Lezione 20 – Applicazioni Android

Componenti android

- Activity: ogni schermata dell'applicazione,
- Service: attività in background
- Content Providers: fornitori di contenuti condivisi
- Intents: Meccanismo di Message passign
- Broadcas Receiver: Consumatori di Intenti
- Notifications: Notificazioni agli utenti

Manifesto dell'applicazione

Il file AndroidManifest.xml permette di definire la struttura e i metadati dell'applicazione.

Include un nodo per ogni componente (Attività, Servizi, Content Providers e Broadcast Receiver) e attraverso gli Intent Filter e i Permission determina come ogni componente interagisce con gli altri e le altre applicazioni.

La root del file xml è il tag “manifest”:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="it.unive.dsi.android"
android:versionCode="1"
android:versionName="1.0"> … </manifest>
```

I tag presenti dentro al manifesto sono:

- application: un solo tag di tipo application che specifica i metadati dell'applicazione(es. `<application android:icon="@drawable/icon" android:label="@string/app_name">`). Questo tag può contenere nidificati i seguenti tag:
- activity: un tag per ogni schermata dell'applicazione (es. `<activity android:name=".Prova" android:label="@string/app_name">`). android.name la classe dell'attività Ogni schermata deve essere dichiarata (altrimenti viene lanciata un'eccezione. Ogni nodo attività supporta il tag intent-filter:
  (es. `<intent-filter> <action android:name="android.intent.action.MAIN" /> <category android:name="android.intent.category.LAUNCHER" /></intent-filter>`) . Gli intent filter indicano quali sono i componenti che possono gestire i vari intenti. In questo caso, si dice che questa attività è quella che può gestire l'intento MAIN di categoria LAUNCHER (ovvero questa attività verrà mostrata quando si fa partire l'applicazione).
- service: un tag per ogni servizio (es .`<service android:name=".MyService" android:enabled="true"></service>`). Anche i service possono avere dei intent-filter.
- provider: un tag per ogni provider
- receiver
- uses-permissions: `<uses-permission android:name="android.permission.INTERNET"></uses-permission>`
- permission:
- instrumentation

Esempio completo file AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="it.unive.dsi.android"
android:versionCode="1"
android:versionName="1.0">
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
android:name="android.permission.SIGNAL_PERSISTENT_PROCESSES"></uses-permission>


<application android:icon="@drawable/icon" android:label="@string/app_name">
<activity android:name=".Prova"
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>
```

Risorse

Esternalizzazione delle risorse

Vantaggi:

1. diventano più facili da mantenere, aggiornare e gestire

2. si possono gestire più facilmente risorse diverse per hardware diversi e l'internazionalizzazione. (in automatico viene scelta la risorsa più idonea in base alla lingua, localizzazione, tastiera, schermo, senza scrivere una linea di codice).

3. Possiamo anche gestire l'aspetto dell'applicazione (layout).

Con Android possiamo gestire l'esternazionalizzazione di stringhe, colori, dimensioni, stili e temi, disegnabili (immagini), layout, animazioni.

Esempio file “string.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Ciao, comincia a navigare</string>
<string name="app_name">Chat</string>
<string name="textEdit">Inserisci l\'url qui</string>
</resources>
```

Utilizzo delle risorse nel codice

Le risorse sono accessibili tramite la classe R che viene generata automaticamente a partire dai file xml che contengono le definizioni delle risorse. La classe R contiene sottoclassi statiche per ognuno dei tipi di risorsa per cui e' stata definita almeno una risorsa. Ognuno delle sottoclassi ha come variabili statiche le risorse dichiarate nel rispettivo file xml il cui nome e' lo stesso dell'identificativo della risorsa (attibuto name del tag). Il valore di queste variabili e' un riferimento alla corrispondente locazione nella tabella delle risorse (*identificatore di risorsa*) (e non una istanza della risorsa stessa). Alcune metodi accettano direttamente degli identificatori di risorsa (come setContentView che abbiamo gia' visto).

```
package it.unive.dsi.android;

public final class R {
public static final class attr {
}
public static final class drawable {
public static final int icon=0x7f020000;
}
public static final class id {
public static final int editText1=0x7f050001;
public static final int listView1=0x7f050000;
}
public static final class layout {
public static final int main=0x7f030000;
}
public static final class string {
public static final int app_name=0x7f040001;
public static final int hello=0x7f040000;
public static final int textEdit=0x7f040002;
}
}
```

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

Quando si ha bisogno di un'istanza della risorsa, si usa un metodo opportuno per estrarre l'istanza della risorsa dalla tabella delle risorse.

```
Resources res = getResources();

String s = res.getString(R.string.app_name);

CharSequence cs = res.getText(R.string.app_name);
```

Utilizzo delle risorse in altre risorse

```
attribute= "@packagename:resourcetype/resourceidentifier"
```

oppure se il package è lo stesso dell'applicazione:

```
attribute= "@resourcetype/resourceidentifier"
```

esempio: `android:text="@string/textEdit"`

system resource

`packagename=android`

esempio in codice: `getString(android.R.string.selectAll)`

esempio in file risorse: `attribute="@android:string/selectAll"`

Creare risorse per vari linguaggi e hardware

Progetto/

res/

values/

values-it/

values-it-CH/

La lista seguente mostra quali sono i qualificatori con cui e' possibile personalizzare le risorse:

- Mobile Country Code e Mobile Network Code (MCC/MNC)
- lingua
- regione (r minuscola piu' il codice dello stato maiuscolo
- Screen Size One of small (smaller than HVGA), medium (at least HVGA and smaller than VGA), or large (VGA or larger).
- Screen Width/Length Specify long or notlong for resources designed specifically for widescreen (e.g., WVGA is long, QVGA is notlong).
- orientamento dello schermo port/land/square
- densita' di pixel dello schermo pixel per pollice (92dpi)
- Tipo di touchscreen notouch, stylus, finger
- disponibilita' della tastiera keysexposed, keyshidden
- tipo di tastiera: nokeys, querty, 12key
- tipo navigazione notouch, dpad, trackball, wheel
- risoluzione schermo: risoluzione in pixel con la piu' grande per prima (320x240)

Bisogna seguire l'ordine e si possono specificare quelli che servono. Al run time android quando ricerca una risorsa restituisce quella che soddisfa piu' criteri, in caso di parita' sul numero di criteri, si sceglie come importanza seguendo l'ordine della lista.

Cambiamenti di configurazione al runtime

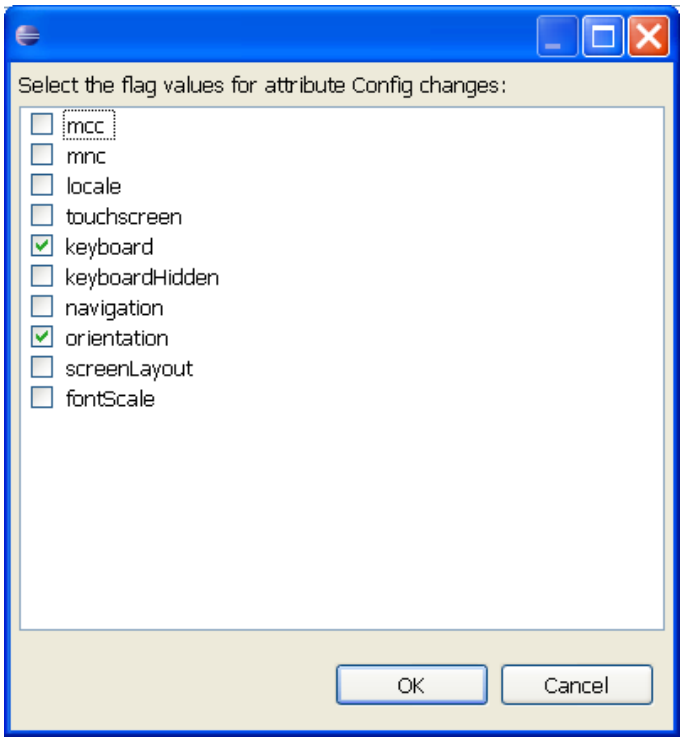
Android gestisce cambiamenti al run time di linguaggio, location e hardware terminando e facendo ripartire l'attività. Quando questo comportamento non ci e' gradito bisogna:

1. nel manifesto aggiungere nell'attivita' l'attributo android:configChanges specificando i cambiamenti di configurazione che si intende gestire:

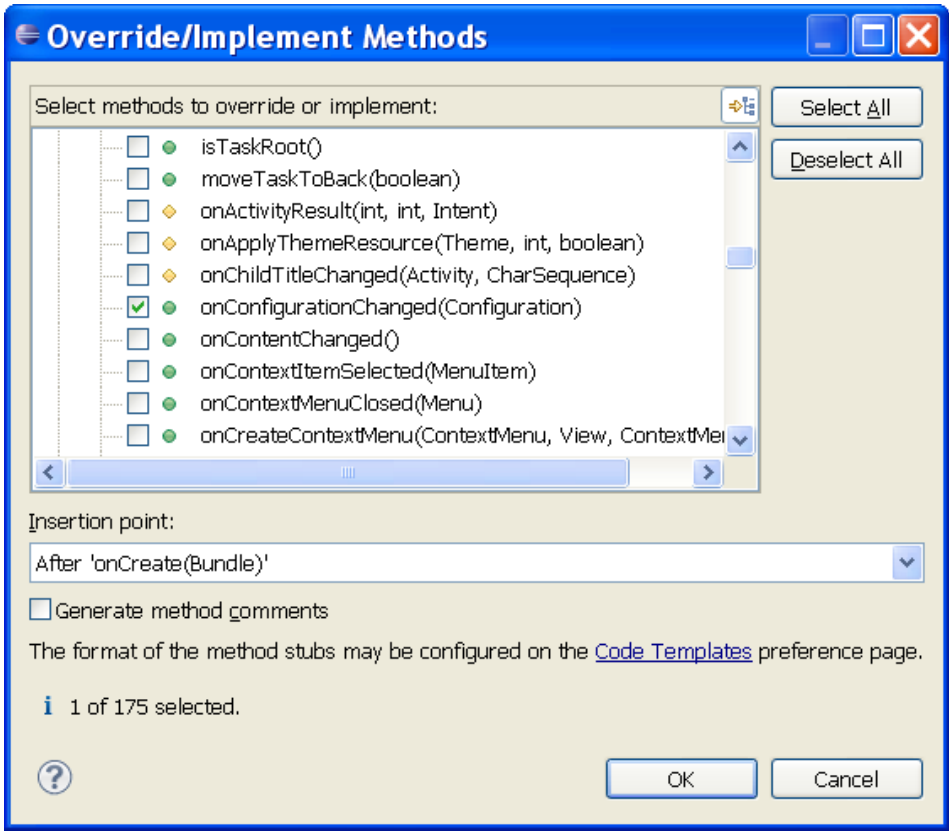
- orientation
- keyboardHidden
- fontScale
- locale
- keyboard
- touchscreen
- navigation

si possono selezionare più eventi da gestire separando i valori con |.

Per esempio `android:configChanges="keyboard|orientation"`



2. sovrascrivere il metodo onConfigurationChanged



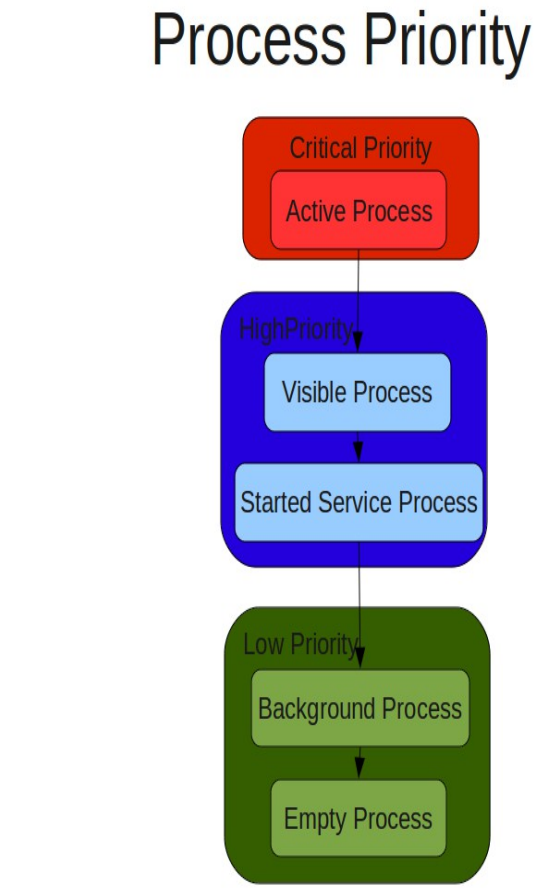

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
super.onConfigurationChanged(newConfig);

if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
//orientamento landscape
}
else {
//orientamento portrait
}

if (newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
//tastiera visibile
}
else {
//tastiera nascosta
}
}
```

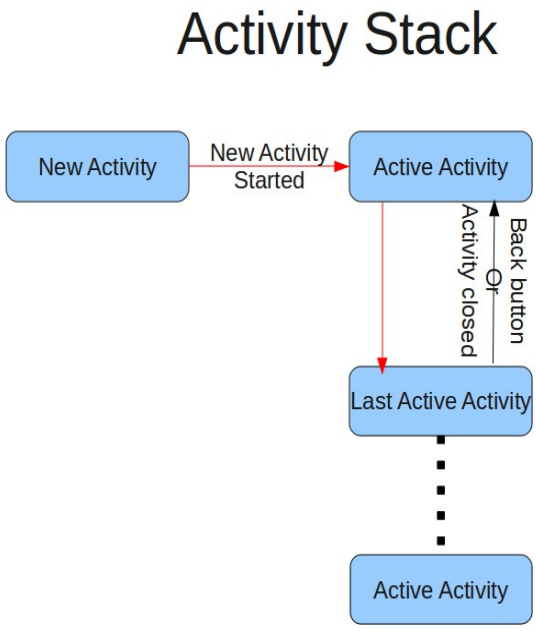
Ciclo di vita delle applicazioni Android

Active Process: processi che ospitano applicazioni con componenti attualmente in iterazione con l'utente. Questi processi sono gli ultimi a essere eliminati. I componenti attualmente in iterazione includono:

- Attività in primo piano e che stanno rispondendo a eventi dell'utente.
- Attività, servizi o broadcast receviver che stanno eseguendo un `onReceive`
- Serivzi che stanno eseguendo `onStart`, `onCreate` o `onDestroy`

Visible Process: processi che ospitano applicazioni visibili ma inattive cioè che ospitano attività visibili ma che non interagiscono con l'utente. (attività parzialmente oscurate da attività visibili)
Started Service Process: processi che ospitano servizi in esecuzione
Background Process: processi che ospitano attività che non sono visibili e che non hanno servizi che sono in esecuzione che vengono eliminati iniziando dall'ultimo che e' stato “visto”.
Empty Process: processi che sono terminati ma di cui Android mantiene l'immagine della memoria dell'applicazione per farla ripartire più velocemente in caso venga richiesta nuovamente.





Ciclo di vita delle Attività