Lezione 12

Programmazione Android

- Storage temporaneo
  - **Salvataggio temporaneo dello stato**
- Storage permanente
  - **Preferenze**
    - Shared & Private Preferences
    - PreferenceScreen e PreferenceActivity
  - **Accesso al File System**
  - **Accesso a Database**
- Condivisione di dati
  - **Content Provider**

Salvataggio temporaneo dello stato

Stato temporaneo

- Il ciclo di vita di un'Activity prevede casi in cui l'istanza della vostra classe può essere eliminata dalla memoria
  - Anche se *logicamente* l'Activity è ancora “viva”
    - Es.: presente nello stack di un task “vivo”, ma non visibile
- In questi casi, è necessario salvare lo **stato transiente** di un'Activity
  - In modo da ripristinarlo più tardi, quando il sistema istanzierà una nuova copia dell'Activity
    - Es.: un'Activity sopra la vostra viene rimossa con Back

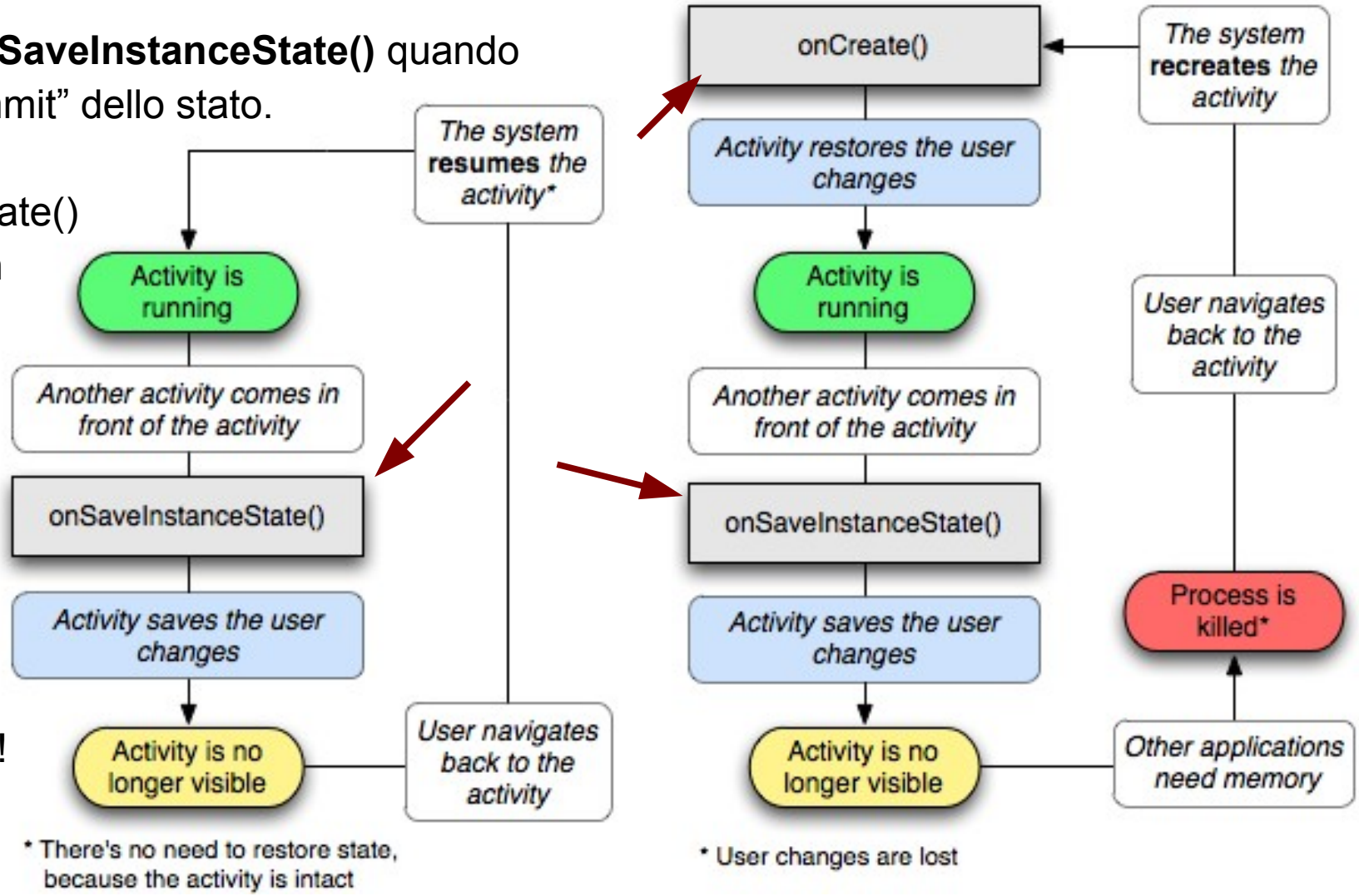
Stato temporaneo

Android invoca **onSaveInstanceState()** quando vuole fare un "commit" dello stato.

onSaveInstanceState() salva lo stato in un **Bundle**.

Se necessario, **onCreate()*** ripristina lo stato dal Bundle.

Può anche non essere necessario!



* O onRestoreInstanceState().

Il Bundle

- La classe **Bundle** è una mappa chiave-valore
  - Chiave è una stringa
  - Valore è un **Parcelable**
    - La classe **Parcel** e l'interfaccia **Parcelable** sono usate come meccanismo di IPC in Android
    - Non è serializzazione piena, ma è molto più efficiente
    - In pratica:
      - Tutti i **tipi base**, **array** degli stessi, **ArrayList**, **String**, altri **Bundle**, altri oggetti che implementano **Parcelable** (circa 350 classi del framework), e oggetti che implementano **Serializable**
  - Abbondanza di metodi get*Tipo*(key) e put*Tipo*(key,value)

Bundle e Parcelable

- Parcelable è un'interfaccia di scrittura…
  - … ma deve contenere un campo statico CREATOR che contenga un oggetto creatore (per la lettura)

```
public class MyP implements Parcelable {
  private int mData;

  public int describeContents() {
    return 0;
  }

  public void writeToParcel(Parcel out, int flags) {
    out.writeInt(mData);
  }

  public static final Parcelable.Creator<MyP> CREATOR
  = new Parcelable.Creator<MyP>() {
    public MyP createFromParcel(Parcel in) {
      return new MyP(in);
    }

    public MyParcelable[] newArray(int size) {
      return new MyP[size];
    }
  };

  private MyP(Parcel in) {
    mData = in.readInt();
  }
}
```

Bitmask di flag
(1 = l'oggetto contiene un file descriptor)

Bundle e Serializable

- Se vi trovate a voler mettere nel Bundle degli oggetti che implementano Serializable, forse *lo state facendo sbagliato!*

Implement Serializable Judiciously

Refer to *Effective Java*'s chapter on serialization for thorough coverage of the serialization API. The book explains how to use this interface without harming your application's maintainability.

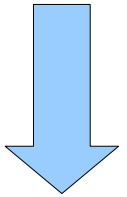
Recommended Alternatives

JSON is concise, human-readable and efficient. Android includes both a `streaming API` and a `tree API` to read and write JSON. Use a binding library like GSON to read and write Java objects directly.

Il Bundle

- Nota:
  - Il Bundle è un meccanismo **generico** per passare valori
  - È possibile istanziare un proprio Bundle, inserire dei valori, e poi metterlo come Extra in un Intent che viene spedito ad altri

```
String s = "pippo";
Bundle b = new Bundle();
b.putString("Account",s);
Intent i = new Intent(this, FetchAct.class);
i.putExtras(b);
startActivity(i);
```



```
Intent j = getIntent();
String acc = j.getStringExtra("Account");
```

Stato temporaneo

- Salvataggio

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    // Save away the original text, so we still have it if the activity
    // needs to be killed while paused.
    outState.putString(ORIGINAL_CONTENT, mOriginalContent);
}
```

- Ripristino

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
      ...
    // If an instance of this activity had previously stopped, we can
    // get the original text it started with.
    if (savedInstanceState != null) {
        mOriginalContent = savedInstanceState.getString(ORIGINAL_CONTENT);
    }
}
```

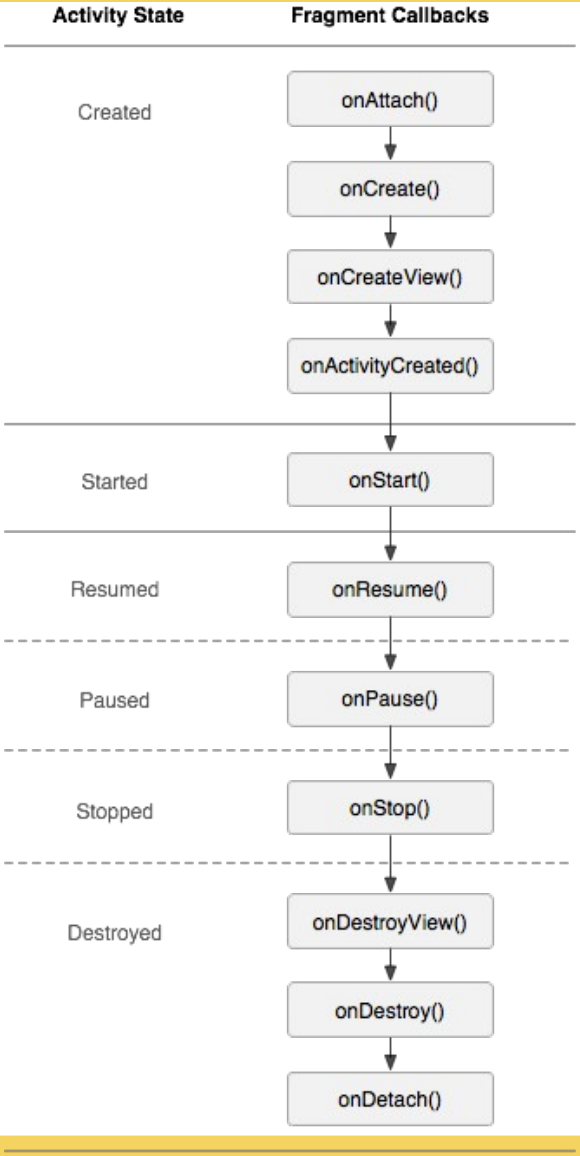
Stato temporaneo

- Il Bundle preparato da onSaveInstanceState() è mantenuto dal sistema...
  - … solo se si prevede di far ripartire **la particolare istanza** dell'Activity
    - Ovvero, se essa viene scaricata per mancanza di memoria
    - **NON** viene chiamato onSaveInstanceState() se l'Activity è terminata, con un Back o con un finish()
  - … in maniera **non permanente**
    - Il Bundle viene tenuto in RAM, in caso di riavvio va tutto perso
      - Ma nulla esclude che il sistema prima o poi decida di tenerlo in NVRAM – meglio essere preparati a tutto

In effetti l'hanno fatto :-)

Stato nei Fragment

- Quando un Fragment è *attached* a una Activity, ne può seguire la sorte
- onSaveInstanceState() del Fragment salva lo stato dell'istanza del fragment in un Bundle
- Il bundle viene preservato e poi passato a
  - onCreate()
  - onCreateView()
  - onActivityCreated()
- Questo consente un ripristino *staged*



Le preferenze

Preferenze & Impostazioni

- Per memorizzare dati in maniera **permanente** si possono usare varie altre tecniche
- Un caso frequente è quello in cui si vogliano memorizzare le **Preferenze** dell'utente
  - Settings, Options, configurazioni, ecc.
  - Si possono usare gli stessi meccanismi anche per memorizzare dati “propri” dell'app, non visibili all'utente
    - es.: data e ora dell'ultimo update, statistiche di utilizzo dell'applicazione
- Android offre un supporto specializzato

SharedPreferences

- Android offre un framework per la gestione generalizzata di preferenze
- La classe **SharedPreference** rappresenta una mappa chiavi-valori
  - Simile al Bundle, ma con alcune importanti differenze:
    - Le preferenze sono memorizzate in maniera **permanente**
    - I valori possono essere solo **tipi base**, **String** o **Set<String>**
    - Il sistema gestisce l'**aggiornamento atomico** (commit)
    - È disponibile un sistema di **notifiche** per essere avvisati quando le preferenze cambiano
    - Le preferenze possono essere **per-Activity** o **globali**

SharedPreferences

- Le preferenze vengono salvate su un file
- Se ne possono avere molte!
  - Il programmatore può dare un *nome* a un insieme di preferenze per l'App
  - Oppure, ogni Activity può usare le proprie
    - In pratica: il nome della classe diventa il nome delle preferenze
- Path sul file system:
  - /data/data/*package*/shared_prefs/*nome*.xml
  - /data/data/*package*/shared_prefs/*package*_preferences.xml

Area dati dell'app identificata dal nome del *package*

SharedPrefs di default

SharedPreferences

- A differenza che nei Bundle, nel caso delle preferenze le scritture sono **transazionali**
  - Prima viene fatto un insieme di aggiornamenti
  - Poi si effettua il **commit** che le scrive tutte insieme
- Ciò garantisce:
  - **Consistenza**: non può capitare che alcuni campi vengano aggiornati e altri no; il commit è atomico
  - **Coalescing** delle notifiche: chi è in attesa viene notificato una volta sola per l'intero insieme di modifiche, non ad ogni campo cambiato

Preferences Editor

- Le modifiche alle preferenze vengono fatte tramite un **editor** (che viene ottenuto dalle preferenze stesse)
- L'**editor** offre i metodi **put*Tipo*(chiave, valore)** per inserire coppie chiave-valore nella sua tabella temporanea
- Per copiare la tabella temporanea sulle preferenze sono disponibili due metodi dell'editor
  - **commit()** – aggiunge la tabella dell'editor a quella delle preferenze, e salva immediatamente su disco
    - Sicura, in caso di errore restituisce un codice d'errore, meno efficiente
  - **apply()** – aggiunge la tabella dell'editor a quella delle preferenze in memoria, e schedula la scrittura asincrona del risultato su disco
    - Meno sicura, non verifica gli errori, più efficiente

Scrittura di preferenze

- Ottenere un oggetto SharedPreferences
  - Dotato di nome: **getSharedPreferences(*nome*,*modo*)**
    - Metodo di Context
    - Il *modo* può essere
      - MODE_PRIVATE
      - MODE_WORLD_READABLE
      - MODE_WORLD_WRITEABLE
      - MODE_MULTI_PROCESS (in OR con i precedenti)

Deprecati da API level 17

Rimossi da API level 24

Da Android 7 in poi lanciano una SecurityException

In Jetpack c'è una classe EncryptedSharedPreferences (attualmente in alpha)

  - Privato: **getPreferences(*modo*)**
    - Metodo di Activity
    - Il *modo* può essere come sopra, tranne MULTI_PROCESS

Scrittura di preferenze

- Ottenere un oggetto SharedPreferences
  - Di default per un dato Context: **PreferenceManager.getDefaultSharedPreferences(*ctx*)**
    - Il context può essere vario
      - Context.getApplicationContext() → context dell'applicazione intera
      - Activity → context della specifica activity
      - Service, BroadcastReceiver, ContentProvider → idem
      - e *molti* altri!
        - Wallpaper animati, metodo di input (=tastiera virtuale), agente di backup, home screen, spell checker di sistema, ecc.

Scrittura di preferenze

- Scrittura

```
SharedPreferences pref=getPreferences(MODE_PRIVATE);
Editor editor=pref.edit();
editor.putString(K_LOGIN, login);
editor.putString(K_PWD, password);
editor.commit();
```

- Lettura

```
SharedPreferences pref=getPreferences(MODE_PRIVATE);
login=pref.getString(K_LOGIN, "guest");
password=pref.getString(K_PWD,"123456");
```

Default da usare se la chiave non esiste

Notifiche sulle preferenze

- Le SharedPreferences possono essere (ovviamente) condivise fra più activity
- Ciascuna di questa può scriverci dentro
  - Ricordate: il commit è atomico
- Le altre activity possono voler essere informate dei cambiamenti apportati alle preferenze
  - Ricordate: si chiamano preferenze, ma possono essere dati qualunque, anche interni!
    - Purché esprimibili con tipi base, String, Set<String>

Notifiche sulle preferenze

- Si usa il solito meccanismo dei **Listener**
  - prefs.**registerOnSharedPreferenceChangeListener(*listener*)**
- Il listener deve implementare l'interfaccia **OnSharedPreferencesChangeListener**
  - Ha un unico metodo: **onSharedPreferenceChanged(*prefs*, *chiave*)**
    - No comment sui nomi...
    - La *chiave* può essere stata aggiunta, modificata o rimossa (e quindi non esistere più in *prefs*!)
- Per rimuovere un listener, ***ovviamente*** si chiama
  - prefs.**unregisterOnSharedPreferenceChangeListener(*listener*)**

Gestione strutturata delle preferenze

- I metodi che abbiamo visto sono relativamente semplici e piuttosto comodi per salvare un po' di dati generici
- Tuttavia, quando le preferenze rappresentano davvero preferenze dell'utente, e devono essere editabili, le cose si complicano
  - Serve un'Activity per fornire la GUI
  - Vanno gestiti tutti gli eventi relativi
- Android fornisce però un framework ad-hoc!

PreferenceScreen

- Note quali preferenze servono, si può costruire la GUI corrispondente in maniera sistematica
- Conviene allora definire le preferenze in maniera **dichiarativa** e lasciar fare al sistema
  - In pratica... tramite un file XML
- L'editing, l'aggiornamento, le notifiche, l'undo (con Back) ecc. sono gestiti autonomamente
- All'applicazione non resta da fare altro che registrare un listener (e reagire)!

PreferenceScreen è con noi da Android 1, ma è stato deprecato in Android 10 (API Level 29).
Al suo posto, si può usare la nuova API **androidx.preference**.

PreferenceScreen

- Il file che descrive le preferenze va in res/xml
  - **Non** è un layout, anche se il sistema ne deriva un layout
- L'elemento radice è **<PreferenceScreen>**
- All'interno, può contenere
  - **<PreferenceScreen>** – struttura gerarchica
  - **<PreferenceCategory>** – raggruppa opzioni correlate
  - **<CheckBoxPreference>**, **<EditTextPreference>**, **<ListPreference>**, **<RingtonePreference>**
  - Altri editor di preferenze custom
    - Sottoclassi di Preference o dei precedenti

PreferenceScreen

- I nodi hanno alcuni attributi standard
  - **android:key** – la chiave di questa entry nelle pref
  - **android:title** – il titolo del'entry
  - **android:summary** – la descrizione dell'entry
  - **android:icon** – un riferimento a risorsa per l'icona
  - **android:defaultValue** – valore di default dell'entry
- Altri sono specifici di certi tipi, o di uso più raro
  - es.: ordinamento, subordinazione all'attivazione di altre entry, layout custom

PreferenceScreen esempio: prefs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

   <PreferenceCategory android:title="Audio" >
      <CheckBoxPreference
         android:defaultValue="true"
         android:key="audio_in"
         android:summary="Abilita la registrazione dell'audio dal microfono incorporato."
         android:title="Attiva microfono" />
      <CheckBoxPreference
         android:defaultValue="true"
         android:key="audio_out"
         android:summary="Abilita la riproduzione dell'audio dall'altoparlante incorporato."
         android:title="Attiva altoparlante" />

      <ListPreference
         android:dialogTitle="Scegli voce"
         android:entries="@array/voci"
         android:entryValues="@array/valori"
         android:key="Voce"
         android:summary="Puoi scegliere che tipo di voce utilizzare"
         android:title="Tipo voce" />
   </PreferenceCategory>
```

PreferenceScreen esempio: prefs.xml

```
<PreferenceCategory android:title="Video" >
    <CheckBoxPreference
        android:defaultValue="true"
        android:key="video"
        android:summaryOff="Abilita il video (occupa lo schermo)"
        android:summaryOn="Disabilita il video (solo audio)"
        android:title="Abilita il video" />

    <EditTextPreference
        android:defaultValue="Roboto"
        android:key="nome_pg"
        android:summary="Puoi scegliere il nome da dare al tuo personaggio"
        android:title="Nome personaggio" />

    <PreferenceScreen android:title="Avanzate..." >

        ...

    </PreferenceScreen>
</PreferenceCategory>

</PreferenceScreen>
```

PreferenceScreen uso di intent

- È possibile avere “preferenze” che non consentono di impostare alcunché, ma si limitano a inviare un Intent quando selezionate
  - Per esempio: un “About” messo fra le preferenze

```
<PreferenceScreen>
    …
    <Preference android:title="@string/about" >
        <intent   android:action="android.intent.action.VIEW"
                  android:data="http://www.di.unipi.it" />
    </Preference>
    …
</PreferenceScreen>
```

La PreferenceActivity

- La classe PreferenceActivity si occupa di
  - leggere il nostro prefs.xml
  - generare al volo un layout adeguato
  - interagire con l'utente
    - Anche in maniera complessa: navigazione in sotto-schermi, dialog, ecc.
  - salvare le impostazioni nelle SharedPrefs alla fine
    - Il che può far partire delle notifiche

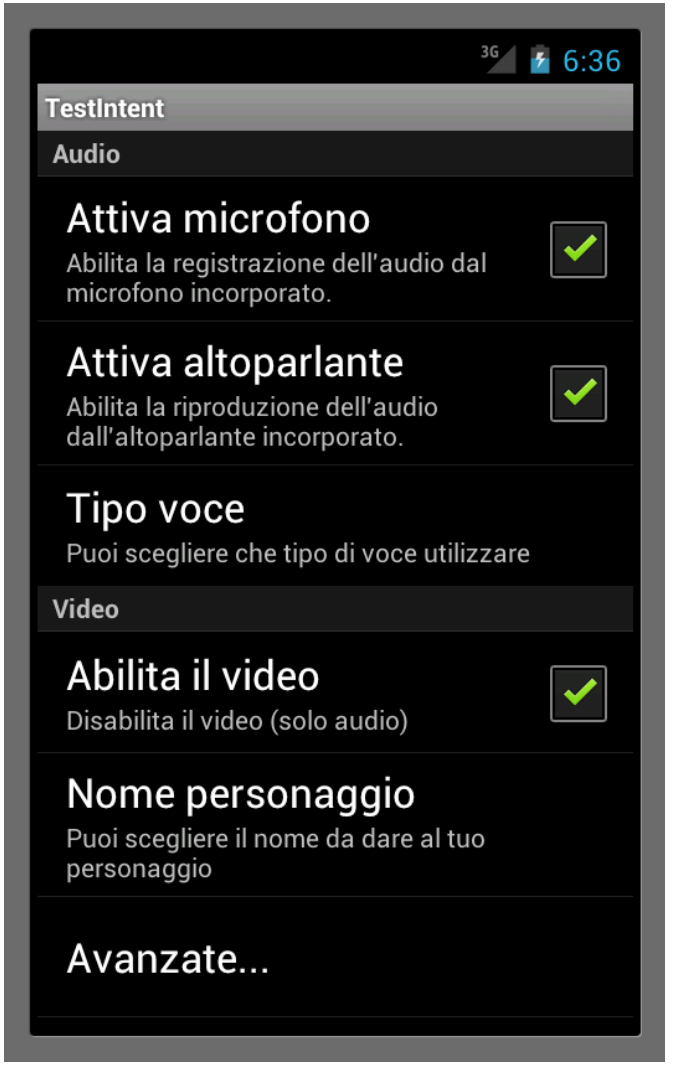
La PreferenceActivity esempio

- Non si può dire che ci si ammazzi di lavoro...
- La classe può opzionalmente implementare un change listener, e reagire opportunamente
  - O, meglio, lasciare che si registrino le altre activity

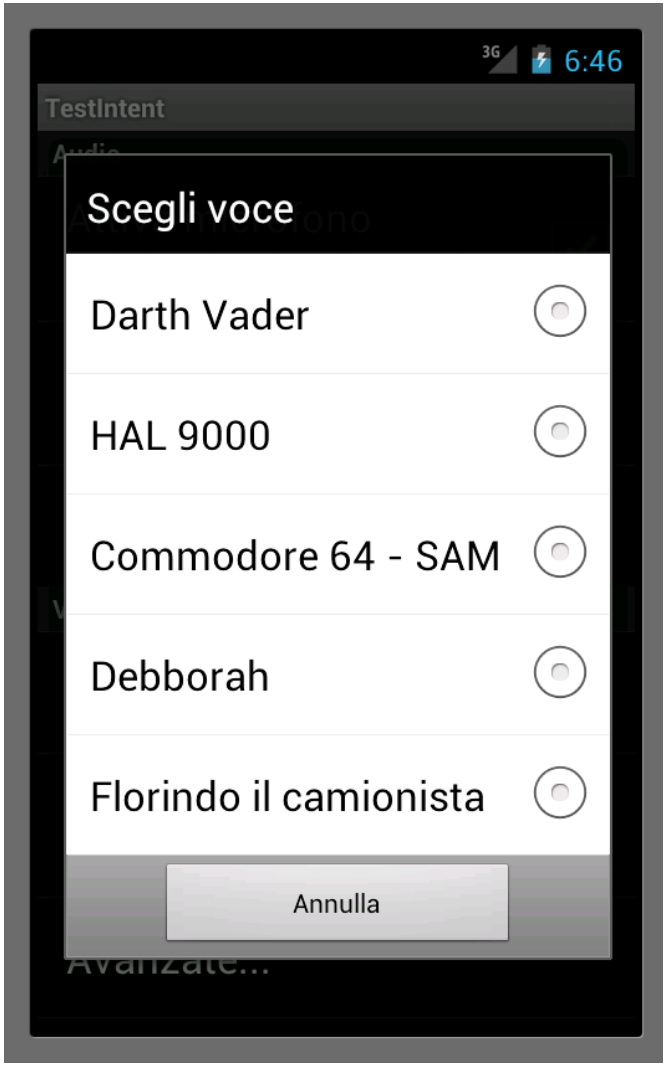
```
public class TestIntentActivity extends
PreferenceActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.prefs);
    }
}
```

La PreferenceActivity esempio

- L'activity parte, presenta la familiare interfaccia dei settings all'utente
- Come bonus ulteriore, si adatta automaticamente al tema corrente
- L'uso di riferimenti a risorse nel file XML rende anche il tutto facilmente localizzabile
  - Noi non l'abbiamo fatto, ma...



La PreferenceActivity esempio



- Anche la gestione dei dialog è completamente invisibile
- I valori scelti però verranno scritti alla fine nelle preferenze
- Vengono usate sempre le **preferenze globali dell'applicazione**

```
Context
c=getApplicationContext();
SharedPreferences prefs=
   PreferenceManager.
   getDefaultSharedPreferences(c);
```

PreferenceFragment

- Da Android 3.0 esiste la versione “Fragment” della PreferenceActivity
- Vantaggi
  - Può essere inclusa in schermate più complesse
    - Anche dinamicamente
  - Può supportare layout più diversificati
  - Maggiore integrazione con transazioni e transizioni
- Svantaggi
  - Un po' di codice in più da scrivere

PreferenceFragment

- La definizione del Fragment è del tutto standard

```
public class TestPrefFrag extends PreferenceFragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.prefs);
    }
}
```

- L'aggiunta a una activity non lo è di meno

```
getFragmentManager().beginTransaction()
      .replace( ... , new TestPrefFrag())
      .commit();
```

- Vale quanto detto per i Fragment in generale

Condividere preferenze

- Si possono invocare dalle proprie preferenze quelle di sistema (o di altre app)

In prefs.xml

```
<PreferenceScreen android:title="Avanzate..." >
   <intent android:action="android.settings.SOUND_SETTINGS"  />
</PreferenceScreen>
```

- Si possono rendere le proprie preferenze invocabili da altre app (già visto!)

In AndroidManifest.xml

```
<activity
   android:name=".TestIntentActivity"
   android:label="@string/app_name" >
   <intent-filter>
      <action android:name="it.unipi.di.masterapp.PG_PREFS"/>
   </intent-filter>
</activity>
```

Preferenze di sistema

ACTION_ACCESSIBILITY_SETTINGS	Activity Action: Show settings for accessibility modules.
ACTION_ADD_ACCOUNT	Activity Action: Show add account screen for creating a new account.
ACTION_AIRPLANE_MODE_SETTINGS	Activity Action: Show settings to allow entering/exiting airplane mode.
ACTION_APN_SETTINGS	Activity Action: Show settings to allow configuration of APNs.
ACTION_APPLICATION_DETAILS_SETTINGS	Activity Action: Show screen of details about a particular application.
ACTION_APPLICATION_DEVELOPMENT_SETTINGS	Activity Action: Show settings to allow configuration of application development-related settings.
ACTION_APPLICATION_SETTINGS	Activity Action: Show settings to allow configuration of application-related settings.
ACTION_BLUETOOTH_SETTINGS	Activity Action: Show settings to allow configuration of Bluetooth.
ACTION_DATA_ROAMING_SETTINGS	Activity Action: Show settings for selection of 2G/3G.
ACTION_DATE_SETTINGS	Activity Action: Show settings to allow configuration of date and time.
ACTION_DEVICE_INFO_SETTINGS	Activity Action: Show general device information settings (serial number, software version, phone number, etc.).
ACTION_DISPLAY_SETTINGS	Activity Action: Show settings to allow configuration of display.
ACTION_INPUT_METHOD_SETTINGS	Activity Action: Show settings to configure input methods, in particular allowing the user to enable input methods.
ACTION_INPUT_METHOD_SUBTYPE_SETTINGS	Activity Action: Show settings to enable/disable input method subtypes.
ACTION_INTERNAL_STORAGE_SETTINGS	Activity Action: Show settings for internal storage.
ACTION_LOCALE_SETTINGS	Activity Action: Show settings to allow configuration of locale.
ACTION_LOCATION_SOURCE_SETTINGS	Activity Action: Show settings to allow configuration of current location sources.
ACTION_MANAGE_ALL_APPLICATIONS_SETTINGS	Activity Action: Show settings to manage all applications.
ACTION_MANAGE_APPLICATIONS_SETTINGS	Activity Action: Show settings to manage installed applications.
ACTION_MEMORY_CARD_SETTINGS	Activity Action: Show settings for memory card storage.
ACTION_NETWORK_OPERATOR_SETTINGS	Activity Action: Show settings for selecting the network operator.
ACTION_NFCSHARING_SETTINGS	Activity Action: Show NFC sharing settings.
ACTION_PRIVACY_SETTINGS	Activity Action: Show settings to allow configuration of privacy options.
ACTION_QUICK_LAUNCH_SETTINGS	Activity Action: Show settings to allow configuration of quick launch shortcuts.
ACTION_SEARCH_SETTINGS	Activity Action: Show settings for global search.
ACTION_SECURITY_SETTINGS	Activity Action: Show settings to allow configuration of security and location privacy.
ACTION_SETTINGS	Activity Action: Show system settings.
ACTION_SOUND_SETTINGS	Activity Action: Show settings to allow configuration of sound and volume.
ACTION_SYNC_SETTINGS	Activity Action: Show settings to allow configuration of sync settings.
ACTION_USER_DICTIONARY_SETTINGS	Activity Action: Show settings to manage the user input dictionary.
ACTION_WIFI_IP_SETTINGS	Activity Action: Show settings to allow configuration of a static IP address for Wi-Fi.
ACTION_WIFI_SETTINGS	Activity Action: Show settings to allow configuration of Wi-Fi.
ACTION_WIRELESS_SETTINGS	Activity Action: Show settings to allow configuration of wireless controls such as Wi-Fi, Bluetooth and Mobile networks.

Accesso al file system

Il file system di Android

- Android è uno strato “sopra” Linux
- Il file system vero e proprio è quindi gestito interamente da quest'ultimo
  - File, directory, hard/soft-link, diritti, proprietario, ecc.
- La gestione dei file di Android è dunque uno strato (interfaccia Java) “sopra” il file system vero
- Accedere direttamente ai file su Android **è raro**
  - **Si usano più spesso le preferenze o i database**
  - **Oppure, in sola lettura, le risorse (raw) e gli asset**

java.io.* e java.nio.*

- È disponibile la gestione dei file standard di Java tramite le classi dei package
  - java.io – accesso ai file tradizionale
  - java.nio – accesso asincrono ai file
- Sono poi disponibili tutte le consuete facility
  - Wrapper di vario tipo
    - Reader/Writer, Buffered(Input/Output)Stream, Object(Input/Output)Stream e la serializzazione, StringReader, StringTokenizer, ecc.
  - Navigazione con gli oggetti File (= file e directory)

Pecularità di Android

- Tuttavia, Android presenta delle peculiarità:
  - L'utente **non siete voi**, ma la vostra App o la coppia (*user*,*app*)
    - Ogni app è un utente diverso, i file sono mutuamente segregati in directory distinte
    - Per default, i file sono “privati” all'app (ma possono essere resi pubblici)
  - I dispositivi distinguono **memoria “interna”** (al telefono) e **memoria “esterna”** (scheda SD e simili)
  - L'ambiente definisce **posti standard** in cui memorizzare vari tipi di dato condivisi
    - Musica, video, ebook, suonerie, foto, podcast, ...

Pecularità di Android

- Ogni App ha una **base directory** in **memoria interna** e una in **memoria esterna** (se c'è)
  - Queste directory vengono cancellate se l'app viene disinstallata
  - L'App può crearvi file e sotto-directory a piacere
- Per certi usi, sono definiti **nomi convenzionali**
  - In questo modo, il Media Scanner di sistema può trovare e classificare i dati multimediali
- L'App ha una **cache directory** per i file temporanei
  - Il sistema la cancella se ha bisogno di spazio – ma meglio limitarsi!
- Il sistema fornisce una **directory condivisa** per dati pubblici
  - Nuovamente, all'interno si usano i nomi convenzionali

File in memoria interna

- Il Context offre metodi di utilità per accedere ai file nella base dir interna
  - FileOutputStream **openFileOutput(*nome*, *modo*)**
  - FileInputStream **openFileInput(*nome*)**
  - String [] **fileList()**
  - void **deleteFile(*nome*)**
- E altri metodi per gestire le directory
  - File **getFilesDir()** – restituisce il path alla base dir
  - File **getDir(*nome*, *modo*)** – apre o crea una sottodir
  - File **getCacheDir()** – restituisce il path alla cache dir

File in memoria esterna

- La scheda SD potrebbe mancare, o essere stata estratta, oppure montata via USB su un PC
- Prima di accedere alla memoria esterna, è bene controllare che sia presente
  - String **getExternalStorageState()**
  - Valori restituiti: costanti stringa definite in Environment
  - I file sono accessibili se viene restituto
    - Environment.MEDIA_MOUNTED – tutto ok!
    - Environment.MEDIA_MOUNTED_READ_ONLY – solo lettura

Metodo (statico) di **Environment**

File in memoria esterna

Altri stati possibili

MEDIA_BAD_REMOVAL	if the media was removed before it was unmounted.
MEDIA_CHECKING	if the media is present and being disk-checked
MEDIA_MOUNTED	if the media is present and mounted at its mount point with read/write access.
MEDIA_MOUNTED_READ_ONLY	if the media is present and mounted at its mount point with read only access.
MEDIA_NOFS	if the media is present but is blank or is using an unsupported filesystem
MEDIA_REMOVED	if the media is not present.
MEDIA_SHARED	if the media is present not mounted, and shared via USB mass storage.
MEDIA_UNMOUNTABLE	if the media is present but cannot be mounted.
MEDIA_UNMOUNTED	if the media is present but not mounted.

File in memoria esterna

- File getExternalFilesDir(*tipo*) – restituisce la directory in cui l'App dovrebbe salvare i file del *tipo* indicato
  - Tipicamente, /Android/data/*package*/files/...
  - Tuttavia, la memoria esterna è condivisa, non ci sono diritti né protezioni: è solo una convenzione
- Se il *tipo* è **null**, viene restituita la base dir della vostra app
- La base dir e tutto il contenuto vengono cancellati se l'app viene disinstallata

File in memoria esterna

DIRECTORY_ALARMS	Standard directory in which to place any audio files that should be in the list of alarms that the user can select (not as regular music).
DIRECTORY_DCIM	The traditional location for pictures and videos when mounting the device as a camera.
DIRECTORY_DOWNLOADS	Standard directory in which to place files that have been downloaded by the user.
DIRECTORY_MOVIES	Standard directory in which to place movies that are available to the user.
DIRECTORY_MUSIC	Standard directory in which to place any audio files that should be in the regular list of music for the user.
DIRECTORY_NOTIFICATIONS	Standard directory in which to place any audio files that should be in the list of notifications that the user can select (not as regular music).
DIRECTORY_PICTURES	Standard directory in which to place pictures that are available to the user.
DIRECTORY_PODCASTS	Standard directory in which to place any audio files that should be in the list of podcasts that the user can select (not as regular music).
DIRECTORY_RINGTONES	Standard directory in which to place any audio files that should be in the list of ringtones that the user can select (not as regular music).

Anche in questo caso, si tratta di costanti definite nella classe **Environment**

File condivisi e cache in memoria esterna

- Un'applicazione che voglia esplicitamente **condividere file** (tipicamente, media), può salvarli nella directory condivisa di sistema
  - File **getExternalStoragePublicDirectory(*tipo*)**
- Questi file **non** vengono cancellati quando l'applicazione viene disinstallata
- Se l'app ha bisogno di una **cache ampia**, può utilizzare la memoria esterna
  - File **getExternalCacheDir()**
- La cache esterna verrà svuotata alla disinstallazione
  - Ma non in caso di SD piena – la gestione è a carico vostro!

Interno vs. Esterno

Internal storage:

- It's always available.
- Files saved here are accessible by only your app.
- When the user uninstalls your app, the system removes all your app's files from internal storage.

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage:

- It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
- It's world-readable, so files saved here may be read outside of your control.
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from `getExternalFilesDir()`.

External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

★ **Tip:** Although apps are installed onto the internal storage by default, you can allow your app to be installed on external storage by specifying the **`android:installLocation`** attribute in your manifest. Users appreciate this option when the APK size is very large and they have an external storage space that's larger than the internal storage. For more information, see App Install Location.

URI file:// & FileProvider

- È sempre possibile accedere a file tramite tutte le API che accettano URI, usando lo schema file://
  - Uri.*fromFile*() per ottenere l'URI (File.toURI() è deprecato)
- Il **FileProvider** è un particolare **ContentProvider** fornito dal sistema dedicato alla gestione dei file
  - Può essere configurato (tramite file XML) per fornire alle applicazioni una visione "virtuale" di file system
  - Vedremo i dettagli nella prossima lezione, dopo aver illustrato i ContentProvider in generale