Lezioni 4 e 5

Programmazione Android

- Componenti di un'applicazione
- AndroidManifest.xml
- Le Activity
  - Definizione
  - Ciclo di vita
  - Layout & View
  - Interazione
- Laboratorio

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)

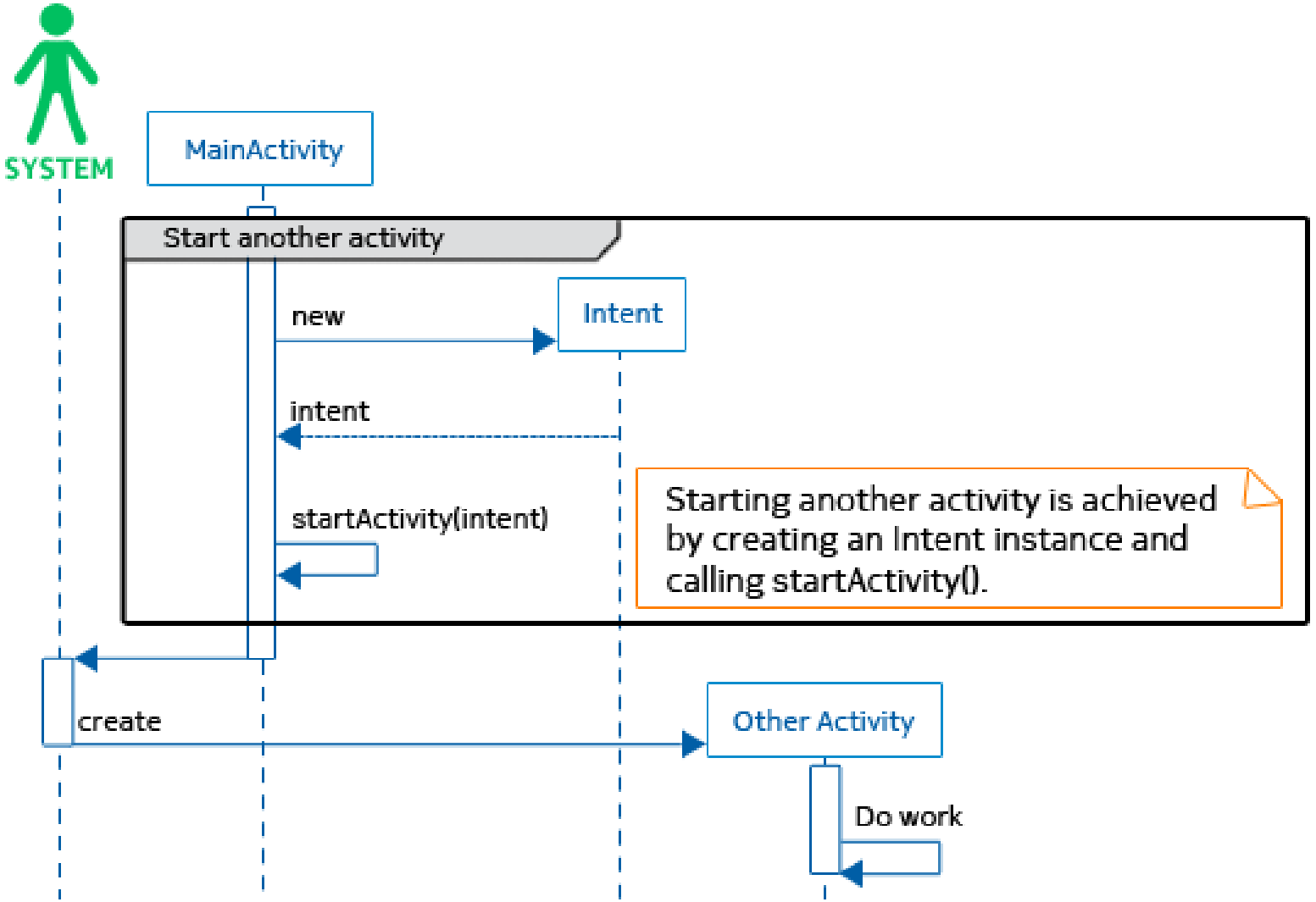
Componenti di un'applicazione

- I componenti di un'applicazione dialogano attraverso un **sistema di messagistica** 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 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



Da developer.nokia.com

L'avvio di una Activity

Explicit Intent

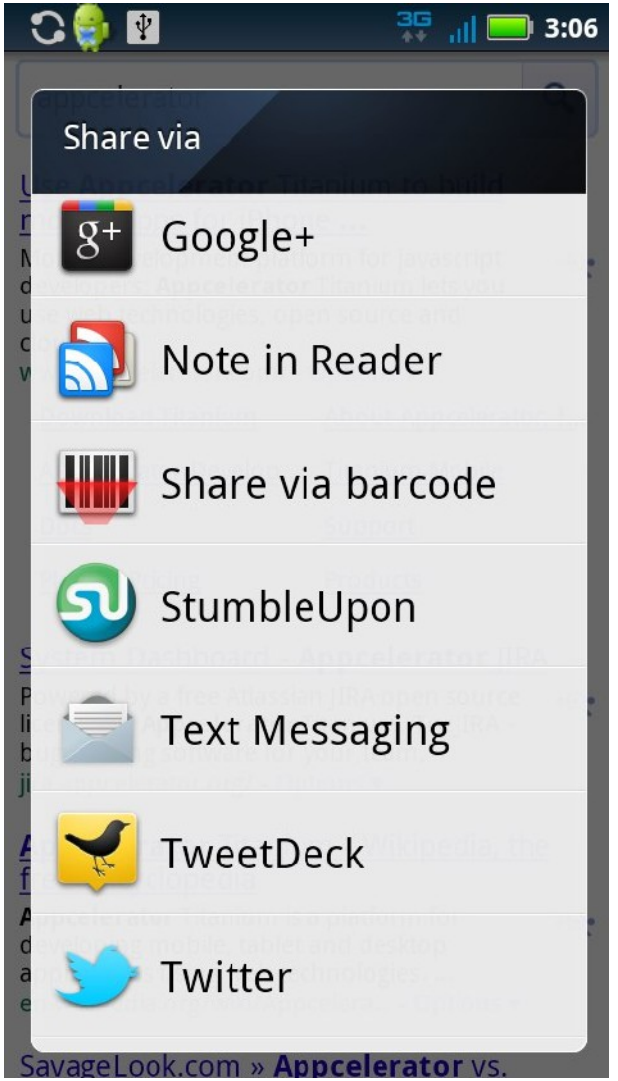
- Possiamo creare un Intent che chiede **una Activity**
  - Se l'Activity in questione non è in esecuzione, viene lanciata
  - Se è già in esecuzione, viene "svegliata"
  - Viene recapitato l'Intent e l'Activity destinataria diventa "attiva"

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
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_DISCONNEC
TED
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_MEDIA_SCANNER_SCAN_FILE
ACTION_MEDIA_SCANNER_STARTED
ACTION_MEDIA_SHARED
ACTION_MEDIA_UNMOUNTABLE
ACTION_MEDIA_UNMOUNTED
ACTION_MY_PACKAGE_REPLACED
ACTION_NEW_OUTGOING_CALL
ACTION_PACKAGE_ADDED
ACTION_PACKAGE_CHANGED
ACTION_PACKAGE_DATA_CLEARED
ACTION_PACKAGE_FIRST_LAUNCH
ACTION_PACKAGE_FULLY_REMOVED
ACTION_PACKAGE_INSTALL
ACTION_PACKAGE_NEEDS_VERIFICATI
ON
ACTION_PACKAGE_REMOVED
ACTION_PACKAGE_REPLACED
ACTION_PACKAGE_RESTARTED
ACTION_PASTE
ACTION_PICK
ACTION_PICK_ACTIVITY
ACTION_POWER_CONNECTED
ACTION_POWER_DISCONNECTED
ACTION_POWER_USAGE_SUMMARY
ACTION_PROVIDER_CHANGED
ACTION_REBOOT
ACTION_RUN
ACTION_SCREEN_OFF
ACTION_SCREEN_ON
ACTION_SEARCH
ACTION_SEARCH_LONG_PRESS
ACTION_SEND
ACTION_SEND_MULTIPLE
ACTION_SENDTO
ACTION_SET_WALLPAPER
ACTION_SHUTDOWN
ACTION_SYNC
ACTION_SYSTEM_TUTORIAL
ACTION_TIME_CHANGED
ACTION_TIME_TICK
ACTION_TIMEZONE_CHANGED
ACTION_UID_REMOVED
ACTION_UMS_CONNECTED
ACTION_UMS_DISCONNECTED
ACTION_UNINSTALL_PACKAGE
ACTION_USER_PRESENT
ACTION_VIEW
ACTION_VOICE_COMMAND
ACTION_WALLPAPER_CHANGED
ACTION_WEB_SEARCH

Implicit Intent di sistema (Android 4.0)

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_APPLIC
NAVAILABLE
ACTION_FACTORY_TEST
ACTION_GET_CONTENT
ACTION_GTALK_SERVICE_C
D
ACTION_GTALK_SERVICE_D
TED
ACTION_HEADSET_PLUG
ACTION_INPUT_METHOD_CH
ACTION_INSERT
ACTION_INSERT_OR_EDIT
ACTION_INSTALL_PACKAGE
ACTION_LOCALE_CHANGED
ACTION_MAIN
ACTION_MANAGE_NETWOR
ACTION_MANAGE_PACKAGE
E
ACTION_MEDIA_BAD_REMO
ACTION_MEDIA_BUTTON
ACTION_MEDIA_CHECKING
ACTION_MEDIA_EJECT
ACTION_MEDIA_MOUNTED
ACTION_MEDIA_NOFS
ACTION_MEDIA_REMOVED
ACTION_MEDIA_SCANNER_

PRESS
R
AL
NGED
D
CTED
KAGE
D
ANGED

Le più frequenti

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

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 (opzionale)
- **Component** – componente a cui è indirizzato il messaggio (opzionale)
- **Extras** – un Bundle (mappa stringhe → valori) di ulteriori campi (opzionale)
- **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: un'altra App 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

AndroidManifest.xml

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.sam.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>
```

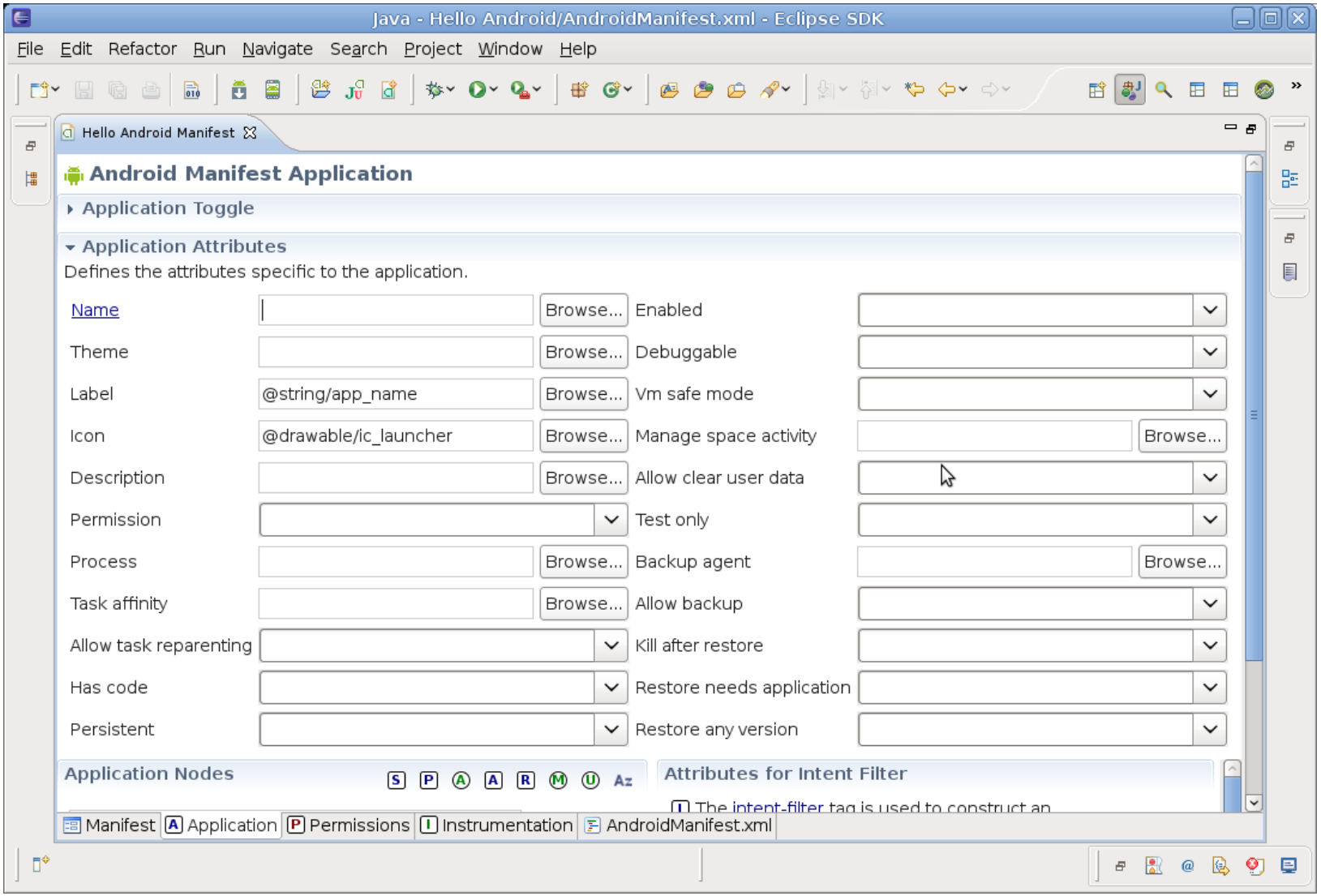
Java
Risorse
Letterali
XML

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

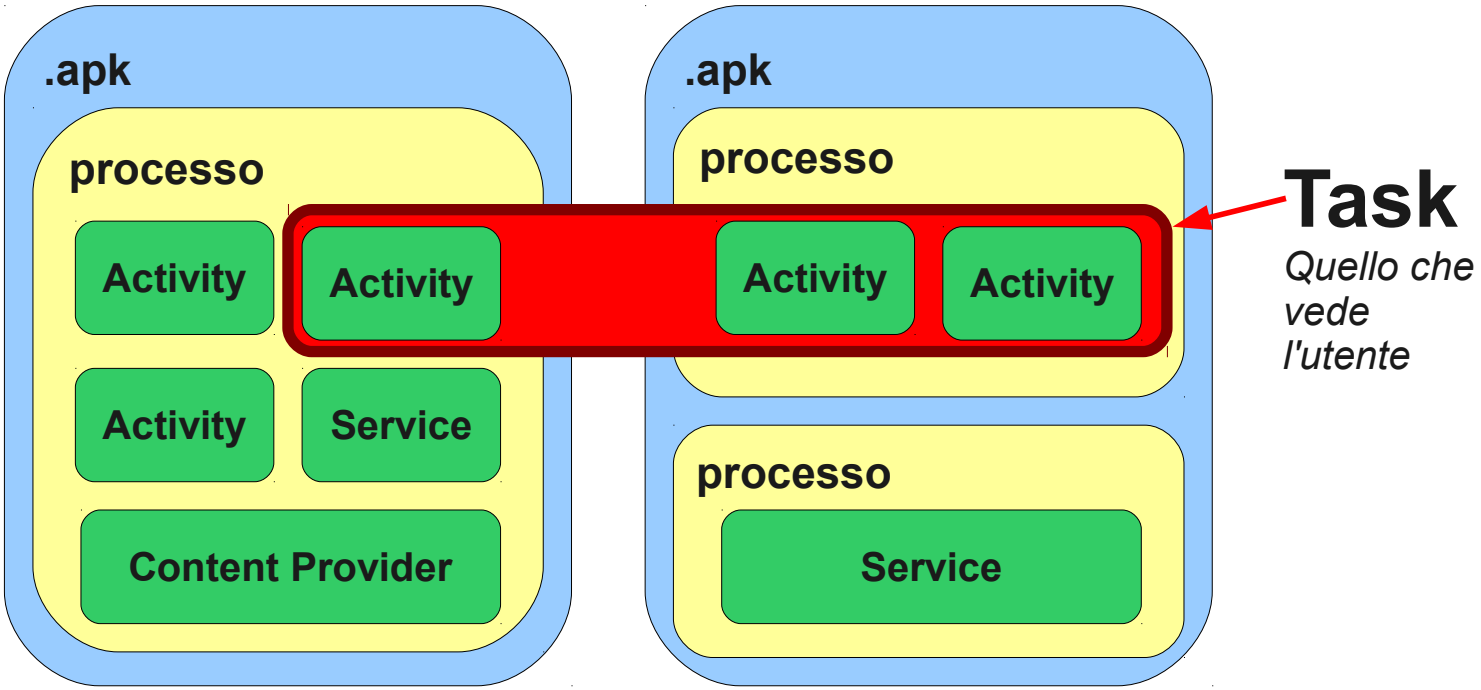
- Fortunatamente, Eclipse fornisce un editor strutturato dedicato ad AndroidManifest.xml
- Lo vedremo in un esempio *in vivo*
- Di norma, si compone il manifesto passo-passo, man mano che si scrivono i componenti



Activity

Activity

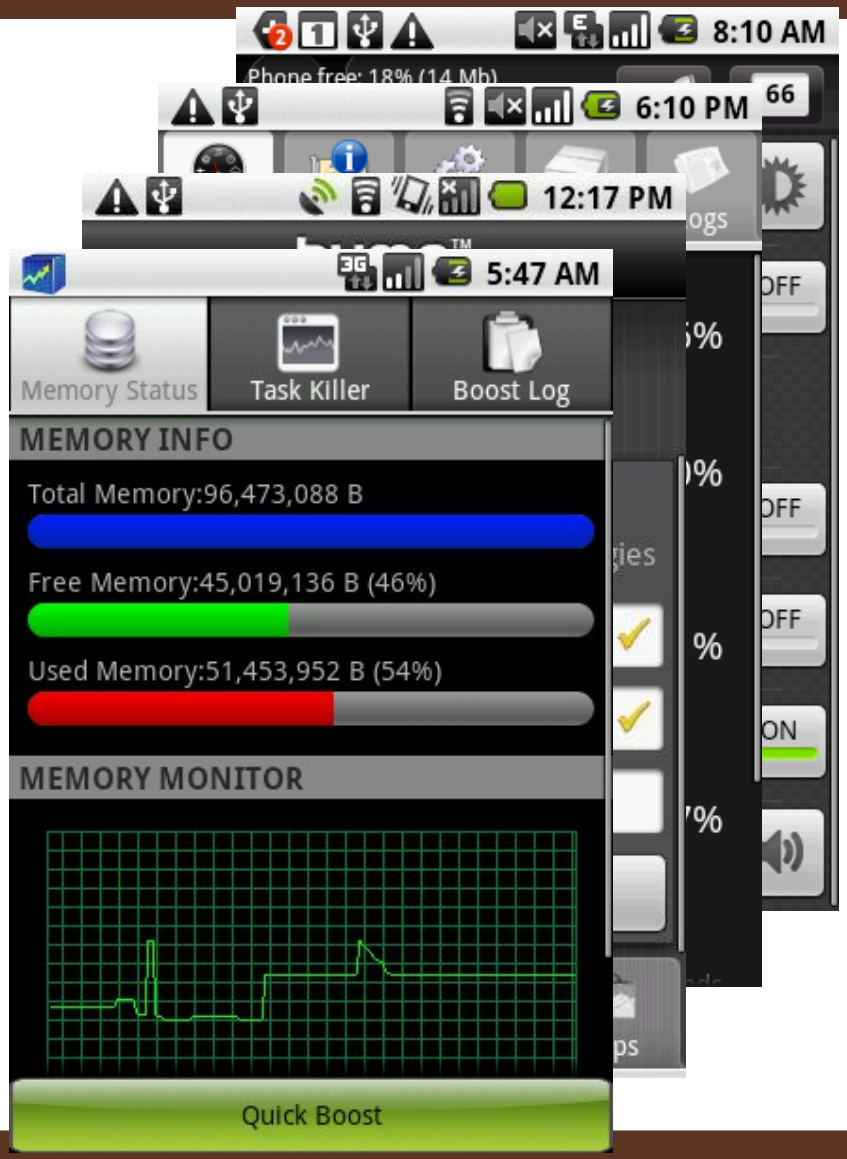
- Le activity sono
  - Uno dei componenti di un'applicazione
  - Uno dei componenti di un task (come visto dall'utente)



Activity stack

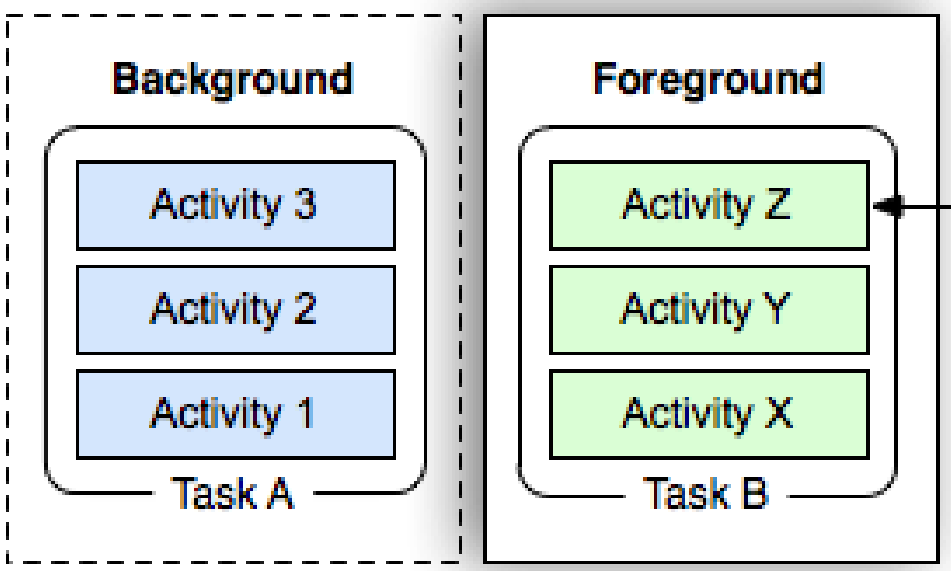
- L'utente passa da un'activity all'altra nel corso del suo task
  - *Avanti*: usando l'applicazione, si passa di schermata in schermata secondo il flusso del lavoro
  - *Indietro*: usando il tasto Back
  - *Richiamo*: dalla lista delle “applicazioni recenti”, da notifiche
  - *Interruzioni:* da eventi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

- Quando si ritorna a un'activity che era sospesa, possono accadere varie cose
- Per default:
  - 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

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

Modo	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.

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 lanciar… generica… se ne indica l'… intent

```
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

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

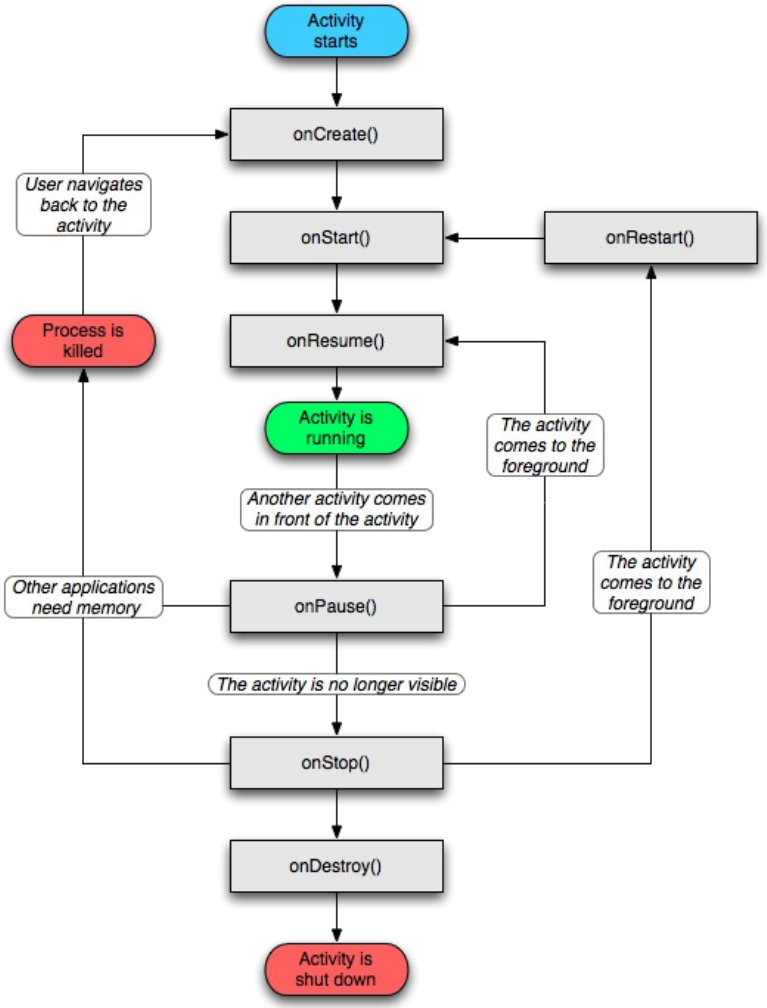
La gestione delle risorse (limitatamente alla activity)

- 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 (limitatamente alle activity)

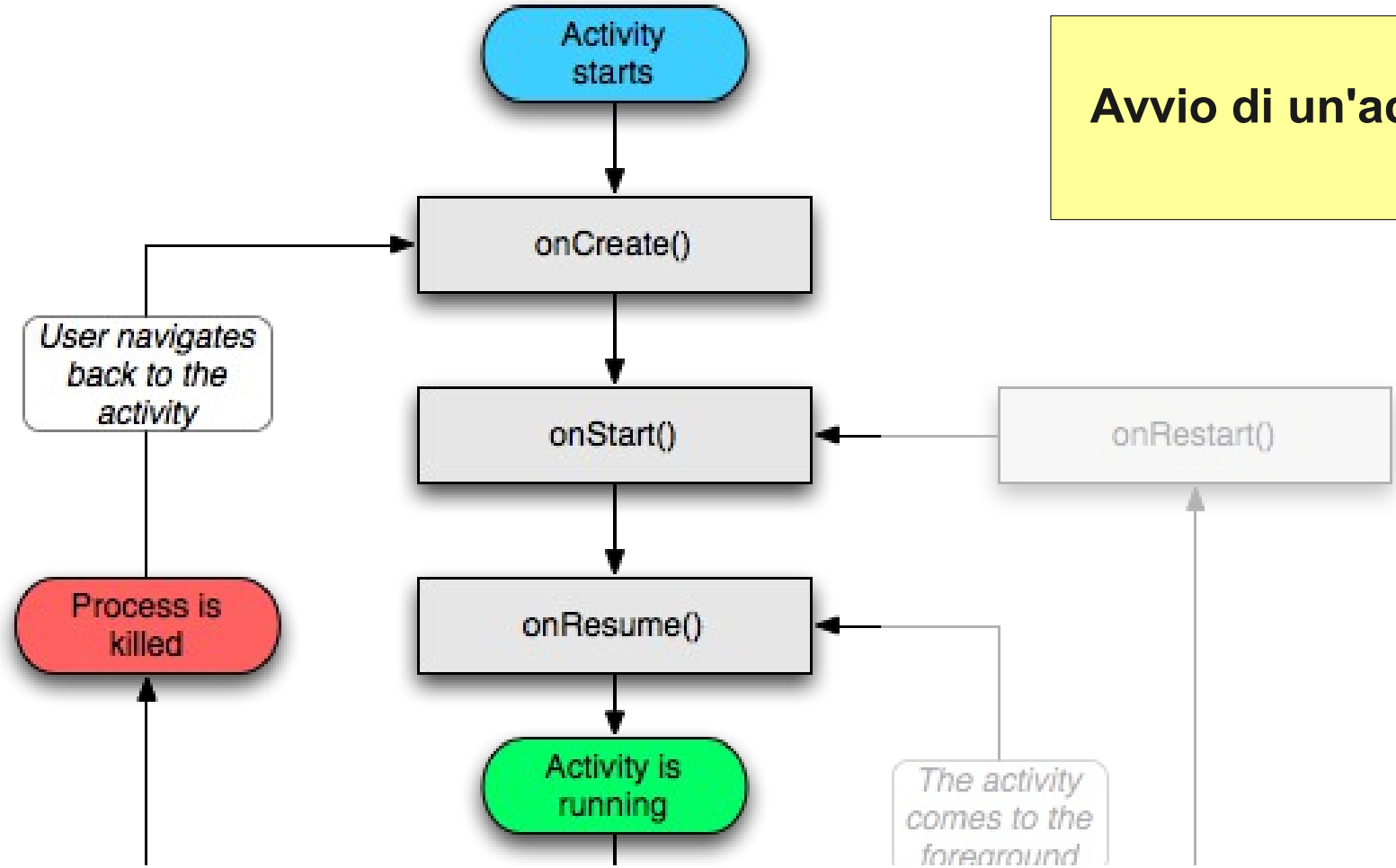
- Android gestisce dispositivi con memoria limitata
  - Niente memoria virtuale
- È **del tutto comune** che il processo che esegue la vostra activity venga ucciso
  - Per consentire all'utente di continuare nel suo task
- Bisogna quindi essere preparati a **salvare lo stato** e **ripristinarlo** se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

Il ciclo di vita di un'Activity

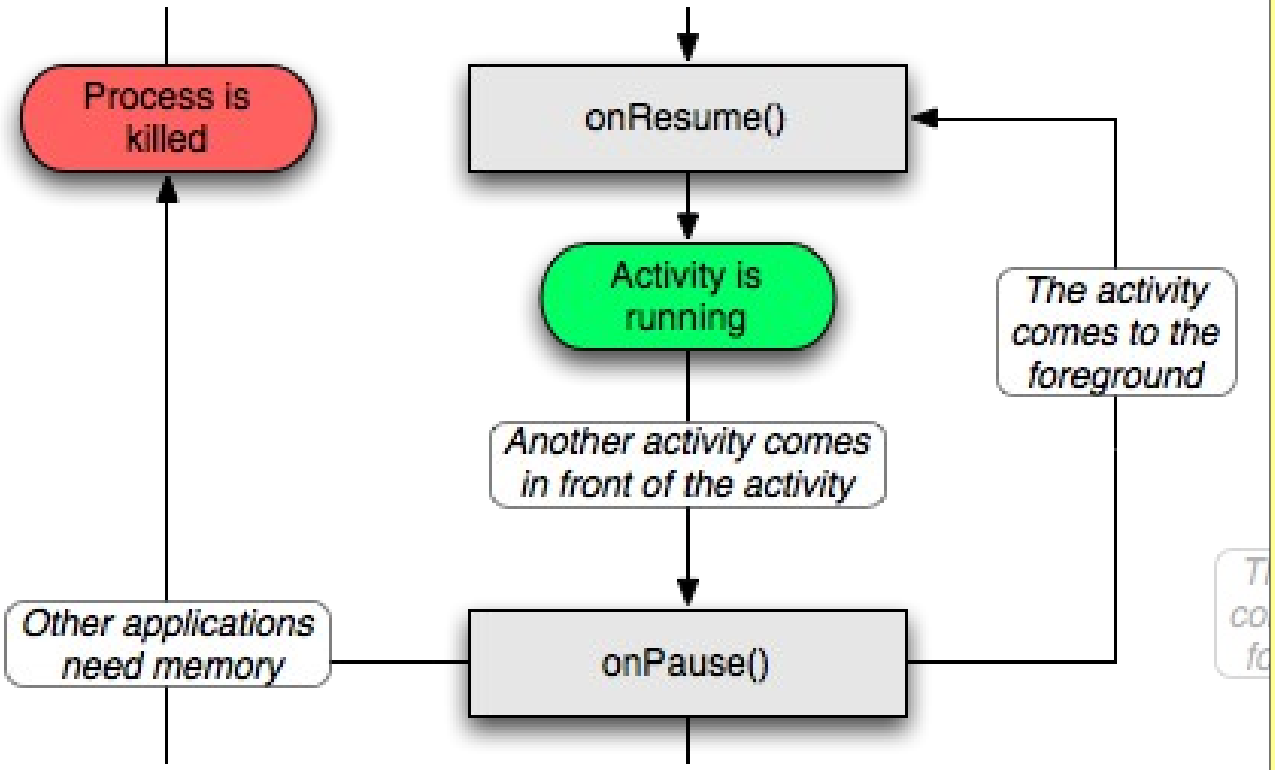


- Questo automa a stati è **fondamentale** per una corretta implementazione di un'Activity
- È una delle figure classiche, la ritroverete su qualunque testo/sito
- La vedremo a parti...

Il ciclo di vita di un'Activity



Il ciclo di vita di un'Activity

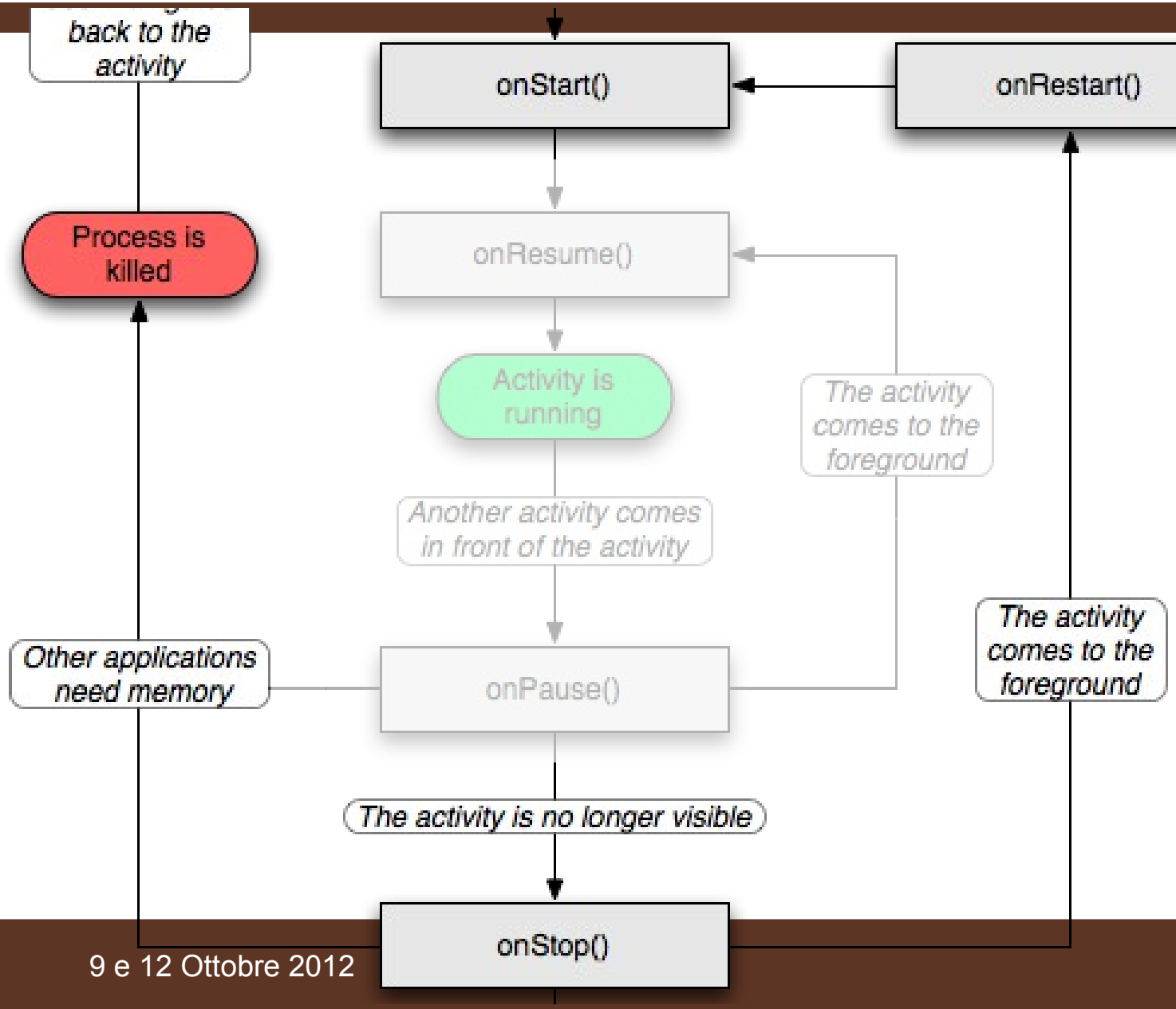


La vita adulta

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

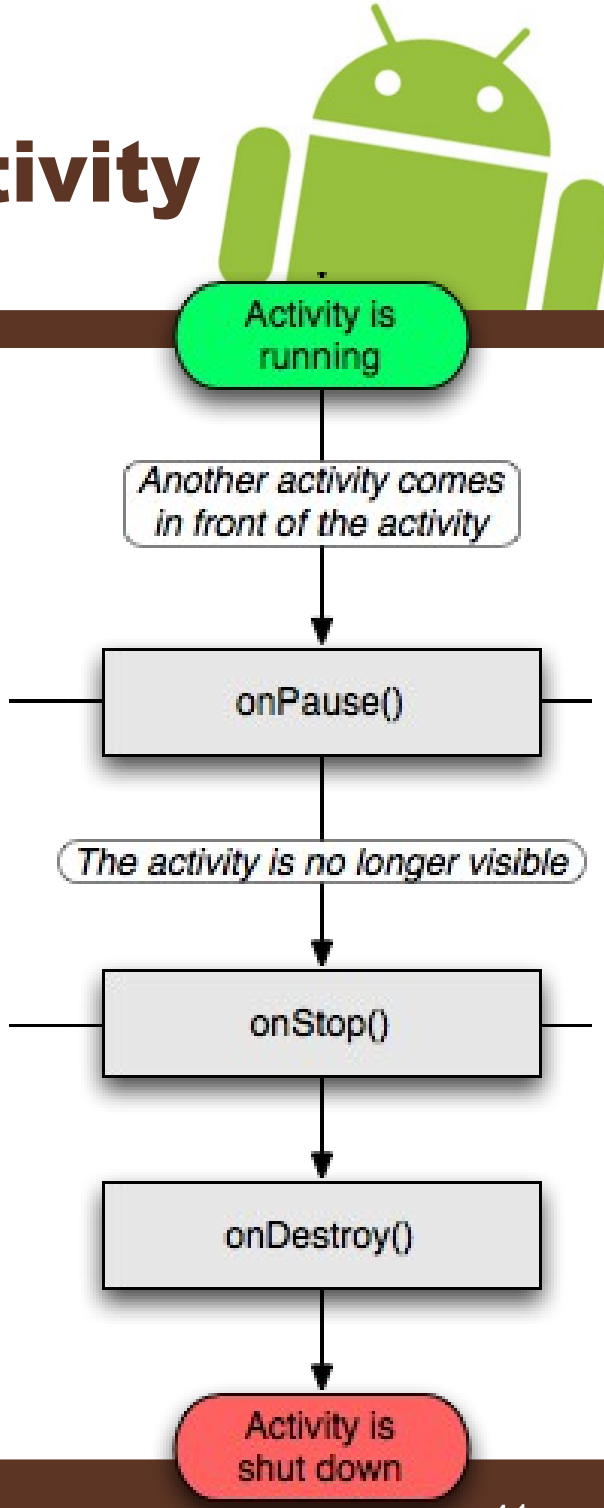
Se l'activity non solo non è in cima, ma non è neanche visibile, passa nello stato di **stop** (l'uccisione in questo caso diventa un po' più probabile).

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!



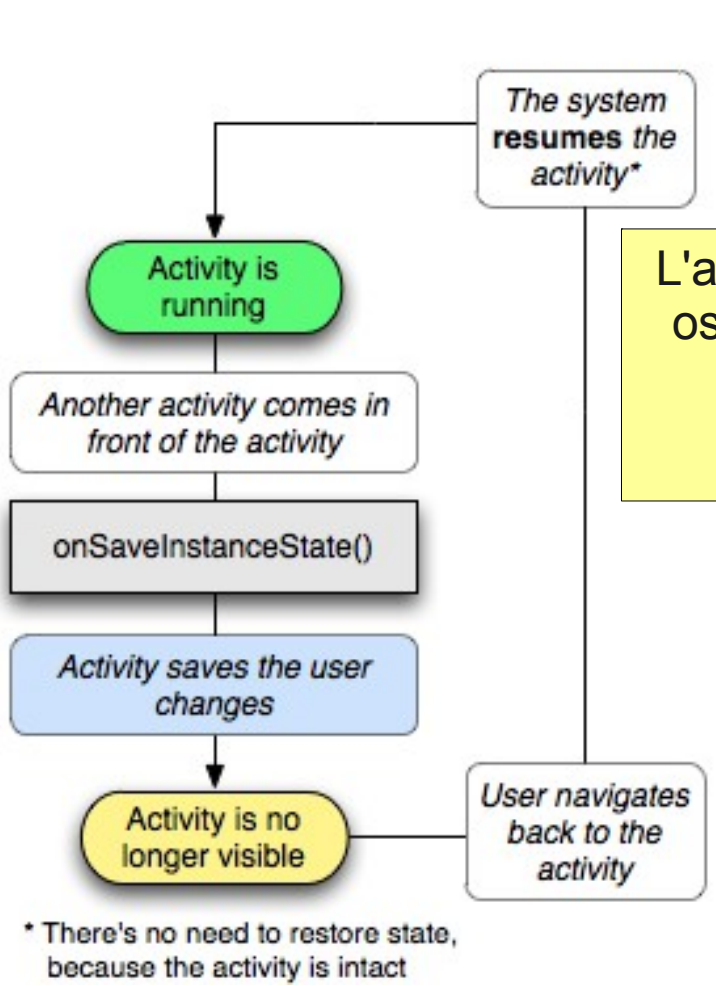
Il ciclo di vita di un'Activity

- **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)

Salviamo lo Stato!

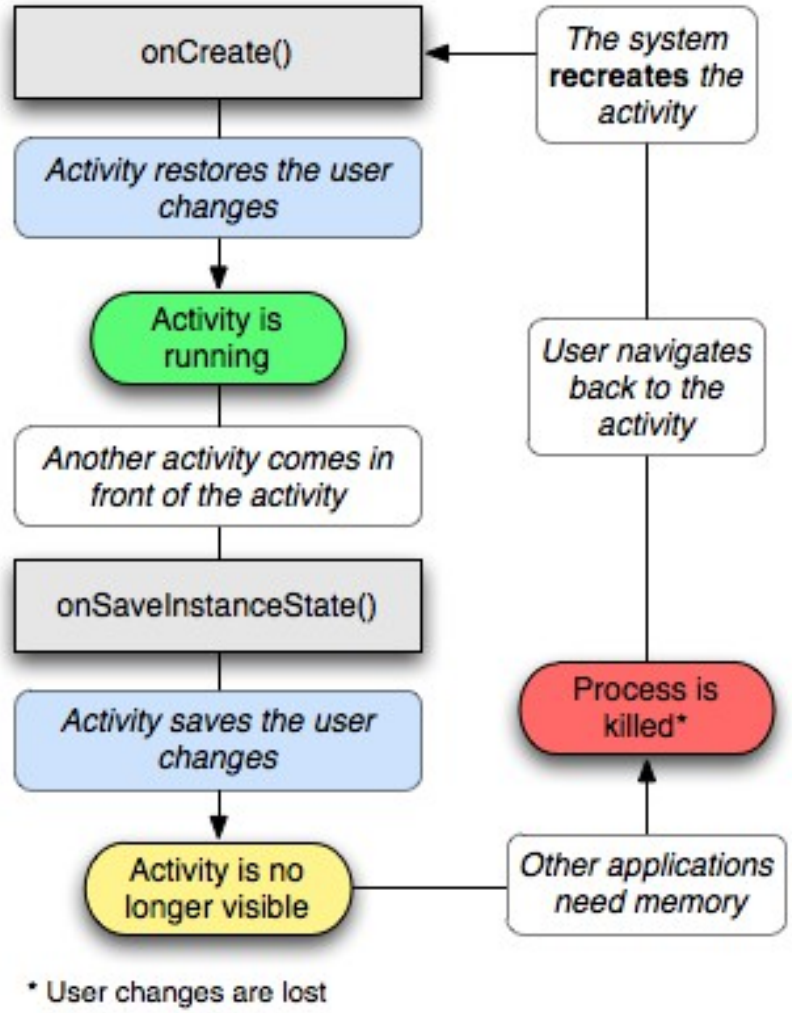
- Il callback **onSaveInstanceState()** viene chiamato dal sistema quando è necessario salvare una parte dello stato dell'activity per il ripristino a breve
  - Prima di **onStop()** e di **onPause()**
  - Ma può anche non essere chiamato se l'utente abbandona l'activity con back!
    - In questo caso, 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!



L'activity viene oscurata, ma tenuta in memoria

L'activity viene distrutta, poi ricreata



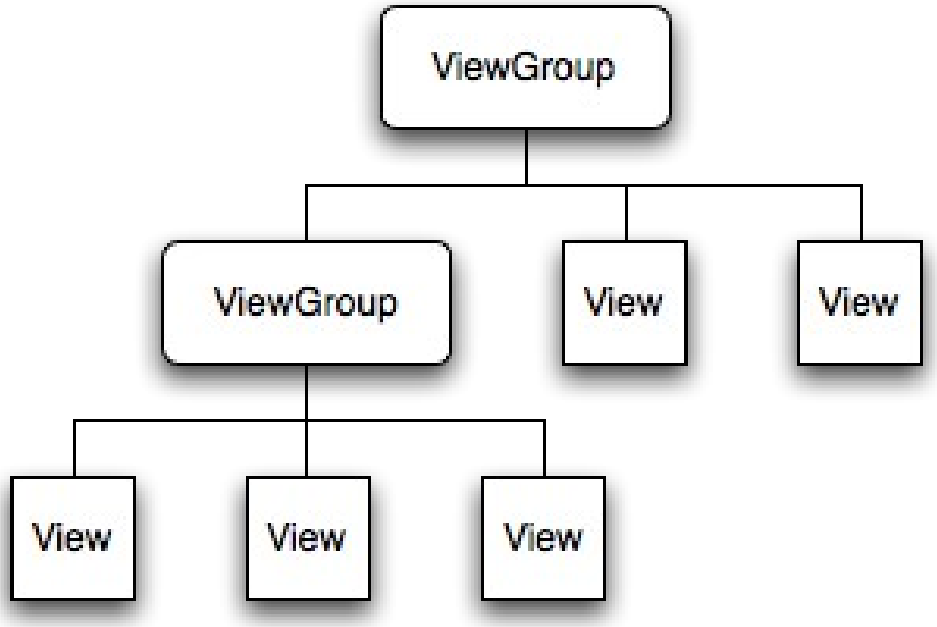
Salviamo lo Stato!

- 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()

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, lan                    ty
    }
};

protected void onCreate(Bundle stat
    ...
    // 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
    }
    ...
}
```