20
Sviluppo di Applicazioni Mobile in Android

Andrea Santi
a.santi@unibo.it

C.D.L. Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2013/2014

Outline

Goal della lezione

- Overview generale su Android
  - Che cos'è
  - Un po' di storia
  - Market share
- Architettura di una applicazione Android
  - I componenti principali dell'SDK e per cosa devono essere usati
- Sviluppo di semplici app

Outline

Outline

Che Cos'è Android?

Android in a Nutshell

Android è una piattaforma/software stack open-source per lo sviluppo di applicazioni (principalmente) per dispositivi mobile

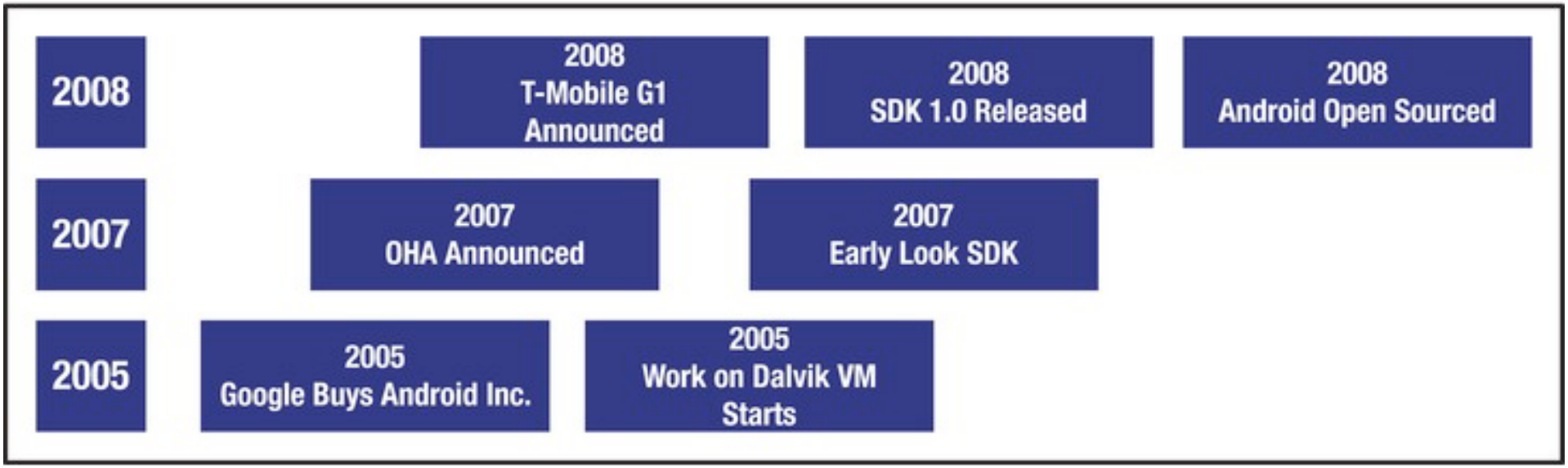
Il software stack di Android è caratterizzato/composto da:

- Un sistema operativo basato su Linux Kernel
- Un middleware composto da un insieme di librerie di varia utilità
- Un set di applicazioni base pre-installate
- Un Software Development Kit (SDK) con cui sviluppare applicazioni

Un po' di Storia [11, 12]

- **Ottobre 2003**: nasce la Android inc. (fondata da Andy Rubin)
- **Agosto 2005**: Google acquisisce la Android inc.
- **Novembre 2007**: nasce la Open Handset Alliance, viene presentato ufficialmente il progetto Android e il primo SDK (IDE per vari OS, documentazione, emulatore e esempi di semplici app)
- **Settembre 2008**: lancio di Android 1.0 Apple Pie
- **Maggio 2010**: viene introdotto il Just In Time compiler (JIT)
- **Gennaio 2011**: nasce la versione 3.0 Honeycomb (solo per tablet)
- **Ottobre 2011**: versione tablet e smartphone si uniscono e nasce Android 4.0 Ice Cream Sandwich
- **Ottobre 2012**: viene rilasciato Android 4.1 Jelly Bean, viene introdotto il supporto multiutente
- **Ottobre 2013**: viene rilasciato Android 4.4 Kit-Kat che include una nuova virtual machine sperimentale (ART [8])

Un po' di Storia: Timeline [11]



Numero di Attivazioni di Device nel Tempo [4]

Market Share of Mobile Operating Systems (Q3 2013) [10]

Global Smartphone OS Shipments (Q3 2013)

Global Smartphone Operating System Shipments (Millions of Units)	Q3 '12	Q3 '13
Android	129.6	204.4
Apple	26.9	33.8
Microsoft	3.7	10.2
BlackBerry	7.4	2.5
Others	5.2	0.5
Total	172.8	251.4

- I numeri si riferiscono a milioni di unità vendute

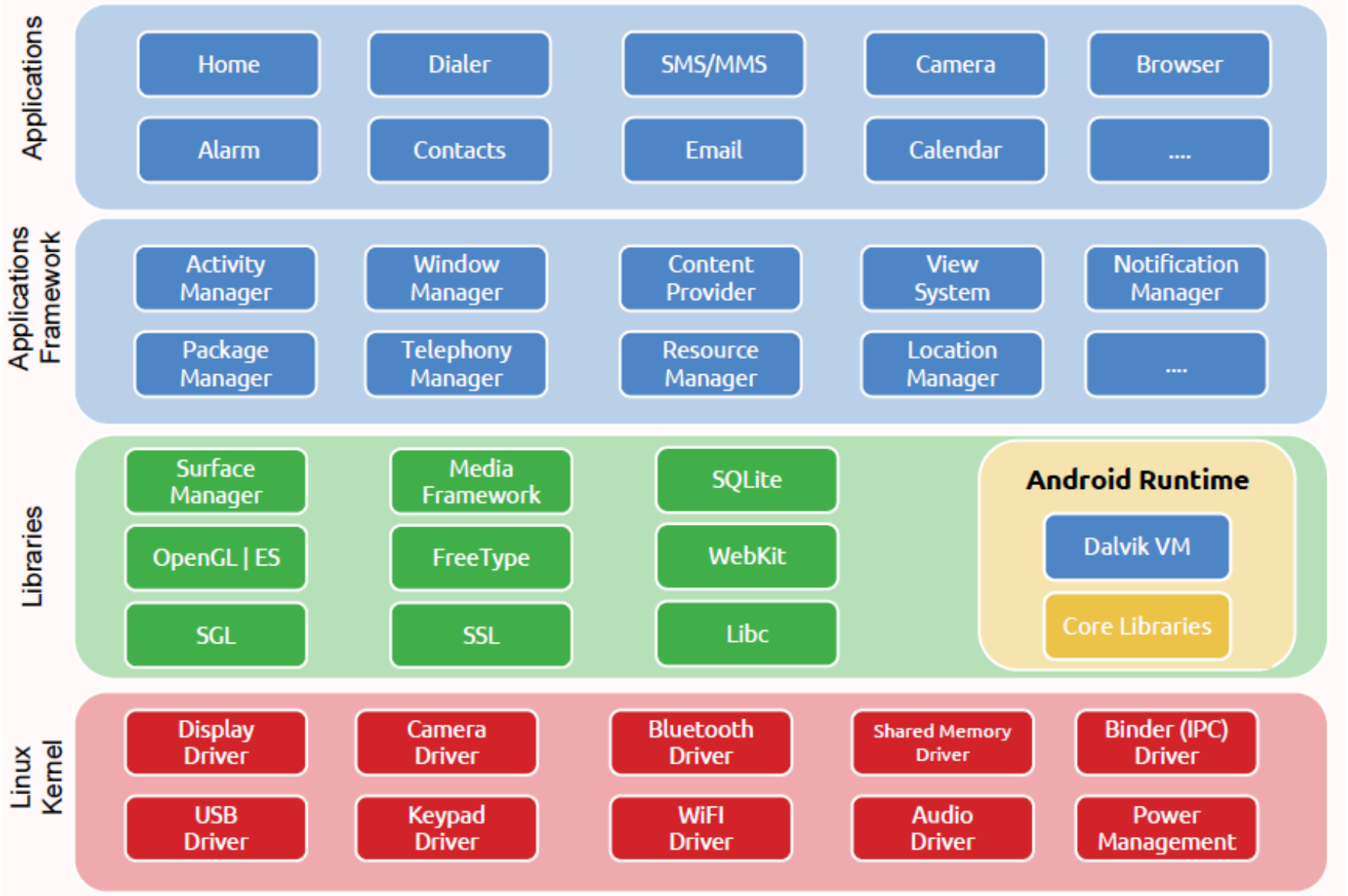
Market Share of Mobile Operating Systems (Q3 2013) [10]

Global Smartphone OS Market Share (Q3 2013)

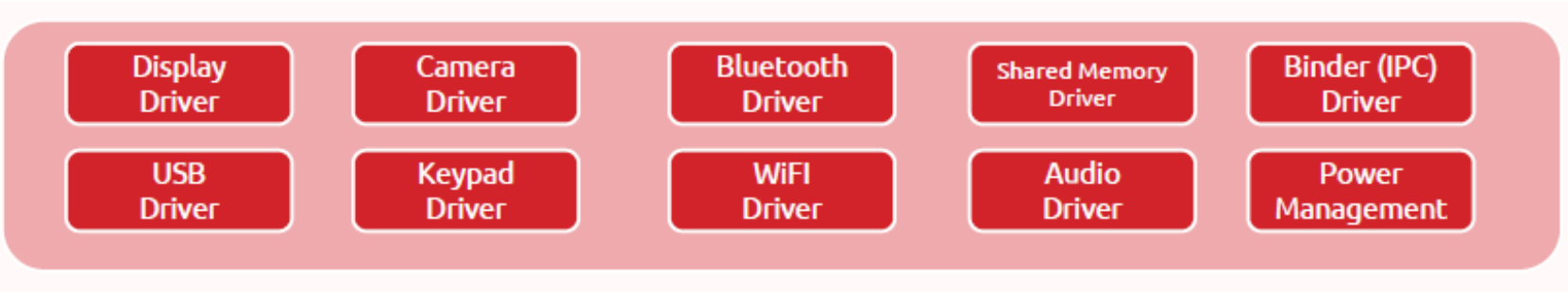
Global Smartphone Operating System Marketshare %	Q3 '12	Q3 '13
Android	75.0%	81.3%
Apple	15.6%	13.4%
Microsoft	2.1%	4.1%
BlackBerry	4.3%	1.0%
Others	3.0%	0.2%
Total	100.0%	100.0%

Outline

Android Architecture: General Overview

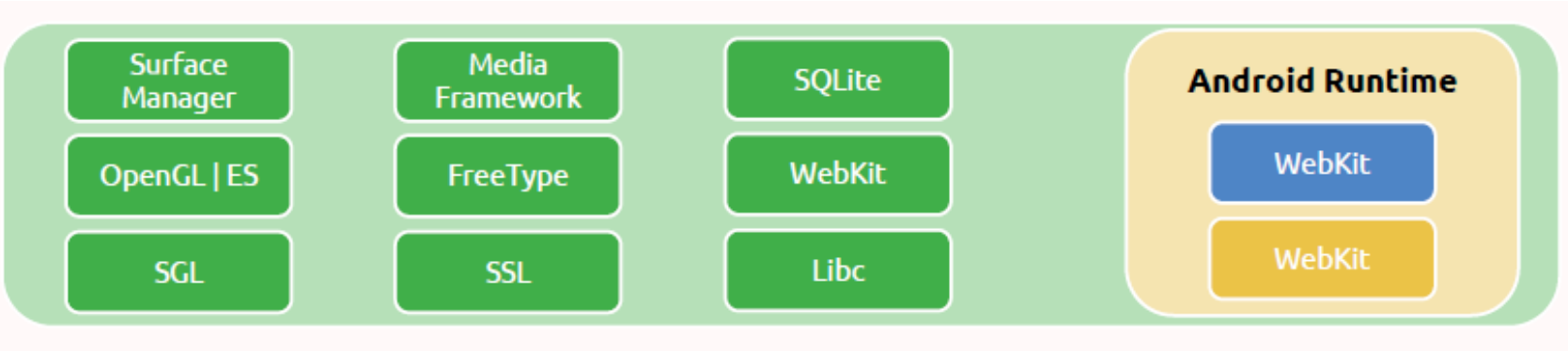


Android Architecture: Focus sul Livello Kernel



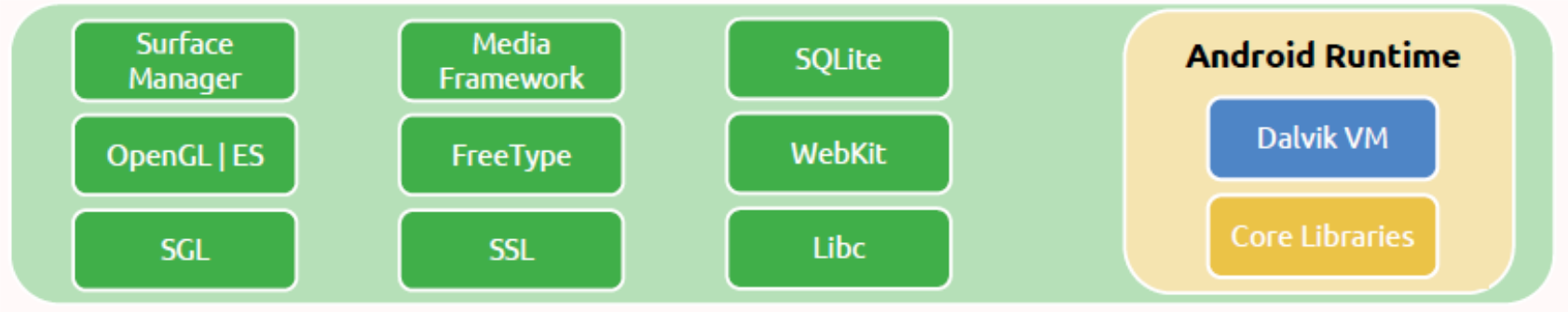
- Inizialmente Linux Kernel 2.6.x, ad oggi 3.4.x
  - Gestione dei processi, della memoria, network stack, drivers, etc.
- Drivers per: display, camera, keypad, WiFi, etc.
- Low Memory Killer
- Power Manager
- Binder (IPC) Driver: gestisce la comunicazione tra processi

Android Architecture: Focus sul Livello Librerie



- Surface Manager: gestisce il sottosistema di visualizzazione (2D-3D)
- Media Framework: gestisce i codec video e audio per i formati multimediali mp3, jpg..
- SSL: gestisce il secure socket layer
- Webkit: browser engine
- Libc: libreria C ottimizzata per dispositivi embedded basati su Linux
- ...

Android Architecture: Focus sul Runtime

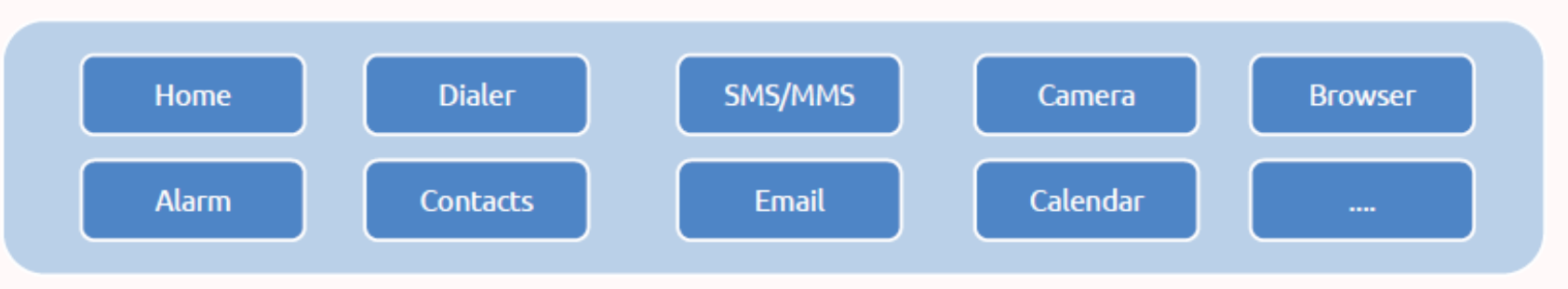


- Dalvik Virtual Machine
  - Java VM ottimizzata per dispositivi mobile
    - Esecuzione efficiente di più istanze della VM
  - Si appoggia a Linux Kernel per la gestione di processi thread e altri aspetti di basso livello (come la JVM)
  - Il *"dalvik bytecode"* è più leggero (occupa meno spazio su disco)
  - JIT compiler (introdotto dalla versione 2.2 di Android)
- Core Libraries
  - Subset delle librerie standard che troviamo nel JDK
  - Librerie specifiche per Dalvik
  - Librerie di terze parti (p.e. Apache HTTPClient)

Android Architecture: Focus sull'Application Framework

- Aspetti chiave del framework
  - Riuso di componenti e funzionalità
    - Per mezzo di **Intent** e **IntentFilter**
  - Ogni applicazione espone un set di funzionalità (**IntentFilters**) utilizzabili da una qualunque altra applicazione (via **Intent**)
- Servizi di sistema alla base di ciascuna applicazione
  - View System: per la creazione e gestione di UI
  - Content Providers: per la condivisione di dati tra applicazioni
  - Resource Manager: per la gestione delle risorse (immagini, layout UI, file multimediali, etc.)
  - Notification Manager: per gestire le notifiche da inviare all'utente
  - Activity Manager: gestice il ciclo di vita delle applicazioni

Android Architecture: Focus sul Livello Applicativo



- Applicazioni base (di sistema o di default)
  - Mail client, calendar, browser, contacts manager, etc.
- Le applicazioni (o parti di esse) possono essere scritte utilizzando
  - SDK [6]: per lo sviluppo di parti *"classiche"*, **interamente in Java** [6]
  - NDK [5]: per sviluppare componenti in codice nativo, a runtime però è sempre la Dalvik vm a gestirne l'esecuzione
    - Utile in particolare per implementare aspetti di basso livello
  - ADK [3]: per interagire con hardware esterno, solitamente per applicazioni domotiche/hobbistiche Arduino-based [1]
- Applicazioni distribuite in forma pachettizzata (**apk**)
  - Installabili sia tramite l'Android Market che manualmente

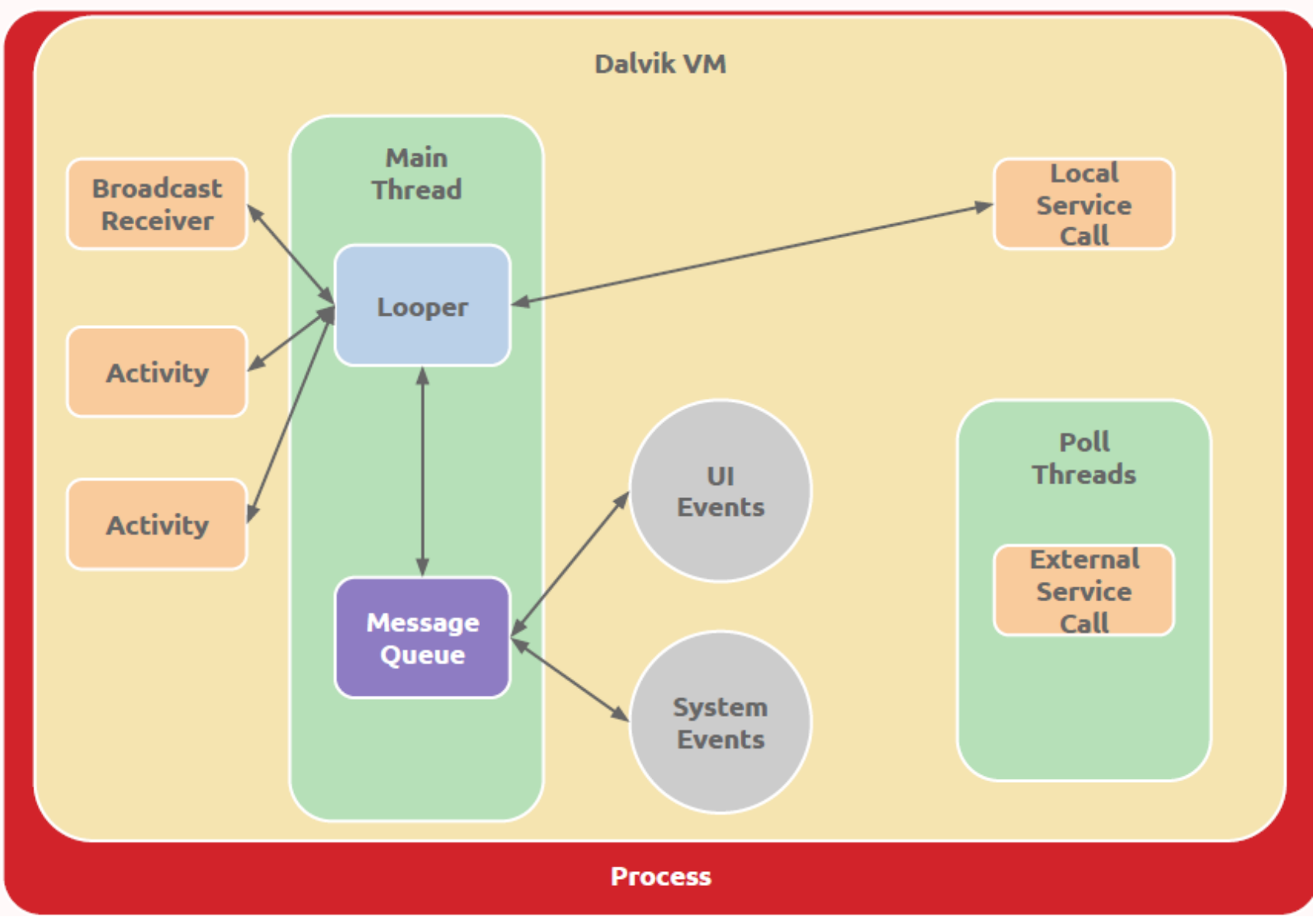
Outline

Componenti Base di una Applicazione Android

I componenti base forniti dall'Android SDK per sviluppare una app sono:

- **Activity**
  - Una generica view di una applicazione
- **Service**
  - Componente utilizzato per l'esecuzione in background di operazioni a lungo termine (p.e. playback di un mp3)
    - Non dispone di una interfaccia utente
- **Content Provider**
  - Componente che consente di accedere/modificare/esporre un set di dati in maniera indipendente dal formato di memorizzazione utilizzato
    - SQLite database, filesystem, web, etc.
  - Diverse applicazioni possono accedere allo stesso set di dati
- **Broadcast Receiver**
  - Componente che consente di ricevere notifiche relative a eventi di sistema e non

Architettura Generale di una Applicazione Android



Intent & Intent Filters 1/2

Intent & Intent Filters

Meccanismo chiave di Android per supportare il late-binding a run-time tra componenti della stessa applicazione o anche di applicazioni differenti

- Activities, services e broadcast receivers sono attivati da **Intent**

Punti chiave del meccanismo di late-binding

- I componenti di una applicazione dichiarano l'insieme di operazioni che sono in grado di svolgere definendo opportuni **IntentFilters**
- Un applicazione che ha necessità di svolgere una data operazione (p.e. mostrare la gui XXX, avviare la navigazione verso un certo punto, etc.) *"lancia"* uno specifico **Intent** con cui richiedere al sistema la funzionalità di interesse

Intent & Intent Filters 2/2

Concretamente, cos'è un Intent?

Un Intent è un oggetto che tiene traccia della descrizione astratta di una operazione che deve essere svolta (o nel caso di broadcast di un *"evento"* di interesse che viene notificato)

Chi gestisce gli Intent?

E l'Android runtime che gestisce il mapping tra un triggered Intent e i componenti in grado di soddisfarlo

Come lanciare un Intent?

Come vedremo, esistono diverse modalità di triggering di un Intent, dipendenti dallo specifico componente target in grado di soddisfarlo

Manifest File

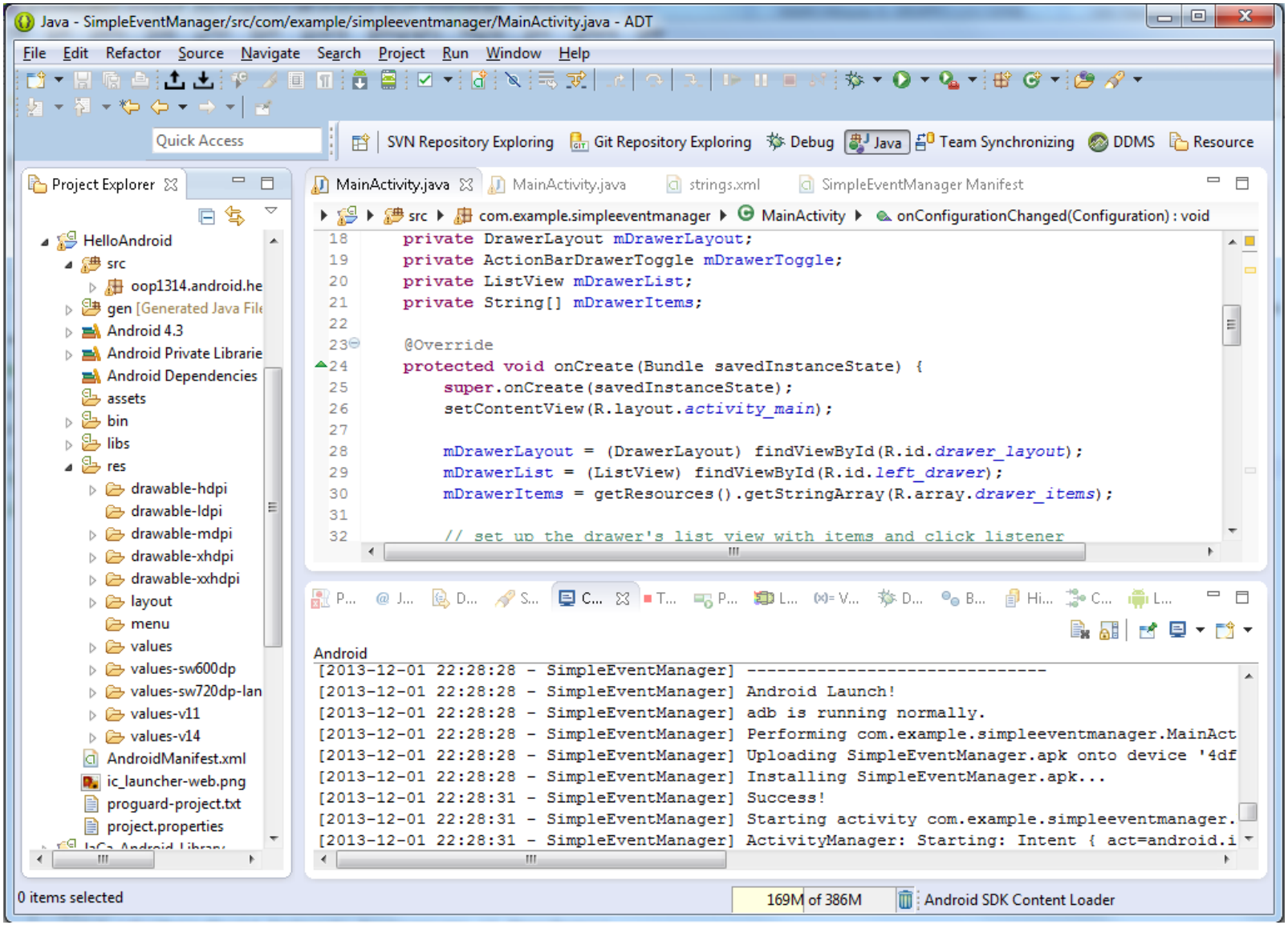
AndroidManifest.xml

Ogni applicazione deve avere un file **AndroidManifest.xml** (con questo esatto nome) nella root directory

Contiene informazioni essenziali relative all'applicazione:

- I componenti dell'applicazione (activities, services, etc.) e le loro capacità (IntentFilters)
- I permessi richiesti dall'applicazione
  - Per poter accedere alle varie funzionalità del device: connettività, contatti, etc.
- API level: indica il livello minimo di API che l'applicazione richiede e il livello massimo su cui è stata testata con successo
- La lista di librerie usate dall'applicazione

Android Integrated Development Environment

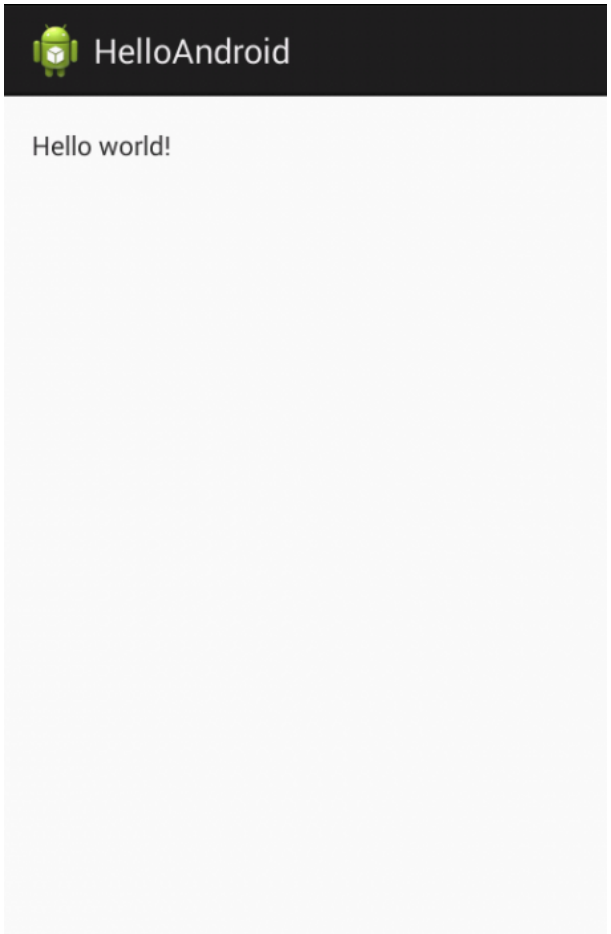


Struttura di Una Applicazione (base)?

- Deve avere almeno una activity
  - p.e. `MainActivity`
- Deve avere un manifest file (`AndroidManifest.xml`)
  - In cui `MainActivity` sia dichiarata
  - In cui vengono registrati opportuni `IntentFilter` per fare in modo che `MainActivity` sia lanciata come prima vista dell'applicazione (`MainActivity` diventa una sorta di *"main"* dell'applicazione)
- Può avere eventuali componenti aggiuntivi (activity, service, etc.)
  - Che dovranno essere dichiarati nel manifest

Outline

Primo Esempio di Applicazione: HelloWorld Android!



- Classica applicazione iniziale
  - Stampa del messaggio "Hello World!"
- L'intera applicazione è una sola Activity

HelloWorld Android: Analizziamo il Manifest

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="oop1313.android.hello"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk
8         android:minSdkVersion="8"
9         android:targetSdkVersion="18" />
10
11    <application
12        android:allowBackup="true"
13        android:icon="@drawable/ic_launcher"
14        android:label="@string/app_name"
15        android:theme="@style/AppTheme" >
16        <activity
17            android:name="oop1313.android.hello.HelloWorldActivity"
18            android:label="@string/app_name" >
19            <intent-filter>
20                <action android:name="android.intent.action.MAIN" />
21
22                <category android:name="android.intent.category.LAUNCHER" />
23            </intent-filter>
24        </activity>
25    </application>
26
27 </manifest>
```

Componenti dell'applicazione

Versione minima di API richiesta & versione massima di API supportata

Definizione di HelloWorldActivity

Definizione degli IntentFilters di HelloWorldActivity

- Gli IntentFilter specificati indicano che `HelloWorldActivity` è il *"main"* dell'app

HelloWorld Android: Analizziamo HelloWorldActivity

```
1  package oop1313.android.hello;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class HelloWorldActivity extends Activity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_hello_world);
12     }
13 }
```

Punti Chiave

- `onCreate` viene invocato da Android al momento della creazione dell'activity
  - E' una sorta di *"costruttore"*
- `setContentView` carica nella view il layout indicato da `R.layout.activity_hello_world`

HelloWorld Android: Analizziamo il Layout dell'Activity

Definizione di layout traminte file XML

Diversamente da Swing, SWT, etc. Android adotta un approccio XML-based per la definizione di GUI

- Ne parleremo più in dettaglio nella prossima lezione

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".HelloWorldActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Gestione delle Risorse in Android

Approccio di gestione delle risore dell'applicazione

Android definisce un approccio (semplice?) ed organizzato per la gestione delle risorse di una applicazione (p.e. stringhe, layout, immagini, etc.), tutte memorizzate in sotto-cartelle dedicate all'interno della cartella `res`

- Referencing delle risorse per mezzo del file `R.java` nei sorgenti java
  - Generato automaticamente
- `R.layout.XX` per accedere a views
- `R.strings.XX` per accedere a stringhe

Gestione automatica di diversi aspetti *"tediosi"*

- Scelta automatica dei valori sulla base della lingua del device
  - Valori specifici per ogni lingua definiti in: `values-it`, `values-en`..
- Scelta automatica del layout, stile etc. sulla base del valore di API
  - Valori specifici per ogni livello di API in: `values-v14`, `values-v15`..

HelloWorld Android: Analizziamo il File `strings.xml`

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloAndroid</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

- `hello_world`: stringa relativa al messaggio mostrato dall'app
- `app_name`: stringa relativa al nome dell'applicazione

Outline

Outline

Ciclo di vita di una Activity 1/2

Stato di una activity

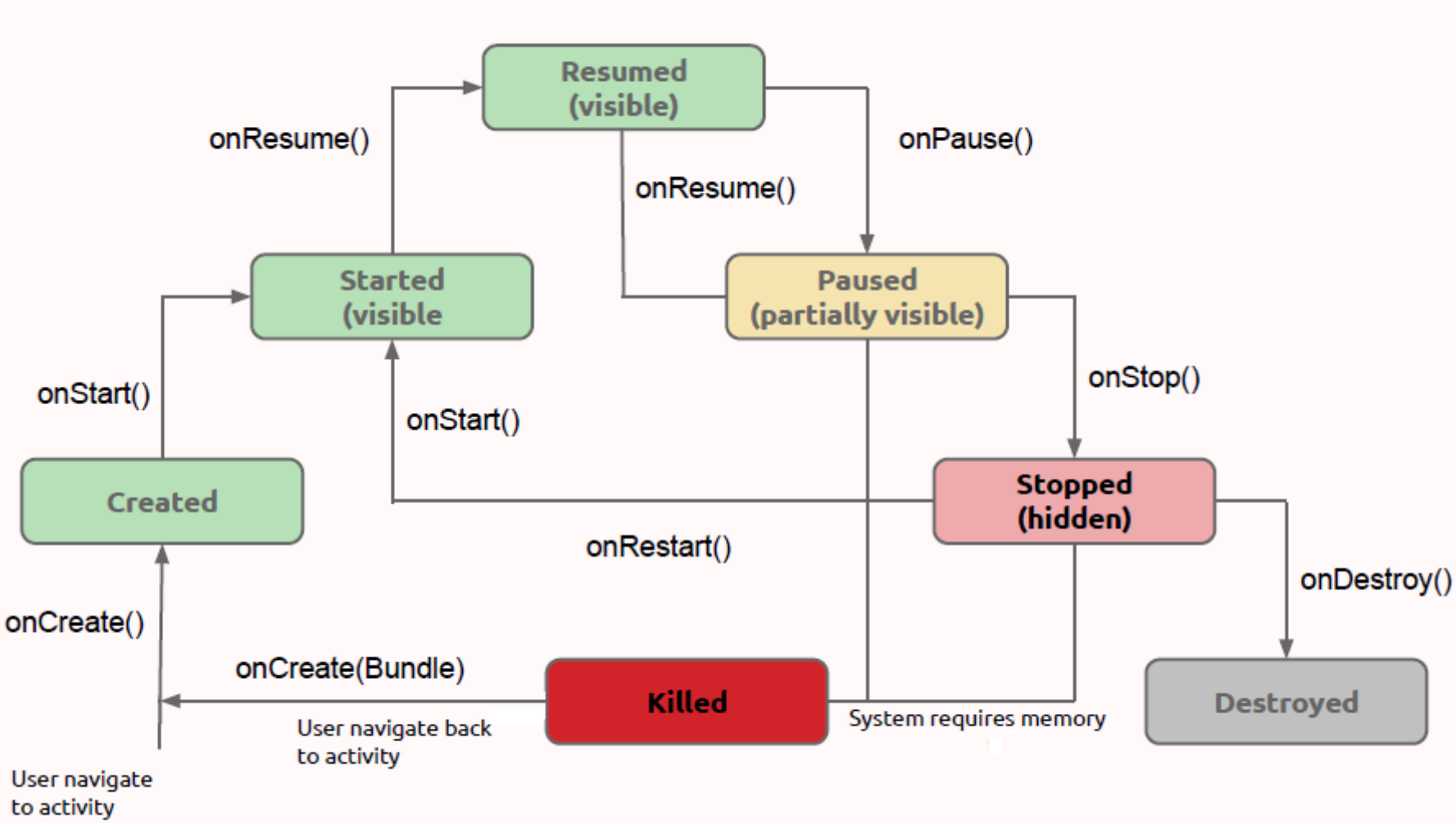
Una activity si può trovare in uno dei seguenti stati: active (o running), paused, stopped, killed

Activity Lifecycle

Sulla base delle interazioni dell'utente (e del sistema) con la view, lo stato di una activity varia da uno all'altro dei quattro appena citati

- Durante le transizione di stato Android invoca una serie di *lifecycle methods* predefiniti
- Lo sviluppatore deve fare l'override dei metodi di interesse per personalizzare secondo le sue esigenze il comportamento dell'activity durante le transizioni di stato

Ciclo di vita di una Activity 2/2

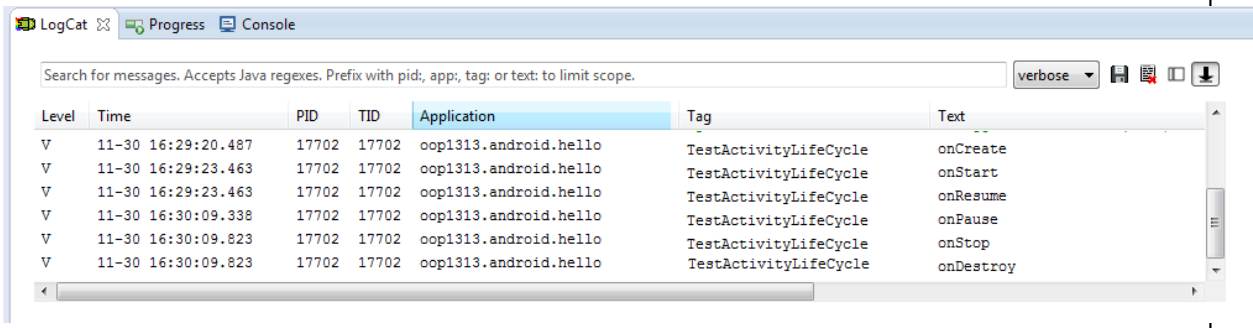


Activity Life Cycle Methods

- **onCreate**
  - Invocato al momento della creazione dell'activity
  - Deve occuparsi di: creare le view, fare il bind dei dati ai vari componenti, etc.
- **onStart**
  - Invocato quando l'activity sta per essere resa visibile all'utente
- **onResume**
  - Invocato prima di portare l'activity in foreground
- **onPause**
  - Invocato quando il sistema sta effettuando il resume di una activity
  - Seguito dall'invocazione di `onStop` quando l'activity corrente viene spostata in background
  - Seguito dall'invocazione di `onResume` quando l'activity corrente viene spostata in foreground
- **onStop**
  - Invocato prima di portare l'activity in background
- **onDestroy**
  - Invocato prima della terminazione dell'activity

Test Activity Life Cycle Methods

```
1  public class TestActivityLifeCycle extends Activity {
2    private static final String TAG = "TestActivityLifeCycle";
3    @Override
4    protected void onCreate(Bundle savedInstanceState) {
5      super.onCreate(savedInstanceState); Log.v(TAG, "onCreate");
6    }
7    @Override
8    protected void onStart() {
9      super.onStart(); Log.v(TAG, "onStart");
10   }
11   @Override
12   protected void onResume() {
13     super.onResume();
14     Log.v(TAG, "onResume");
15   }
16
17   @Override
18   protected void onPause() {
19     super.onPause();
20     Log.v(TAG, "onPause");
21   }
22   @Override
23   protected void onStop() {
24     super.onStop();
25     Log.v(TAG, "onStop");
26   }
27   @Override
28   protected void onDestroy() {
29     super.onDestroy();
30     Log.v(TAG, "onDestroy");
31 }}
```



Lancio di una Nuova Activity

Come aprire una nuova view?

Per lanciare una view di una applicazione (o di una applicazione differente) è necessario appoggiarsi al meccanismo degli intent

Start di una nuova view con intent esplicito

Un intent esplicito fa riferimento direttamente alla specifica activity da lanciare (indipendentemente dalle specifiche capacità offerte)

- E' noto a priori il componente (activity) che verrà lanciato

Start di una nuova view con intent implicito

Un Intent implicito specifica la funzionalità/capacità richiesta, indipendentemente dallo specifico componente in grado di realizzarla

- Più componenti potrebbero offrire la stessa funzionalità
- In caso di più opzioni, Android lascia all'utente la scelta

Lancio di una Nuova Activity Tramite Intent Esplicito

```
1  package oop1313.android.hello;
2
3  public class MainActivity extends Activity {
4    private Button mBtn;
5    @Override
6    protected void onCreate(Bundle savedInstanceState) {
7      super.onCreate(savedInstanceState);
8      setContentView(R.layout.activity_main);
9      mBtn = (Button) findViewById(R.id.open_btn);
10     mBtn.setOnClickListener(new OnClickListener() {
11       @Override
12       public void onClick(View v) {
13         Intent intent = new Intent(getApplicationContext(), HelloWorldActivity.class);
14         startActivity(intent);
15       }
16     });
17 }}
```

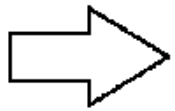
- `MainActivity` ha il compito di lanciare `HelloWorldActivity` a seguito del click su un pulsante
- Dovremo aggiornare di conseguenza il manifest e il layout

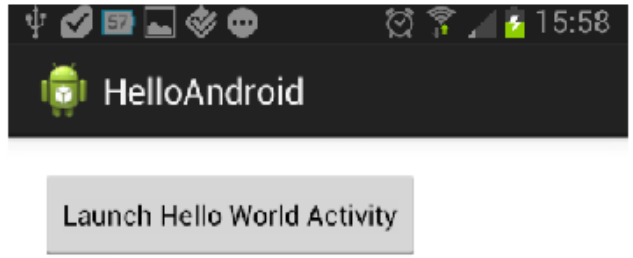
Modifiche a Manifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="oop1313.android.hello"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="oop1313.android.hello.HelloWorldActivity"
            android:label="@string/app_name" />

        <activity
            android:name="oop1313.android.hello.MainActivity"
            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>
```

Modifiche al Layout

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      tools:context=".HelloWorldActivity" >
10
11     <Button
12         android:id="@+id/open_btn"
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/open_hello_world" />
16
17 </RelativeLayout>
```





Id dei Componenti e loro Lookup

Definizione di id per i componenti di interesse

- Ai componenti grafici (bottoni, checkbox, etc.) definiti nei layout può essere assegnato un id
  - Univoco all'interno dell'applicazione, fondamentale per il loro lookup
- Assegnamento per mezzo dell'attributo XML `id`
  - `<Button android:id=@+id/open_btn .. >`
- L'operatore '+' va utilizzato solo al momento della dichiarazione, per generare un nuovo id
  - Da lì in poi il componente viene riferito direttamente tramite l'id, senza il '+' (p.e. quando si specifica l'allineamento di altri componenti rispetto a componenti già esistenti)
- Un insieme di valori `R.id.X` sono generati dati gli Id definiti

Lookup dei Componenti

- La classe Activity fornisce il mentodo `findViewById(id)` con cui recuperare una View a partire dall'id specificato

Lancio di una Nuova Activity Tramite Intent Implicito

```
1  package oop1314.android.hello;
2
3  public class MainActivity extends Activity {
4
5    private Button mBtn;
6    private static final String OPEN_HELLO_ACTION ="oop1314.actions.openhello";
7
8    @Override
9    protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.activity_main);
12     mBtn = (Button) findViewById(R.id.open_btn);
13     mBtn.setOnClickListener(new OnClickListener() {
14
15       @Override
16       public void onClick(View v) {
17         Intent i = new Intent(OPEN_HELLO_ACTION);
18         startActivity(i);
19       }
20     });
21 }}
```

- `MainActivity` ha il compito di lanciare una activity in grado di ripondere all'intent a seguito del click su un pulsante
- Dovremo aggiornare di conseguenza il manifest...

Modifiche a Manifest.xml

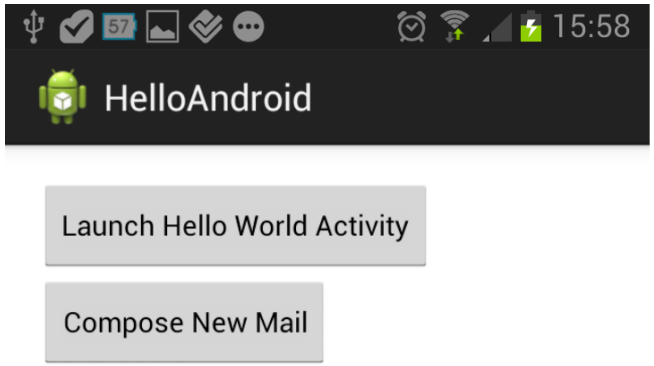
```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2      package="oop1313.android.hello"
3      android:versionCode="1"
4      android:versionName="1.0" >
5      ...
6      <application
7          android:allowBackup="true"
8          android:icon="@drawable/ic_launcher"
9          android:label="@string/app_name"
10         android:theme="@style/AppTheme" >
11
12         <activity
13             android:name="oop1314.android.hello.HelloWorldActivity"
14             android:label="@string/app_name">
15             <intent-filter>
16                 <action android:name="oop1314.actions.openhello"/>
17                 <category android:name="android.intent.category.DEFAULT"/>
18             </intent-filter>
19         </activity>
20
21         <activity
22             android:name="oop1314.android.hello.MainActivity"
23             android:label="@string/app_name">
24             <intent-filter>
25                 <action android:name="android.intent.action.MAIN" />
26                 <category android:name="android.intent.category.LAUNCHER" />
27             </intent-filter>
28         </activity>
29
30     </application>
31 </manifest>
```

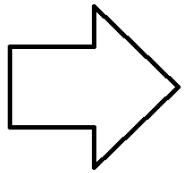
Intent Implicito: Activity per Scrivere una Mail 1/2

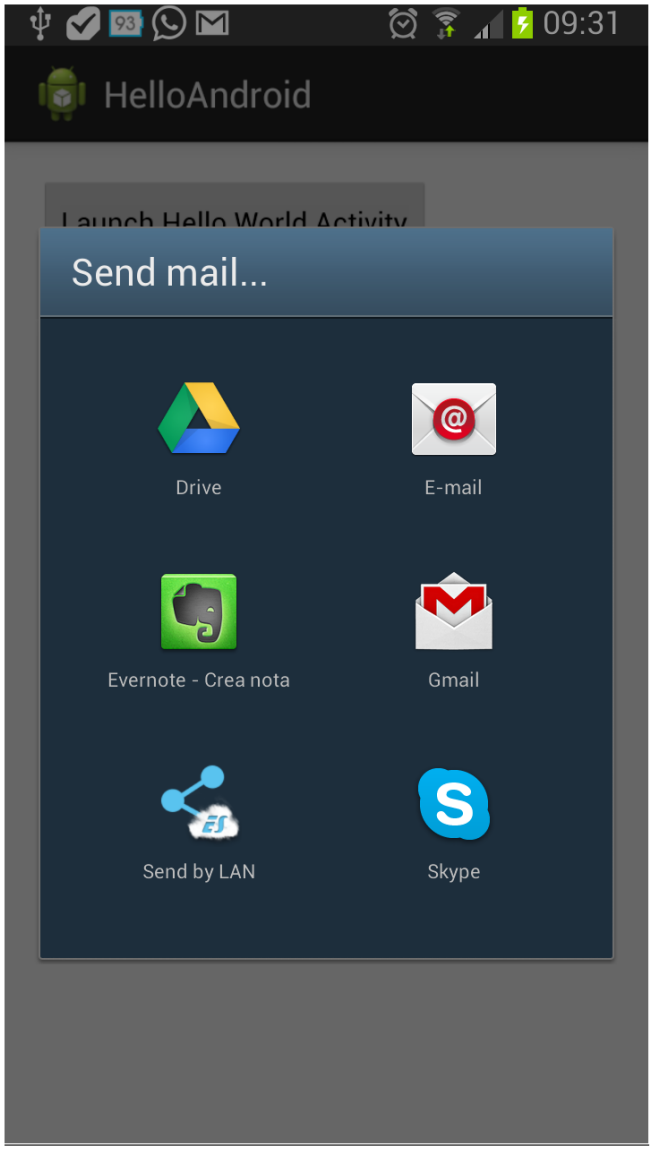
```
1  package oop1314.android.hello;
2
3  public class MainActivity extends Activity {
4    private Button mBtnComposeMail;
5    ...
6    @Override
7    protected void onCreate(Bundle savedInstanceState) {
8      super.onCreate(savedInstanceState);
9      setContentView(R.layout.activity_main);
10     mBtnComposeMail = (Button) findViewById(R.id.compose_mail);
11     ...
12     mBtnComposeMail.setOnClickListener(new OnClickListener() {
13       @Override
14       public void onClick(View v) {
15         Intent i = new Intent(Intent.ACTION_SEND);
16         i.setType("message/rfc822");
17         i.putExtra(Intent.EXTRA_EMAIL, "example@example.com");
18         i.putExtra(Intent.EXTRA_SUBJECT, "This will be the mail subject");
19         i.putExtra(Intent.EXTRA_TEXT, "This will be the mail text");
20         startActivity(Intent.createChooser(i, "Send mail..."));
21       }
22     });
23 }}
```

- `MainActivity` ha ora un ulteriore bottone e relativo listener
- Al click, viene lanciato un intent *"targetting"* una activity in grado di gestire il compose di una mail

Intent Implicito: Activity per Scrivere una Mail 2/2







Task and Back Stack 1/3 [7]

Premessa

Data la natura dei dispositivi, fornire una esperienza d'uso semplice e immediata è un aspetto fondamentale

- Evitare che l'utente "si perda" tra l'insieme di attività/app in esecuzione
- Necessità di un razionale organizzativo relativo alle attività/app

La risposta di Android: Task & Back Stack

- Un task è una collezione di activity con cui l'utente interagisce quando esegue un certo compito
- Le activity di un task sono memorizzate in uno stack (detto *"back stack"*) nell'ordine in cui sono state aperte

Task and Back Stack 2/3 [7]

Esempio di Dinamica di costruzione di Task e Back Stack

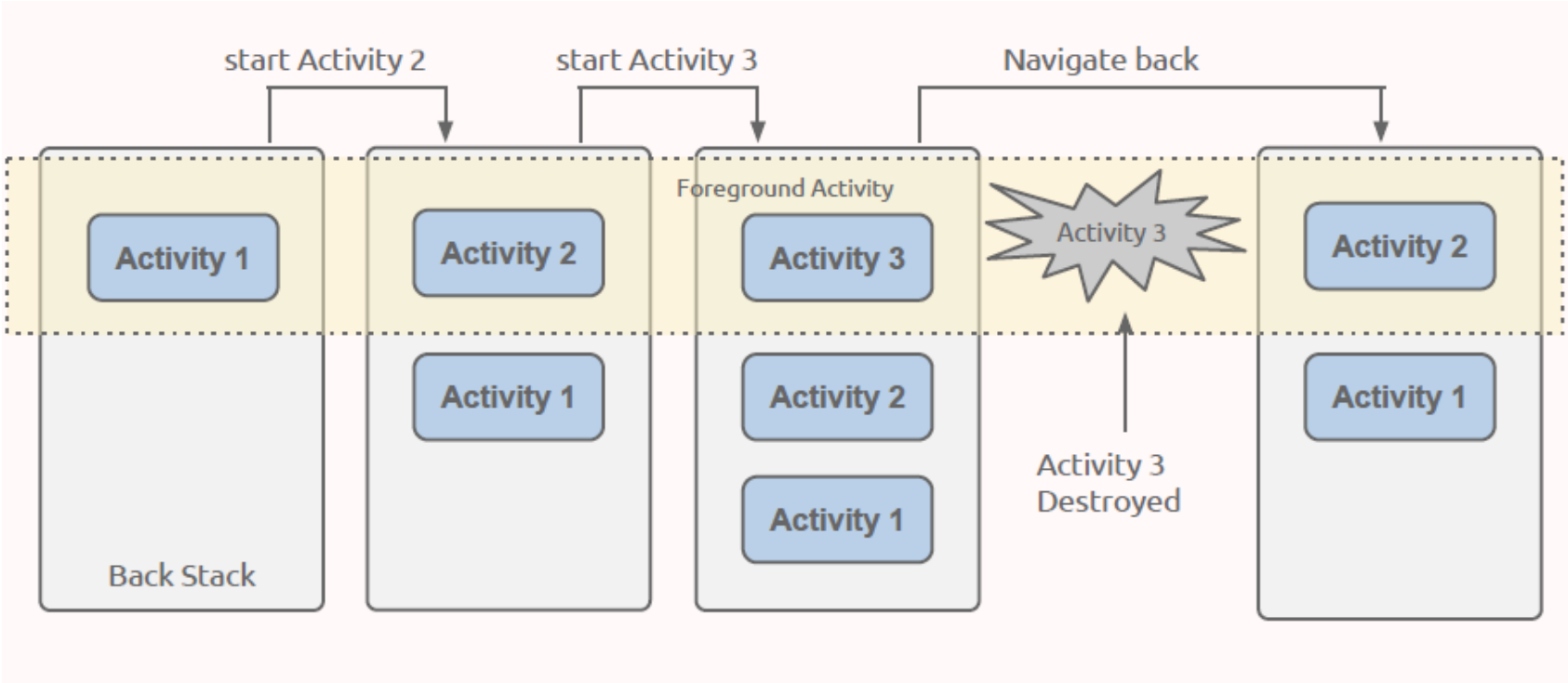
- L'utente sceglie una applicazione dalla home
- Viene creato un nuovo task e la prima activity della app lanciata viene messa sullo stack
- Vengono lanciate due nuova activity che vengono impilate sullo stack
- Alla pressione del pulsante back viene tolta dallo stack l'activity *"top"*

Creazione di un nuovo Task

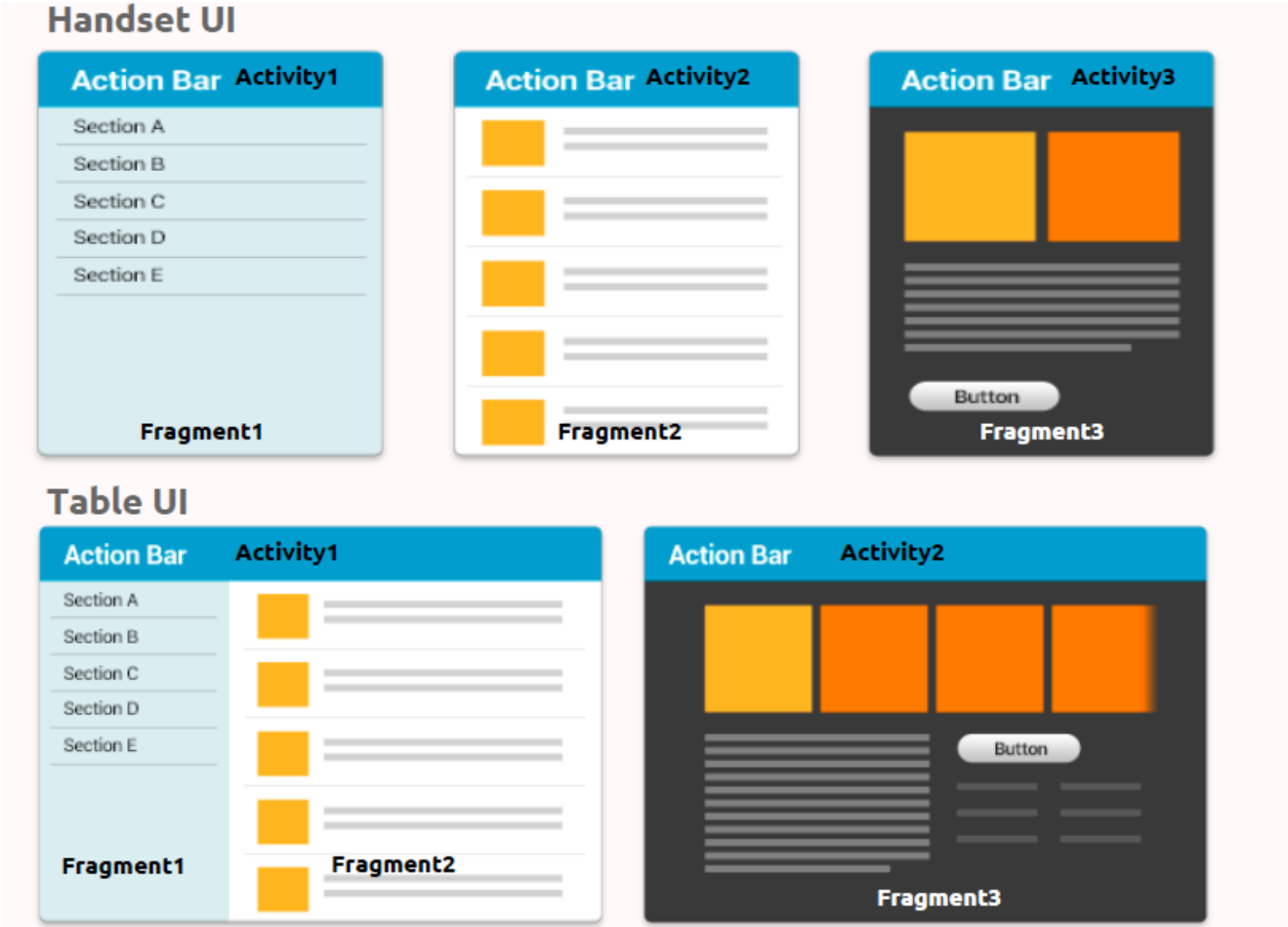
In ogni momento l'utente ha la facoltà di avviare un nuovo task (con relativo back stack) semplicemente premento il tasto *"home"*

- Lo stack precedente viene *"congelato"* e portato in background
- Viene creato un nuovo stack (in foreground)
- C'è la possibilità di riprendere l'esecuzione di task precedenti portati in background (p.e. scegliendoli dal task manager)

Task and Back Stack 3/3 [7]



Activities & Fragments



Ne parleremo più in dettaglio la prossima lezione

Outline

Service

Cos'è un service?

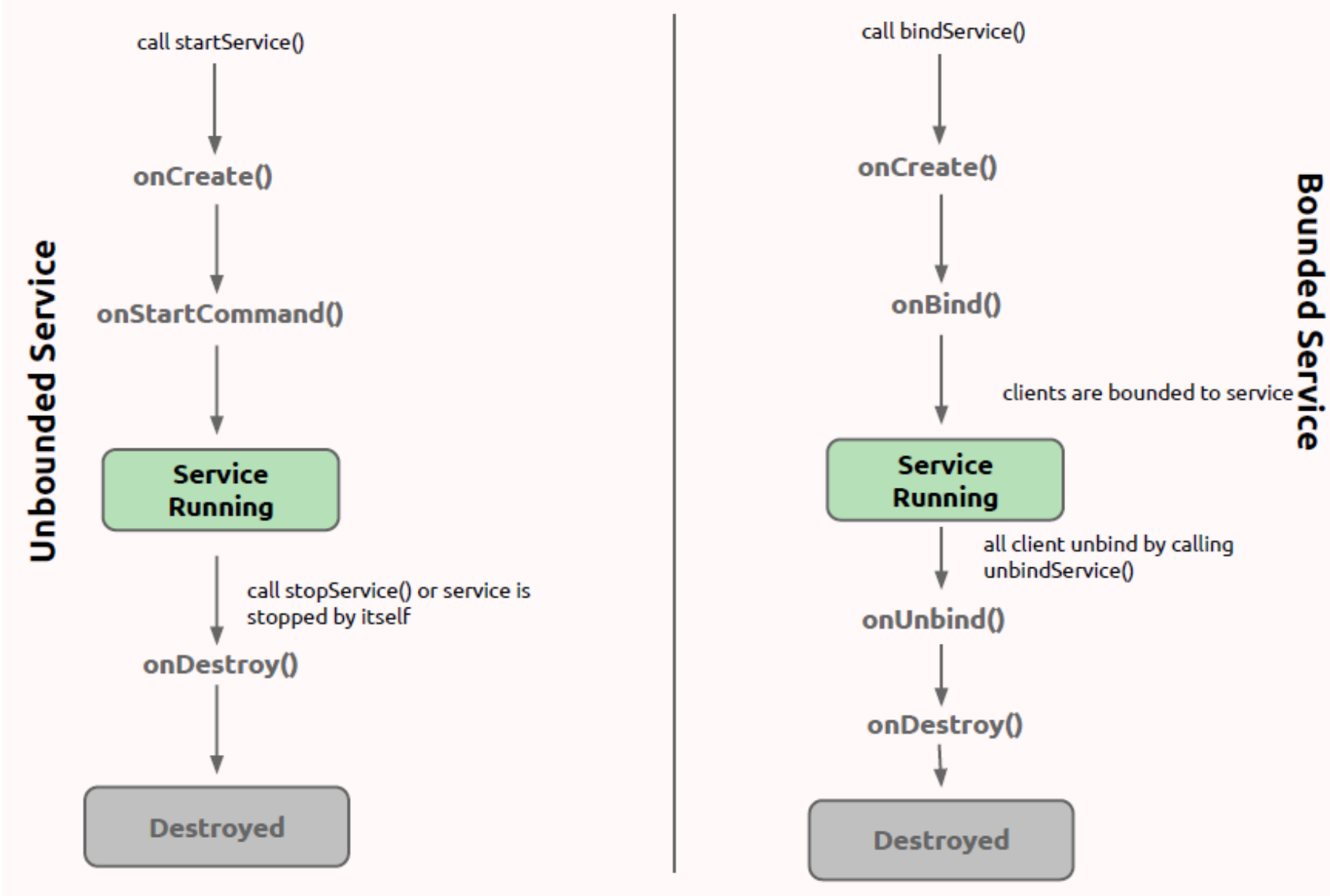
Componente utilizzato per l'esecuzione in background di operazioni a lungo termine

- Eseguito nello stesso thread dell'applicazione (di default)
- Ciclo di vita indipendente dal componente che l'ha creato

Due diversi tipi di servizi

- **Started Service**
  - Utilizzato (solitamente) per eseguire singole operazioni in background
- **Bound Service**
  - Utilizzato come un server in una architettura client/server
  - Inizialmente i client fanno la bind al servizio, e vi interagiscono invocando i metodi esposti (una sorta di pattern proxy)
    - Le richieste possono coinvolgere anche IPC
  - Alla fine i client fanno l'unbind dal servizio

Ciclo di Vita di un Servizio



Un Esempio Concreto: FileDownloaderService

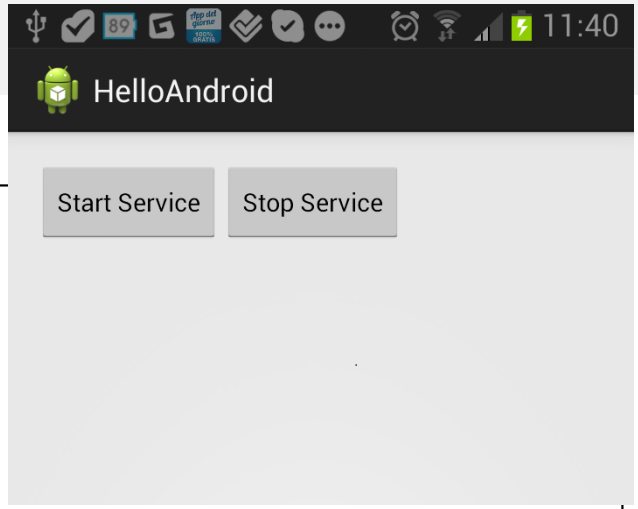
Utilizziamo un service per gestire il download in background di un file

- La relazione d'esempio di ACME-Exams

Note

- Utilizzeremo un IntentService [9]
  - Estensione della classe Service già pensata per gestire in un thread separato le richieste inviate
  - L'utente deve solo preoccuparsi di fare l'override del metodo `onHandleIntent`
    - Le varie richieste inviate al servizio vengono memorizzate in una coda interna e passate sequenzialmente al metodo `onHandleIntent`, eseguito dal thread dedicato
- `android.permission.INTERNET`
  - Va aggiunto al manifest per poter accedere alla rete
- Non ci cureremo di notificare all'utente la fine del download
- Il file è fisso (sempre la relazione)

WorkWithService Activity




```
1  public class WorkWithService extends Activity
2                    implements OnClickListener{
3    private Button mStartSrvBtn;
4    private Button mStopSrvBtn;
5    @Override
6    protected void onCreate(Bundle savedInstanceState) {
7      super.onCreate(savedInstanceState);
8      setContentView(R.layout.activity_work_with_service);
9      mStartSrvBtn = (Button) findViewById(R.id.start_service);
10     mStopSrvBtn = (Button) findViewById(R.id.stop_service);
11     mStartSrvBtn.setOnClickListener(this);
12     mStopSrvBtn.setOnClickListener(this);
13   }
14    @Override
15   public void onClick(View v) {
16     switch (v.getId()){
17       case R.id.start_service:
18         startService(new Intent(this, FileDownloaderService.class));
19       break;
20       case R.id.stop_service:
21         stopService(new Intent(this, FileDownloaderService.class));
22       break;
23 }}}
```

- Avvia/termina il servizio a seguito del click sui pulsanti

FileDownloaderService

```
1  public class FileDownloaderService extends IntentService {
2    private static final String FILE_URL = "https://...relazione.pdf";
3
4    public FileDownloaderService(){ super("FileDownloaderService"); }
5
6    @Override
7    protected void onHandleIntent(Intent arg0) {
8      BufferedInputStream in = null; OutputStream out = null;
9      try {
10       URL url = new URL(FILE_URL);
11       in = new BufferedInputStream(url.openStream());
12       out = new FileOutputStream(Environment.getExternalStorageDirectory()
13                                  .getPath()+File.separator+"relazione.pdf");
14       byte data[] = new byte[1024];
15       int count =0;
16       while ((count = in.read(data)) != -1) {
17         out.write(data, 0, count);
18       }
19     } catch (Exception e) {
20     } finally {
21       ... /* closing streams */
22   }}
23
24   @Override
25   public void onDestroy() {
26     super.onDestroy();
27     Log.v("FileDownloaderService", "destroyed!");
28 }}
```

Modifiche Apportate al Manifest

```
1  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2      package="oop1313.android.hello"
3      android:versionCode="1"
4      android:versionName="1.0" >
5
6      <uses-permission android:name="android.permission.INTERNET" />
7      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
8
9      <uses-sdk
10         android:minSdkVersion="8"
11         android:targetSdkVersion="18" />
12
13     <application
14         android:allowBackup="true"
15         android:icon="@drawable/ic_launcher"
16         android:label="@string/app_name"
17         android:theme="@style/AppTheme" >
18         <activity
19             android:name="oop1314.android.hello.WorkWithService"
20             android:label="@string/app_name" >
21             <intent-filter>
22                 <action android:name="android.intent.action.MAIN" />
23
24                 <category android:name="android.intent.category.LAUNCHER" />
25             </intent-filter>
26         </activity>
27         <service android:name="oop1314.android.hello.FileDownloaderService" />
28     </application>
29
30 </manifest>
```

Avvio di un Service In Maniera Implicita/Esplicita

- Nell'esempio appena mostrato abbiamo avviato un Service in maniera implicita
  - Specificando nell'intent la classe del Service da avviare
- E' possibile avviare servizi anche in maniera implicita analogamente a quanto fatto per le activity
  - Appoggiandosi a Intent & IntentFilters
- In pratica però l'avvio di servizi in maniera implicita è poco comune

Outline

Content Provider

Cos'è un Content Provider

Componente che gestisce l'accesso/modifica/cancellazione/inserimento di un insieme di dati dell'applicazione, rendendoli disponibili - al bisogno - anche ad applicazioni esterne

- Un content provider può anche essere visto come un REST [2] enabled data provider..
- .. cioè consente di accedere con un approccio REST ai dati gestiti
- Sono solitamente usati come *wrapper* per database
  - Possono in realtà esporre anche dati memorizzati su file, etc.

Quando usarli?

Principalmente quando vi è la necessità di esporre i dati (o parte di essi) della nostra applicazione

Uso di Content Provider di Default

Content Provider di Default

Android fornisce due importanti content provider di default: calendar & contacts provider

Contacts Provider

Gestisce i dati relativi ai contatti dell'utilizzatore del device

- Tutti i contatti (da account google, provenienti dalla SIM, etc.)
- Utilizzabile da applicazioni (con i giusti permessi) per: recuperare informazioni sui contatti, editarli, etc.

Calendar Provider

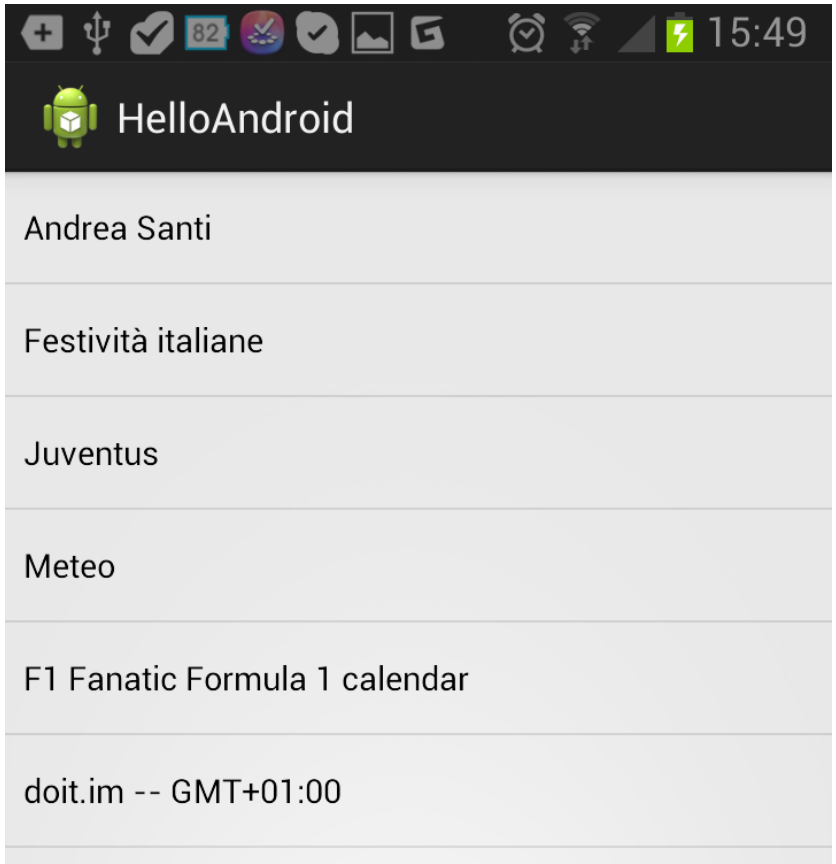
Gestisce i dati relativi ai calendari dell'utente

- Tutti i calendari (google, exchange, etc.)
- Utilizzabile da applicazioni (con i giusti permessi) per: recuperare informazioni su eventi, cancellarli, etc.

Esempio d'Uso del Calendar Provider

Obiettivo

Mostrare l'elenco dei calendari presenti sul telefono



ListCalendars Activity

```
1  package oop1314.android.hello;
2
3  public class ListCalendars extends ListActivity{
4
5    public void onCreate(Bundle icicle) {
6      super.onCreate(icicle);
7
8      final String[] EVENT_PROJECTION = new String[] {
9        Calendars.CALENDAR_DISPLAY_NAME,
10       Calendars._ID
11     };
12
13     // Run query
14     Cursor cur = null;
15     ContentResolver cr = getContentResolver();
16     Uri uri = Calendars.CONTENT_URI;
17     String selection = "(" + Calendars.ACCOUNT_TYPE + " = ?)";
18     String[] selectionArgs = new String[] {"com.google"};
19     // Submit the query and get a Cursor object back.
20     cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
21     //Update the listview data
22     setListAdapter(new SimpleCursorAdapter(this, android.R.layout.simple_list_item_1,
23         cur, EVENT_PROJECTION, new int[]{android.R.id.text1}, 0));
24     }
25 }
```

Cosa Abbiamo utilizzato di Nuovo?

- `ListActivity`
  - Classe di utility per realizzare view contenenti liste di elementi
- `ContentResolver`
  - Oggetto che consente di effettuare query su content provider
- `Cursor`
  - Oggetto che tiene traccia degli elementi recuperati dal content provider
    - Analogo a un Iterator
  - Ne approfondirete l'uso nel corso di basi di dati
- `SimpleCursorAdapter` (pattern Adapter)
  - Gestisce il mapping dei dati memorizzati nel cursore in componenti della view
- `R.layout.simple_list_item_1` & `android.R.id.text1`
  - Fanno parte di un set di view e id di base presenti in Android
- `android.permission.READ_CALENDAR`
  - Nuovo permesso da aggiungere al manifest

Outline

Broadcast Receiver

Cos'è un Broadcast Receiver

E' un componente che consente di ricevere eventi (intent) inviati dal sistema o da applicazioni

Quando usarli?

- Per ricevere eventi di sistema di interesse (altrimenti non intercettabili)
  - Lo schermo è stato spento
  - Cambiamento del livello di carica della batteria
  - ...
- Per gestire eventi provenienti da altri componenti (p.e. ricevere un broadcast da un service che ha finito il suo compito)

Esempio di Broadcast Receiver

Obiettivo

Monitorare i cambiamenti di connettività (WiFi/reti mobili) sul device

```
1  public class MyConnectivityStatusReceiver extends BroadcastReceiver {
2    @Override
3    public void onReceive(Context context, Intent intent) {
4      Log.v("MyConnectivityStatusReceiver", "action: " + intent.getAction());
5      Log.v("MyConnectivityStatusReceiver", "component: " + intent.getComponent());
6      Bundle extras = intent.getExtras();
7      if (extras != null) {
8        for (String key: extras.keySet()) {
9          Log.v("MyConnectivityStatusReceiver", "key [" + key + "]: " + extras.get(key));
10       }
11     }
12   }
13 }
```

- E' necessario estendere la classe base `BroadcastReceiver`
- Va fatto l'override del metodo `onReceive` in cui viene gestita la ricezione di intent di interesse

Registrazione di un Broadcast Receiver nel Manifest

```
<receiver android:name="oop1314.android.hello.MyConnectivityStatusReceiver" >
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

- Definizione degli intent che siamo interessati a ricevere

Outline

Bibliography I

[1] Arduino Inc.

Arduino Official Webpage.

Online, available at: `http://arduino.cc/` – Last Retrieved: November 29, 2013.

[2] Roy T Fielding and Richard N Taylor.

Principled design of the modern web architecture.

ACM Transactions on Internet Technology (TOIT), 2(2):115–150, 2002.

[3] Google Inc.

Android ADK Official Webpage.

Online, available at: `http://developer.android.com/tools/adk/index.html` – Last Retrieved: November 29, 2013.

Bibliography II

[4] Google Inc.

Android General Information Webpage.

Online, available at: http://developer.android.com/about/index.html – Last Retrieved: November 29, 2013.

[5] Google Inc.

Android NDK Official Webpage.

Online, available at: http://developer.android.com/tools/sdk/ndk/index.html – Last Retrieved: November 29, 2013.

[6] Google Inc.

Android SDK Official Webpage.

Online, available at: https://developer.android.com/sdk/index.html – Last Retrieved: November 29, 2013.

Bibliography III

[7] Google Inc.

Android Task & Back Stack Official Documentation Page.

Online, available at: http://developer.android.com/guide/components/tasks-and-back-stack.html – Last Retrieved: November 29, 2013.

[8] Google Inc.

ART Virtual Machine Official Documentation.

Online, available at: http://source.android.com/devices/tech/dalvik/art.html – Last Retrieved: November 27, 2013.

[9] Google Inc.

IntentService Official Documentation.

Online, available at: http://developer.android.com/reference/android/app/IntentService.html – Last Retrieved: November 30, 201

Bibliography IV

[10] International Data Corporation (IDC).

Worldwide Quarterly Mobile Phone Tracker (Q3 2013).

Online, available at: http://en.wikipedia.org/wiki/Android_%28operating_system%29 – Last Retrieved: November 27, 2013.

[11] Satya Komatineni and Dave MacLean.

Pro Android 4.

Apress, 2012.

[12] Wikimedia Foundation Inc.

Official Android Page on Wikipedia.

Online, available at: http://en.wikipedia.org/wiki/Android_%28operating_system%29 – Last Retrieved: November 27, 2013.