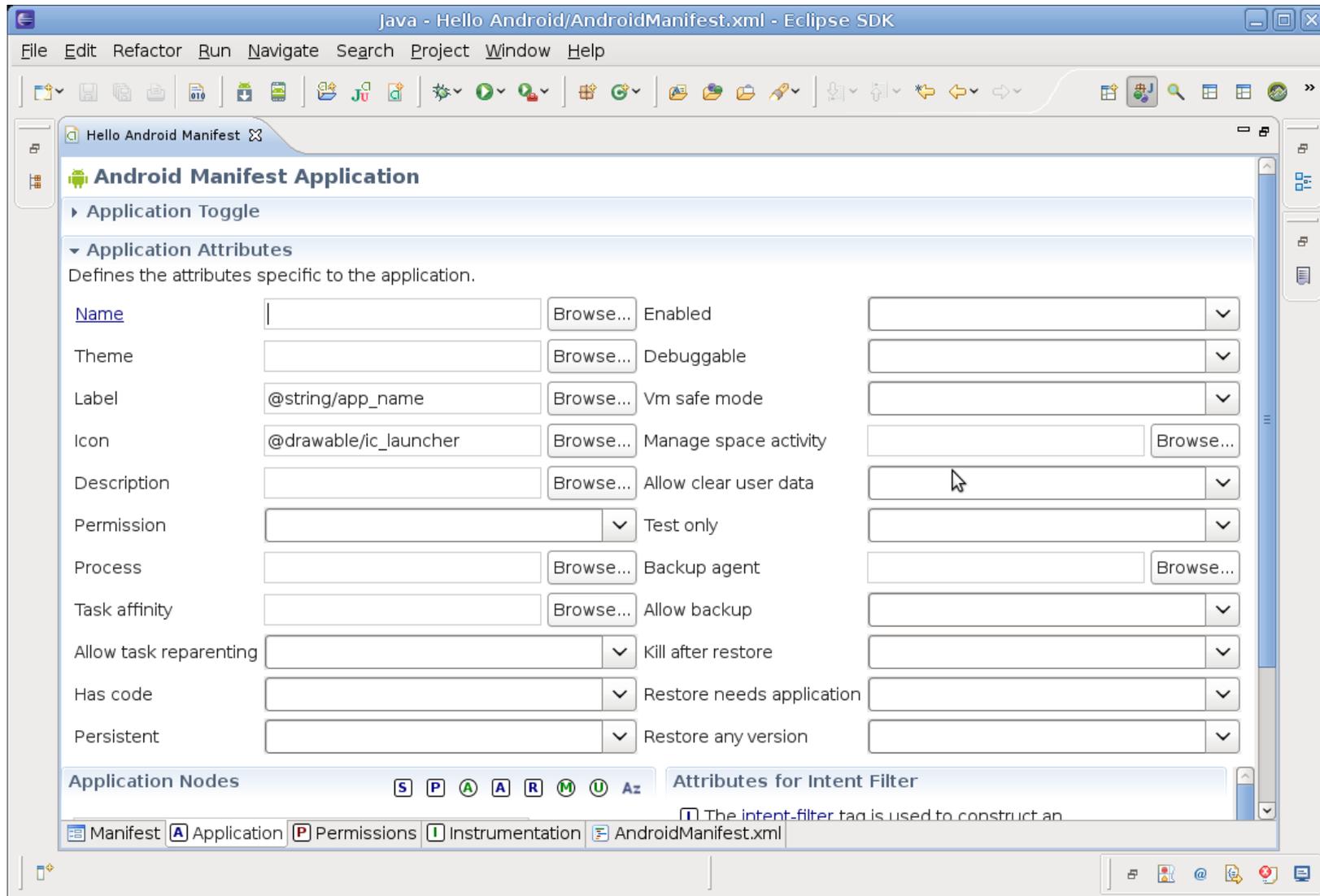
  <application>
    <activity>
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
      <meta-data />
```

```
</activity>
  <activity-alias>
    <intent-filter>..</intent-filter>
    <meta-data />
  </activity-alias>
  <service>
    <intent-filter>..</intent-filter>
    <meta-data/>
  </service>
  <receiver>
    <intent-filter>..</intent-filter>
    <meta-data />
  </receiver>
  <provider>
    <grant-uri-permission />
    <meta-data />
  </provider>

  <uses-library />
</application>
</manifest>
```

# Editing strutturato

- Eclipse fornisce un editor strutturato dedicato ad AndroidManifest.xml
- Lo vedremo in un esempio *in vivo*
- Di norma, si compone il manifesto passo-passo, man mano che si scrivono i componenti





# Manifest su AndroidStudio



Sviluppo Applicazioni Mobili  
V. Gervasi – a.a. 2018/19

- Android Studio *non* fornisce una GUI per l'editing
- Però il code completion funziona sui tag XML
  - Abbastanza comodo comunque
- Inoltre, il sistema di build effettua il **manifest merging** (per esempio, inserendo automaticamente nel manifest informazioni da Gradle)

The screenshot shows the Android Studio IDE with the `AndroidManifest.xml` file open. The XML code is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unipi.di.masterapp14.helloandroid" >

    <uses-permission android:name="" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="Hello Android"
        android:theme="@style/AppTheme" >

        <activity
            android:name="it.unipi.di.masterapp14.MainActivity"
            android:label="Hello Android"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

A code completion popup is visible over the `android:name=""` attribute, listing various permissions. The selected item is `android.permission.ACCESS_COARSE_LOCATION`. To the right, a documentation window for `ACCESS_COARSE_LOCATION` is open, showing the following details:

```
Documentation for ACCESS_COARSE_LOCATION
< Android API 21 Platform >

public static final String ACCESS_COARSE_LOCATION
Added in API level 1
Allows an app to access approximate location derived from network location
sources such as cell towers and Wi-Fi.
Constant Value: "android.permission.ACCESS_COARSE_LOCATION"
```

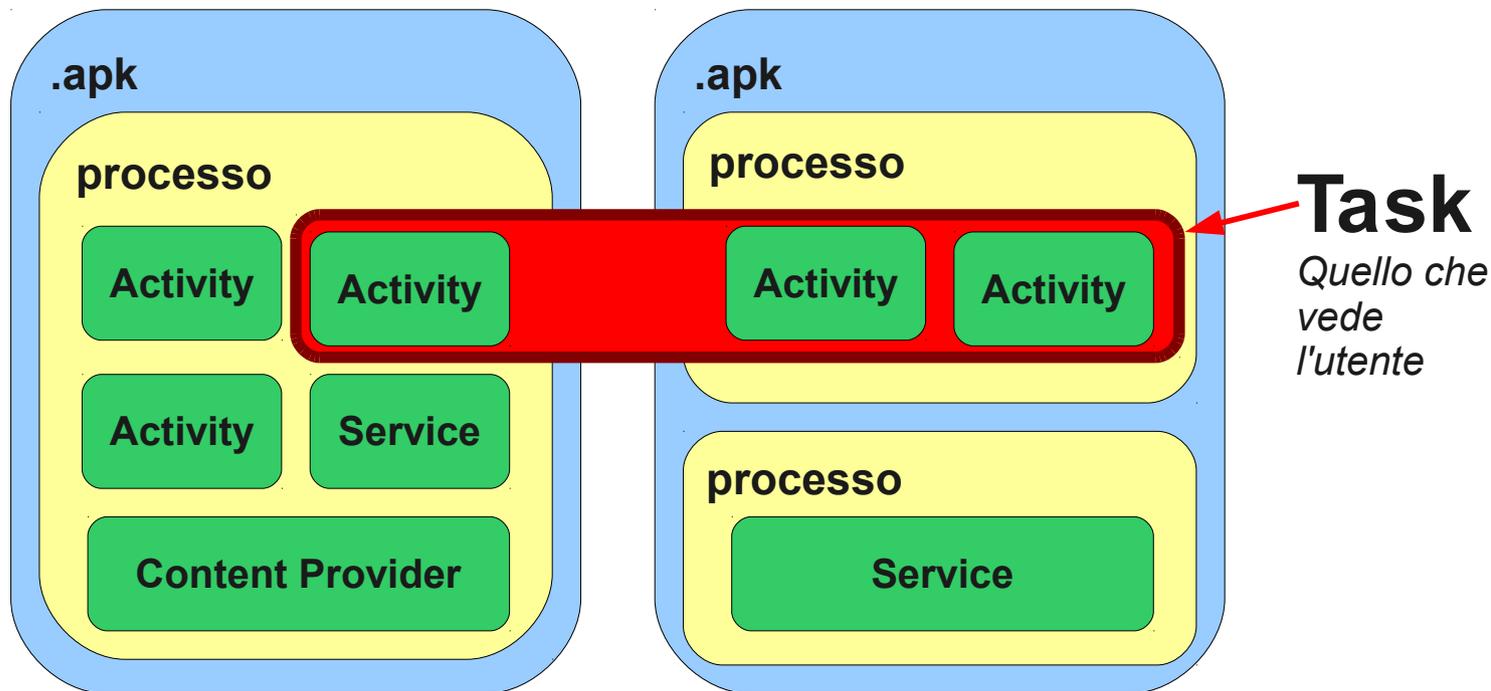


# Activity

# Activity



- Le activity sono
  - Uno dei componenti di un'applicazione
  - Uno dei componenti di un task (come visto dall'utente)





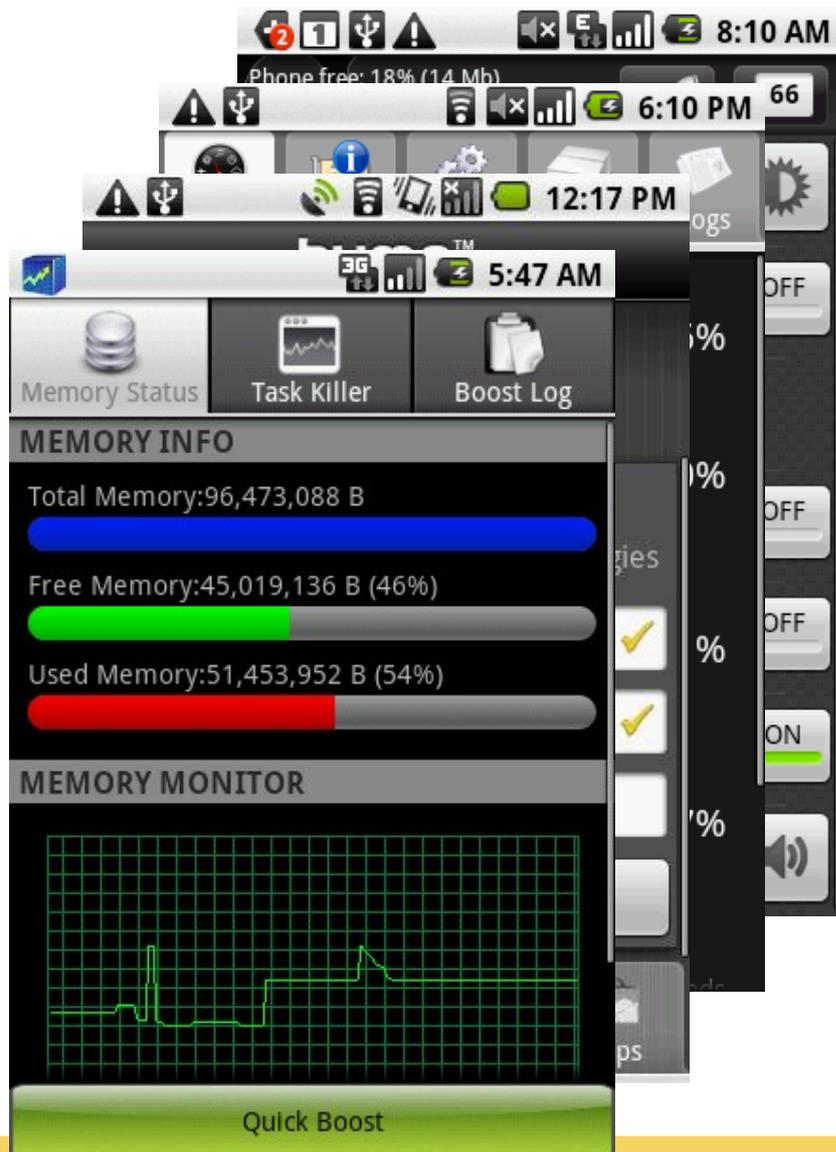
# Activity stack

- L'utente passa da un'activity all'altra nel corso del suo task
  - *Avanti*: usando l'applicazione, si passa di schermata in schermata secondo il flusso del lavoro
  - *Indietro*: usando il tasto Back
  - *Richiamo*: dalla lista delle “applicazioni recenti”, da notifiche
  - *Interruzioni*: da eventi esterni (es.: telefonata in arrivo)
  - *Sospensioni*: usando il tasto Home

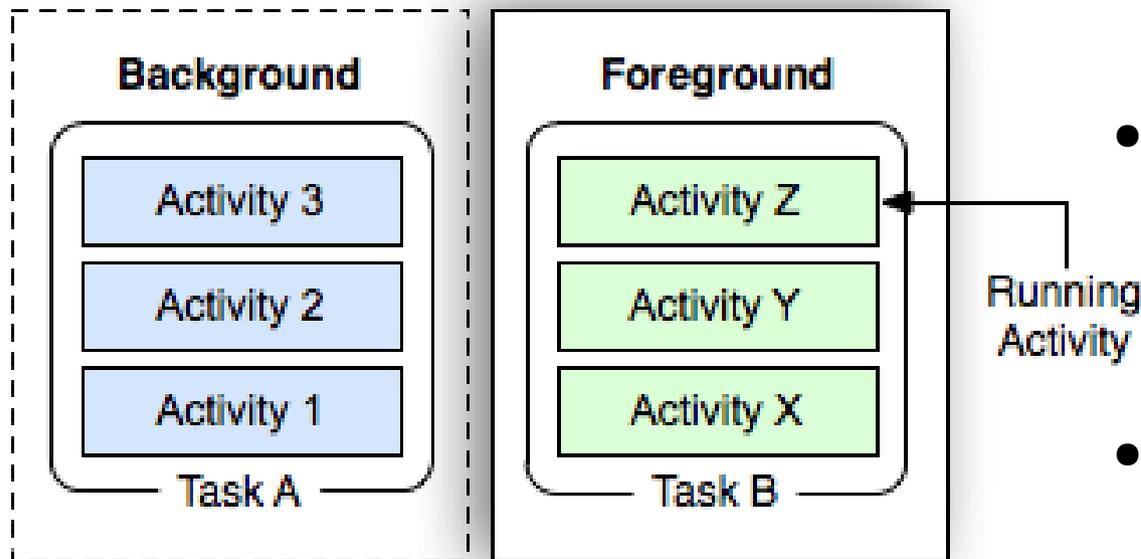
# Activity stack



- Si viene quindi a creare uno **stack** di Activity
  - Analogo alla history nei browser
- L'utente vede solo l'activity in cima allo stack
  - Interagisce **solo** con la top
  - Può vedere **anche** le altre, se la top include delle parti trasparenti



# Stack multipli

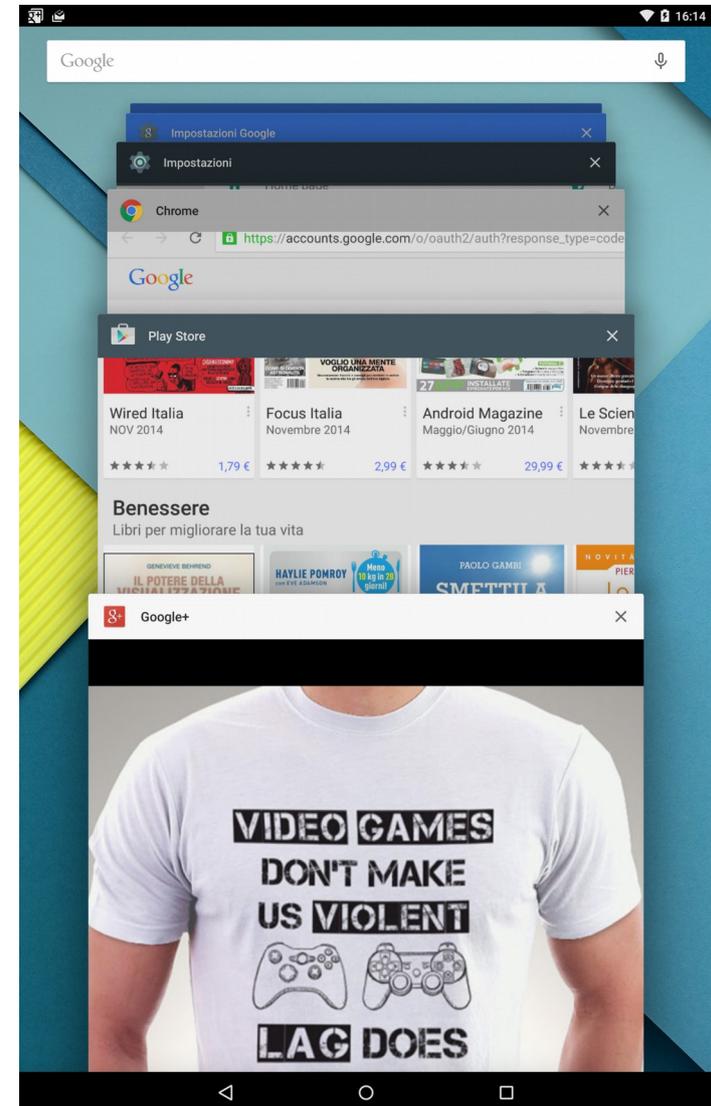


- Quando l'utente preme Home, “sospende” il task corrente
- L'intero stack va in background, e viene iniziato un nuovo stack
- Quando l'utente ritorna a un'activity del vecchio stack, il suo task torna in foreground

# Stack multipli



- Quando si attiva il task switcher di sistema, si vedono *gli stack*
  - Ciascuno rappresentato dall'activity in cima allo stack
- L'utente può selezionare uno stack, che diventa quello corrente
  - *Back* opera sulla history dello stack corrente



# Stack multipli

- Android 5.0 ha aggiunto la possibilità per una app di integrarsi più strettamente con il task switcher
  - Che infatti ora si chiama “Overview”, più generico
- Più istanze della stessa activity possono essere presentate come task distinti (“documenti” distinti)
  - Tab di Google Chrome, documenti di Google Drive, ...
- Una app può inserire elementi arbitrari nella lista
- Una app può chiedere di tenere un elemento nella lista anche dopo che l'app è terminata

# Stack multipli

- Android 5.0 ha aggiunto la possibilità per una app di integ
  - Che i
  - Più ista
  - Tab d
  - Una ap
  - Una ap
  - lista an
- Android *Nougat* aggiunge alcune ulteriori funzionalità:
- Split screen → più di un'activity può essere “top”
  - Rapid switch → scambiare le due activity usate più recentemente
- Android *Oreo* non è da meno:
- Picture-in-picture → più di un'activity può essere “top”



# Stack multipli

## Istanze multiple



- Quando si avvia un'activity possono accadere varie cose
- Per default:
  - Se non esisteva un task contenente l'activity, viene creato un nuovo task che ha una nuova istanza dell'activity come unico elemento
  - Se viene riavviata un'activity che era top del suo stack, si torna a quella particolare istanza
  - Se viene riavviata un'activity che **non** era top del suo stack, viene lanciata una nuova istanza e messa in cima allo stack
- Questo è il comportamento che gli utenti *imparano* a considerare naturale
- Per default... sarebbe meglio non alterarlo senza buoni motivi!

# Stack multipli

## Istanze multiple



- Il comportamento in fase di riavvio può essere controllato dal programmatore
  - Dall'activity lanciata, tramite flag nel suo `<activity>`
  - Dall'activity che lancia, tramite flag nel suo Intent

launchMode	Effetto
Standard	Crea sempre una nuova istanza dell'activity nel task di destinazione
SingleTop	Se l'activity è top, riavvia l'istanza esistente; altrimenti crea una nuova istanza (che diventa top dello stack)
SingleTask	Può esistere una sola istanza dell'activity. Se ce n'è già una in qualche task, viene riavviata quella. Altrimenti, si crea un nuovo task, che ha l'activity come unico elemento.
SingleInstance	Come sopra, ma se si crea un nuovo task, non consente di creare ulteriori activity al suo interno.



# Stack multipli

## Istanze multiple



- Tramite i vari `FLAG_ACTIVITY_*` dell'Intent che lancia un'activity, è anche possibile essere più specifici
  - Lanciare senza animazione (transizione) fra schermi
  - Lanciare senza che l'activity compaia nella history
  - Decidere se l'activity riavviata deve essere spostata in cima allo stack a cui appartiene, o lasciata al suo posto originale
  - ecc.



# Flag di lancio più comuni



- **FLAG\_ACTIVITY\_NEW\_TASK**
  - L'activity viene lanciata in un nuovo stack, di cui è l'unico membro
  - Flag usato tipicamente dai launcher (Home screen)
- **FLAG\_ACTIVITY\_CLEAR\_TOP**
  - Se l'activity esiste nello stack corrente, tutte quelle sopra di essa vengono chiuse, e l'activity diventa top
- **FLAG\_ACTIVITY\_SINGLE\_TOP**
  - Se l'activity è già quella top nello stack corrente, non viene lanciata



# Lanciare un'activity



- Abbiamo visto che la “prima” Activity di un'app è indicata da AndroidManifest.xml
  - Non esiste realmente **un** main; piuttosto, tutte le `<activity>` che dichiarano di accettare un Intent `MAIN/LAUNCHER` sono potenziali main
  - Tipicamente, i launcher aggiungono una icona per ognuna di queste activity nei loro menu, elenchi, ecc.
  - Al **primo** avvio di una Activity corrisponde una chiamata a `onCreate()` - quello è il nostro punto di partenza



# Lanciare un'activity



- Per lanciare una specifica activity (nostra), se ne indica la classe: explicit intent

```
Intent intent = new Intent(this, miaActivity.class);  
startActivity(intent);
```

- Per lanciare una activity generica (nostra o no), se ne indica l'azione: implicit intent

```
Intent intent = new Intent(Intent.ACTION_...);  
startActivity(intent);
```

# Lanciare un'activity

- Per lanciare una specifica activity (nostra), se ne indica la classe: explicit intent

```
Intent intent = new Intent(this, miaActivity.class);  
startActivity(intent);
```

## Context

Ricordate dalla lezione sulle risorse? Activity è una sottoclasse di Context...

## Classe da lanciare

Deve essere un componente in grado di ricevere l'Intent

- Per lanciarne una generica intent se ne indica l'azione

```
Intent intent = new Intent(Intent.ACTION_...);  
startActivity(intent);
```

# Lanciare un'activity

- Per lanciare una specifica activity (nostra), se ne indica la class

```
Intent intent =  
startActivity();
```

Questo è un costruttore che crea un Intent quasi vuoto.  
Di solito, seguono chiamate a  
`intent.set...()`  
per impostare tutti gli altri campi, prima dello  
`startActivity()`.

- Per lanciare una activity generica (nostra o no), se ne indica l'azione: implicit intent

```
Intent intent = new Intent(Intent.ACTION_...);  
startActivity(intent);
```



# Lanciare un'activity e ottenere un risultato



- Alcune activity hanno senso solo se possono inviare un risultato di qualche tipo a chi le ha invocate
  - Per esempio: l'activity di sistema per scegliere un contatto dalla rubrica

```
Intent intent = new Intent(...);  
startActivityForResult(intent, codice_richiesta);
```

- Al termine dell'activity lanciata, viene invocato il metodo `onActivityResult()` del chiamante, con argomenti che incapsulano la risposta



# Activity, Fragment e back



- Normalmente, un'activity ha un layout (contenuti della schermata) impostato con `setLayout()`
- Il contenuto della schermata può cambiare dinamicamente
  - Alterando il layout
  - Manipolando i **Fragment** che compongono l'activity
    - Vedremo i dettagli più avanti
- Nel secondo caso, ogni **transazione** relativa ai Fragment viene inserita nel back stack
  - Con Back, l'utente fa l'undo della transazione e ritorna all'aspetto precedente dell'Activity



# Uccidere un'activity



- Un'activity può *suicidarsi* chiamando il proprio metodo **finish()**
  - È praticamente un return al chiamante; invocando **setResult()** prima di **finish()** si stabilisce il valore di ritorno
- È possibile *uccidere* un'altra activity (lanciata con **startActivityForResult(intent, codr)**) chiamando il proprio metodo **finishActivity(codr)**
- **Il sistema può uccidere activity per liberare risorse**



# La gestione delle risorse (activity)



- In particolare, Android distingue 3 classi di priorità
- Priorità critica – non vengono mai uccisi
  - Activity in cima allo stack corrente (in uso!)
- Priorità alta – uccisi alla disperata
  - Activity visibili tramite le trasparenze di quella top
- Priorità bassa – uccisi in ordine LRU se serve
  - Activity non visibili, in fondo allo stack corrente
  - Activity non visibili, in altri stack

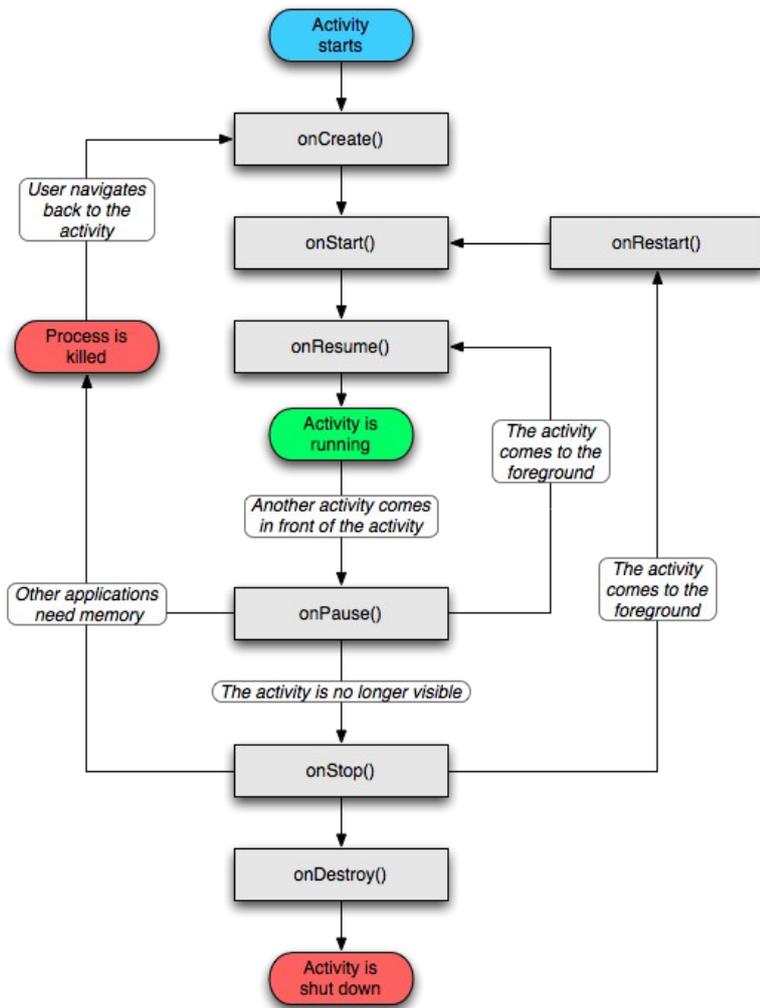


# La gestione delle risorse (activity)



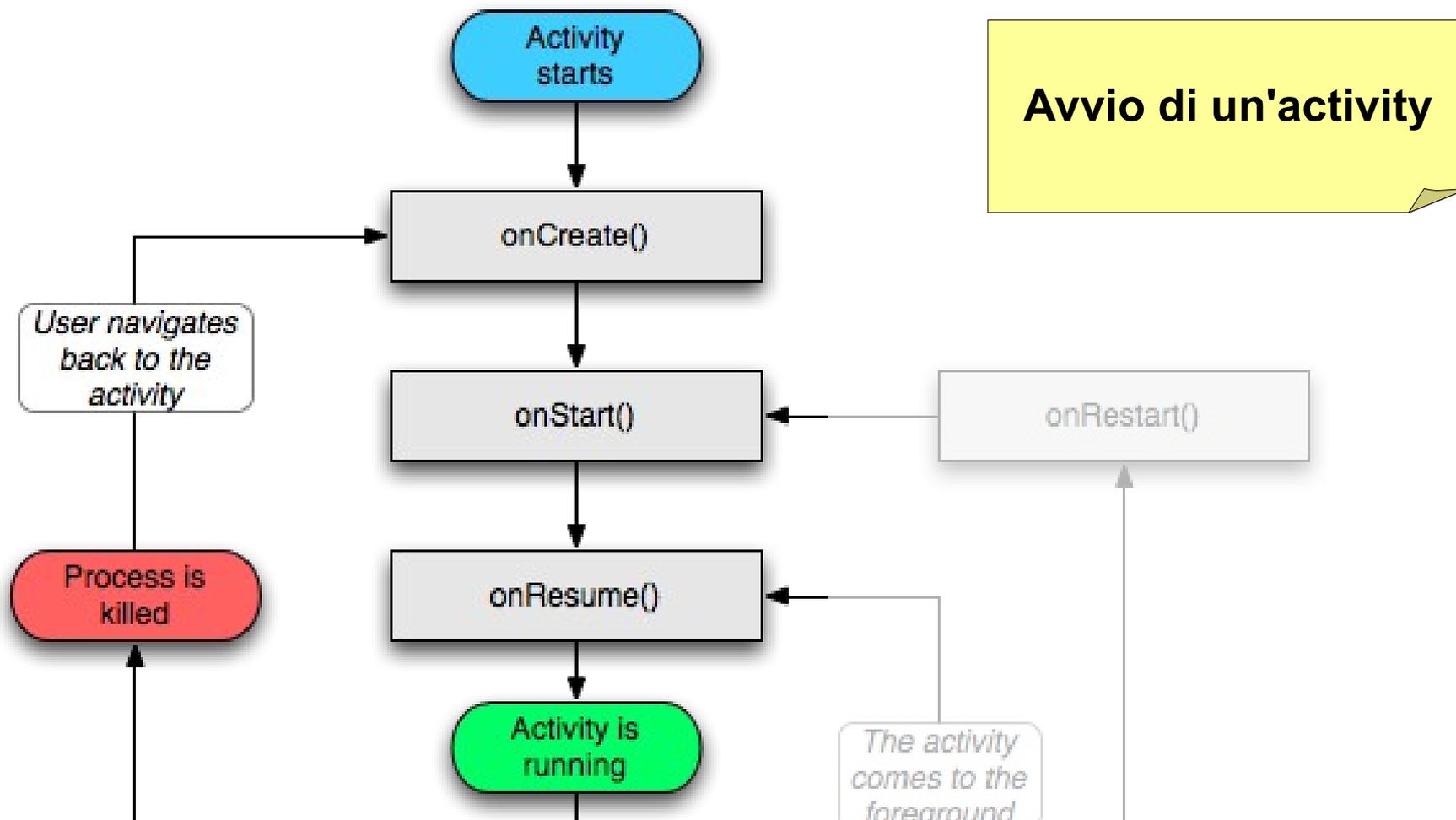
- Android gestisce dispositivi con memoria limitata
  - Niente memoria virtuale
- **È del tutto comune** che il processo che esegue la vostra activity venga ucciso
  - Per consentire all'utente di continuare nel suo task
- Bisogna quindi essere preparati a **salvare lo stato e ripristinarlo** se necessario
- Android vi aiuta chiamando dei metodi *callback* al momento opportuno

# Il ciclo di vita di un'Activity

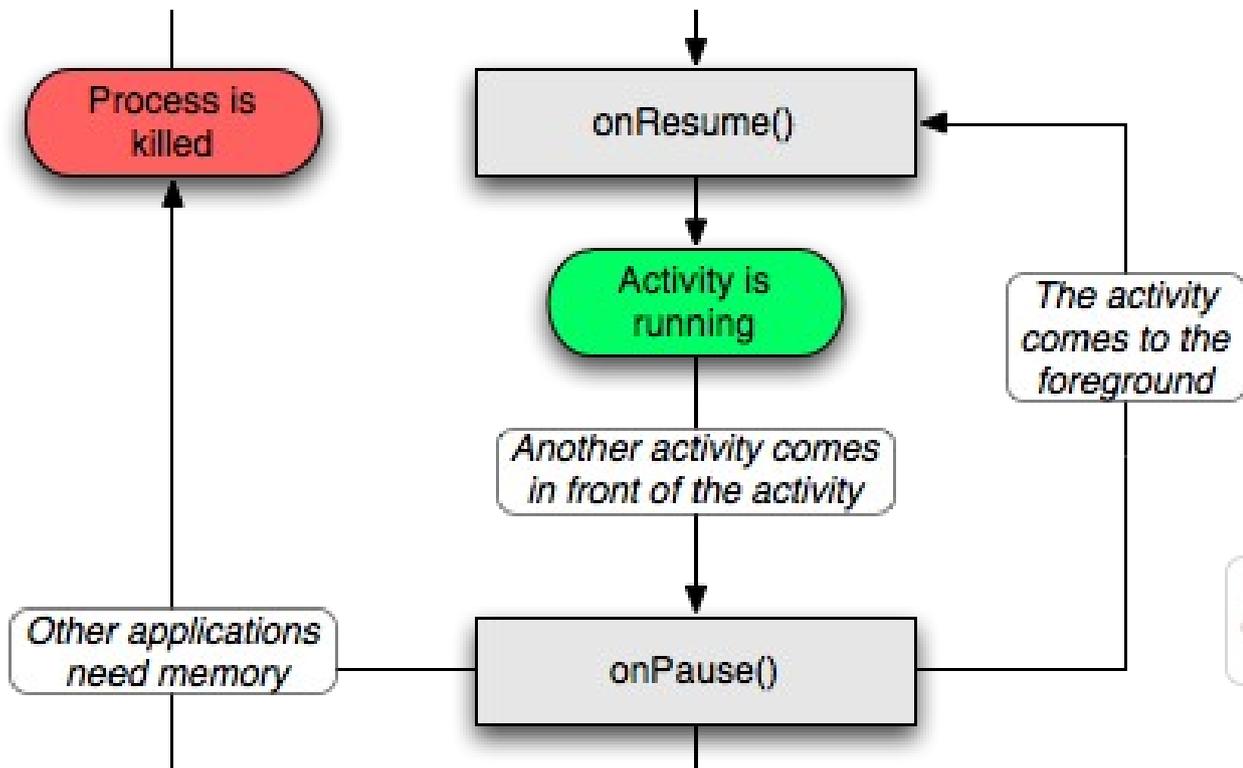


- Questo automa a stati è **fondamentale** per una corretta implementazione di un'Activity
- È una delle figure classiche, la ritroverete su qualunque testo/sito
- La vedremo a parti...

# Il ciclo di vita di un'Activity



# Il ciclo di vita di un'Activity

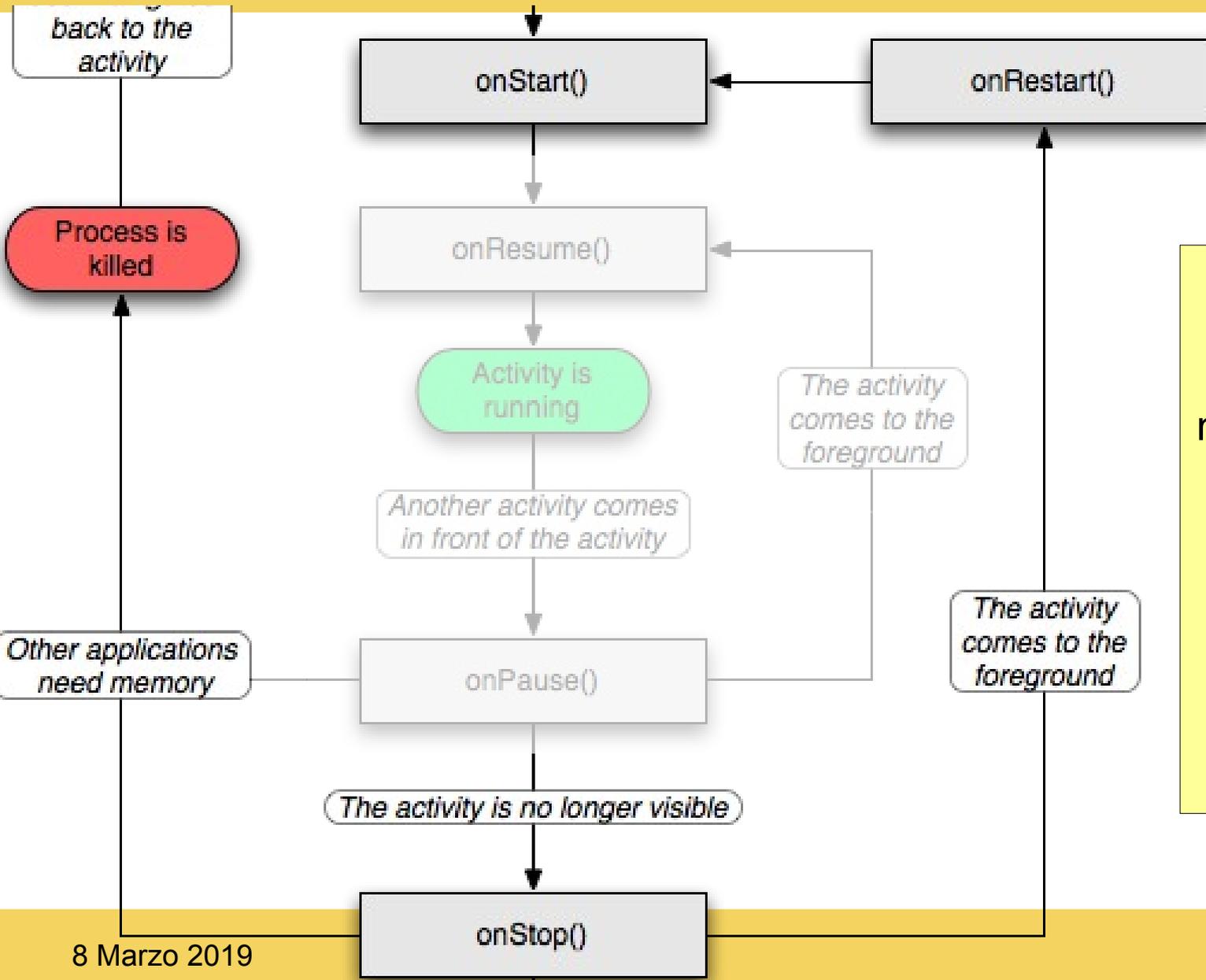


## La vita adulta

L'activity cicla fra essere quella in cima allo stack (con cui l'utente interagisce) ed essere **in pausa** mentre altre activity occupano la cima dello stack.

Solo durante una pausa si può essere uccisi (se proprio necessario).

# Il ciclo di vita di un'Activity



**La vita adulta**

Se l'activity non solo non è in cima, ma non è neanche visibile, passa nello stato di **stop** (l'uccisione in questo caso diventa un po' più probabile).

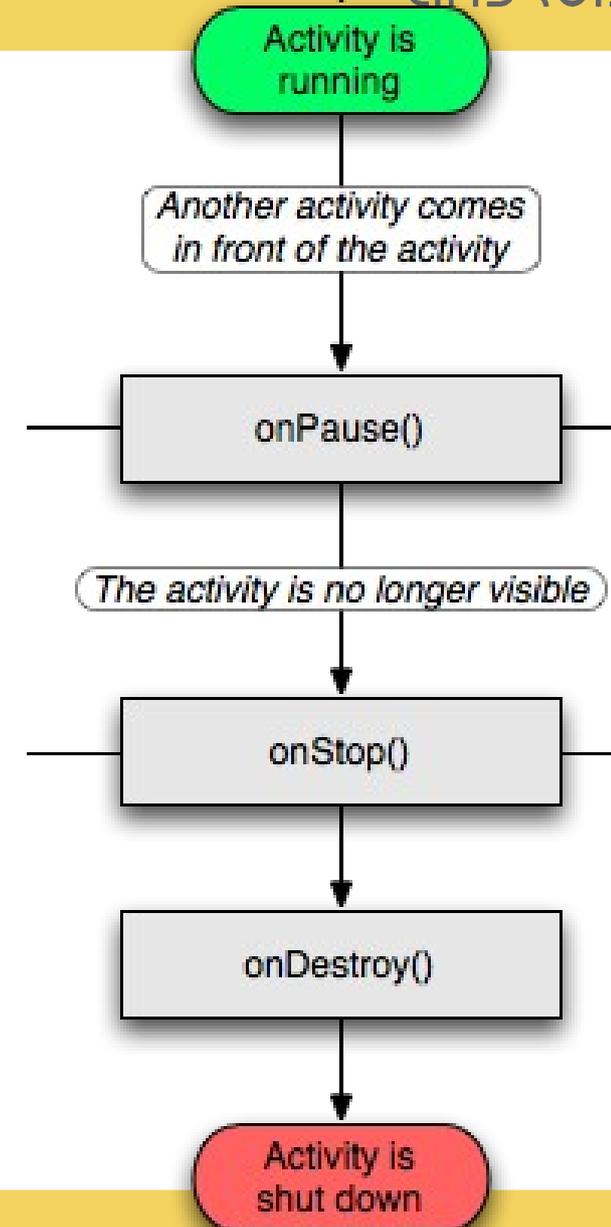
# Il ciclo di vita di un'Activity



## Il momento che arriva per tutti

Una activity in stato di stop può finalmente essere distrutta dal sistema. La memoria viene disallocata, il processo viene ucciso (oppure messo in un pool di processi vuoti, pronti per essere riutilizzati per un'altra activity, secondo il classico approccio dei thread pool).

Prima di uccidere l'activity, il sistema chiamerà `onDestroy()`: questa è l'ultima possibilità di salvare lo stato dell'activity in maniera permanente!





# Il ciclo di vita di un'Activity



- **onCreate()** - inizializziamo lo stato
- **onStart()** - stiamo per essere resi visibili, UI pronta!
- **onResume()** - stiamo per diventare top dello stack, da ora in poi riceviamo input dall'utente
- **onPause()** - un'altra activity sta per diventare top. Salviamo lo stato dell'UI
- **onStop()** - non siamo più visibili, non dobbiamo preoccuparci di tenere la UI aggiornata
- **onDestroy()** - o abbiamo finito (con finish()) o stanno per distruggerci. Si salvi chi può (UI non importa, dati si)

# Il ciclo di vita di due Activity



- L'ordine con cui vengono invocati i *lifecycle methods* di diverse activity è ben definito
  - Ma è prudente non farci troppo affidamento!
- Esempio: switch da **A** a **B**
  - **A.onPause()**
  - **B.onCreate()**
  - **B.onStart()**
  - **B.onResume()**
  - **A.onStop()**

Se A vuole salvare qualcosa che B deve leggere in `B.onCreate()`, deve farlo in `A.onPause()` – non in `A.onStop()`.

Solo se la UI di B non ha parti trasparenti attraverso cui si vede la UI di A

# Il ciclo di vita di un'Activity



- **IMPORTANTE:** La vostra re-implementazione di questi metodi **deve** chiamare il metodo corrispondente della superclasse:

```
@Override
```

```
protected void onStart()
```

```
{
```

```
    super.onStart();
```

```
    // codice vostro
```

```
}
```

- In questo modo, il framework può continuare a fare il suo processing di default

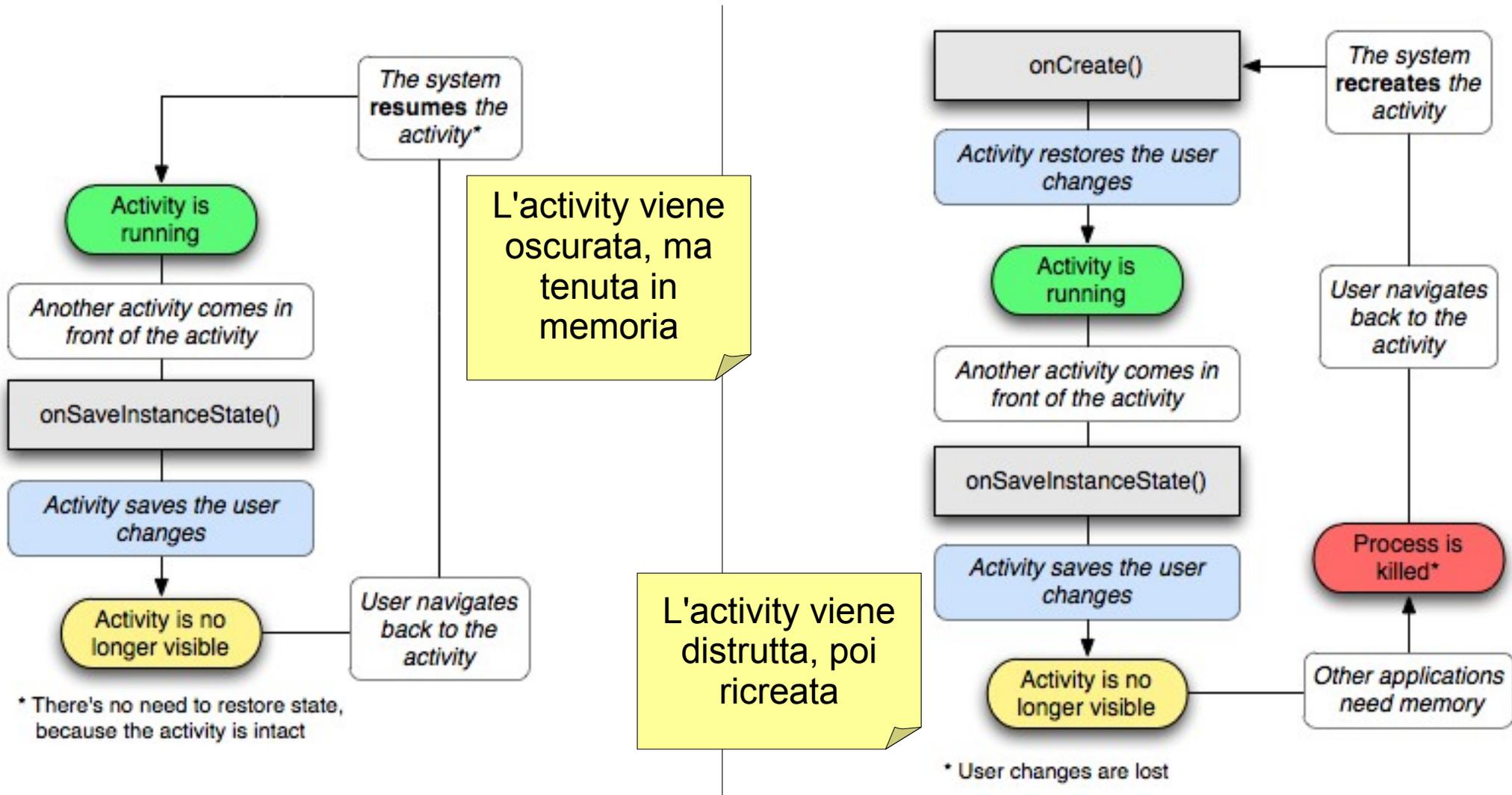


# Salviamo lo Stato!



- Il callback **onSaveInstanceState()** viene chiamato dal sistema quando è necessario salvare una parte dello stato dell'activity per il ripristino a breve
  - Prima di **onStop()** e di **onPause()**
  - Ma può anche non essere chiamato se l'utente abbandona l'activity con back!
    - In questo caso, l'utente sta uscendo; per il ripristino a lunga si usa **onDestroy()**
    - Oppure, se non è un'operazione costosa, anche **onStop()** o **onPause()** stessi

# Salviamo lo Stato!



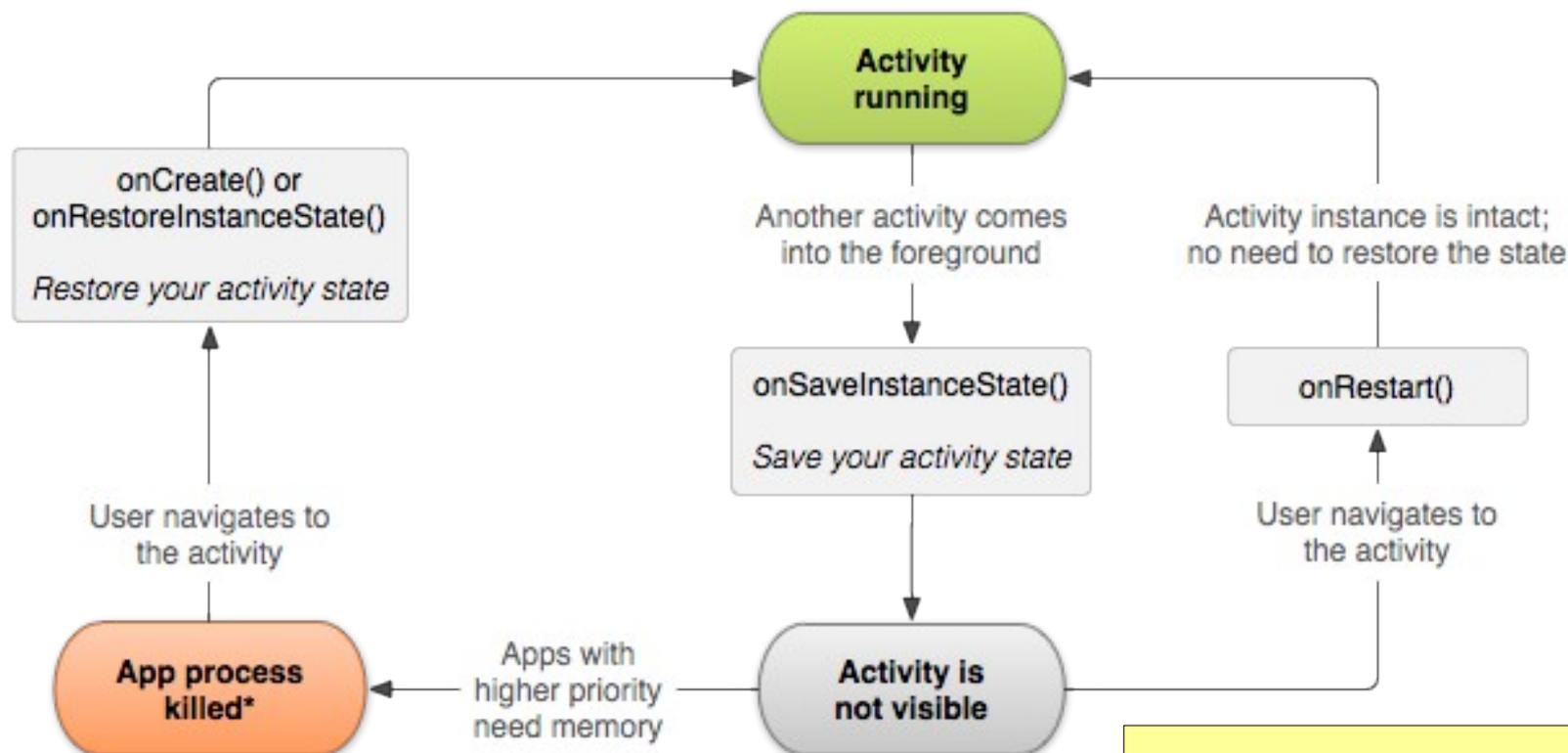


# Salviamo lo Stato!



- A `onSaveInstanceState()` viene passato un **Bundle**
- Lo abbiamo già visto: coppie chiave-valore
- Il metodo dovrebbe salvare dentro il Bundle tutto quello che può servire
  - Metodi `putString(chiave, stringa)`, `putInt(chiave, intero)`, ecc. - c'è anche `putBundle(chiave, bundle)`!
- L'activity può essere distrutta, ma il bundle sopravvive!
  - Praticamente, è l'anima dell'activity :-)
- Il Bundle sarà poi passato a `onCreate()`
  - E anche a `onRestoreInstanceState()`, chiamata dopo `onStart()`

# Salviamo lo Stato!



\*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

Non sempre lo stato salvato nel Bundle viene ripristinato: a volte semplicemente non è necessario.

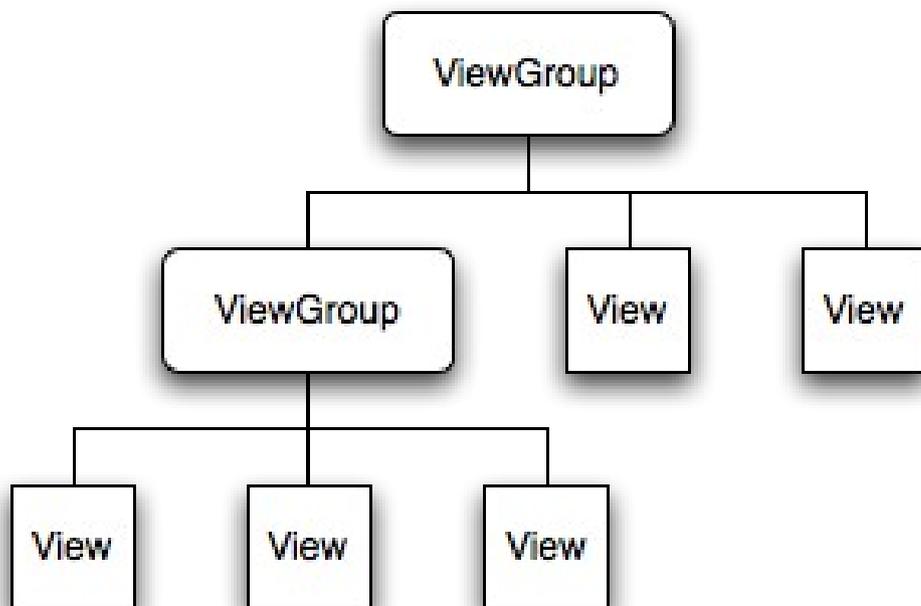


# Implementazione di default



- La classe Activity ha una implementazione di default del salvataggio/ripristino dello stato
- Scorre il suo **Layout**, e salva nel bundle lo stato di tutte le **View** (componenti di UI) chiamano il loro `onSaveInstanceState()`, in ordine
  - Solo per le View che hanno un campo **id**, però!
- Se la nostra activity ha solo UI, basta così
- Conviene praticamente **sempre** chiamare `super.onSaveInstanceState()` anche se si devono salvare ulteriori pezzi di stato (non-UI)

# Layout & View



- Una UI Android è un albero con foglie di classe `View` e nodi intermedi di classe `ViewGroup`
  - Come già visto, tipicamente definito in XML
- Ogni `View` è una classe Java con nome uguale al tag XML relativo



# Layout & View

- A run-time, esiste un albero di oggetti Java che creato a partire dall'albero XML del layout
- Gli oggetti possono ricevere input dall'utente (si interfacciano col sistema touch)
- Quando si verifica un **evento** significativo, viene chiamato un *handler*
  - La vostra Activity può registrare propri handler
  - In Java, sono *inner interfaces* dentro la classe View
  - Ogni interfaccia definisce un metodo **on...Listener()**



# Esempio di Listener



```
private OnClickListener listener = new OnClickListener()
{
    public void onClick(View v) {
        // reazione: per esempio, lanciamo una Activity
    }
};

protected void onCreate(Bundle stato) {
    ...
    // prendi un riferimento al pulsante di nome "b"
    Button b = (Button)findViewById(R.id.b);
    // registra il listener per il click di b
    b.setOnClickListener(listener);
    ...
}
```

# Esempio di Listener

```
private OnClickListener listener = new OnClickListener()  
{  
    public void onClick(View v) {  
        // reazione: per esempio, lanciata  
    }  
};  
  
protected void onCreate(Bundle savedInstanceState)  
{  
    ...  
    // prendi un riferimento al pulsante di nome "b"  
    Button b = (Button) findViewById(R.id.b);  
    // registra il listener per il click di b  
    b.setOnClickListener(listener);  
    ...  
}
```

Non s'era detto  
di evitare la  
new?

# Esempio di Listener



```
public class act extends Activity
    implements OnClickListener {

    protected void onCreate(Bundle stato) {
        ...
        Button b = (Button) findViewById(R.id.b);
        b.setOnClickListener(this);
    }

    public void onClick(View v) {
        // reazione: per esempio, lanciamo una Activity
    }
    ...
}
```



# Un esempio completo

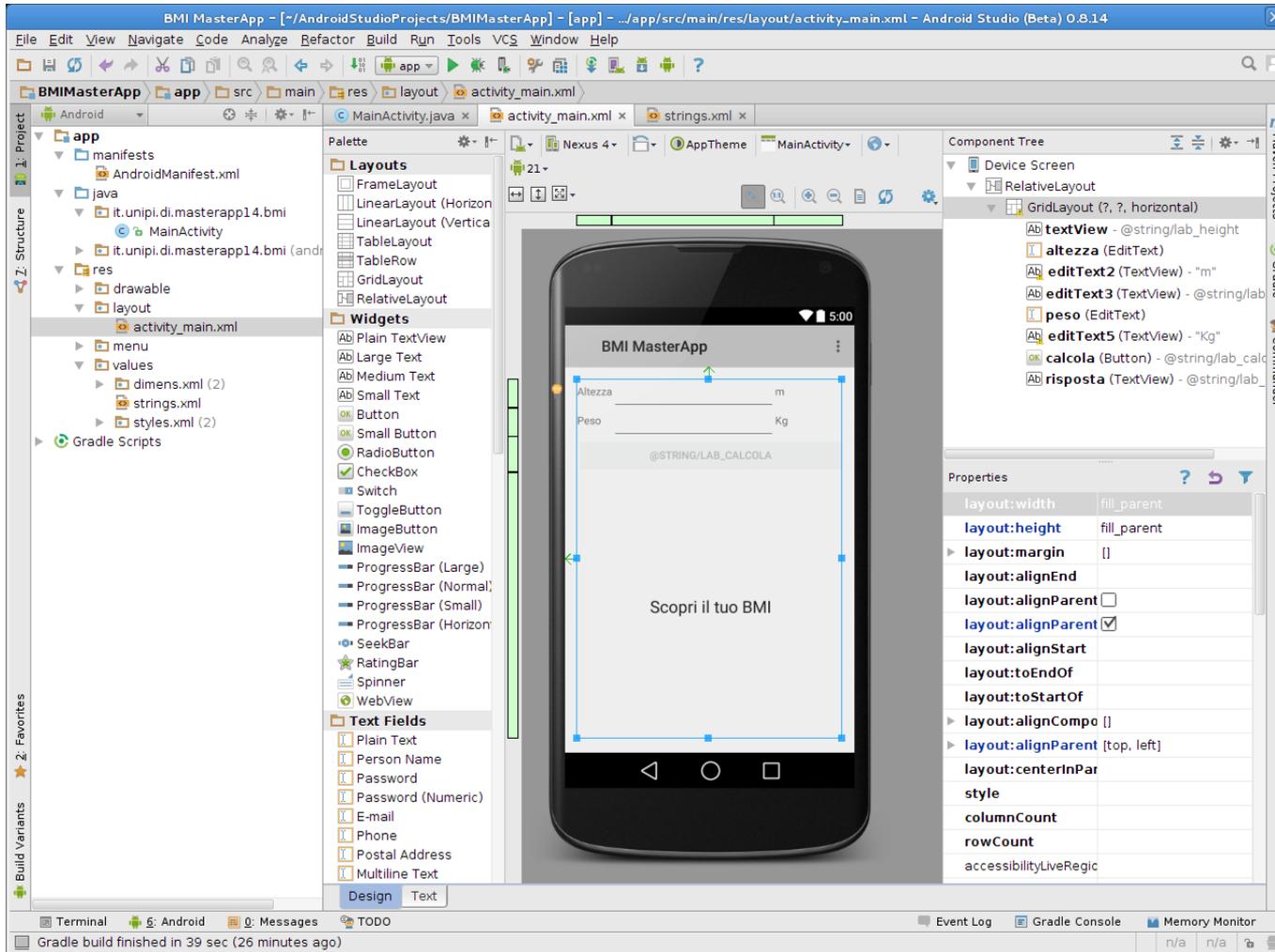


# BMI Calc



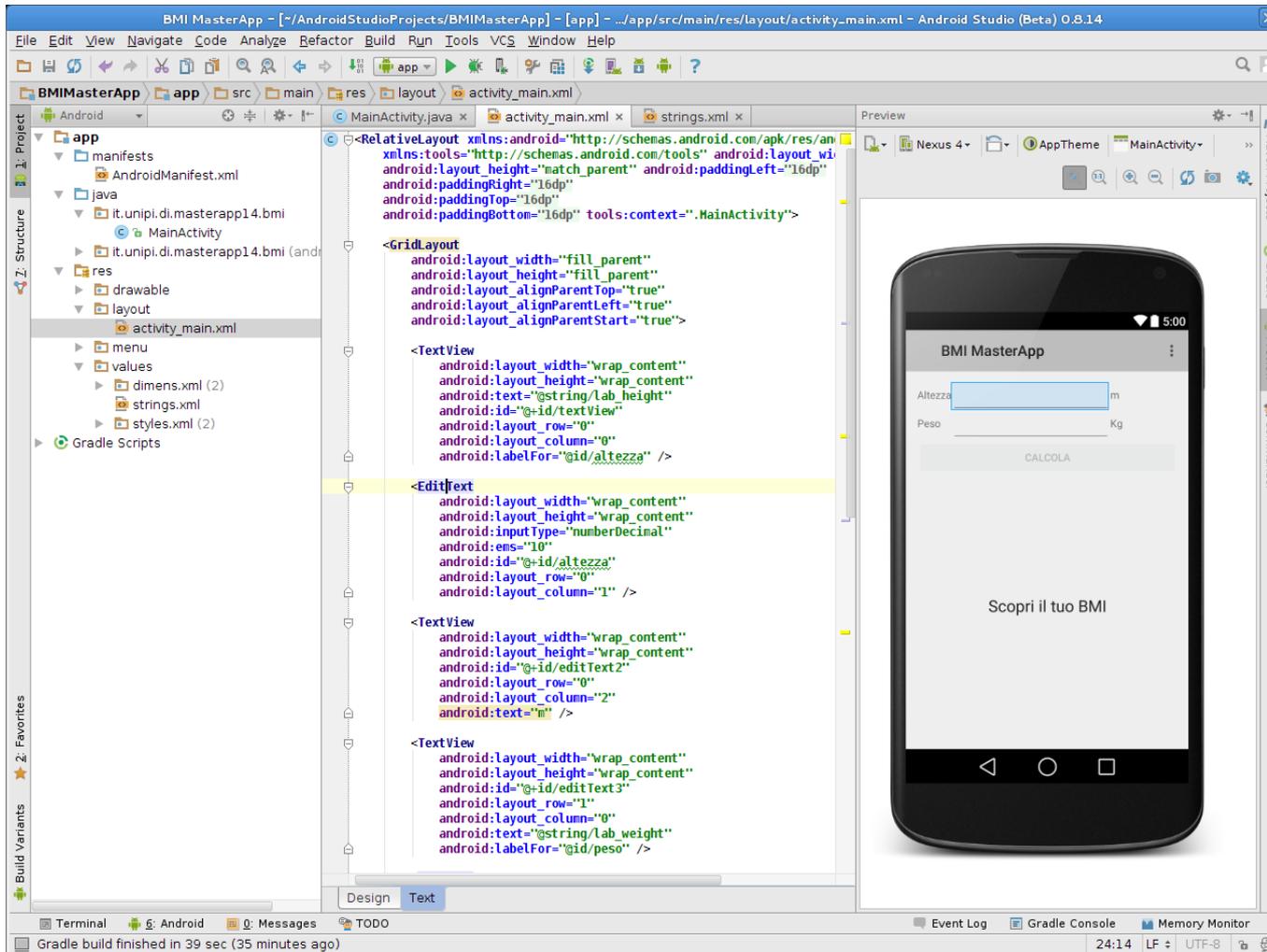
- Facciamo un esempio completo
  - La più banale cosa che viene in mente: BMI calc
  - Due campi di testo, in cui l'utente immette altezza e peso
  - Un pulsante calcola, che fa il semplice conto del Body Mass Index
  - Un'area di testo, in cui mostrare il risultato
  - (c'è chi c'ha fatto i soldi...)

# BMI calc – layout



- Ha senso usare l'editor grafico per creare il layout

# BMI calc – layout



- Ha senso usare l'editor grafico per creare il layout
- Però lo illustriamo nella forma XML

# res/layout/activity\_main.xml



## <RelativeLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:paddingLeft="@dimen/activity_horizontal_margin"  
android:paddingRight="@dimen/activity_horizontal_margin"  
android:paddingTop="@dimen/activity_vertical_margin"  
android:paddingBottom="@dimen/activity_vertical_margin"  
tools:context=".MainActivity" >
```

**xmlns**: definisce i namespace XML utilizzati in questo file.

**android**: dati del framework di Android

**tools**: dati dei tool (nel nostro caso, Android Studio)

## <GridLayout

```
android:layout_width="fill_parent"  
android:layout_height="fill_parent"  
android:layout_alignParentTop="true"  
android:layout_alignParentLeft="true"  
android:layout_alignParentStart="true" >
```

Ricordate? È il formato per riferire le risorse all'interno di un file XML:

@tipo/nome

*... contenuti veri ...*

</GridLayout>

</RelativeLayout>

# res/layout/activity\_main.xml



## <TextView

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="@string/lab_height"  
android:id="@+id/textView"  
android:layout_row="0" android:layout_column="0"  
android:labelFor="@id/altezza" />
```

Etichetta per "Altezza".  
Notate che il testo della  
stringa è esternalizzato  
come risorsa.

## <EditText

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:inputType="numberDecimal"  
android:ems="10"  
android:id="@+id/altezza"  
android:layout_row="0" android:layout_column="1" />
```

Campo di testo editabile, di  
tipo numerico, per inserire il  
valore dell'altezza (in metri).

## <TextView

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:id="@+id/editText2"  
android:layout_row="0" android:layout_column="2"  
android:text="m" />
```

Etichetta per l'unità di  
misura. Questa volta text  
**non** è una risorsa: "m" =  
"metro", e la nostra formula  
richiede l'input in metri.

# res/layout/activity\_main.xml



La seconda riga del layout, per il peso, è del tutto analoga e non la mostriamo.

## <Button

```
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:text="@string/lab_calcola"  
android:id="@+id/calcola"  
android:layout_row="2" android:layout_column="0" android:layout_columnSpan="3"  
android:enabled="false" />
```

Pulsante "Calcola" – inizialmente disabilitato

## <TextView

```
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:textAppearance="?android:attr/textAppearanceLarge"  
android:text="@string/lab_initresult"  
android:id="@+id/risposta"  
android:layout_row="3" android:layout_column="0" android:layout_columnSpan="3"  
android:layout_gravity="fill"  
android:gravity="center_vertical|center_horizontal" />
```

Questo è un riferimento a un *attributo* di uno *stile* di sistema

?[package:][tipo]nome

Campo di testo per la risposta. Occupa tutto lo spazio rimanente, con il testo centrato.

# res/values/[...].xml



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
<string name="app_name">BMI MasterApp</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="lab_height">Altezza</string>
<string name="lab_weight">Peso</string>
<string name="lab_initresult">Scopri il tuo BMI</string>
<string name="lab_calcola">Calcola</string>
```

res/values/strings.xml

(ovviamente, dovrebbe essere  
res/values-it/strings.xml)

```
</resources>
```

res/values/dimens.xml

```
<resources>
  <!-- Default screen margins, per the Android Design guidelines. -->
  <dimen name="activity_horizontal_margin">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```



# AndroidManifest.xml



- Una volta create le risorse (layout e valori) per la nostra App, ci rimangono da creare i componenti (codice)
- In un caso così semplice, basta un solo componente
  - La singola Activity che gestisce la schermata di input e output
- Definiamola subito nel manifesto:



# AndroidManifest.xml



```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="it.unipi.di.sam.bmi" >
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="it.unipi.di.sam.bmi.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```



# MainActivity.java



```
package it.unipi.di.sam.bmi;
import ...
public class MainActivity extends Activity implements OnClickListener {
    private Button calcola;
    private EditText altezza, peso;
    private TextView risposta;

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

        calcola=(Button)findViewById(R.id.calcola);
        altezza=(EditText)findViewById(R.id.altezza);
        peso=(EditText)findViewById(R.id.peso);
        risposta=(TextView)findViewById(R.id.risposta);

        calcola.setOnClickListener(this);
        calcola.setEnabled(true); // dovremmo farlo solo dopo l'input negli altri campi...
    }
}
```



# MainActivity.java



```
@Override
public void onClick(View v) {
    if (v==calcola) {
        try {
            double a = Double.parseDouble(altezza.getText().toString());
            double p = Double.parseDouble(peso.getText().toString());
            double bmi = p / (a * a);
            risposta.setText(String.format("%2.1f", bmi));
        } catch (Exception e) {
            risposta.setText("?");
        }
    }
}
```

Per testare:

- telefonino abilitato da sviluppatore
- telefonino in modalità Debug USB abilitata
- telefonino collegato via USB al PC
- Ctrl-Alt-F10 su Android Studio

# BMI calc – finale

