Services

Servizi

- Componenti che avviano operazioni in background
  - i loro thread sopravvivono di più di quelli delle activity nascoste
- Specializzazioni della classe Context
  - come le activity
- Le operazioni dei Service vengono eseguite nello stesso processo che li ha invocati.
  - i thread creati

Dichiarazione

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Tipi di servizi

- Due tipi di servizi:
  - Started
    - vengono avviati e sono "indipendenti"
  - Bounded
    - accessibili da componneti dall'app o di altre app

- Interfaccia Servizi started:
  - permette lo start e lo stop
- Interfaccia Servizi bounded
  - ottengo il riferimento al servizio e mi ci collego (binding)

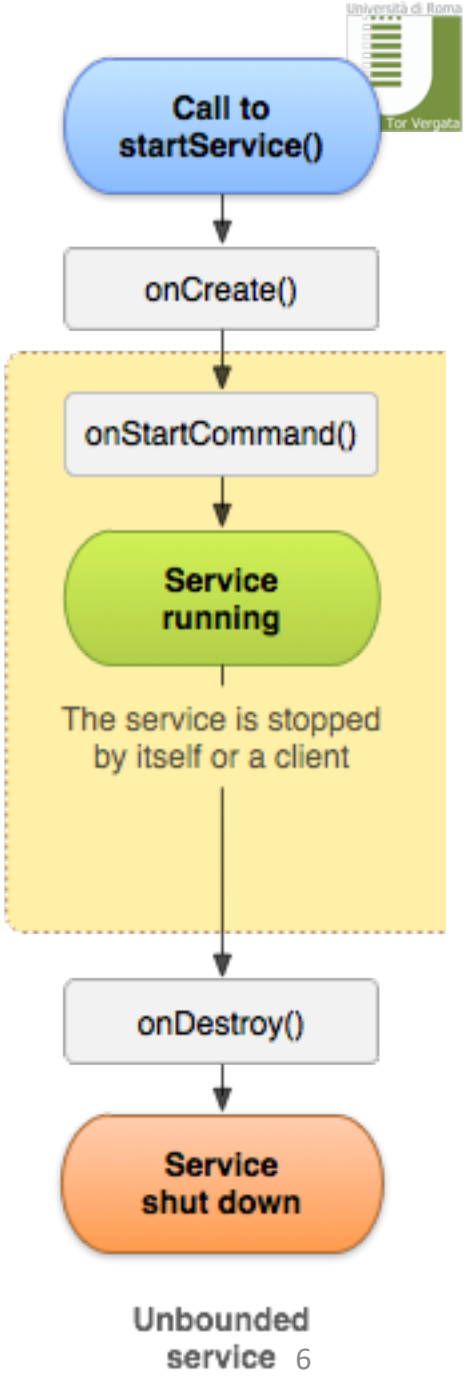
Avvio e binding

- Servizi locali: start e stop
  - public abstract ComponentName startService (Intent service)
  - public abstract boolean stopService (Intent service)

- Collegamento a servizi “remoti”
  - public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)
  - public abstract void unbindService (ServiceConnection conn)

Ciclo di vita – servizi started

- public void onCreate ()
  - eseguito alla creazione del servizio
- public int onStartCommand (Intent intent, int flags, int startId)
  - eseguito all'avvio del servizio
    - in corrispondenza di ogni startService()
  - flags
    - START_FLAG_REDELIVERY
    - START_FLAG_RETRY
- public void onDestroy()
  - eseguito alla distruzione del servizio



onStart command

Costanti	Descrizione
`START_STICKY`	Il Service viene mantenuto nello stato di started, ma l'Intent ricevuto non viene mantenuto. L'ambiente proverà successivamente a ricreare il servizio invocando nuovamente il metodo `onStartCommand()`, passando però un Intent a null, a meno che non ve ne siano di pendenti.
`START_NOT_STICKY`	Il Service, se non ci sono Intent in attesa, non viene ricreato fino a un'esplicita invocazione del metodo `startService()`. Questo garantisce che l'Intent passato sia sempre diverso da null.
`START_REDELIVER_INTENT`	In questo caso viene rischedulata una nuova partenza del Service, con un reinvio dello stesso Intent.
`START_STICKY_COMPATIBILITY`	Indica la non garanzia nella chiamata al metodo `onStartCommand()` nel caso di gestione con `START_STICKY`.

Esempio

```
public class LocalServiceTestActivity extends Activity {
    private Intent serviceIntent;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                serviceIntent = new Intent(this,MyLocalService.class);
    }
    public void startLocalService(View button){
                startService(serviceIntent);
        }

        public void stopLocalService(View button){
                stopService(serviceIntent);
        }
}
```

Esempio (cont.)

```
public class MyLocalService extends Service {
    private BackgroundThread backgroundThread;

    public void onCreate() {
        super.onCreate();
        backgroundThread = new BackgroundThread();
        backgroundThread.start();
        // …
    }

    private final class BackgroundThread extends Thread {
        public void run() {
            // …
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Bounded Services

"A bound service allows components (such as activities) to bind to the service, send requests, receive responses, and even perform interprocess communication (IPC)"

- Metodi per collegarsi ad un servizio
  - public abstract boolean bindService (Intent service, ServiceConnection conn, int flags)
  - public abstract void unbindService (ServiceConnection conn)

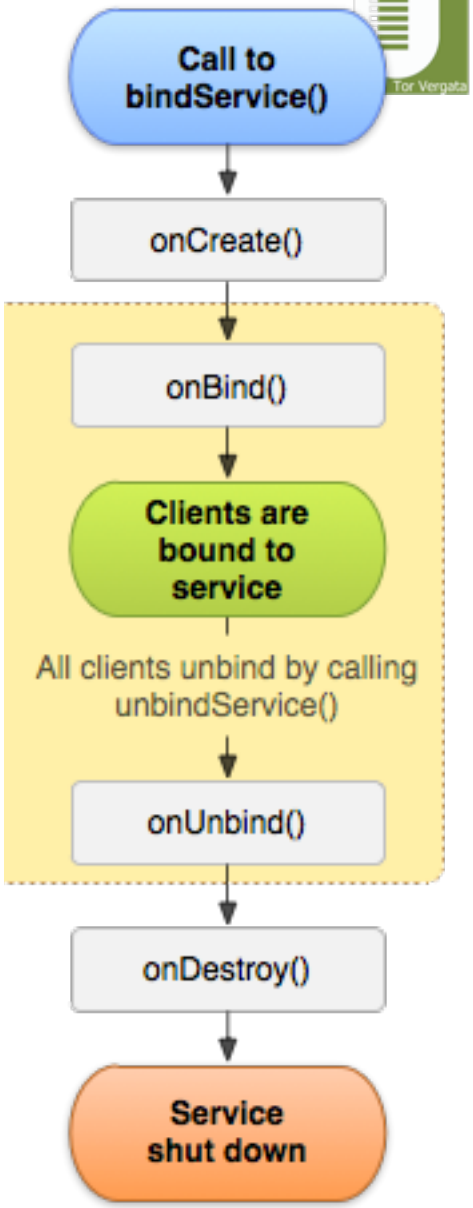
- Interfaccia ServiceConnection
  - public abstract void onServiceConnected (ComponentName name, IBinder service)
  - public abstract void onServiceDisconnected (ComponentName name)

Bind a Service

- Implementare i due metodi di ServiceConnection
  - onServiceConnected()
    - chiamato per fornire un IBinder
  - onServiceDisconnected()
    - The Android system calls this when the connection to the service is unexpectedly lost, such as when the service has crashed or has been killed. This is *not* called when the client unbinds.
- Chiamare il metodo bindService()
  - Passandogli la ServiceConnection
- Usare il servizio
  - Il sistema chiama onServiceConnected()
- Disconnettersi dal servizio unbindService()

Ciclo di vita

- public void onCreate ()
  - eseguito alla creazione del servizio
- public abstract IBinder onBind (Intent i)
  - eseguito al primo binding
- public boolean onUnbind (Intent i)
  - tutti i client se ne sono andati
- public boolean onRebind (Intent i)
  - è tornato un client
    - chiamata a seconda sel valore tornato da onUnbind
- public void onDestroy()
  - eseguito alla distruzione del servizio



Bounded service

IBinder

- Contiene l'interfaccia per accedere al servizio o interagire con questo
- Ci sono tre modalità
  - Estendere la classe Binder
    - soluzione singolo processo / singolo thread
  - Usare un Messenger
    - soluzione IPC / singolo thread
  - Usare AIDL
    - soluzione IPC multi-thread

Esempio 1

```
public class LocalService extends Service {
    private final IBinder mBinder = new LocalBinder();

    private final Random mGenerator = new Random();

    public class LocalBinder extends Binder {
        LocalService getService() {
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    public int getRandomNumber() {
      return mGenerator.nextInt(100);
    }
}
```

Esempio 1 (cont)

```
public class BindingActivity extends Activity {

    LocalService mService;
    boolean mBound = false;

    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className,
                IBinder service) {
            LocalBinder binder = (LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBound = false;
        }
    };

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

Esempio 1 (cont)

```
    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }

    public void onButtonClick(View v) {
        if (mBound) {
            int num = mService.getRandomNumber();
            Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
        }
    }


}
```

Usare un Messenger

- Il servizio implementa un Handler
- L'Handler è usato per creare il Messenger
  - che mantiene il riferimento all'Handler
- Il Messenger crea l'IBinder che viene restituito nel metodo onBind
- Il Client usa IBinder per istanziare un suo Messenger (che referenzia l'Handler) che è usato per mandare Message al Servizio
- Il Service riceve i Message e li gestisce nel metodo handleMessage dell'Handler

Esempio 2

```
public class MessengerService extends Service {
    static final int MSG_SAY_HELLO = 1;

    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_SAY_HELLO:
                    Toast.makeText(getApplicationContext(), "hello!", Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    final Messenger mMessenger = new Messenger(new IncomingHandler());

    @Override
    public IBinder onBind(Intent intent) {
        Toast.makeText(getApplicationContext(), "binding", Toast.LENGTH_SHORT).show();
        return mMessenger.getBinder();
    }
}
```

Esempio 2

```
public class ActivityMessenger extends Activity {
    Messenger mService = null;

    boolean mBound;

    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            mService = new Messenger(service);
            mBound = true;
        }

        public void onServiceDisconnected(ComponentName className) {
            mService = null;
            mBound = false;
        }
    };

    public void sayHello(View v) {
        if (!mBound) return;
        Message msg = Message.obtain(null, MessengerService.MSG_SAY_HELLO, 0, 0);
        try {
            mService.send(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

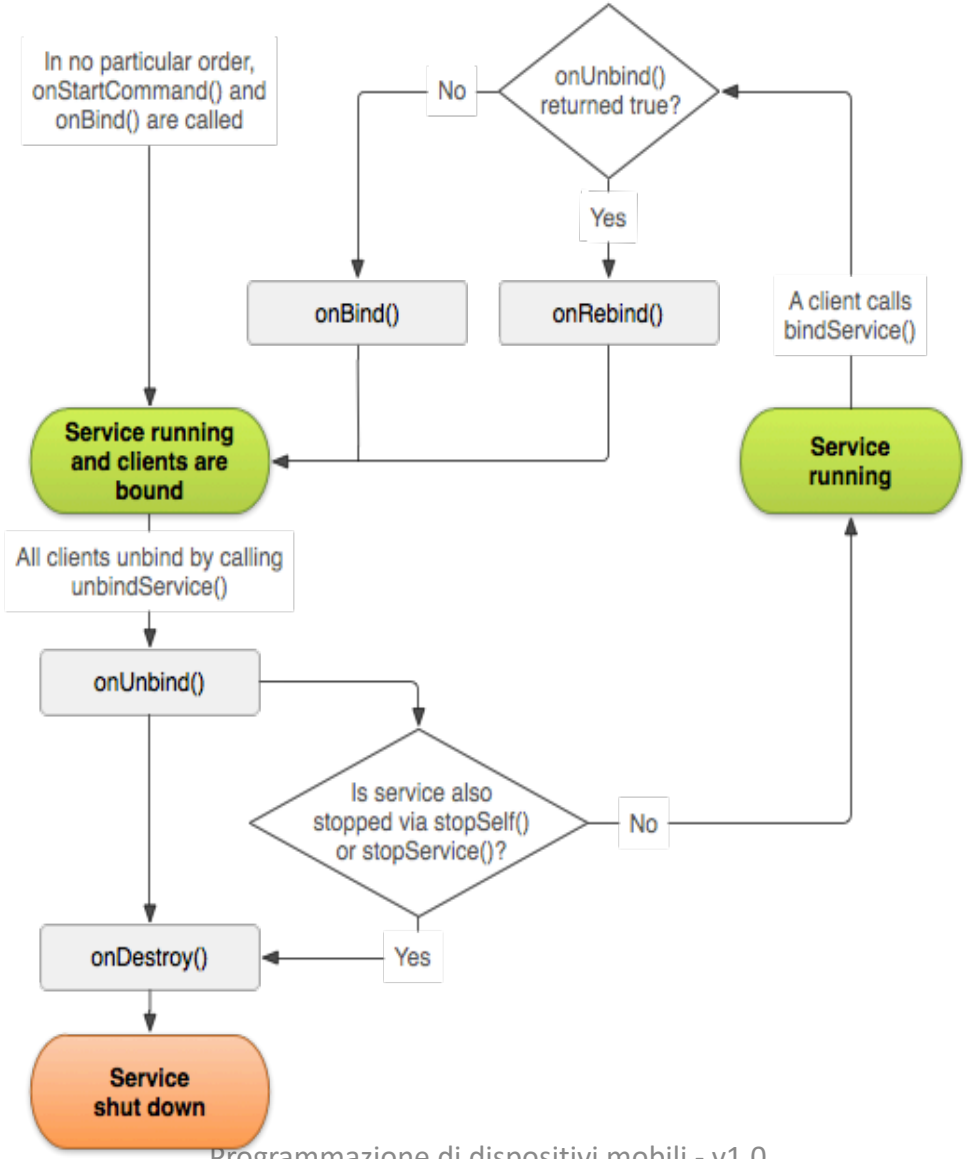
Esempio 2

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

    @Override
    protected void onStart() {
        super.onStart();
        bindService(new Intent(this, MessengerService.class), mConnection,
            Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }
}
```

Ciclo di vita completo



AIDL IBinder

- Per creare un servizio con interfaccia AIDL:
- Creare il file .aidl
  - fornirà l'interfaccia al servizio
- Implementare l'interfaccia
  - il compilatore genera un file java
  - il file contiene un'interfaccia contenente una inner class astratta di nome Stub che estender Binder e definisce i metodi dichiarati in aidl
  - implementare i metodi astratti della classe Stub
- Esporre l'interfaccia al client

Esempio 3

```
// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things with it. */
    int getPid();

    /** Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,
            double aDouble, String aString);
}
```

Esempio 3 (cont.)

```
private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
    public int getPid(){
        return Process.myPid();
    }
    public void basicTypes(int anInt, long aLong, boolean aBoolean,
        float aFloat, double aDouble, String aString) {
        // Does nothing
    }
};
```

Esempio 3 (cont.)

```
public class RemoteService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public IBinder onBind(Intent intent) {
        // Return the interface
        return mBinder;
    }

    private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
        public int getPid(){
            return Process.myPid();
        }
        public void basicTypes(int anInt, long aLong, boolean aBoolean,
            float aFloat, double aDouble, String aString) {
            // Does nothing
        }
    };
}
```

Esempio 3 (cont.)

```
IRemoteService mIRemoteService;
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder service) {
        // Following the example above for an AIDL interface,
        // this gets an instance of the IRemoteInterface, which we can use to c
        mIRemoteService = IRemoteService.Stub.asInterface(service);
    }

    // Called when the connection with the service disconnects unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "Service has unexpectedly disconnected");
        mIRemoteService = null;
    }
};
```

Broadcast Receiver

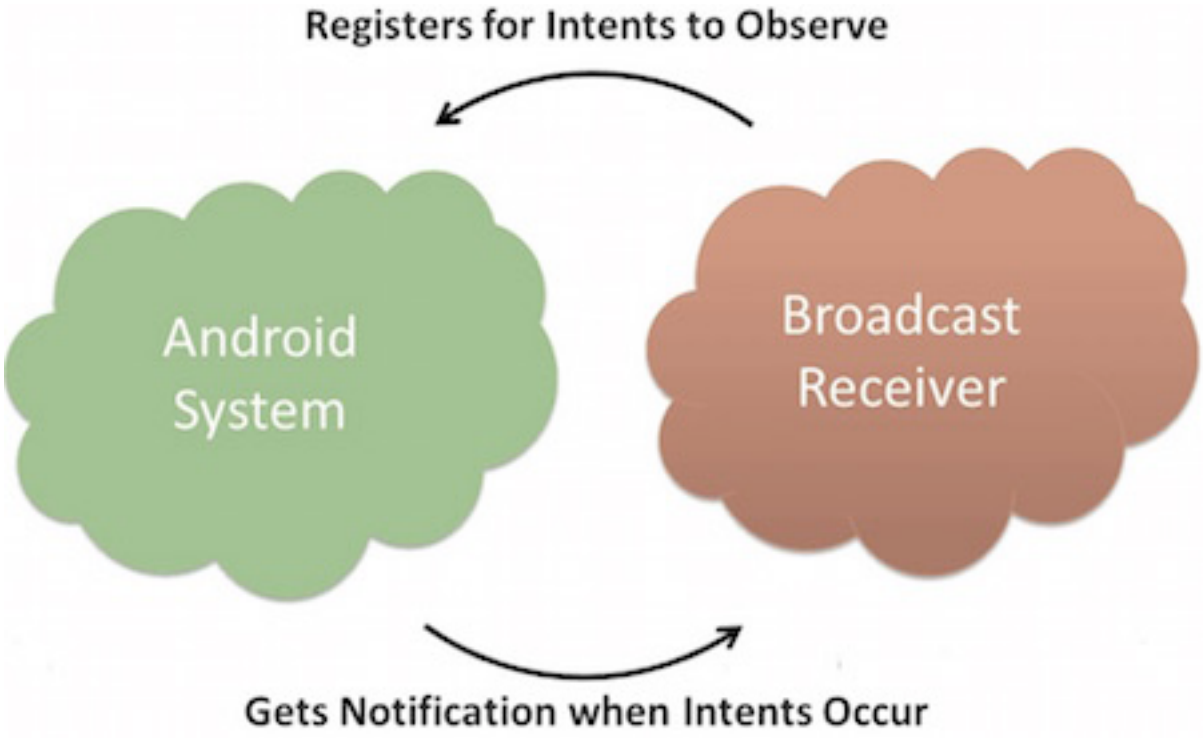
Broadcast receiver

- Il BR è il componente android che permette di ascoltare (ricevere) gli eventi di sistema o dell'applicazione
- Esempio eventi:
  - Intent.ACTION_BOOT_COMPLETED
  - Intent.ACTION_POWER_CONNECTED
  - Intent.ACTION_POWER_DISCONNECTED
  - Intent.ACTION_BATTERY_LOW
  - Intent.ACTION_BATTERY_OKAY
  - ...

- http://developer.android.com/reference/android/content/Intent.html

Implementazione

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
    }
}
```



Registrazione

- Manifest

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED">
         </action>
      </intent-filter>

   </receiver>
</application>
```

- Dinamica
  - registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
  - unregisterReceiver(BroadcastReceiver receiver)

Esempio

```
public class BroadcastReceiverTestActivity extends Activity {

    private final BroadcastReceiver timeBroadcastReceiver = new BroadcastReceiver(){

        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(BroadcastReceiverTestActivity.this,
                           "BroadCast Intent Receiver", Toast.LENGTH_SHORT).show();
        }

    };
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(timeBroadcastReceiver);
    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(timeBroadcastReceiver,new IntentFilter(Intent.ACTION_TIME_TICK));
    }

}
```

Creare eventi

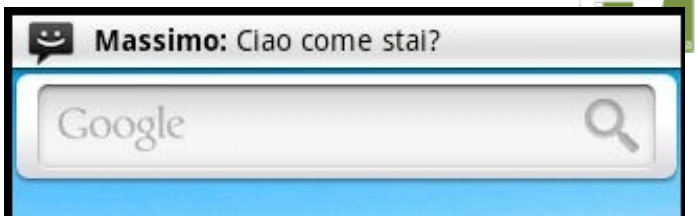
- sendBroadcast
- sendOrderedBroadcast
  - i receiver hanno una priorità impostata nel filtro
    - 0 default
    - intero tra 1000 e -1000

- LocalBroadcastManager.getInstance(this)
  - registerReceiver
  - unregisterReceiver
  - sendBroadcast

Notification service

Notification

- messaggi di notifica visualizzati:
  - nella barra superiore
  - nella zona notifiche



- Serve ai componenti in backgroud a comunicare con l'utente

Creare Notification

- Vanno specificati
  - un'icona
  - il messaggio da visualizzare nella versione breve ed espansa
  - l'Intent da lanciare nel caso di selezione nella forma di un PendingIntent
- Opzionalmente
  - un messaggio stile Ticker da visualizzare nella status bar
  - un suono di alert
  - modalità di vibrazione
  - modalità di lampeggio dei led

Pending Intent

- Problema: voglio attivate un componente di un app con un intent esplicito ma da un app diversa
  - non è possibile in modo diretto
- Idea:
  - la mia app memorizza l'azione (Intent) in un oggetto (PendingIntet)
  - un'altra app può eseguire l'azione

- Creare il PendingIntent
  - getActivity(Context c, int requestCode, Intent intent, int flags)
  - getBroadcast (Context c, int requestCode, Intent intent, int flags)
  - getService (Context c, int requestCode, Intent intent, int flags)

Flag Pending Intent

Costante flag	Descrizione
`FLAG_ONE_SHOT`	Indica che il PendingIntent può essere utilizzato solamente una volta. Ogni ulteriore tentativo porta alla generazione di un errore.
`FLAG_NO_CREATE`	Se il corrispondente PendingIntent non esiste già, il metodo di creazione restituisce null.
`FLAG_CANCEL_CURRENT`	Se il corrispondente PendingIntent esiste già, viene eliminato a favore di uno nuovo.
`FLAG_UPDATE_CURRENT`	Se il corrispondente PendingIntent esiste già, viene mantenuto e ne vengono aggiornate solamente le informazioni relative all'Extra.

Esempio

```
// Creiamo la Notification
Notification notification = new Notification(R.drawable.icon,
                                            "Simple Notification",
                                            System.currentTimeMillis());
notification.flags |= Notification.FLAG_AUTO_CANCEL;
// Impostiamo le altre informazioni tra cui l'Intent
Intent intent = new Intent(this, NotificationActivity.class);
intent.putExtra("notificationType", "Simple Notification");
PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent,
                                                  PendingIntent.FLAG_CANCEL_CURRENT);
notification.setLatestEventInfo(this, "Simple Notification",
                                      "Simple Notification Extended",
                                      pIntent);
// La lanciamo attraverso il Notification Manager
notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(SIMPLE_NOTIFICATION_ID, notification);
```