

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
C.d.S. in Ingegneria e Scienze Informatiche, Campus di Cesena

Programmazione in Android

GUI, Risorse e Shared Preferences

Angelo Croatti
a.croatti@unibo.it

Sistemi Embedded e Internet of Things
A.A. 2019 – 2020

Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse
- 5 Shared Preferences



Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse
- 5 Shared Preferences



Approccio alle GUI in Android

- In linea teorica, una GUI per una Activity potrebbe essere descritta e creata interamente per via programmatica (ovvero, direttamente nel codice sorgente).
 - ▶ Similmente a quanto si può fare in Java.
 - ▶ Si creano le istanze degli oggetti dei componenti grafici (es. JButton), li si associano ad un layout (es. JPanel) e si aggancia il layout ad una finestra (es. JFrame).
- In Android, tuttavia, si preferisce descrivere (e gestire) in altro modo le interfacce grafiche.
 - ▶ Definendo una GUI, e tutti i suoi componenti, secondo la metafora delle **risorse**.
 - ▶ La descrizione delle interfacce passa per la scrittura di **file XML**.
 - ▶ Da Android 5.0, si dovrebbe seguire la *filosofia* del **Material Design**.

Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse
- 5 Shared Preferences



Definizione (informale) di Risorsa

- Il solo codice sorgente non è sufficiente per scrivere una buona (ben progettata) applicazione Android.
- **Le risorse sono file aggiuntivi per l'applicazione**
 1. scritti in linguaggio XML,
 2. con un contenuto statico,
 3. ai quali è possibile far riferimento dal codice sorgente,
 4. che consentono di descrivere una serie di elementi e caratteristiche dell'applicazione.
- Si possono creare risorse per: **layout** e componenti delle interfacce grafiche (**drawable**), **menu**, testo per la localizzazione e dimensioni (**values**),...

Principio di indipendenza delle risorse

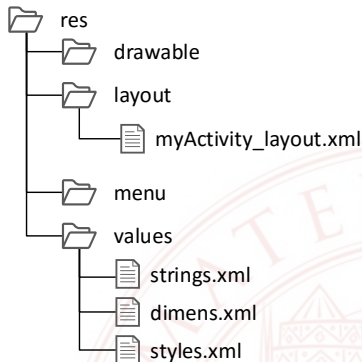
- **Definire alcuni elementi come risorse significa, soprattutto, mantenerli indipendenti dalla logica dell'applicazione.**
 - ▶ Le risorse possono essere scelte (mediante identificatori) e “attivate” a runtime.
- ES1. *Se la GUI di una Activity è definita sotto forma di risorsa, potranno essere definite più risorse per la stessa GUI e potrà quindi essere scelta quella più opportuna in funzione delle dimensioni dello schermo del device sul quale l'applicazione sarà seguita.*
- ES2. *Definire tutti i messaggi e i riferimenti testuali di un'applicazione in modo separato (tramite risorse) consente di agevolare la localizzazione dell'applicazione (ovvero, il passaggio tra una lingua ed un'altra).*

Il file R.java

- L'Android Development Kit genera automaticamente un file (**R.java**) con tutti i riferimenti alle risorse definite nel progetto.
- In questo modo, è possibile accedere alle risorse con un nome logico.
 - ▶ I nomi logici sono definiti nel file R come gerarchia di classi e sono trattati come campi statici di tipo Integer.
- *Esempio.* Durante la creazione di una activity deve essere passato al metodo `setContentView()` il riferimento alla risorsa che ne definisce il layout.
 - ▶ `setContentView(R.layout.myactivity_layout);`
 - ▶ Significa che esiste la risorsa `myactivity_layout` descritta in un file XML omonimo che definisce un GUI layout.

Resource Tree di un'applicazione

- Nella home directory del progetto è presente una directory **res** per le risorse, organizzata in sub-directory.
- Possono essere aggiunte altre directory secondarie per contenere risorse specifiche.



Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse
- 5 Shared Preferences



Android User Interface

- Rappresenta tutto ciò che l'utente può vedere e con cui può interagire direttamente.
 - ▶ Generalmente una UI è associata al componente Activity.
- Esistono diversi componenti predefiniti per creare ed organizzare il layout di un'applicazione.
 - ▶ A ciascun componente sono associati uno o più eventi, gestibili attraverso opportuni listener.
 - ▶ Cambiano i nomi, ma il meccanismo di gestione delle GUI è (dal punto di vista logico) lo stesso di quello adottato in Swing in Java.

Android UI – Alcuni Componenti

- Componenti per il Layout
 - ▶ LinearLayout, RelativeLayout, FrameLayout, TableLayout,...
- Componenti per l'input
 - ▶ Button, TextField, CheckBox, RadioButton, Spinner, ToggleButton,...
- Componenti per l'output
 - ▶ TextView, ListView, GridView, ImageView, SurfaceView,...
- ...

» <http://developer.android.com/guide/topics/ui/index.html>

Definizione di una GUI – Esempio

myactivity_layout.xml

```
<?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/mytextview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView"/>

    <Button
        android:id="@+id/mybutton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button"/>

</LinearLayout>
```

Definizione di una GUI – Esempio (note) I

- È stata definita una risorsa denominata `myactivity_layout` nell'omonimo file XML.
 - ▶ Automaticamente aggiunta al file R mediante un campo statico omonimo nella sottoclasse `layout`.
- La GUI è strutturata mediante un linear layout principale, di tipo verticale (i cui componenti sono allineati uno sotto l'altro).
 - ▶ Nota. Il componente più esterno di qualunque layout deve definire la proprietà che qualifica il file come risorsa Android.
`xmlns:android="http://schemas.android.com/apk/res/android"`
- Nel linear layout principale sono stati definiti due componenti: una textview e un button.

Definizione di una GUI – Esempio (note) II

- Sia per il layout sia per i componenti sono stati specificati alcuni valori per alcune proprietà dei componenti stessi.
- A ciascun componente è stato associato un nome logico che sarà quello reperibile nel file R.
 - ▶ Es. Per il bottone, alla proprietà `android:id` è stato assegnato il valore `@+id/mybutton`
 - ▶ In questo modo si dichiara di voler aggiungere al file R, un campo statico per la sottoclasse `id` denominato `mybutton`.

Gestione degli elementi della GUI I

- Come visto in fase di descrizione del componente Activity, l'associazione di una GUI (un layout) deve essere fatto in fase di creazione dell'Activity.
 - ▶ **Facendo riferimento all'ID della risorsa layout che si vuole associare.**

Esempio – Associazione di una GUI ad una Activity

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

    /* ... */
}
```


Gestione degli elementi della GUI II

- Il riferimento agli elementi della GUI può essere richiesto in qualunque punto dell'activity attraverso il metodo `findViewById(int id)` fornito dalla classe `Activity`.
 - ▶ Dato l'identificativo del componente, restituisce la relativa istanza, di tipo generico `View` che deve essere opportunamente convertita al tipo specifico del componente.

Esempio – Accesso ad elementi della GUI

```
Button btn = (Button) findViewById(R.id.mybutton);  
TextView txt = (TextView) findViewById(R.id.mytextview);
```

- Ottenuta l'istanza dell'oggetto associato al componente grafico, è possibile gestirne proprietà ed eventi.
 - ▶ Ad esempio, associare listener a specifici eventi oppure ottenere/modificare i valori delle singole proprietà.

Gestione degli elementi della GUI III

Esempio – Associazione di un Listener al click del bottone

```
Button btn = (Button) findViewById(R.id.mybutton);  
btn.setOnClickListener(new OnClickListener(){  
    @Override  
    public void onClick(View v) {  
        //do something  
    }  
});
```

Esempio – Modifica del testo della TextView

```
TextView txt = (TextView) findViewById(R.id.mytextview);  
txt.setText("nuovo testo");
```

- Recuperata l'istanza dell'oggetto del componente d'interesse, la gestione di tale elemento può essere fatta analogamente a quanto si farebbe in Java.

Dialogs

- Un **Dialog** rappresenta una popup utilizzato generalmente per comunicare messaggi all'utente, per proporre una scelta e/o richiedere una conferma
- In Android esistono diversi tipi di dialogs:
 - ▶ **AlertDialog**, **DialogFragment**, **ProgressDialog**, ...
- Ciascuno può essere personalizzato sia nel contenuto che nel layout (attraverso una risorsa definita in XML)

» <https://developer.android.com/guide/topics/ui/dialogs.html>

Alert Dialog I

- Si tratta dell'alert più semplice e immediato disponibile in Android (è rappresentato dal tipo `AlertDialog`)
- Può essere costruito mediante il pattern Builder (attraverso un oggetto di tipo `AlertDialog.Builder`)
 - ▶ Il Builder deve essere istanziato con il riferimento all'activity che dovrà visualizzare il dialog
 - ▶ A tale scopo, può essere utilizzato il metodo della classe `Activity` `getActivity()` che restituisce il riferimento all'activity corrente
 - ▶ L'ultimo metodo da chiamare sul Builder è il metodo `create()` che restituisce l'istanza dell'AlertDialog

Alert Dialog II

Esempio

```
AlertDialog dialog = new AlertDialog.Builder(getActivity())
    .setTitle("File unsaved...")
    .setMessage("Do you want to save this file?")
    .setCancelable(false)
    .setPositiveButton("YES", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            // save file
        }
    })
    .setNeutralButton("Undo", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) { /*...*/ }
    })
    .setNegativeButton("NO", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) { /*...*/ }
    })
    .create();
```

Notifiche

- Una notifica è un messaggio che può essere proposto all'utente da un applicazione al di fuori del layout dell'applicazione stessa
 - ▶ Appare come elemento nella barra di notifica del device
- In Android, esistono due macro-tipi per le notifiche:
 - ▶ Notifiche Standard
 - ▶ Notifiche Toast

» developer.android.com/guide/topics/ui/notifiers/notifications.html

» developer.android.com/guide/topics/ui/notifiers/toasts.html

Notifiche Tradizionali I

- Anche per le notifiche, il meccanismo di creazione si avvale del pattern builder
- Gli eventi relativi alle notifiche sono gestiti tramite intent
 - ▶ In particolare, tramite una specializzazione della classe `Intent` denominata `PendingIntent`
- Le notifiche possono essere visualizzate avvalendosi dell'istanza relativa al `NotificationManager`
 - ▶ Istanza sulla quale è possibile richiamare il metodo `notify(int id, Notification n)` per inviare la notifica
 - ▶ L'istanza del Notification Manager è ottenibile attraverso l'API di sistema `getSystemService()` richiamata con il parametro `Context.NOTIFICATION_SERVICE`

Notifiche Tradizionali II

Esempio – Creazione e visualizzazione di Notifiche

```
final int NOTIFICATION_ID = 1234;

PendingIntent tapPendingIntent = PendingIntent.getActivity(this, 0,
    new Intent(this, MyActivity.class),
    PendingIntent.FLAG_UPDATE_CURRENT);

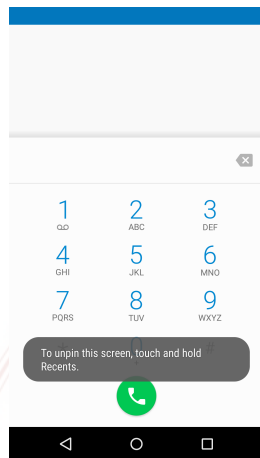
Notification notification = new NotificationCompat.Builder(this)
    .setContentTitle("File Downloaded")
    .setContentText("The requested file has been downloaded")
    .setAutoCancel(true)
    .setContentIntent(tapPendingIntent)
    .build();

NotificationManager notificationManager = (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);

notificationManager.notify(NOTIFICATION_ID, notification);
```


Notifiche Toast I

- Rappresentano una forma di comunicazione diretta limitata nel tempo con l'utente dell'applicazione.
 - ▶ Una sorta di Pop-up che appare in foreground sul display per un tempo predefinito occupando solo lo spazio richiesto per il testo della notifica.
 - ▶ Richiamabili anche da processi in background (es. Service, AsyncTask, ...).
- Implementate in Android mediante istanze di oggetti di tipo `android.widget.Toast`.
- Trattandosi di componenti della UI, tali notifiche devono essere create ed eseguite nell'ambito del Main Thread.



Notifiche Toast II

Esempio – Creazione e visualizzazione di Toast

```
CharSequence text = "Hello toast!";

Toast toast = Toast.makeText(getApplicationContext(), text, Toast.
    LENGTH_SHORT);
toast.show();
```

- La durata della notifica può essere specificata mediante `Toast.LENGTH_SHORT` oppure `Toast.LENGTH_LONG`.
- L'istanza dell'oggetto di tipo `Toast` propone alcune funzionalità con cui personalizzare la notifica.

Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse**
- 5 Shared Preferences



Stringhe come risorse

- In Android è best practice non inserire mai messaggi testuali (stringhe di testo) direttamente nel codice sorgente o come valore per le proprietà dei componenti della GUI
 - ▶ Per separare i contenuti
 - ▶ Per favorire la localizzazione del testo delle applicazioni nelle diverse lingue
- Specificando l'impostazione per la lingua, viene caricato automaticamente il file di risorse per il testo relativo (se disponibile)
 - ▶ Nel codice sorgente si farà riferimento alla risorsa specifica per ciascun messaggio e non al valore specifico.

File strings.xml

- Contiene tutte le risorse di tipo **string** per l'applicazione.
- Deve essere inserito nella directory **values** interna alla directory **res** del progetto dell'applicazione.
 - ▶ Eventuali altre versioni dello stesso file (con lo stesso nome) dovranno essere contenute in directory allo stesso livello di **values** con nomi quali **values-it**, **values-fr**,...

Esempio di file strings.xml

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

    <string name="appName">MyFirstApp</string>
    <string name="btn01text">Click me!</string>
    <string name="settingsMenuLabel">Settings</string>
    <string name="myActivityTitle">MyActivity</string>

</resources>
```

Uso delle risorse per il testo

- Dalle altre risorse (es. layout) le risorse di tipo string possono essere richiamate facendo diretto riferimento al nome della risorsa con il prefisso `@string`

Uso di risorse testuali in altre risorse

```
<Button android:id="@+id/mybutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/btn01text"/>
```

- Dal codice sorgente è possibile ricorrere al metodo `getString(int id)` fornito dalla classe Context.

Uso di risorse testuali nel codice sorgente

```
String s = getString(R.string.myActivityTitle);
```

Outline

- 1 Introduzione alle GUI
- 2 Risorse Android
 - Il file R.java
 - Resource Tree
- 3 Android GUI e Layout
 - Gerarchia dei Componenti
 - Gestione degli Elementi della GUI
 - Dialogs e Notifiche
- 4 Altri tipi di Risorse
- 5 Shared Preferences**



Shared Preferences

- Rappresentano il meccanismo più naïve per salvare informazioni (dati) persistenti in un'applicazione Android
 - ▶ Persistenti finché l'app non viene reinstallata nel sistema
- Si tratta di dati condivisi solo dai componenti di una stessa applicazione oppure accessibili da tutto il sistema, secondo il livello di accesso scelto
 - ▶ `Context.MODE_PRIVATE` – lei dati potranno essere lette solo dall'applicazione corrente
 - ▶ `Context.MODE_WORLD_READABLE` – tutte le applicazioni avranno permessi Read-Only sui dati
 - ▶ `Context.MODE_WORLD_WRITEABLE` – tutte le applicazioni avranno permessi R/W sui dati
- Per ogni applicazione, le Shared Preferences devono essere associate ad un nome identificativo (una stringa di testo)

» developer.android.com/reference/android/content/SharedPreferences.html

Shared Preferences – Accesso in Scrittura

Esempio – Salvataggio di elementi nelle Shared Preferences

```
final String PREFERENCES_ID = "my-app-preferences";

SharedPreferences preferences =
    getSharedPreferences(PREFERENCES_ID, Context.MODE_PRIVATE);

preferences.edit()
    .putString("USERNAME", "mario.rossi")
    .putString("PASSWORD", "abc123")
    .putBoolean("LOGGED", true)
    .commit();
```

- Il riferimento alle SP è ottenibile mediante l'API di sistema `getSharedPreferences(String id, int mode)`
 - ▶ Su tale riferimento è possibile aggiungere (salvare) dati nella forma key-value, richiamando l'Editor delle SP
 - ▶ (1) invocazione del metodo `edit()`, (2) invocazione dei vari metodi `putX(...)`, (3) invocazione del metodo `commit()`

Shared Preferences – Accesso in Lettura

Esempio – Verifica e Recupero di elementi precedentemente salvati

```
final String PREFERENCES_ID = "my-app-preferences";

SharedPreferences preferences =
    getSharedPreferences(PREFERENCES_ID, Context.MODE_PRIVATE);

String username = preferences.getString("USERNAME", "unavailable");
boolean logged = preferences.getBoolean("LOGGED", false);
```

- Sul riferimento alle SP, è possibile invocare i metodi `getX(...)` per ottenere il valore relativo alla key specificata
- Qualora il valore non fosse disponibile (ovvero la key non è presente nel keyset), sarà restituito il valore del secondo parametro del metodo `getX` come default

Shared Preferences – Note





- Sul riferimento alle SP (ottenuto tramite `getSharedPreferences()`) può essere invocato il metodo `getAll()` che restituisce una mappa (di tipo `Map<String, ?>`) contenente tutti gli elementi delle SP
- Può essere registrato un listener che reagisca alle modifiche alle SP
 - ▶ metodo `registerOnSharedPreferenceChangeListener()`

Esempio – SP Callback

```
SharedPreferences.OnSharedPreferenceChangeListener spcl =
    new SharedPreferences.OnSharedPreferenceChangeListener() {
        @Override
        public void onSharedPreferenceChanged(SharedPreferences sp,
            String s) {
            //do something
        }
    };

preferences.registerOnSharedPreferenceChangeListener(spcl);
```

Riferimenti - Risorse Online

-  **Android Developers - Guide**
» <https://developer.android.com/guide/>
-  **Android Developers - API Reference**
» <https://developer.android.com/reference/>
-  **Android Developers - Samples**
» <https://developer.android.com/samples/>
-  **Android Developers - Design & Quality**
» <https://developer.android.com/design/>



Riferimenti - Libri

-  Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura
Programming Android
O'Reilly, 2011
-  Chris Haseman, Kevin Grant
Beginning Android Programming: Develop and Design
Peachpit Press, 2013
-  Ronan Schwarz, Phil Dutson, James Steele, Nelson To
The Android Developer's Cookbook : Building Applications with the Android SDK
Addison-Wesley, 2013
-  Theresa Neil
Mobile Design Pattern Gallery: UI Patterns for Smartphone App
O'Reilly, Second Edition, 2014