1-The Android Platform

- Android Jetpack
- La piattaforma Android
- Kotlin
- Android visto ad alto livello
- La macchina virtuale Dalvik
- Applicazioni Sandbox
  - Sicurezza e Permessi
    - Permessi delle applicazioni
    - Limited Ad-Hoc Permissions
    - Application Signing
- Programmazione tradizionale comparata ad Android
  - Activities
    - Il ciclo di vita delle attività
    - Salvare lo stato delle attività
      - Esempio creazione attività
  - Servizi
    - Esempio creazione servizio
  - Provider di contenuti
  - Intents
  - Ricevitori broadcast
- Android Application Context
- Android Application Manifest
  - Esempio
- R object
  - Esempio
- Application Logs
  - Buona pratica
- Model View Controller
- Model
- View
- Controller
- Android User Interface
- View Hierarchy
- Layout
  - Esempio
  - Layout Attributes
    - Layout ID Attribute
      - Esempio
- Widgets
- Il ruolo chiave delle Activities in Android
  - Activity
    - Esempio
- Input events
  - Esempio
  - Button Events
    - Esempio
  - Launch/Finish Activity

Android Jetpack

Jetpack è una collezione di componenti software per Android che rende più facile lo sviluppo di app Android di alta qualità. Questi componenti aiutano gli sviluppatori a seguire le migliori pratiche, liberandoli dalla scrittura di codice ripetitivo e semplificando compiti complessi, consentendo loro di concentrarsi sul codice che realmente conta.

I componenti di Jetpack sono parte delle librerie `androidx.*` ed essi sono svincolati dalle API della piattaforma Android.
Questo significa che offrono compatibilità con versioni precedenti e vengono aggiornati più frequentemente rispetto alla piattaforma Android, garantendo che si abbia sempre accesso alle versioni più recenti e avanzate dei componenti Jetpack.

La piattaforma Android

Android è una piattaforma software per dispositivi mobili che include un sistema operativo, middleware e applicazioni chiave. La SDK di Android fornisce le componenti necessarie per lo sviluppo di applicazioni sulla piattaforma Android utilizzando il linguaggio di programmazione Java (ora anche Kotlin).

> Nota
> Questo sistema operativo è basato su Linux ed è destinato a dispositivi mobili come smartphone e tablet, attualmente sviluppato dall'Open Handset Alliance, guidata da Google

Il kernel Linux 2.6 gestisce i servizi di base del sistema, funge da strato di astrazione dell'hardware (HAL) tra l'hardware fisico e lo stack software di Android, e si occupa delle funzioni:

- Permessi delle applicazioni e sicurezza, gestisce i permessi concessi alle applicazioni e garantisce la sicurezza del sistema
- Gestione energetica a basso livello, si occupa della gestione dell'energia per ottimizzare l'efficienza del dispositivo
- Gestione dei processi e del threading, gestisce i processi in esecuzione sul dispositivo e la gestione dei thread per garantire un funzionamento fluido e reattivo
- Networking, si occupa della connettività di rete per consentire al dispositivo di comunicare con il mondo esterno.
- Display, input della tastiera, telecamera, memoria Flash, audio e accesso al driver binder (IPC), gestisce l'interazione con l'hardware del dispositivo, inclusi display, input, fotocamera, memoria Flash e audio, nonché la comunicazione tra processi tramite il driver binder.

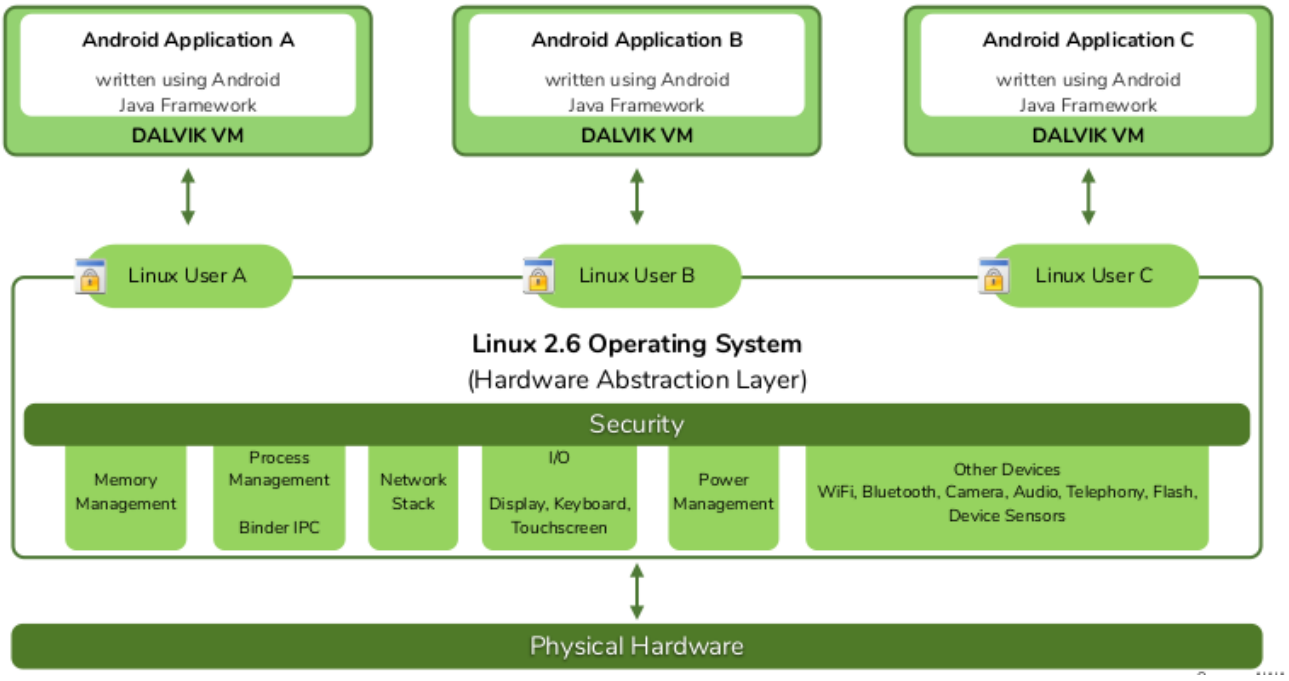
Kotlin

Kotlin è un linguaggio di programmazione ufficiale su Android. Si tratta di un linguaggio a tipizzazione statica che viene eseguito sulla macchina virtuale Java (Java Virtual Machine - JVM) e può essere compilato anche in codice sorgente JavaScript o utilizzare l'infrastruttura del compilatore LLVM. Inoltre si può dire che è un linguaggio:

- Espressivo
- Conciso
- Potente
- Intropeabile con i linguaggi e l'ambiente di runtime esistenti su Androi

---

Android visto ad alto livello



La macchina virtuale Dalvik

Dalvik è la Virtual Machine di Android, basata sulla Java VM ma ottimizzata per i dispositivi mobili. Essa presenta le seguenti caratteristiche:

- Piccola impronta di memoria, Dalvik è progettata per avere un piccolo impatto sulla memoria, il che è cruciale per dispositivi mobili con risorse limitate. Più istanze di Dalvik VM possono essere eseguite simultaneamente sul dispositivo.
- Linguaggio di programmazione principale, i programmi per Android sono principalmente scritti in un dialetto di Java e poi compilati in bytecode. Prima dell'installazione sul dispositivo, vengono convertiti da file .class (compatibili con la Java Virtual Machine) a file `.dex` (Dalvik Executable) compatibili con Dalvik. Il formato compatto dei file `.dex` è progettato per essere adatto a sistemi con limitazioni di memoria e velocità di elaborazione.
- Esecuzione in processi separati, ogni applicazione Android viene eseguita in un processo separato con la propria istanza della virtual machine Dalvik. Ciò migliora l'isolamento tra le applicazioni e aiuta a garantire la stabilità del sistema.
- Ciclo di vita dell'applicazione Android, il ciclo di vita di un'applicazione Android permette al sistema operativo di ottimizzare le prestazioni della garbage collection (ripristino della memoria non utilizzata) e la gestione del recupero della memoria attraverso più heap (spazi di memoria separati).

Applicazioni Sandbox

La sicurezza di Android è strettamente correlata alle restrizioni e ai livelli di sicurezza del sistema operativo Linux, in particolare riguardo ai limiti dei processi e degli utenti. Adotta un approccio di sicurezza che prevede la creazione di un nuovo utente per ogni venditore di applicazioni, ed ogni applicazione viene eseguita con differenti privilegi utente, a meno che non siano firmate dallo stesso venditore.

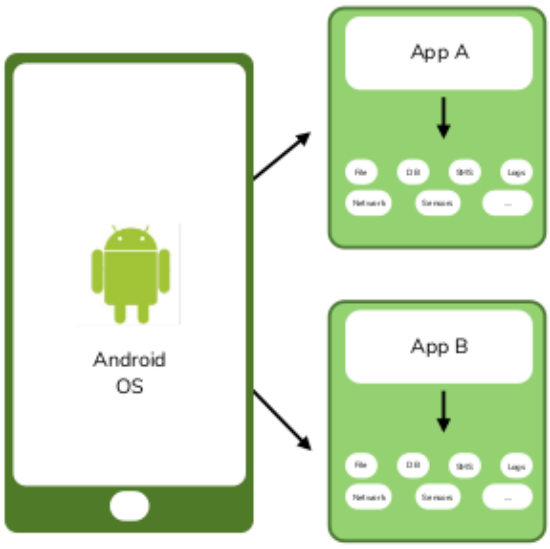
I file posseduti da un'applicazione sono, per default, inaccessibili alle altre applicazioni. L'SDK fornisce i Content Provider e gli Intent per consentire alle applicazioni di comunicare e scambiare dati.

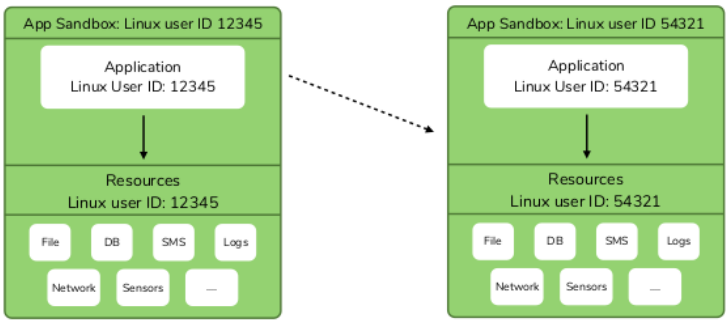
> Nota
> I Content Provider sono utilizzati per condividere dati tra applicazioni in modo sicuro e controllato. Un Content Provider espone un insieme di dati all'esterno e definisce le regole di accesso. Le altre applicazioni possono accedere a questi dati solo se il Content Provider ha reso disponibile l'accesso tramite autorizzazioni appropriate.

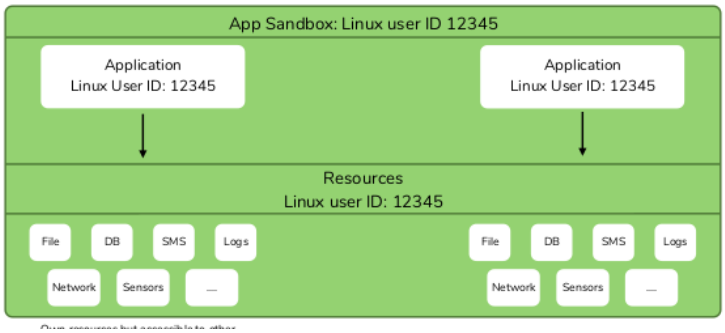
> Nota
> Gli Intent, sono utilizzati per avviare attività o servizi in altre applicazioni. Gli Intent possono anche essere utilizzati per trasmettere dati tra le applicazioni. Quando un'applicazione avvia un Intent per eseguire un'azione o avviare un'altra applicazione, può includere dati aggiuntivi nell'Intent stesso. Questi dati possono essere utilizzati dalla destinazione dell'Intent per svolgere un'azione specifica o visualizzare contenuti correlati.

Questa soluzione è stata adottata per garantire un'elevata sicurezza in un mondo di numerose piccole applicazioni di terze parti provenienti da diversi fornitori.

> Nota
> L'approccio che utilizza sandbox è una soluzione comune per i sistemi operativi mobili come iOS







Sicurezza e Permessi

La piattaforma Android adotta diverse misure di sicurezza e integrità per garantire che i dati dell'utente siano protetti e che il dispositivo non sia soggetto a malware, le applicazioni vengono trattate come utenti del sistema operativo (istanze di Dalvik e sandboxes).

Permessi delle applicazioni

Ogni applicazione si registra per ottenere specifici privilegi per accedere a risorse condivise, alcuni di questi privilegi sono correlati alle funzionalità del telefono, come effettuare chiamate, accedere alla rete, al file system, controllare la fotocamera o gli altri sensori hardware. Ulteriori autorizzazioni sono dedicate all'accesso a informazioni private e personali, come le preferenze, i contatti dell'utente o la posizione, per fare ciò è necessario il consenso dell'utente per ottenere queste autorizzazioni.

Limited Ad-Hoc Permissions

Le applicazioni che utilizzano un Content Provider per condividere apertamente informazioni specifiche con altre applicazioni possono definire delle autorizzazioni "on-the-fly" per concedere o revocare l'accesso a una risorsa.

Application Signing

Tutte le applicazioni Android vengono firmate con un certificato per permettere agli utenti di verificare l'autenticità dell'applicazione, il certificato di sviluppatore può essere utilizzato per sviluppare ma non per pubblicare le applicazioni sullo store.

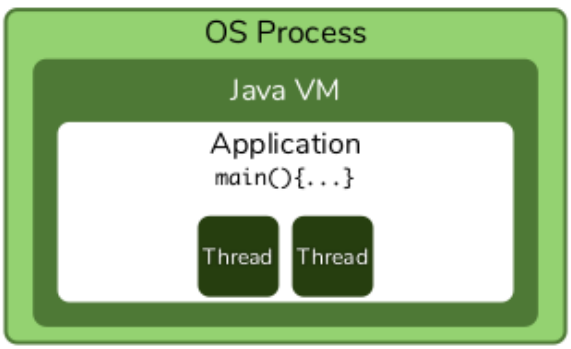
> Nota
> La "Firma delle Applicazioni" (Application Signing) è un processo critico nel ciclo di sviluppo delle applicazioni Android che garantisce l'autenticità e l'integrità dell'applicazione

---

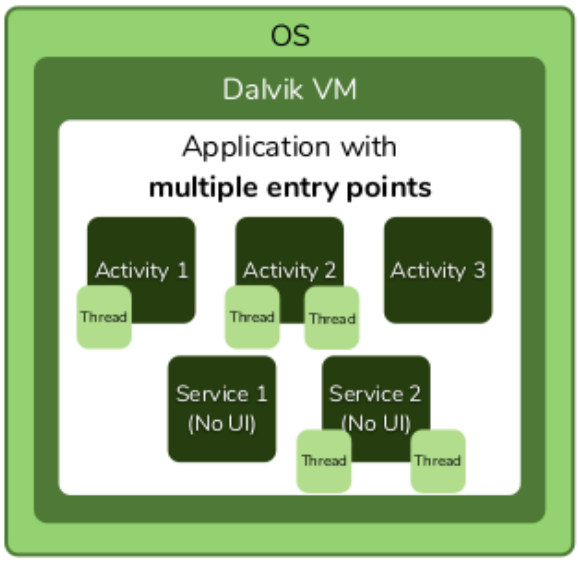
Programmazione tradizionale comparata ad Android

Le applicazioni SO tradizionali utilizzano un punto di ingresso singolo (`main`), il SO carica il programma in un processo e inizia la sua esecuzione.

Le applicazioni basate su Java invece sono gestite in maniera differente, una Java VM viene istanziata in un processo dedicato per caricare tutte le classi usate dall'applicazione e le esegue.



Android introduce un sistema più complesso che supporta più punti di ingresso per le applicazioni. Ogni componente rappresenta un punto diverso attraverso il quale il sistema può accedere all'applicazione. Non tutti i componenti sono veri e propri punti di ingresso per l'utente e alcuni dipendono tra loro, ma ciascuno esiste come entità propria e svolge un ruolo specifico. Ognuno di essi è un mattoncino unico che contribuisce a definire il comportamento complessivo dell'applicazione.



Ci sono quattro diversi tipi di componenti delle applicazioni. Ogni tipo ha uno scopo distinto e un ciclo di vita specifico che definisce come il componente viene creato e distrutto. I principali punti di ingresso dell'applicazione sono:

- Activities (attività)
- Services (servizi)
- Content Providers (contenuti)
- Broadcast Receivers (ricevitori di broadcast)

Activities

Un'attività (Activity) è sia un'unità di interazione con l'utente (tipicamente associata a una view), sia un'unità di esecuzione. Per creare un'attività in Android, è necessario creare una sottoclasse di `Activity` (o una sottoclasse di una sua estensione, ad esempio `MapActivity`)

> Nota
> La classe `Activity` è una classe base fornita dal framework Android ed è responsabile della gestione del ciclo di vita e dell'interazione dell'attività con l'utente

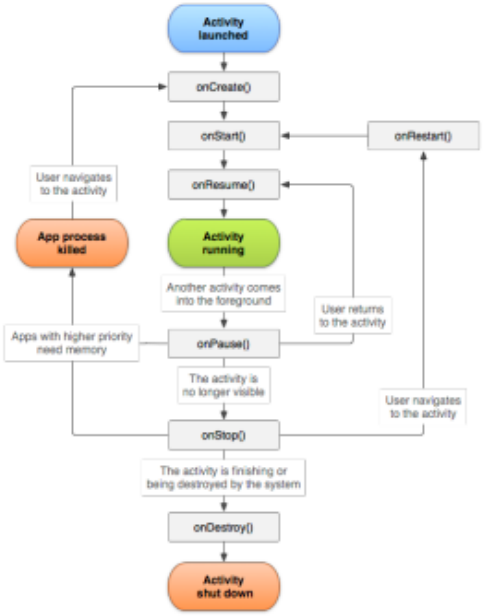
Le varie attività (comprese quelle appartenenti a diverse applicazioni) sono organizzate in una struttura a pila (`stack`) in cui nuovi elementi possono essere inseriti (`push`) o rimossi (`pop`) in seguito a specifiche azioni dell'utente, come interazioni con l'interfaccia utente mediante l'utilizzo del pulsante "Indietro" (`back button`).

Anche se lavorano insieme per formare un'esperienza utente coerente all'interno di un'applicazione, ognuna di esse è indipendente dalle altre. Di conseguenza, un'applicazione diversa può avviare qualunque di queste attività. Ad esempio, un'applicazione fotocamera può avviare l'attività dell'applicazione email che compone una nuova email, in modo che l'utente possa condividere una foto.

Il ciclo di vita delle attività

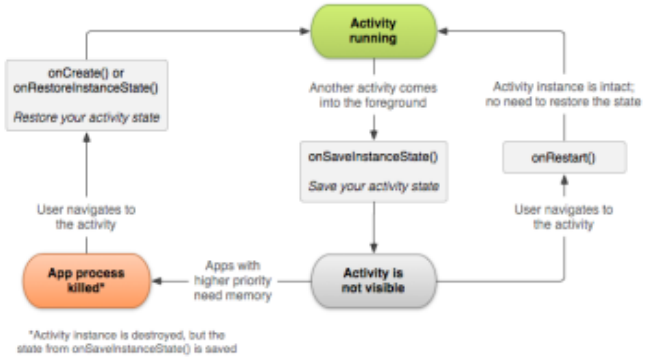
Metodo	Descrizione	Uccidibile	Chiamata successiva
`onCreate()`	Chiamato quando l'attività viene creata per la prima volta. Qui dovresti eseguire tutte le operazioni di inizializzazione statica: creare le viste, collegare dati alle liste, ecc.	No	`onStart()`
`onRestart()`	Chiamato dopo che l'attività è stata fermata, appena prima di essere riavviata.	No	`onStart()`
`onStart()`	Chiamato poco prima che l'attività diventi visibile per l'utente.	-	`onResume()` o `onStop()`
`onResume()`	Chiamato poco prima che l'attività inizi a interagire con l'utente. In questo momento, l'attività è in cima alla pila delle attività e riceve l'input dell'utente.	No	`onPause()`
`onPause()`	Chiamato quando il sistema sta per riprendere un'altra attività. Questo metodo è tipicamente utilizzato per salvare le modifiche non salvate ai dati persistenti, interrompere le animazioni e altre attività che possono consumare la CPU, ecc. Dovrebbe eseguire le sue operazioni molto velocemente, poiché la prossima attività non verrà ripresa finché non termina.	Sì	`onResume()` o `onStop()`

Metodo	Descrizione	Uccidibile	Chiamata successiva
`onStop()`	Chiamato quando l'attività non è più visibile all'utente. Questo può accadere perché sta per essere distrutta o perché un'altra attività (sia una già esistente o una nuova) è stata ripresa e la copre.	Sì	`onRestart()` o `onDestroy()`
`onDestroy()`	Chiamato prima che l'attività venga distrutta. Questa è l'ultima chiamata che l'attività riceverà. Può essere chiamato sia perché l'attività sta finendo, sia perché il sistema sta temporaneamente distruggendo questa istanza dell'attività per liberare spazio.	Sì	-



Salvare lo stato delle attività

Quando il sistema distrugge un'attività per recuperare memoria, l'oggetto dell'attività viene distrutto, quindi il sistema non può semplicemente riprenderla con il suo stato intatto. Se l'utente navigasse nuovamente sull'attività distrutta, il sistema deve ricreare l'oggetto dell'attività, però l'utente non è consapevole del fatto che il sistema abbia distrutto e ricreato l'attività, pertanto probabilmente si aspetta che l'attività sia esattamente com'era.
In questa situazione, il sistema operativo Android fornisce un metodo chiamato `onSaveInstanceState()` per assicurarsi che le informazioni importanti sullo stato dell'attività siano preservate.



Esempio creazione attività

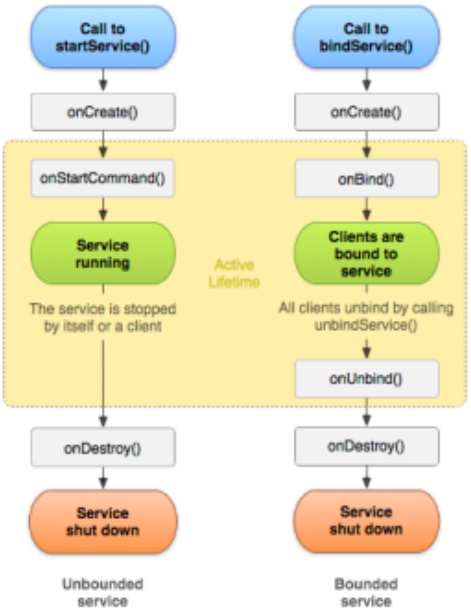
```
import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Definire il layout dell'attività tramite un file XML
        setContentView(R.layout.activity_my);

        // Altre operazioni di inizializzazione e configurazione dell'attività
    }
```

```
    // Altri metodi del ciclo di vita dell'attività e metodi personalizzati possono essere
definiti qui
}
```

Servizi

La classe `Service` di Android è utilizzata per operazioni in background che sono attive ma non visibili all'utente, i servizi vengono utilizzati per eseguire operazioni a lunga durata senza interagire con l'utente o per fornire funzionalità utilizzabili da altre applicazioni.
Un componente può collegarsi (`bind`) a un servizio per interagire con esso e eseguire una comunicazione interprocesso (IPC), ad esempio un servizio potrebbe gestire, in background, transazioni di rete, riprodurre musica, eseguire operazioni di I/O su file o interagire con un content provider.

Il ciclo di vita è più semplice rispetto a quello presentato per le attività, è gestito principalmente attraverso due metodi principali ovvero avvio (`start`), terminazione (`stop`) e riavvio (`restart`).



Esempio creazione servizio

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // Questo metodo deve essere implementato quando il servizio viene definito come
collegabile (bindable)
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("MyService", "Il servizio è stato creato.");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d("MyService", "Il servizio è stato avviato.");

        // Esempio: esegui un'operazione in background
        doBackgroundOperation();

        // Impostiamo il servizio come START_STICKY in modo che venga riavviato
automaticamente
        // se viene terminato dal sistema
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
```

```
        super.onDestroy();
        Log.d("MyService", "Il servizio è stato distrutto.");
    }

    private void doBackgroundOperation() {
        // Esempio di operazione in background (puoi sostituire questo con il tuo codice)
        for (int i = 1; i <= 10; i++) {
            Log.d("MyService", "Esecuzione dell'operazione in background: " + i);
            try {
                Thread.sleep(1000); // Simuliamo un'attesa di 1 secondo tra ogni iterazione
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

DIchiarazione nel file `AndroidManifest.xml`

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <application
        ...>

        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="false" />

    </application>

</manifest>
```

Avvio del servizio da un attività

```
Intent serviceIntent = new Intent(this, MyService.class);
startService(serviceIntent);
```

Provider di contenuti

I Content Provider (Provider di contenuti) gestiscono l'accesso a un insieme strutturato di dati, come file o database, incapsulano i dati e forniscono meccanismi per definire la loro sicurezza. Sono l'interfaccia standard che collega i dati in un processo con il codice in esecuzione in un altro processo.

> Nota
> Altre applicazioni possono accedere alle informazioni di un Content Provider tramite un URI specifico (`content://`)

Le operazioni tipiche di un content provider sono:

- Inserimento, `create`
- Lettura, `read`
- Aggiornamento, `update`
- Cancellazione, `delete`

> Nota
> Android OS fornisce alcuni Content Provider (Provider di contenuti) integrati per il Browser, il Calendario, i Contatti, il Registro delle chiamate, i Media e le Impostazioni, consentendo l'accesso a questi dati da altre applicazioni attive

Intents

Gli intents (intenti) sono messaggi asincroni che consentono ai componenti del sistema operativo Android di richiedere funzionalità da altri elementi del sistema, ad esempio un'Activity può inviare un intent al sistema Android per avviare un'altra Activity.

> Nota
> Possono essere utilizzati per segnalare al sistema che si è verificato un determinato evento, così altri componenti possono registrarsi per questo evento e riceveranno una notifica quando quest'ultimo si verifica

Vengono inviati al sistema, e a seconda di come l'intent è stato costruito, eseguirà una "reciver determination" e stabilirà cosa fare.

> Nota
> Un intent può anche contenere dati, che possono essere utilizzati dalla componente che riceve l'Intent, ad esempio, una applicazione può chiamare una componente del browser tramite un intent, l'URL può essere inviato come dati alla componente del browser

> Nota
> Android supporta intents impliciti ed espliciti

Ricevitori broadcast

I Broadcast Receiver sono il punto di ricezione degli intents, e rappresentano una variante di comunicazione tra diversi processi basata sugli oggetti di intents.
Sono considerabili un punto di ingresso per un applicazione, anche se non in esecuzione, poiché il suo codice può essere eseguito come risultato di un messaggio proveniente dal sistema operativo o un'altra applicazione. È destinato a svolgere un lavoro molto limitato come ad esempio eseguire alcune operazioni in base all'evento ricevuto

> Nota
> Molti broadcast hanno origine dal sistema (ad esempio, un broadcast che annuncia che lo schermo si è spento, la batteria è scarica o è stata scattata una foto)

Le applicazioni possono anche avviare broadcast, ad esempio per informare altre applicazioni che alcuni dati sono stati scaricati sul dispositivo e sono disponibili per l'uso.

> Nota
> I Broadcast Receiver non mostrano un'interfaccia utente, ma possono creare una notifica sulla barra di stato per avvisare l'utente quando si verifica un evento di broadcast

---

Android Application Context

Il contesto di un applicazione è un entità centrale accessibile da tutte le principali applicazioni (top apps) e contiene informazioni globali sull'applicazione e l'ambiente. È una classe astratta fornita dal Android OS che permette di accedere a risorse e classi specifiche dell'applicazione, nonché a richieste di operazioni a livello di applicazione, come l'avvio di attività, l'invio e la ricezione di intents.

I metodi nel contesto possono essere chiamati direttamente dalle classi `Activity` e `Service` perché estendono direttamente la classe `Context`

Alcuni esempi di funzionalità sono:

- Recuperare risorse dell'applicazione, come immagini e altre risorse
- Accedere alle preferenze dell'applicazione
- Avviare attività
- Richiedere un servizio di sistema
- Gestire i file, le directory e i database privati dell'applicazione
- Ispezionare e applicare le autorizzazioni dell'applicazione

---

Android Application Manifest

Il manifesto delle applicazioni android descrive il loro contenuto e si tratta di un file `XML` chiamato `AndroidManifest.xml`. Utilizzandolo lo sviluppatore può dichiarare la presenza di attività, servizi, provider di contenuti, ricevitori di broadcast, permessi e altri elementi come la versione del codice, il nome dell'applicazione e i requisiti SDK.

> Nota
> I file del manifesto sono condivisi con il sistema operativo per caricare ed eseguirli in un ambiente gestito

Esempio

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
                        package="it.unipr.ce.dgt.android"
                        android:versionCode="1"
                        android:versionName="1.0">
        <uses-sdk android:minSdkVersion="7" />

        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

```
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.GPS" />
        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

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

                <uses-library android:name="com.google.android.maps" />
                <activity android:name=".WarningMessageActivity"></activity>
                <activity android:name=".IncomingWarningMessageActivity"></activity>
                <activity android:name=".PeerInfoActivity"></activity>

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

R object

Le risorse di un'applicazione vengono indicizzate e assegnate un identificatore univoco tramite il quale possono essere accedute, l'IDE e la SDK lavorano insieme per creare la cartella `gen` che contiene una classe specifica chiamata R, la quale si trova all'interno del pacchetto dell'applicazione Java indicato nel Manifest. La class R contiene campi che identificano univocamente tutte le risorse nella struttura del pacchetto dell'applicazione

> Nota
> Tutte le risorse dell'applicazione non possono essere accedute direttamente tramite il loro nome di file, ma attraverso l'oggetto R

Utilizzando l'oggetto `Context` (`Context.getResources`), è possibile ottenere un'istanza di `android.content.res.Resources` che contiene direttamente le risorse dell'applicazione.

> Nota
> L'oggetto R viene anche utilizzato per recuperare i file di layout dell'interfaccia utente e gli elementi correlati

Esempio

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
*
* This class was automatically generated by the
* aapt tool from the resource data it found. It
* should not be modified by hand.
*/
package it.unipr.ce.dgt.android;

public final class R {
        public static final class attr {
        }
        public static final class drawable {
                public static final int accident_marker=0x7f020000;
                // ...
        }
        public static final class id {
                public static final int accidentButton=0x7f050018;
                // ...
        }
        public static final class layout {
                public static final int alert_message_dialog=0x7f030000;
                // ...
        }
        public static final class string {
                public static final int app_name=0x7f040001;
                // ....
        }
}
```

Application Logs

Il framework Android fornisce un modo strutturato per consentire allo sviluppatore di registrare eventi correlati alla sua applicazione utilizzando la classe chiamata `Log`, che fornisce cinque metodi di log statici con la stessa struttura

```
// VERBOSE
Log.v(TAG, <log>);

// DEBUG
Log.d(TAG, <log>);

// INFO
Log.i(TAG, <log>);

// WARNING
Log.w(TAG, <log>);

// ERROR
Log.e(TAG, <log>);
```

> Nota
> L'ordine in termini di verbosità, dal meno al più verboso, è "ERROR", "WARNING", "INFO", "DEBUG", "VERBOSE".
> I log di livello VERBOSE non dovrebbe mai essere incluso nell'applicazione durante la fase di compilazione finale o in produzione, ma dovrebbe essere utilizzato solo durante lo sviluppo per ottenere informazioni dettagliate sul flusso dell'esecuzione e il comportamento dell'applicazione.
> I log di livello DEBUG sono inclusi nella compilazione dell'applicazione, ma vengono rimossi a runtime quando l'applicazione viene eseguita in un ambiente di produzione.
> I log di livello ERROR, WARN e INFO sono sempre mantenuti nell'applicazione a runtime, indipendentemente dal fatto che l'applicazione sia in fase di sviluppo o in produzione.

Buona pratica

Un approccio buono e consigliato è di dichiarare una costante `TAG` nella classe e utilizzarla nelle chiamate di log

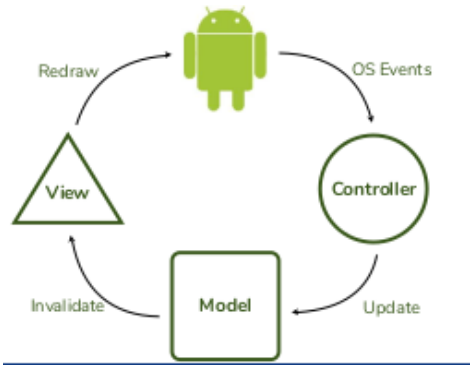
```
public class MyClass{

    private static final String TAG = "MyClass";

    // ...

    // Esempi di log usando la costante TAG
    Log.d(TAG, "Questo è un messaggio di log di livello DEBUG");
    Log.e(TAG, "Questo è un messaggio di log di livello ERROR");
    Log.i(TAG, "Questo è un messaggio di log di livello INFO");
    // ...
}
```

Model View Controller

Il Model-View-Controller (MVC) è un pattern architetturale software utilizzato per isolare l'interfaccia utente (input e presentazione) dalla "logica" di un'applicazione (la logica dell'applicazione per l'utente), il suo utilizzo consente lo sviluppo, il testing e la manutenzione indipendenti delle diverse componenti.

> Nota
> Il framework UI di Android, come altri framework Java, è organizzato attorno a un modello MVC comune

Fornisce struttura e strumenti per costruire un controller che gestisce l'input dell'utente (tasti premuti e tocchi dello schermo) e una view che visualizza informazioni grafiche sullo schermo.



Model

Il modello gestisce il comportamento e i dati dell'applicazione, risponde alle richieste di informazioni sul suo stato (di solito dalla vista) e risponde alle istruzioni per cambiare stato (di solito dal controller). Nei sistemi event-driven, il modello notifica gli osservatori (di solito le viste) quando le informazioni cambiano in modo che possano reagire.

Il modello riguarda tutti i dati dell'applicazione, memorizzati in un database, un file o una struttura dati generica, fornisce tutti i metodi necessari per accederci.

Invece la view ed il controller rifletteranno necessariamente il modello che manipolano, un singolo modello potrebbe essere utilizzato da diverse applicazioni, alcuni esempi semplici potrebbero essere la lista degli indirizzi dei contatti o il database della musica sul dispositivo.

> Nota
> Applicazioni come un lettoreo un convertitore MP3 hanno funzionalità diverse, ma si basano entrambe sullo stesso modello legato al formato del file MP3

View

La view è la visualizzazione del modello, in generale è un componente dell'applicazione responsabile della visualizzazione dell'interfaccia utente, dell'invio di audio allo speaker, della generazione di feedback tattili e così via.
Nel framework Android, come in altre piattaforme, il framework organizza e rappresenta le viste in una gerarchia (un albero), chiedendo a ciascun componente di disegnarsi mediante un attraversamento in preordine (ogni vista si disegna da sola e quindi chiede a ciascuno dei suoi figli di fare lo stesso).

Un oggetto `view` è una struttura dati le cui proprietà memorizzano i parametri di layout e il contenuto per una specifica area rettangolare dello schermo, gestisce la propria misurazione, layout, disegno, cambio di focus, scrolling e interazioni con tasti/gestures per l'area rettangolare dello schermo in cui risiede.

> Nota
> Come oggetto nell'interfaccia utente, una view è anche un punto di interazione per l'utente e il destinatario degli eventi di interazione

---

Controller

Il Controller è la parte dell'applicazione che risponde agli eventi esterni: tocco dello schermo, pressione di tasti, chiamate in arrivo, ecc.

È implementato come una coda di eventi, ogni azione esterna è rappresentata come un evento univoco nella coda, il framework rimuove ogni evento dalla coda e lo smista chiamando il metodo handler corretto della vista correlata all'evento.

Android User Interface

L'interfaccia utente delle applicazioni Android è costruita utilizzando oggetti view e viewgroup.

Il framework fornisce diversi tipi di view e view group, ognuno dei quali è un discendente della classe `View` che è la base per le sottoclassi chiamate widget, che offrono oggetti UI completamente implementati, come campi di testo e pulsanti.

La classe `ViewGroup` funge da base per le sottoclassi chiamate layout, che offrono diversi tipi di architetture di layout, come lineari, tabellari e relativi.

> Nota
> Come oggetto nell'interfaccia utente, una View è anche un punto di interazione per l'utente e il ricevitore degli eventi di interazione

---

View Hierarchy

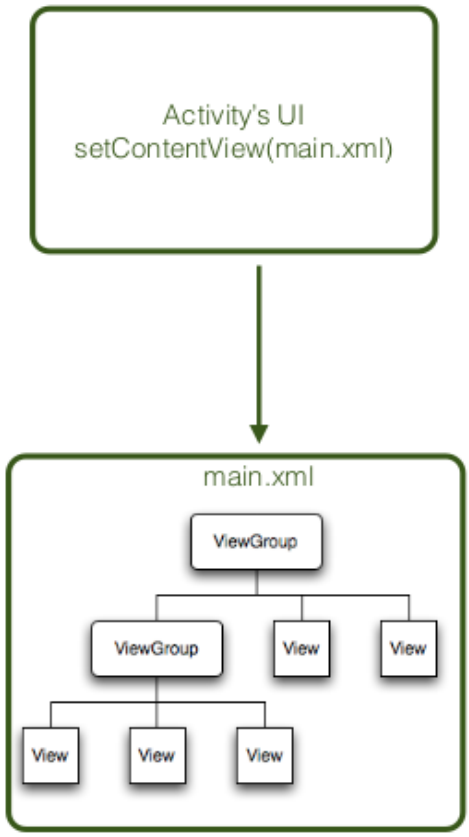
Sulla piattaforma Android, si definisce l'interfaccia utente di un'activity utilizzando una gerarchia di nodi `View` e `ViewGroup`. Questa gerarchia può essere semplice o complessa a seconda delle necessità, e può essere costruita utilizzando il set di widget e layout predefiniti, oppure con view personalizzate create dall'utente.

> Note
> Gli oggetti `View` sono foglie dell'albero, mentre gli oggetti `ViewGroup` sono i nodi interni dell'alber.

Per collegare l'albero della gerarchia di view allo schermo per il rendering, l'activity deve chiamare il metodo `setContentView()` e passare il riferimento alla radice dell'albero (layout XML principale).

Come detto in precedenza, il sistema riceve questo riferimento e lo utilizza per invalidare, misurare e disegnare l'albero. La radice della gerarchia richiede ai suoi nodi figli di disegnarsi, a loro volta, ciascun nodo di view group è responsabile di chiamare ognuna delle sue view figlie per disegnarsi.

> Note
> Dato che le viste vengono disegnate in ordine, se ci sono elementi che si sovrappongono in posizione, l'ultimo disegnato si sovrapporrà agli altri disegnati precedentemente nello stesso spazio.



Layout

Il modo più comune per definire il layout ed esprimere la vista gerarchica è un file di layout XML, il quale offre una struttura per il layout, leggibile dall'umano, tipo HTML. Ogni elemento nell'XML è un oggetto `View` o `ViewGroup` (o suo discendente).

Esistono diverse modalità per disporre le viste. Utilizzando molti e differenti tipi di view group, è possibile strutturare le view e i view group in un numero infinito di modi. Alcuni view group predefiniti offerti da Android (chiamati layout) includono `LinearLayout`, `RelativeLayout`, `TableLayout`, `GridLayout` e altri.
Ognuno offre un set unico di parametri di layout che vengono utilizzati per definire le posizioni delle view figlie e la struttura del layout.

Esempio

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        <TextView android:id="@+id/text"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
        <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

Layout Attributes

Tutti gli oggetti `View` e `ViewGroup` supportano una varietà di attributi XML, alcuni di questi sono specifici di un oggetto `View`, ad esempio `TextView` supporta l'attributo `textSize`.

> Nota
> Questi attributi sono ereditati da eventuali oggetti View che possono estendere questa classe

Alcuni attributi invece sono comuni a tutti gli oggetti `View`, perché sono ereditati dalla classe `View` radice (come l'attributo *id*), altri sono considerati "parametri di layout", che sono attributi che descrivono determinate disposizioni del layout dell'oggetto View, come definito dall'oggetto ViewGroup genitore.

> Nota
> Gli attributi di layout possono essere modificati direttamente nel file XML o utilizzando l'apposita API del framework quando il riferimento all'oggetto View è stato recuperato nel codice

Layout ID Attribute

Qualsiasi oggetto View può avere associato un ID intero per identificarlo univocamente all'interno dell'albero.

> Nota
> Quando l'applicazione viene compilata, questo ID è referenziato come un intero, ma di solito l'ID viene assegnato nel file XML di layout come una stringa, nell'attributo id

Il simbolo `@` all'inizio della stringa indica che il parser XML dovrebbe analizzare ed espandere il resto della stringa ID e identificarla come una risorsa ID, il simbolo `+` significa che questo è un nuovo nome di risorsa che deve essere creato e aggiunto alle nostre risorse (nel file R.java)

Quando la risorsa viene aggiunta all'oggetto R, può essere utilizzata per recuperare l'elemento view in un'activity utilizzando il metodo `context.findViewById(R.<resource_id>)`.

Esempio

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

> Nota
> Il testo definito da `android:text= ...` può essere una stringa o un riferimento a una risorsa stringa come segue

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="my_button_text">Hello!</string>
</resources>
```

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Widgets

Un widget è un oggetto `View` che serve come interfaccia per l'interazione con l'utente. Android ne fornisce un insieme completamente implementati:

- Pulsanti
- Caselle di controllo

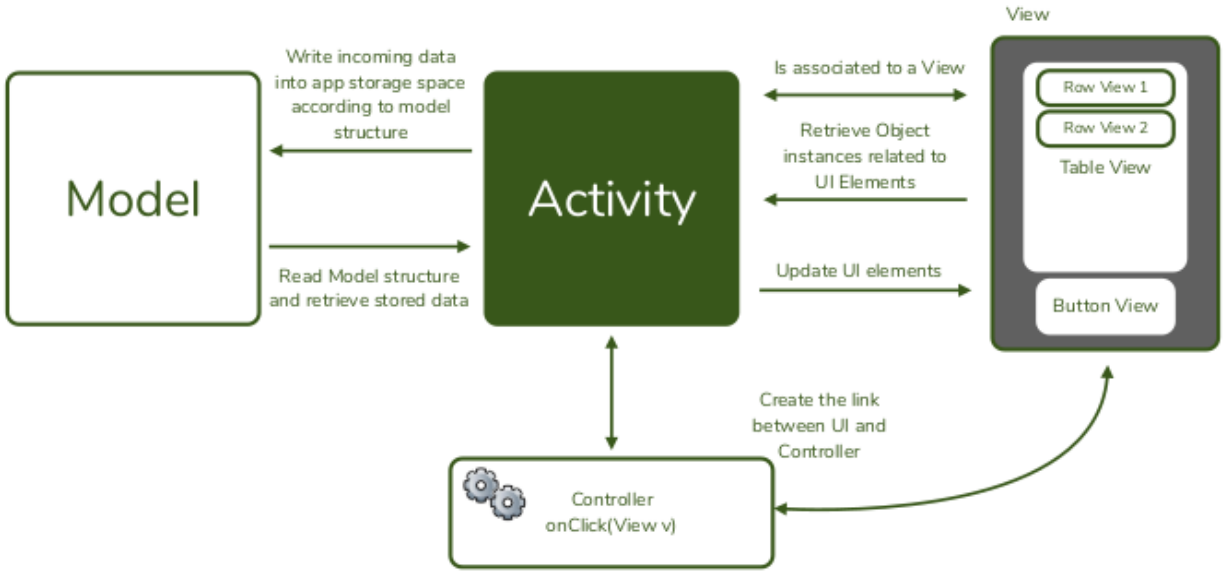
- Campi di inserimento testo

Alcuni sono anche abbastanza complessi:

- Selettore di date
- Orologio
- Controlli di zoom

C'è anche la possibilità di fare qualcosa di più personalizzato creando elementi azionabili definiti dall'utente. Lo sviluppatore può definire i propri oggetti `View` o estendere e combinare i widget esistenti. Esempi di componenti personalizzate sono pulsanti personalizzati definiti dall'utente, barre di avanzamento personalizzate, visualizzazioni di grafici, marcatori sulla mappa, ecc...

Il ruolo chiave delle Activities in Android



Activity

Come detto in precedenza un activity p sia un'unità di interazione con l'utente (tipicamente associata ad una vista), sia un unita di esecuzione

All'interno del metodo di override `onCreate(...),` lo sviluppatore può impostare la view del layout associata all'activity chiamando `setContentView(R.layout.<layout_file>)` utilizzando l'oggetto `R` condiviso per specificare il file delle risorse contenente il layout, quando la view radice è stata impostata, è possibile recuperare gli oggetti associati agli elementi dell'interfaccia utente tramite il metodo `findViewById(R.id.<resource_id>)`.

> Nota
> Un oggetto view come `TextView` può essere utilizzato per aggiornare l'interfaccia utente, per ottenere valori dai campi di input o per impostare listener di eventi utilizzando i metodi `get` e `set` forniti.

Esempio

Vista

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
        apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        <TextView
        android:id="@+id/mainTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

Controller

```
public class HelloWorldActivity extends Activity {
        /** Called when the activity is first created. */
```

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); // Collegamento tra il Layout (View) e l'Activity
(Controller)
        }
}
```

Accesso e modifica degli elementi dell'interfaccia utente

```
TextView mainTextView = (TextView)findViewById(R.id.mainTextView);
mainTextView.setText("Hello World ! :D");
```

Input events

View/widget e in generale l'interfaccia utente non vengono utilizzati solo per mostrare contenuti e informazioni, ma hanno il ruolo importante di consentire all'utente di interagire con l'applicazione eseguendo azioni. Per essere informato degli eventi di input dell'utente, lo sviluppatore deve fare una delle due cose:

1. Definire un listener di eventi e registrarli con la view. La classe `View` contiene una collezione di interfacce nidificate chiamate `OnSomeEventListener`, ognuna con un metodo di callback chiamato `onSomeEvent().`
   - Ad esempio, `View.OnClickListener`(per gestire i "click" su una view), `View.OnTouchListener` (per gestire gli eventi del touch screen in una view) e `View.OnKeyListener` (per gestire le pressioni dei tasti del dispositivo all'interno di una view)
   - Per essere notificati quando la view viene "cliccata" (ad esempio quando viene selezionato un pulsante), implementare `OnClickListener` e definire il suo metodo di callback `onClick()` (dove viene eseguita l'azione al clic), e registrarlo alla view con `setOnClickListener()`
2. Sovrascrivere un metodo di callback esistente per la view. Questo è ciò che si dovrebbe fare quando si è implementata la propria classe `View` e si desidera ascoltare specifici eventi che si verificano al suo interno

Esempio

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
        public void onClick(View v) {
        // do something when the button is clicked
        }
};

protected void onCreate(Bundle savedValues) {
        // Capture our button from layout
        Button button = (Button)findViewById(R.id.corky);
        // Register the onClick listener with the implementation above
        button.setOnClickListener(mCorkyListener);
}
```

Button Events

Android si basa massicciamente (`UIListener` o gestione della posizione) sul principio/design pattern del callback, che sono utili per gestire eventi che possono verificarsi in un momento non specificato (eventi asincroni).

Un metodo di callback è un metodo che viene passato come argomento di un altro metodo per essere successivamente invocato.

> Nota
> In Java i metodi callback sono implementati mediante interfacce

Esempio

```
public class MyActivity extends Activity {
        protected void onCreate(Bundle icicle) {

                super.onCreate(icicle);
                setContentView(R.layout.content_layout_id);
                final Button button = (Button) findViewById(R.id.button_id);

                button.setOnClickListener(new View.OnClickListener() {
                        public void onClick(View v) {
                                // Perform action on click
                        }
                });
        }
}
```

Launch/Finish Activity

Il metodo `startActivity(Intent)` viene utilizzato per avviare una nuova attività, che verrà posizionata in cima allo `stack` delle attività, prende un singolo argomento, un intent, che descrive l'attività da eseguire

```
startActivity(new Intent(getApplicationContext(), MyActivity.class));
```

A volte si desidera ottenere un risultato dall'attività quando essa termina, ad esempio, si può avviare un'attività che consente all'utente di selezionare una persona da un elenco di contatti, quando l'attività termina, restituisce la persona selezionata. Per fare ciò, si chiama la versione di `startActivityForResult(Intent, int)` con un secondo parametro intero per identificare la chiamata, il risultato verrà restituito attraverso il metodo `onActivityResult(int, int, Intent)`.

Quando un'attività esce, può chiamare `setResult(int)` per restituire i dati al suo genitore.

> Nota
> Deve sempre fornire un codice di risultato, che può essere uno dei risultati standard `RESULT_CANCELED`, `RESULT_OK`, o qualsiasi valore personalizzato a partire da `RESULT_FIRST_USER`.

Inoltre si può restituire un intent contenente eventuali dati aggiuntivi, tutte queste informazioni saranno disponibili nel metodo `Activity.onActivityResult()` del genitore, insieme all'identificatore intero fornito originariamente.

Quando l'attività ha terminato il suo scopo, lo sviluppatore può chiamare il metodo `finish()` per chiuderla, il risultato dell'attività viene restituito a chiunque l'abbia avviata tramite `onActivityResult()`.