Preferences

Marco Ronchetti
Università degli Studi di Trento

SharedPreferences

SharedPreferences allows to save and retrieve persistent key-value pairs of primitive data types. This data will persist across user sessions (even if your application is killed).

getSharedPreferences(String name, int mode)

A method of Contex

- Uses multiple preferences files identified by name, which you specify with the first parameter.

getPreferences()

A method of Activity

- Use this if you need only one preferences file for your Activity. This simply calls the underlying getSharedPreferences(String, int) method by passing in this activity's class name as the preferences name

SharedPreferences methods

boolean contains(String key)

Checks whether the preferences contains a preference.

T getT(String key, T defValue)

Value returned
If key does not exist

Retrieve a T value from the preferences where T={int, float, boolean, long, String, Set<String>}.

SharedPreferences.Editor edit()

All changes you make in an editor are batched, and not copied back to the original SharedPreferences until you call commit() or apply()

SharedPreferences.Editor methods

Void apply(), boolean commit()

Commit your preferences changes back (apply is asynchronous)

Editor putT(String key)

Stores a T value in the preferences where T={int, float, boolean, long, String, Set<String>}.

Editor remove(String key)

Mark in the editor that a preference value should be removed

Editor clear ()

Mark in the editor that all preference values should be removed

User Preferences

Shared preferences are not strictly for saving "user preferences," such as what ringtone a user has chosen.

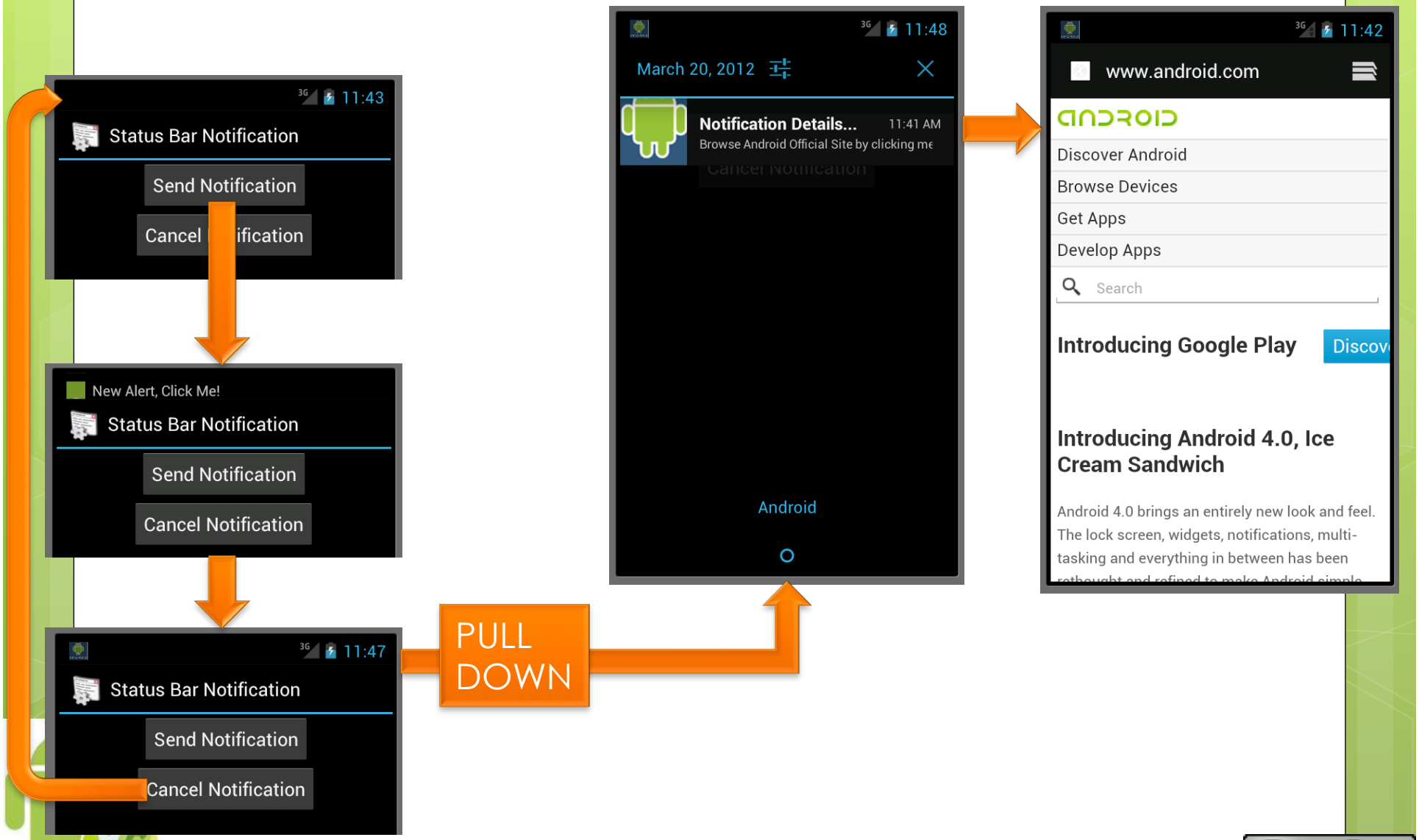
For creating user preferences for your application, you should use PreferenceActivity, which provides an Activity framework for you to create user preferences, which will be automatically persisted (using shared preferences).

It is based on Fragments

Notification

Marco Ronchetti
Università degli Studi di Trento

Notification Bar



SimpleNotification

```
public class SimpleNotification extends Activity {
   private NotificationManager nm;
   private int SIMPLE_NOTIFICATION_ID;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      nm = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
      final Notification notifyDetails = new Notification(
            R.drawable.android,"New Alert, Click Me!",
            System.currentTimeMillis());
      Button cancel = (Button)findViewById(R.id.cancelButton);
      cancel.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            nm.cancel(SIMPLE_NOTIFICATION_ID);
         }});}
```

Adapted from http://saigeethamn.blogspot.it

SimpleNotification – part 2

```
Button start = (Button)findViewById(R.id.notifyButton);
start.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      Context context = getApplicationContext();
      CharSequence contentTitle = "Notification Details...";
      CharSequence contentText = "Browse Android Site by clicking me";
      Intent notifyIntent = new Intent
         (android.content.Intent.ACTION_VIEW,
         Uri.parse("http://www.android.com"));
      PendingIntent intent =
      PendingIntent.getActivity(SimpleNotification.this, 0, notifyIntent,
           android.content.Intent.FLAG_ACTIVITY_NEW_TASK);
      notifyDetails.setLatestEventInfo(context, contentTitle,
          contentText, intent);
      nm.notify(SIMPLE_NOTIFICATION_ID, notifyDetails);
      }
});
}}
```

More on Intents

Marco Ronchetti
Università degli Studi di Trento

Intents

An Android Intent is an object carrying an intent, i.e. a message from one component to another component either inside or outside of the application. Intents can communicate messages among any of the three core components of an application -- Activities, Services, and BroadcastReceivers.

The intent itself, an Intent object, is a passive data structure. It holds an abstract description of an operation to be performed.

Explicit vs. implicit Intents

```
// Explicit Intent by specifying its class name
   Intent i = new Intent(this, TargetActivity.class);
   i.putExtra("Key1", "ABC");
   i.putExtra("Key2", "123");

// Starts TargetActivity
   startActivity(i);


// Implicit Intent by specifying a URI
   Intent i = new Intent(Intent.ACTION_VIEW,
               Uri.parse("http://www.example.com"));

// Starts Implicit Activity
   startActivity(i);
```

Pending Intents

A PendingIntent is a token that you give to a foreign application (e.g. NotificationManager, AlarmManager, Home Screen AppWidgetManager, or other 3rd party applications), which allows the foreign application to use your application's permissions to execute a predefined piece of code.

By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity).

An important difference between Intent and PendingIntent is that by using the first, you want to start / launch / execute something NOW, while by using the second entity you want to execute that something in the future.

Explicit vs. implicit Intents

```
public static PendingIntent getActivity (Context context,
                int requestCode,
                Intent intent,
                int flags)
```

Retrieve a PendingIntent that will start a new activity, like calling Context#startActivity(Intent).

Note that the activity will be started outside of the context of an existing activity, so you must use the Intent#FLAG_ACTIVITY_NEW_TASK launch flag in the Intent.

15

Broadcast receivers

Marco Ronchetti
Università degli Studi di Trento

Bradcast receiver

A component that responds to system-wide broadcast announcements.

Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

Applications can initiate broadcasts—e.g. to let other applications know that some data has been downloaded to the device and is available for them to use.

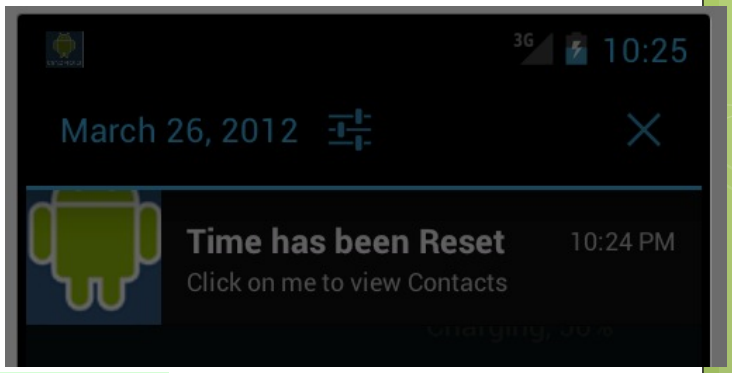
Broadcast receivers don't display a user interface, but they can crate a status bar notification. More commonly, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work e.g. it might initiate a service.

Broadcast receiver

```
public class MyBroadcastReceiver extends BroadcastReceiver {
   …
   public void onReceive(Context context, Intent intent) {
            …
   }
}
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="…I" android:versionCode="1"  android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
       <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
            <action android:name="android.intent.action.TIME_SET"/>
            </intent-filter>
       </receiver>
   </application>
   <uses-sdk android:minSdkVersion="13" />
</manifest>
```

```
>adb shell
# date +%s
1332793443
# date -s +%s 1332793443
time 1332793443 -> 1332793443.0
settimeofday failed Invalid argument
```



Adapted from saigeethamn.blogspot.it

Broadcast receiver

```
public class MyBroadcastReceiver extends BroadcastReceiver {
   private NotificationManager nm;
   private int SIMPLE_NOTIFICATION_ID;

   @Override
   public void onReceive(Context context, Intent intent) {
    nm = (NotificationManager) context.getSystemService
                        (Context.NOTIFICATION_SERVICE);
    Notification n= new Notification(R.drawable.android,"Time Reset!",
                        System.currentTimeMillis());
    PendingIntent myIntent = PendingIntent.getActivity(context, 0,
            new Intent(Intent.ACTION_VIEW, People.CONTENT_URI), 0);
    n.setLatestEventInfo(context, "Time has been Reset",
                        "Click on me to view Contacts", myIntent);
     n|= Notification.FLAG_AUTO_CANCEL;
     n|= Notification.DEFAULT_SOUND;
     nm.notify(SIMPLE_NOTFICATION_ID, n);
     Log.i(getClass().getSimpleName(),"Sucessfully Changed Time");
   }
}
```

Adapted from saigeethamn.blogspot.it

Sending broadcast events

(in Context)

sendBroadcast (Intent intent, String receiverPermission)

Broadcast the given intent to all interested BroadcastReceivers, allowing an optional required permission to be enforced.

This call is asynchronous; it returns immediately, and you will continue executing while the receivers are run.

No results are propagated from receivers and receivers can not abort the broadcast.

Sending ordered broadcast events

(in Context)

sendOrderedBroadcast (Intent intent, String receiverPermission)

Broadcast the given intent to all interested BroadcastReceivers, delivering them one at a time to allow more preferred receivers to consume the broadcast before it is delivered to less preferred receivers.

This call is asynchronous; it returns immediately, and you will continue executing while the receivers are run.

LocalBroadcastManager

Helper to register for and send broadcasts of Intents to local objects within your process.

Advantages of Local vs Global B.M.:

- the data you are broadcasting will not leave your app
  - (you don't need to worry about leaking private data).
- it is not possible for other applications to send these broadcasts to your app
  - (you don't need to worry about having security holes)
- it is more efficient than sending a global broadcast through the system.

Content Providers

Marco Ronchetti
Università degli Studi di Trento

Content Provider

A standard interface connecting a running process with data in another process

Manages access to a structured set of data:

- encapsulate the data
- provide mechanisms for defining data security.

To access data in a content provider, use the **ContentResolver** object in your Context

The ContentResolver object communicates with the provider object, an instance of a class that implements **ContentProvider**. The provider object receives data requests from clients, performs the requested action, and returns the results.

Content Provider

Android includes content providers that manage data such as audio, video, images, and personal contact information..

You can create your own custom content provider to share your data with other packages that works just like the built-in providers.

You need to develop your own provider if

- you intend to share your data with other applications.
- you want to to provide custom search suggestions in your own application.
- you want to copy and paste complex data or files from your application to other applications.

Default content providers

- ContactsContract
  - Stores all contacts information. etc
- Call Log Stores
  - call logs, for example: missed calls, answered calls. etc.
- Browser
  - Use by browser for history, favorites. etc.
- Media Store
  - Media files for Gallery, from SD Card. etc.
- Setting
  - Phone device settings. etc.

Querying a Content Provider

To query a content provider, you provide a query string in the form of a URI, with an optional specifier for a particular row, using the following syntax:

<standard_prefix>://<authority>/<data_path>/<id>

For example, **to retrieve all the bookmarks** stored by your web browsers (in Android), you would use the following content URI:

content://browser/bookmarks

Similarly, **to retrieve all the contacts** stored by the Contacts application, the URI would look like this:

content://contacts/people

To retrieve a particular contact, you can specify the URI with a specific ID:

content://contacts/people/3

Accessing calls

```
package ...
import ...
public class ContentProviderActivity extends Activity {
   /** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Uri allCalls = Uri.parse("content://call_log/calls");
      Cursor c = managedQuery(allCalls, null, null, null, null);
```

package ...
import ...

```
if (c.moveToFirst()) {
        do{
          String callType = "";
          switch (Integer.parseInt(c.getString(
           c.getColumnIndex(Calls.TYPE))))
          {
              case 1: callType = "Incoming";
                break;
              case 2: callType = "Outgoing";
                break;
              case 3: callType = "Missed";
          }
          Log.v("Content Providers",
              c.getString(c.getColumnIndex(Calls._ID))
              + ", " +
              c.getString(c.getColumnIndex(Calls.NUMBER))
              + ", " +
              callType) ;
        } while (c.moveToNext());
      }
   }
}
```

Error!

E/AndroidRuntime(541): java.lang.RuntimeException:
Unable to start activity ComponentInfo{it.unitn.science.latemar/
it.unitn.science.latemar.ContentProviderActivity}:
java.lang.SecurityException: Permission Denial: opening provider
com.android.providers.contacts.CallLogProvider from ProcessRecord{41475a28
541:it.unitn.science.latemar/10041} (pid=541, uid=10041)
requires
android.permission.READ_CONTACTS or
android.permission.WRITE_CONTACTS

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

   <uses-sdk android:minSdkVersion="13" />

   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name" >
      <activity
         android:name=".ContentProviderActivity"
         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>
    <uses-permission
     android:name="android.permission.READ_CONTACTS">
    </uses-permission>
</manifest>
```

Change in Orientation

Marco Ronchetti
Università degli Studi di Trento

Change in orientation

Change in orientation

For devices that support multiple orientations, Android detects a change in orientation:

the display is "landscape" or "portrait".

When Android detects a change in orientation, its default behavior is to destroy and then re-start the foreground Activity.

- Is the screen re-drawn correctly? Any custom UI code you have should handle changes in the orientation.
- Does the application maintain its state? The Activity should not lose anything that the user has already entered into the UI.

Change in configuration

e.g. a change in the availability of a keyboard or a change in system language.

A change in configuration also triggers the default behavior of destroying and then restarting the foreground Activity.

Besides testing that the application maintains the UI and its transaction state, you should also test that the application updates itself to respond correctly to the new configuration.

Save and restore the application state

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
   // Save UI state changes to the savedInstanceState.
   // This bundle will be passed to onCreate if the process is
   // killed and restarted.
   savedInstanceState.putCharSequence("text", button.getText());
   savedInstanceState.putInt("count", count);
   super.onSaveInstanceState(savedInstanceState);
}
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   button = (Button) findViewById(R.id.button1);
   button.setText(savedInstanceState.getCharSequence("text"));
   count=savedInstanceState.getInt("count");
}
```

What can we save in a bundle?

- Primitive data types – arrays of p.d.t.
- String – StringArray - StringArrayList
- CharSequence – CharSequenceArray – CharSequenceArrayList
- Parcelable - ParcelableSequenceArray – ParcelableSequenceArrayList
- Serializable

What to test

Marco Ronchetti
Università degli Studi di Trento

What to test

Change in orientation

Battery life

Techniques for minimizing battery usage are discussed in Optimizing Battery life:

https://developer.android.com/topic/performance/power/index.html

What to test

Dependence on external resources

If your application depends on network access, SMS, Bluetooth, or GPS, then you should test what happens when the resource or resources are not available.

For example, if your application uses the network,it can notify the user if access is unavailable, or disable network-related features, or do both. For GPS, it can switch to IP-based location awareness. It can also wait for WiFi access before doing large data transfers, since WiFi transfers maximize battery usage compared to transfers over 3G or EDGE.

You can use the emulator to test network access and bandwidth. To learn more, please see Network Speed Emulation. To test GPS, you can use the emulator console and LocationManager. To learn more about the emulator console, please see Using the Emulator Console.

Support of multiple versions

Marco Ronchetti
Università degli Studi di Trento

1) Specify Minimum and Target API Levels
2) Check System Version at Runtime
3) Use Platform Styles and Themes

https://developer.android.com/training/basics/supporting-devices/platforms.html

Best practices

Marco Ronchetti
Università degli Studi di Trento

Performance tips

http://developer.android.com/training/articles/perf-tips.html

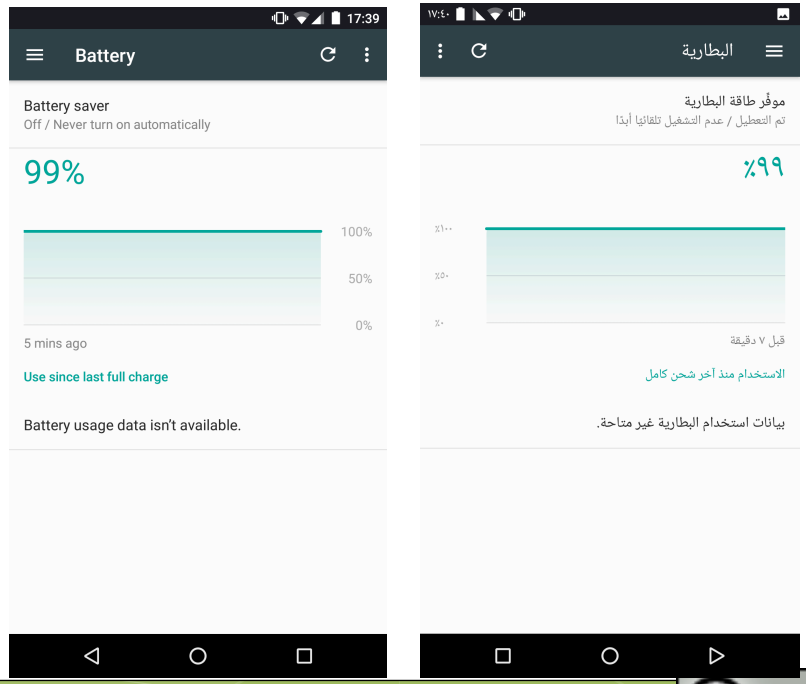
I18n (Internationalization)

Marco Ronchetti
Università degli Studi di Trento

I18n (Internationalization)

https://developer.android.com/training/basics/supporting-devices/languages

- Text
- Number format
- Dates
- Images
- LTR vs RTL
- Layout mirroring



Design

Marco Ronchetti
Università degli Studi di Trento

Android design

http://developer.android.com/design/index.html

Up and running with material design

Android uses a new design metaphor inspired by paper and ink that provides a reassuring sense of tactility. Visit the material design site for more resources.

- Introducing material design
- Downloads for designers
- Articles