

**SUPSI**

# MVC e Activity Lifecycle

Sviluppo di Applicazioni Mobile

Vanni Galli, lecturer and researcher SUPSI

## Obbiettivi

- Familiarizzare con l'MVC su Android
- Comprendere il lifecycle delle applicazioni su Android

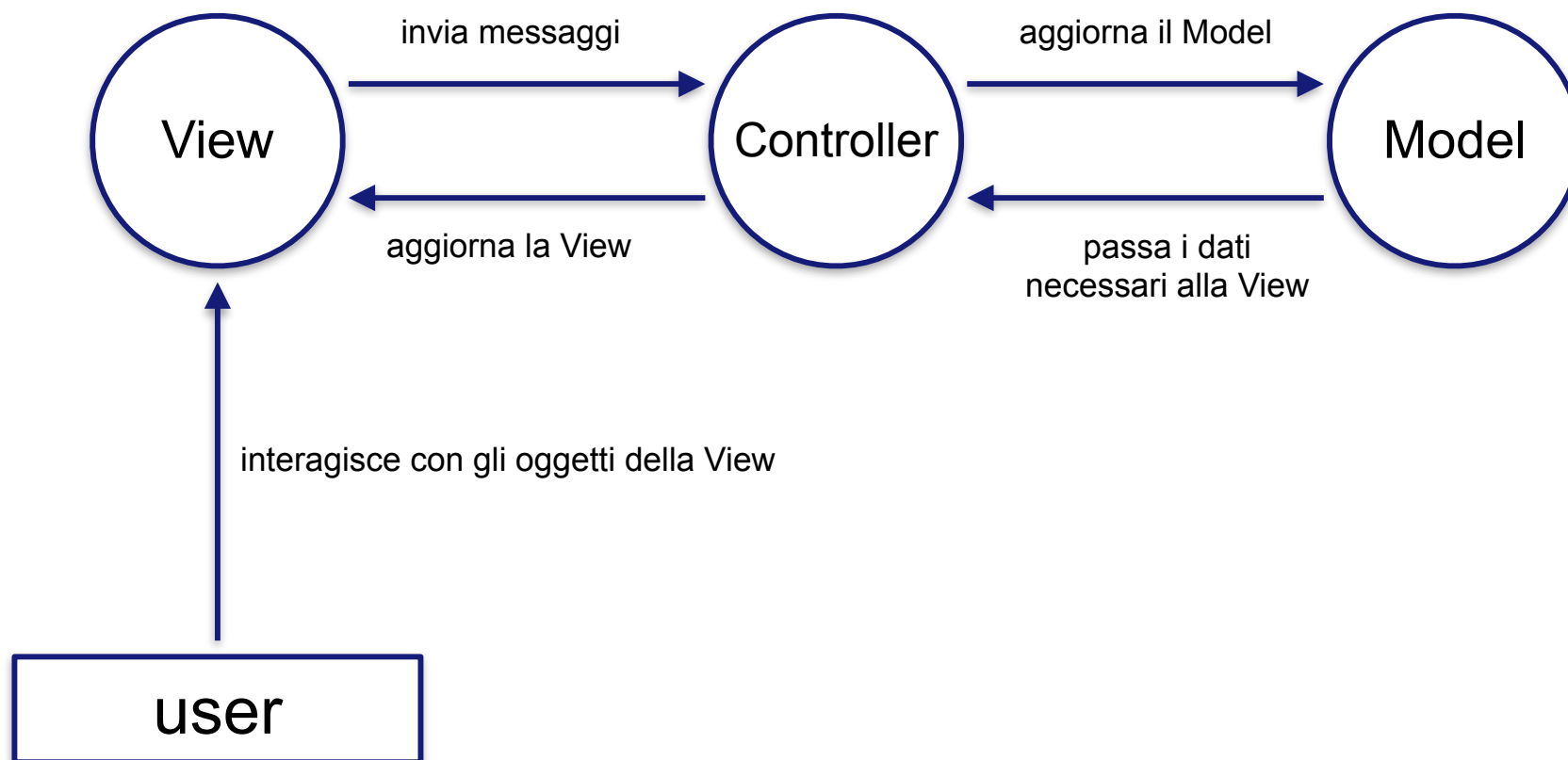
## L'MVC su Android

- Le applicazioni Android vengono disegnate attorno ad un'architettura *MVC* (Model - View - Controller)
- In un'architettura MVC tutti gli oggetti sono un Modello, una View oppure un Controller

## L'MVC su Android

- I modelli contengono i dati dell'applicazione e la “business logic”
  - Gli oggetti del modello non hanno nessuna conoscenza della UI, il loro scopo è quello di gestire e mantenere i dati.
  - Generalmente si tratta di classi sviluppate appositamente
- Le view “sanno come disegnarsi” sullo schermo e sanno come rispondere all'interazione con l'utente
  - Rule of thumb: “se posso vederla sullo schermo, allora è una view”
  - Android mette già a disposizione una grande varietà di classi personalizzabili per le View
- I controller connettono le view e i modelli. Contengono l'“application logic”
  - Sono disegnati per rispondere agli eventi scatenati dalle view e per gestire il flusso di dati dai modelli alle view.
  - In Android i controller sono tipicamente delle sottoclassi di *Activity*, *Fragment*, o *Service*

## L'MVC su Android

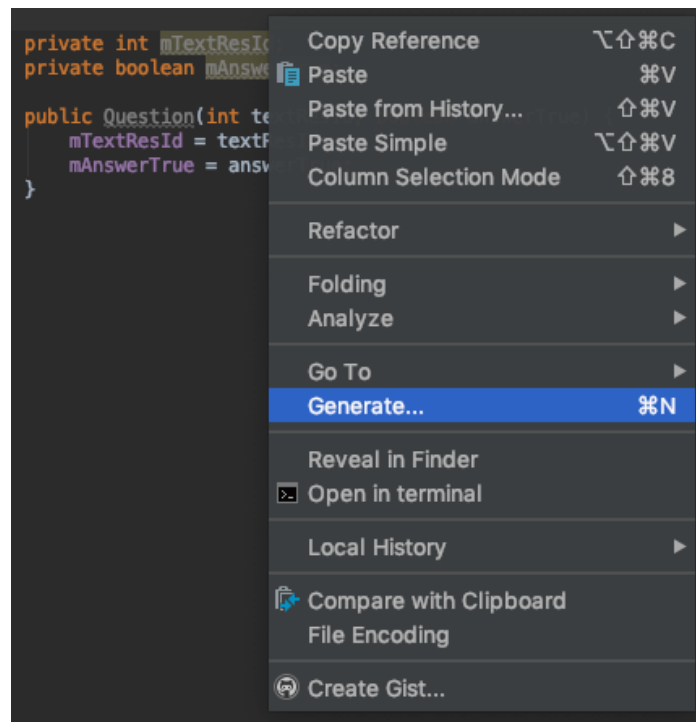


## Benefici del MVC

- Separare le classi in un'architettura Model, View e Controller aiuta a capire l'applicazione nel suo insieme
- È possibile pensare all'applicazione come un insieme di "layers" al posto che di singole classi
- Con l'MVC diventa anche più facile il riutilizzo del codice

## Creazione di una nuova classe e generazione di getter e setter

- Da Android Studio possiamo creare una nuova classe cliccando col tasto destro su di un package e facendo New → Java Class
- È poi possibile generare getter e setter dal menu *Generate*



## Referenziare gli oggetti di una view

- Gli oggetti presenti all'interno di una view possono essere referenziati nel codice tramite il Resource ID

```
<TextView
```

```
    android:id="@+id/my_text_view"
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:padding="24dp" />
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_quiz);
```

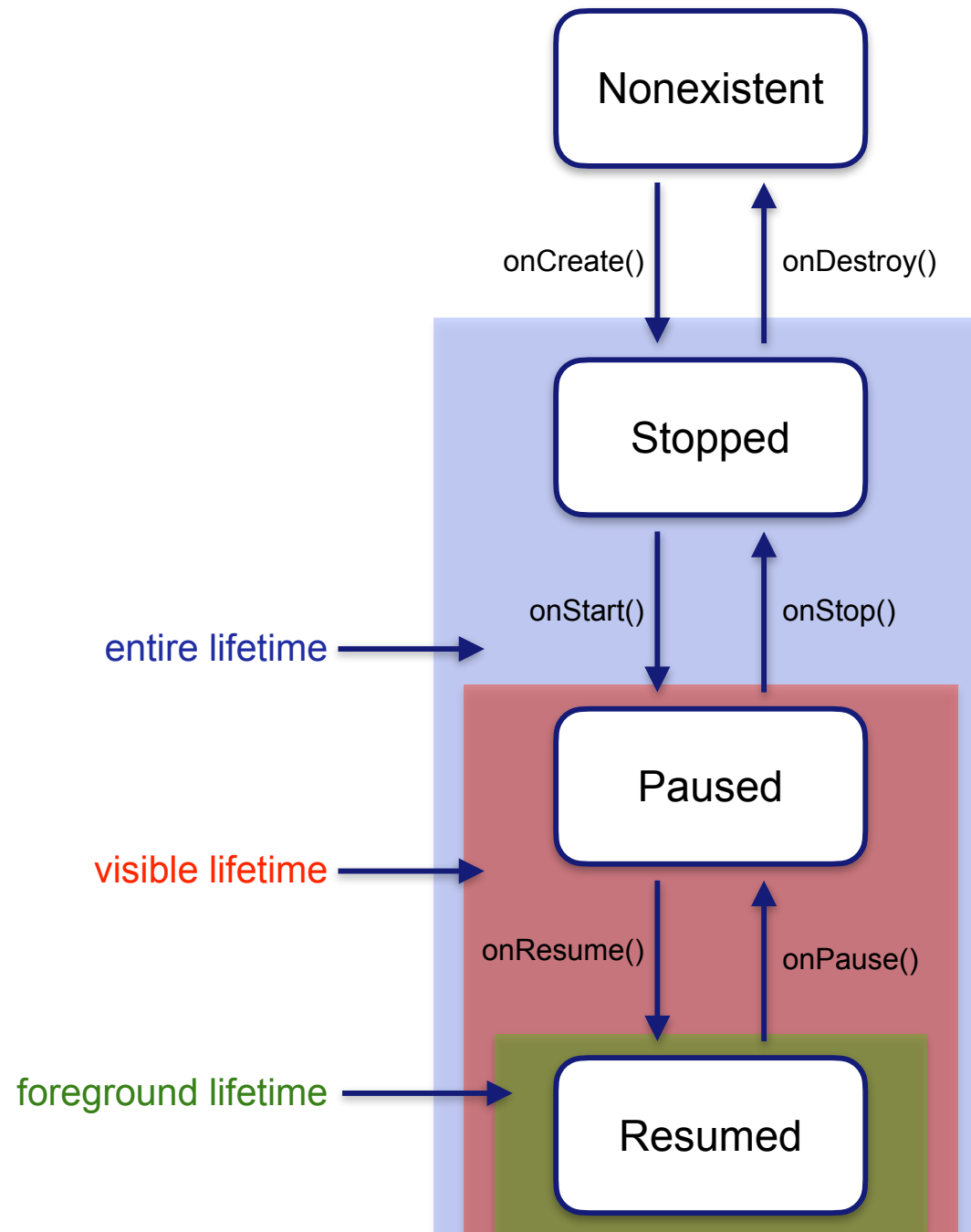
```
        myTextView = (TextView) findViewById(R.id.my_text_view);
```

```
    }
```



## Il lifecycle delle applicazioni

- Ogni istanza di un'attività ha un ciclo di vita
- Durante il ciclo di vita ci sono 4 possibili stati
- Per ogni transizione, c'è un metodo dell'attività che notifica che c'è stato un cambiamento di stato
- I metodi presentati nella prossima slides possono essere sovrascritti (*override*) per svolgere dei compiti durante le varie transizioni



## Tabella riassuntiva dei possibili stati di un'applicazione

Stato	In memoria?	Visibile all'utente?	In foreground?
nonexistent	no	no	no
stopped	si	no	no
paused	si	si (o parzialmente visibile)	no
resumed*	si	si	si

\*: Tra tutte le activity (di tutte le applicazioni installate) sul device può essercene solo una nello stato resumed.

## Il lifecycle in un esempio

- Il programma all'interno del file *LifecycleExample.zip* su iCorsi contiene gli *override* dei vari metodi che vengono chiamati durante il lifecycle dell'applicazione
- Il programma può essere usato per capire cosa succede esattamente mentre si sta usando l'applicazione
  - Cosa succede se premo il tasto *Back*?
  - Cosa succede se premo il tasto *Home*?
  - Cosa succede se premo il tasto delle applicazioni recenti e rientro nell'app?



## Il "bug" della rotazione

- Il progetto *RotationBug.zip* su iCorsi contiene una semplicissima applicazione con un bottone che incrementa un contatore
- Utilizzando l'applicazione e provando a ruotare il device, ci si può accorgere di un "bug" (apposta tra virgolette!)
- Ogni volta che si ruota il device l'istanza dell'attività viene distrutta e poi ricreata

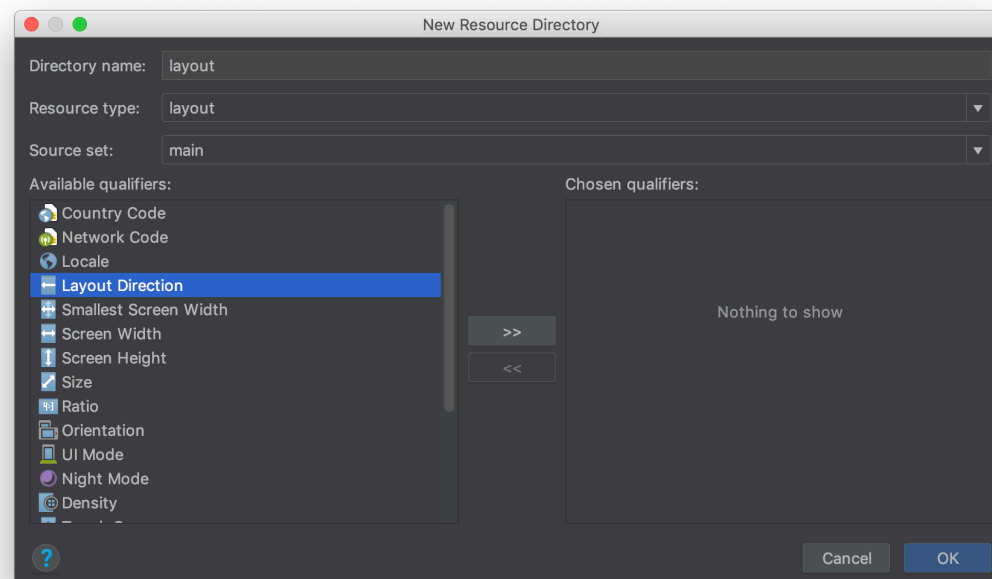


## Le device configurations

- Ruotare il device significa cambiare la *device configuration*
- Una device configuration è un set di caratteristiche che descrivono lo stato corrente del device (orientamento dello schermo, dimensioni, densità di pixels, ...)
- Quando la configuration cambia a runtime, potrebbero esserci delle risorse (e.g. un layout diverso) che più si adattano alla nuova configurazione
- Per questo motivo Android distrugge e ricrea l'activity

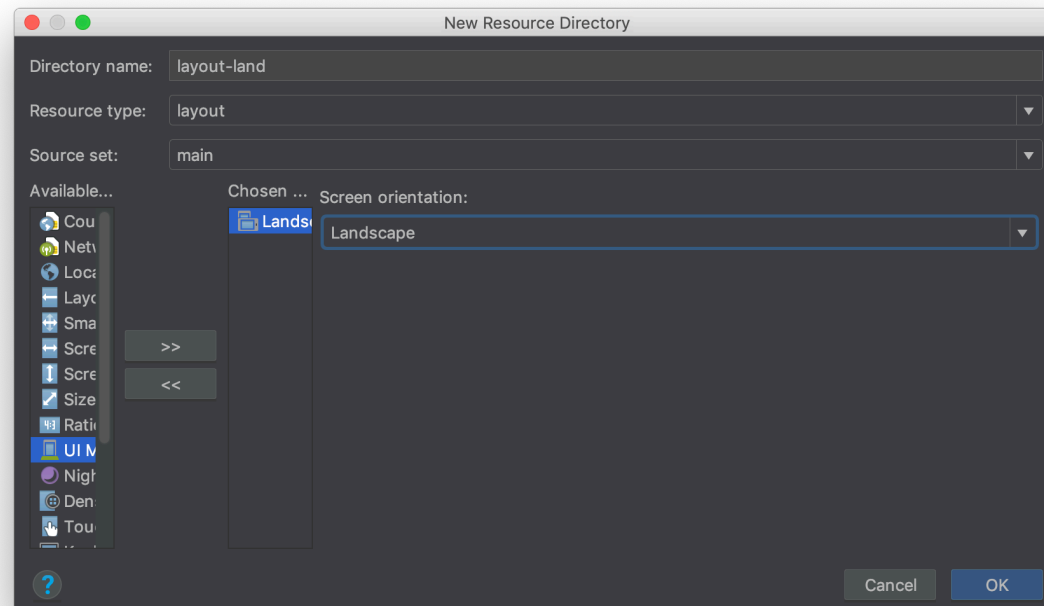
## Creazione di un layout alternativo

- Al momento della rotazione, Android può fare lo switch automatico del layout
- Occorre creare un nuovo layout andando dapprima nella visualizzazione *Project*, cliccando con il tasto destro su *res* e selezionando *New* → *Android resource directory*
  - Occorre poi selezionare *layout* nel dropdown *Resource type*:



## Creazione di un layout alternativo

- È poi possibile selezionare cosa "qualifica" un layout (ad esempio il suo orientamento) dalla lista *available qualifiers*
- Ad esempio selezionando Orientation si può creare un layout per la modalità landscape:





## Creazione di un layout alternativo

- Android Studio aggiunge automaticamente un suffisso al nome del Layout
- Ad esempio per l'orientamento Landscape, il nome diventa *layout-land*
- I *configuration qualifiers* (nelle sottodirectory di *res*) vengono usati da Android per capire quale risorsa è più ideale per una data configurazione
- Se ad esempio copiamo (e poi modifichiamo) il file *activity\_main.xml* da *res/layout* a *res/layout-land*, Android sarà in grado di usare il layout corretto in base all'orientamento del layout
- La lista completa dei *configuration qualifiers* è reperibile presso [developer.android.com/guide/topics/resources/providing-resources.html](https://developer.android.com/guide/topics/resources/providing-resources.html)

## Mantenere lo stato durante un cambio di Layout

- È possibile mantenere lo stato durante i cambi di layout (ad esempio in caso di rotazioni) sovrascrivendo il seguente metodo:

```
protected void onSaveInstanceState(Bundle outState)
```

- Questo metodo viene chiamato prima del metodo `onStop()`
- Eccezione: quando l'utente preme *back* sta dicendo ad Android che ha finito di usare l'attività; l'attività viene quindi cancellata completamente dalla memoria e il metodo `onSaveInstanceState` non viene chiamato

## I Bundle

- L'implementazione di default di `onSaveInstanceState(Bundle)` si aspetta un oggetto *Bundle*
- Un bundle è una struttura che mappa chiavi e valori
- Sovrascrivendo il metodo è possibile salvare dati aggiuntivi in un bundle:

```
private static final String MY_KEY = "key";  
  
@Override  
public void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
    savedInstanceState.putInt(MY_KEY, 10);  
}
```

## I Bundles

- Nei bundle è possibile salvare un qualsiasi dato primitivo, così come classi che implementano l'interfaccia *Serializable* o *Parcelable*
- Inserire oggetti in un bundle è comunque una "bad practice"
- Una migliore pratica è quella di usare un altro sistema per salvare gli oggetti, e salvare poi nel bundle solamente dei dati primitivi per ri-identificare gli oggetti

## Caricare lo stato salvato

- Una volta salvato lo stato attuale dell'applicazione, occorre ricaricarlo quando l'attività viene (ri)creata
- Il caricamento viene fatto direttamente nel metodo `onCreate()`:

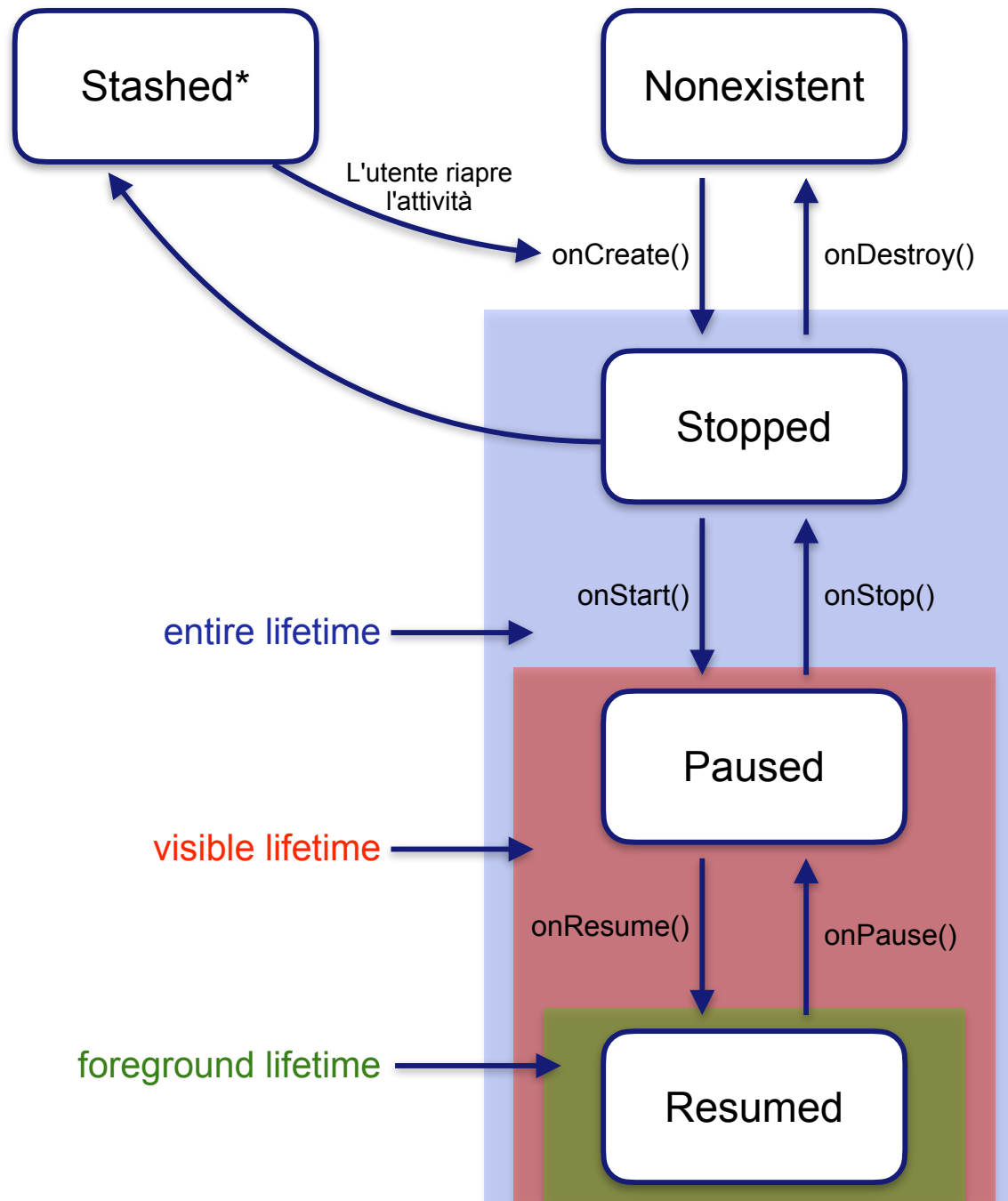
```
private static final String MY_KEY = "key";

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

    if (savedInstanceState != null)
        int myInt = savedInstanceState.getInt(KEY_INDEX, 0);
}
```

## Ancora sul mantenimento dello stato

- Riscrivere il metodo `onSaveInstanceState(Bundle)` non serve solo in caso di rotazioni del device
- In alcune situazioni Android potrebbe necessitare di liberare della memoria
- Un'activity può venire anche distrutta nel caso in cui l'utente esca da questa attività senza premere *Back* (ma premendo ad esempio *Home*)
- Per essere distrutta da Android, l'activity deve però essere **Stopped**
  - Questo significa che, se ci troviamo nello stato Stopped, il metodo `onSaveInstanceState(Bundle)` è stato per forza chiamato!



\*: l'istanza è morta,  
lo stato dell'istanza è salvato

Faccio l'override di  
`onStop()` per salvare i dati  
che voglio siano  
"permanenti".

Faccio l'override di  
`onSaveInstanceState(Bundle)` per salvare dati  
"transienti".

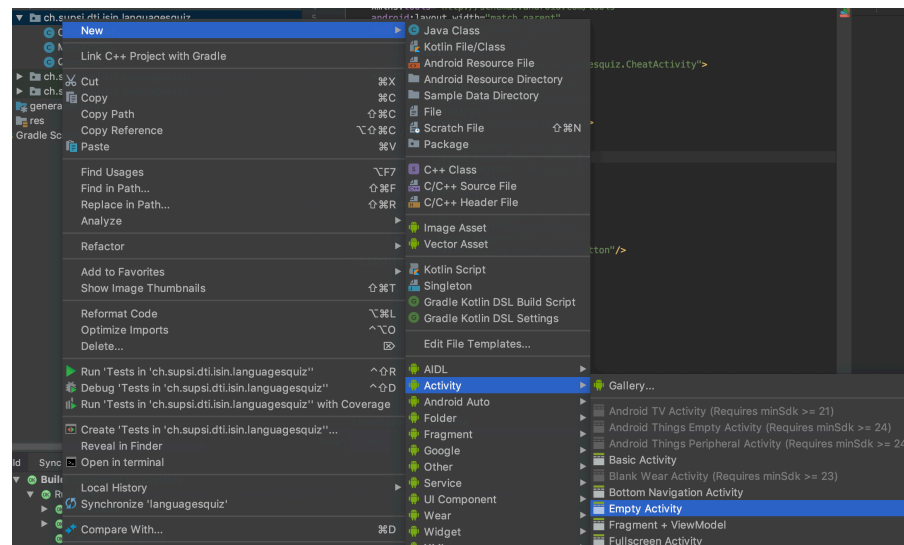
## Testare il mantenimento dello stato

- È possibile simulare una situazione di cui Android libera la memoria
- Nei *Settings* del device, occorre cliccare le *Developer Options* e attivare l'opzione *Don't keep activities*
- A questo punto, uscendo da un'attività con *Home* si simula la chiusura dell'attività da parte di Android
- È dunque possibile testare il proprio metodo *onSaveInstanceState(Bundle)*



## Aggiunta di altre Activity

- Aggiungere una nuova activity comporta di solito la creazione / modifica di 3 files: la classe (*java*), il layout (*xml*) e il *manifest* dell'applicazione
- Per aggiungere ulteriori activity alla propria App, occorre innanzitutto crearle nel progetto, tramite il menu New → Activity (→ Empty Activity)
- Creando una nuova activity vengono generati i corrispettivi file *java* e *xml*



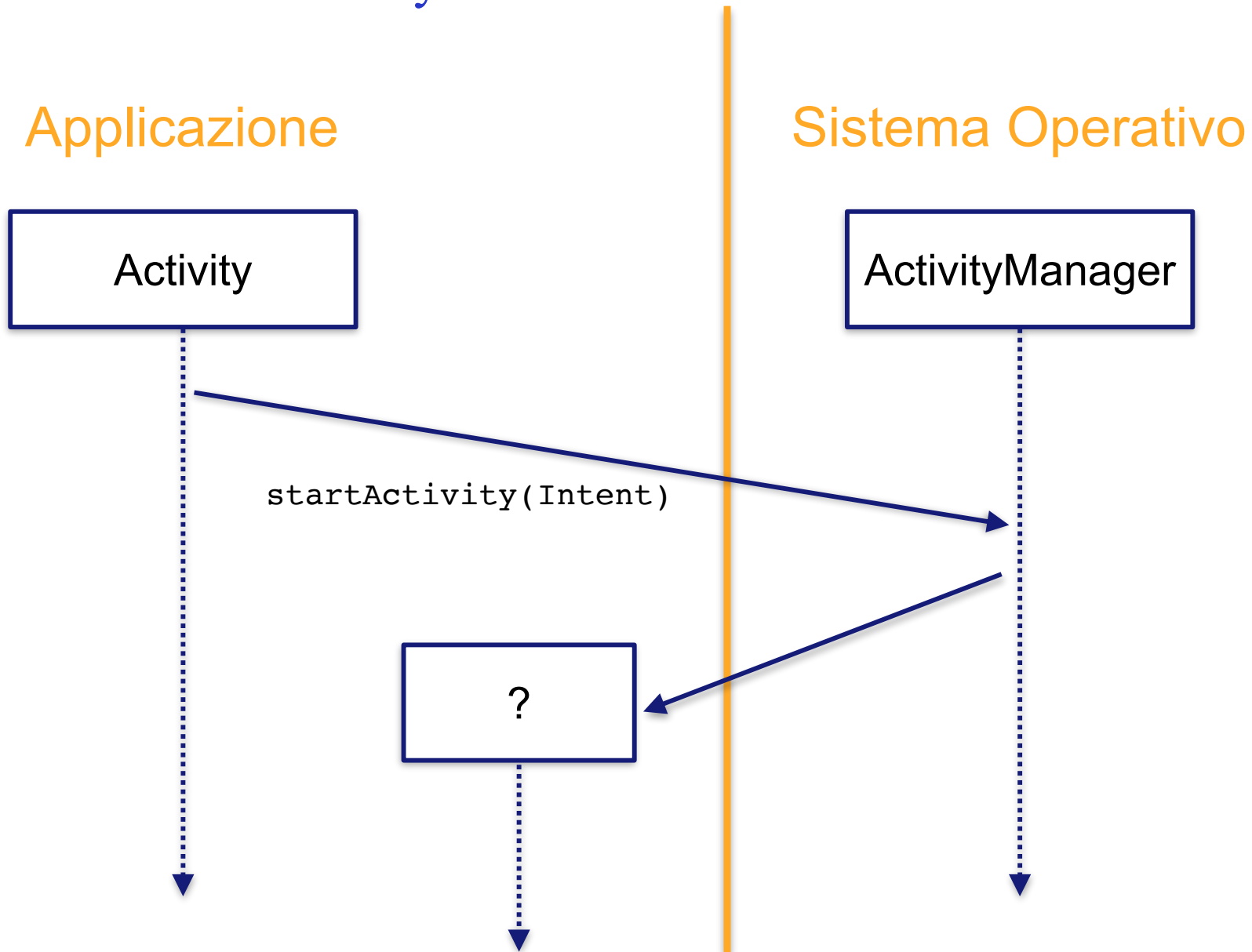
## Dichiarazione dell'Activity nel manifest

- Il *manifest* è un file XML che contiene metadati che descrivono l'applicazione al sistema operativo Android
- Ogni activity deve essere dichiarata nel file manifest!
- Di norma il wizard di Android Studio dichiara già le nuove activity nel file manifest

## Start di una nuova activity

- Il modo più semplice per fare in modo che un'activity ne faccia partire un'altra è il metodo `public void startActivity(Intent intent)`
- Il metodo `startActivity` non viene gestito direttamente dalla classe `activity`!
- Le chiamate a `startActivity` vengono invece inviate all'`ActivityManager` del sistema operativo

## Start di una nuova activity



## Gli Intent

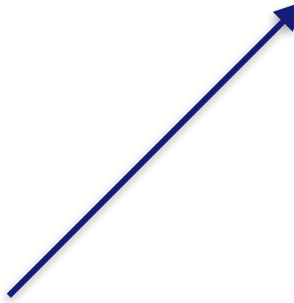
- Un intent è un oggetto che un componente può usare per comunicare con il sistema operativo
- Un componente può essere un'activity, ma anche un *service*, un *broadcast receiver*, ...
- La classe intent mette a disposizione differenti costruttori in base all'utilizzo che si vuole fare con l'intent
- Gli intent possono essere *explicit* (per far partire activity nella propria applicazione) o *implicit* (per activity in altre applicazioni)

## Gli Intent


- Per lanciare una nuova activity si utilizza il metodo  
`public Intent(Context packageContext, Class<?> cls)`

- Ad esempio:

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

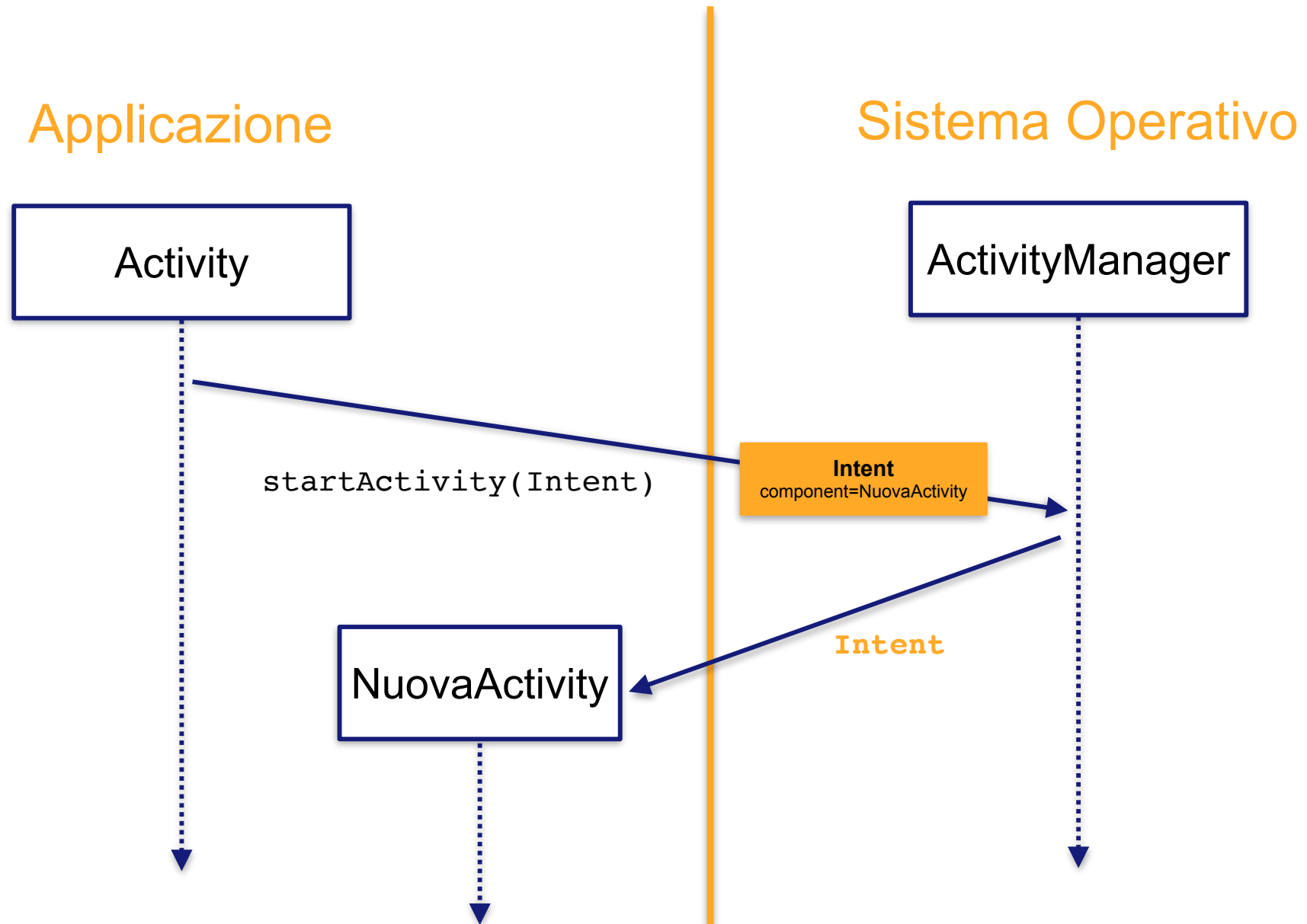


contesto attuale  
(dove trovo la nuova activity)



la nuova activity  
(deve essere presente nel manifest!)

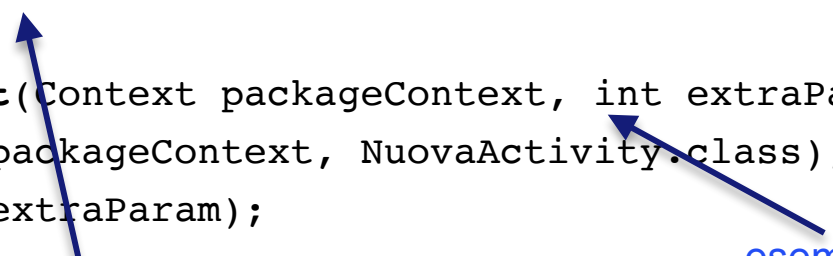
## Start di una nuova activity



## Intent extra - Best Practices

- È possibile creare un intent e aggiungere degli extra direttamente nell'attività chiamante
- L'attività chiamante non deve però per forza conoscere i dettagli dell'implementazione dell'attività chiamata
- Per questo motivo è *best practice* dichiarare un metodo nell'activity chiamata che restituisca l'intent per questa activity:

```
public class NuovaActivity extends AppCompatActivity {  
    private static final String EXTRA_KEY = "ch.supsi.dti.isin.example";  
  
    public static Intent newIntent(Context packageContext, int extraParam) {  
        Intent intent = new Intent(packageContext, NuovaActivity.class);  
        intent.putExtra(EXTRA_KEY, extraParam);  
        return intent;  
    }  
}
```



dichiaro la chiave dell'extra in un posto solo

esempio!



## Intent extra - Best Practices

- Una volta dichiarato, nell'activity chiamata, un metodo che torna un intent, è poi possibile richiamarlo dall'activity chiamante:

```
public void onClick(View v) {  
    Intent intent = NuovaActivity.newIntent(QuestaActivity.this, 1);  
    startActivity(intent);  
}
```

chiedo all'activity child l'intent per chiamarla

- Il parametro passato può essere poi letto nell'activity chiamata:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_cheat);  
    int i = getIntent().getIntExtra(EXTRA_KEY, 0);  
}
```

ritorna sempre l'intent che ha lanciato l'activity

esempio!

default

## Leggere un risultato di ritorno dall'attività chiamata (vecchio metodo)

- Se l'attività chiamante vuole "limitarsi" a chiamarne un'altra, ignorando poi eventuali dati di ritorno, basta utilizzare:

```
public void startActivity(Intent intent)
```

- Se invece si vogliono leggere dei dati "di ritorno" dall'attività chiamata, occorre utilizzare il seguente metodo (con un *request code*):

```
public void startActivityForResult(Intent intent, int  
requestCode)
```

- Questo metodo è *deprecated* in favore di della classe `ActivityResultLauncher`.

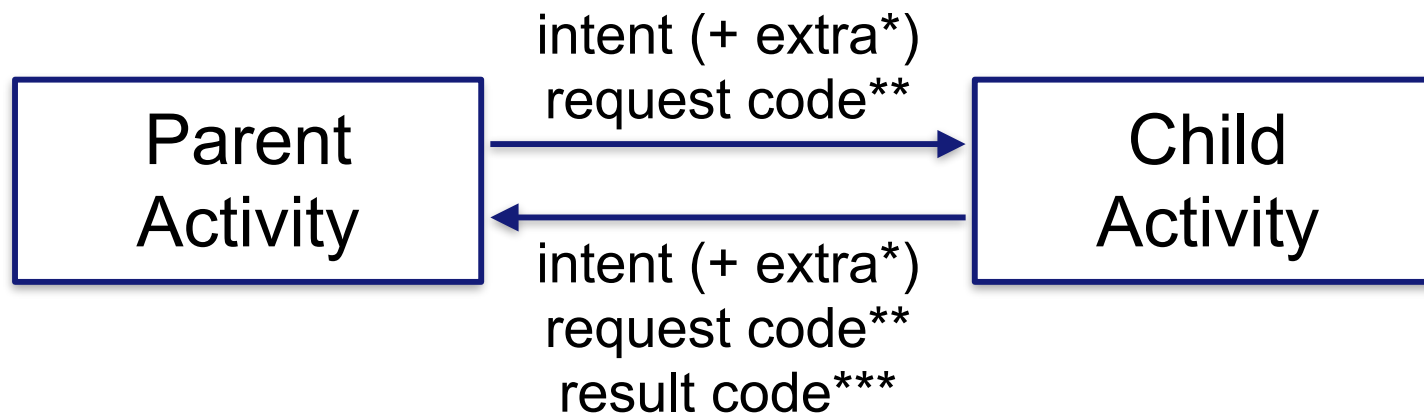
## Utilizzo del requestCode per distinguere le activity figlie

- Se si vogliono leggere i risultati provenienti da un'attività figlia, occorre prima definire un requestCode quando si chiama l'attività figlia:

```
public void startActivityForResult(Intent intent, int  
requestCode)
```

- Il secondo parametro del metodo è il *request code*
- Il request code è un intero che:
  - viene mandato all'activity figlia
  - viene mandato indietro (automaticamente) dall'activity figlia all'activity parent

## Passaggio di dati da un activity all'altra

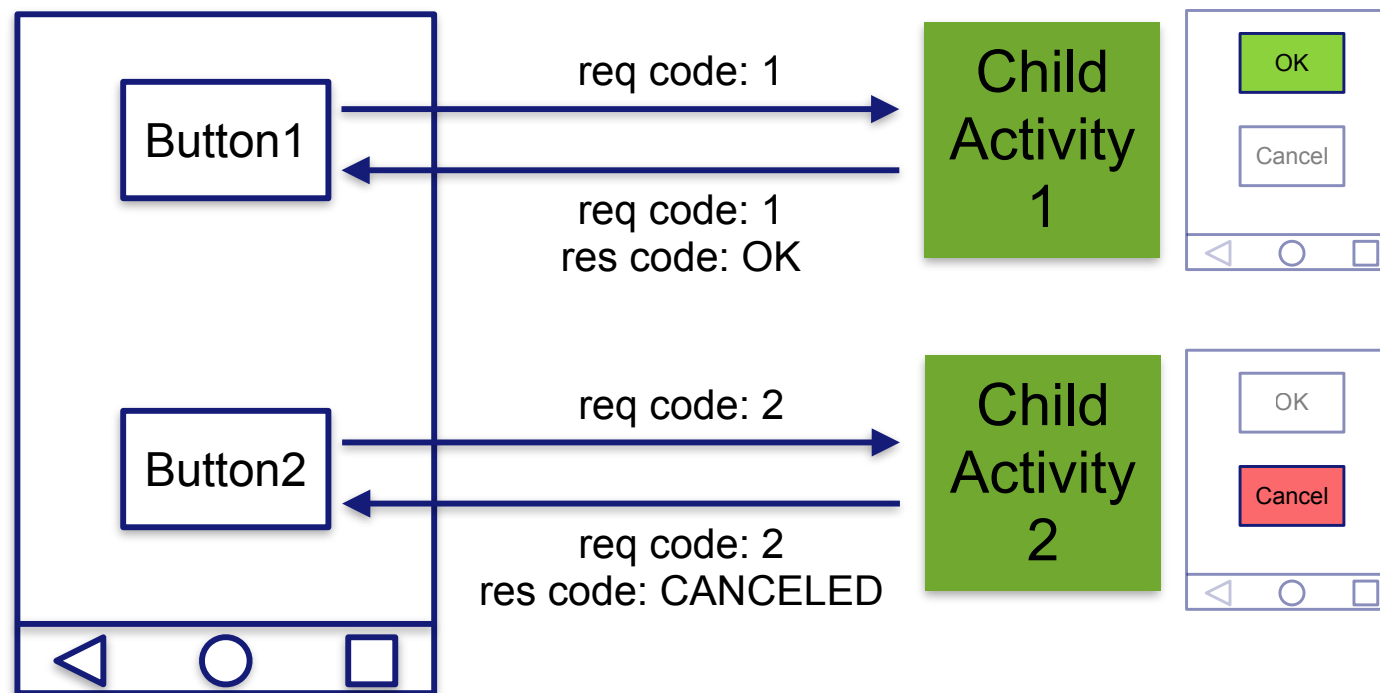


\*: dati aggiuntivi (*key-value pair*) che l'activity può includere in un intent

\*\**: int* usato (principalmente) nei casi in cui un *parent* lanci più *children* e voglia poi sapere chi sta "tornando"

\*\*\*: tipicamente un valore costante che indica se l'activity child è andata a buon fine

## Request code e Result code



## Settare un valore di ritorno nell'attività figlia

- Per settare un risultato di ritorno si possono usare i seguenti metodi:

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

tipicamente una costante tra:

`Activity.RESULT_OK`

`Activity.RESULT_CANCELED`

- Il metodo `setResult` va chiamato nell'activity figlia:

```
private void tornaAllActivityParent() {
    Intent data = new Intent();
    data.putExtra(EXTRA_KEY2, 99);
    setResult(RESULT_OK, data);
    finish(); // concludo questa attività
}
```

## Leggere i risultati di ritorno nell'attività parent

- I risultati inviati dalle activity children, possono poi essere letti nell'activity parent tramite il metodo `onActivityResult`:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        // c'è stato un errore (da gestire)
    }
    if (requestCode == MIO_CODICE) {
        if (data == null) {
            // non ho nessun intent di ritorno
        }
        // controllo gli extra nell'intent...
    }
}
```

il requestCode "originale"

il codice che mi dice se è tutto OK

eventuale intent di ritorno

## Leggere un risultato di ritorno dall'attività chiamata (nuovo metodo)

- `ActivityResultLauncher` è *deprecated*; ora, per leggere il risultato di ritorno da un'attività chiamata occorre usare la classe `ActivityResultLauncher`.
- Il concetto di *request code* va perso, in quanto ogni chiamata ad un'attività figlia viene fatta da una classe diversa.
- Il *result code* invece può / deve ancora venire gestito.



## Leggere un risultato di ritorno dall'attività chiamata (nuovo metodo)

- L'assegnazione di `ActivityResultLauncher` va fatto in `onAttach()` oppure in `onCreate()`, cioè prima che l'attività venga visualizzata:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...

    ActivityResultLauncher<Intent> someActivityResultLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        result -> {
            if (result.getResultCode() != Activity.RESULT_OK) {
                // gestione se risultato non OK
            }
            Log.d("tag", String.valueOf(result.getData().getIntExtra("EXTRA_KEY2", 0)));
        });

    myButton.setOnClickListener(v -> {
        Intent intent = MainActivity2.newIntent(MainActivity.this, 1);
        someActivityResultLauncher.launch(intent);
    });
}
```

## Esempio

- Il progetto *ActivitiesTester.zip* su iCorsi contiene una semplice applicazione che permette di navigare tra 2 attività
- Tra le due attività c'è un passaggio di dati (in entrambe le direzioni)

