
JAVA


Java

Diseño de Interfaces Gráficas en Android

Las interfaces gráficas en Android se construyen usando 2 métodos:

- **Declarativo**: consiste en usar XML para declarar qué mostrará la UI, en forma similar a diseñar páginas web usando HTML. Se escriben *tags* que especifican elementos que aparecerán en la pantalla.

- **Programático**: consiste en escribir código Java para desarrollar la UI. Es similar a escribir una interface gráfica en AWT ó Swing en una aplicación JSE.

  Todo lo que se hace declarativamente se puede hacer programáticamente. Sin embargo en JAVA es posible especificar qué pasará cuando el usuario interactúa con una componente.

Diseño de Interfaces Gráficas en Android

Ventajas del enfoque declarativo:

- Uso de herramientas que facilitan la creación de UI. Android Studio viene equipado con herramientas visuales.
- XML es un lenguaje legible que puede ser entendido por personas no familiarizadas con la plataforma y el framework Android.

Desventaja del enfoque declarativo:

- Su uso es limitado: es ideal para declarar la apariencia de las componentes de UI, pero no provee manejo de eventos producidos por la interacción con los usuarios. Es necesario usar Java en este punto.

Diseño de Interfaces Gráficas en Android

¿Qué enfoque usamos?

La mejor práctica es usar el enfoque declarativo y programático.
Es recomendable usar el enfoque declarativo, XML, para todo lo estático de la UI como el *layout* y las componentes o widgets. Y el enfoque programático, JAVA, para declarar qué pasa cuando el usuario interactúa con los widgets de la UI.
En síntesis en XML programamos cómo se ve un widget y en JAVA cómo reacciona ante la interacción del usuario.

Aunque se utilicen dos enfoques para la creación de UI, el sistema Android creará objetos JAVA a partir de los XML. Finalmente se ejecutará código JAVA.
La clase R es un conjunto de referencias, generada automáticamnte, que ayuda a conectar el mundo JAVA con el mundo XML
R.layout.estado apunta a /res/layout/estado.xml file.

Vistas y Layouts XML

Android organiza los elementos de UI en **vistas** y ***layouts***.

Vistas: es todo lo que se ve en la pantalla, botones, etiquetas, cajas de diálogo, etc. También se los conoce como widgets.

Layouts: organizan y agrupan las vistas. También se los conoce como ViewGroup. Las *vistas* están contenidas en *layouts*.

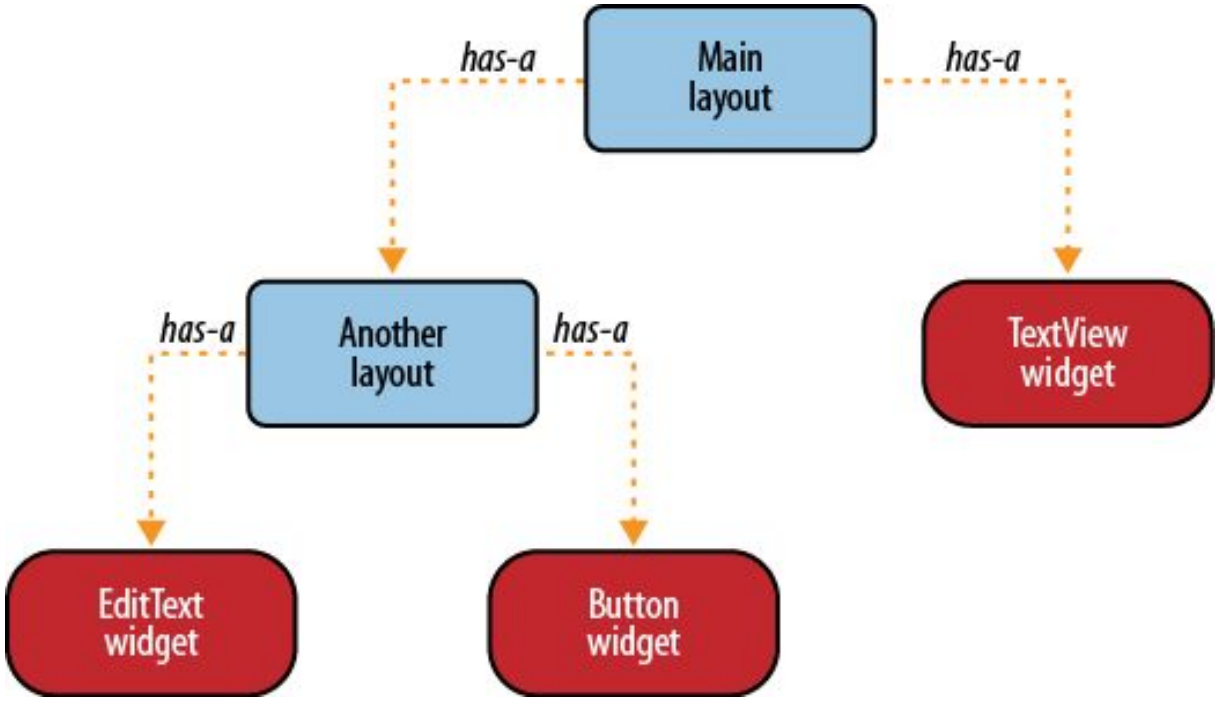
En vocabulario Java, los *layouts* son equivalentes a los **contenedores** y las *vistas* son las **componentes de UI**.

Los *layouts* se guardan en archivos XML ubicados en el directorio **res/layout** de Android.

Los **Activities** usan los *layouts* para mostrar las pantallas de la app.

Layouts XML

Un *layout* puede contener otros *layouts,* llamados hijos, permitiendo diseñar interfaces de usuario complejas.



Los *layouts* más representativos en Android son: **LinearLayout**, **RelativeLayout** y **GridLayout**

LinearLayout

El LinearLayout es un layout que alinea a todos sus hijos en una única dirección, que puede ser vertical u horizontal.

```
<LinearLayout 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:orientation="horizontal" >

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/app_test_boton1" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/app_test_boton2" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/app_test_boton3" />

</LinearLayout>
```

Las hijos se muestran HORIZONTALMENTE

Vistas en el layout

LinearLayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="@string/message" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text="@string/send" />
</LinearLayout>
```

Las hijos se muestran VERTICALMENTE

IMPORTANCIA en relación al espacio que ocupa en la pantalla

ALINEACIÓN

match_parent y wrap_content

"wrap_content": el ancho o la altura de la vista se establece como el tamaño mínimo necesario para adaptar el contenido a esta vista

"match_parent": el ancho o la altura de la vista se amplía hasta coincidir con el tamaño de la vista principal

wrap_content

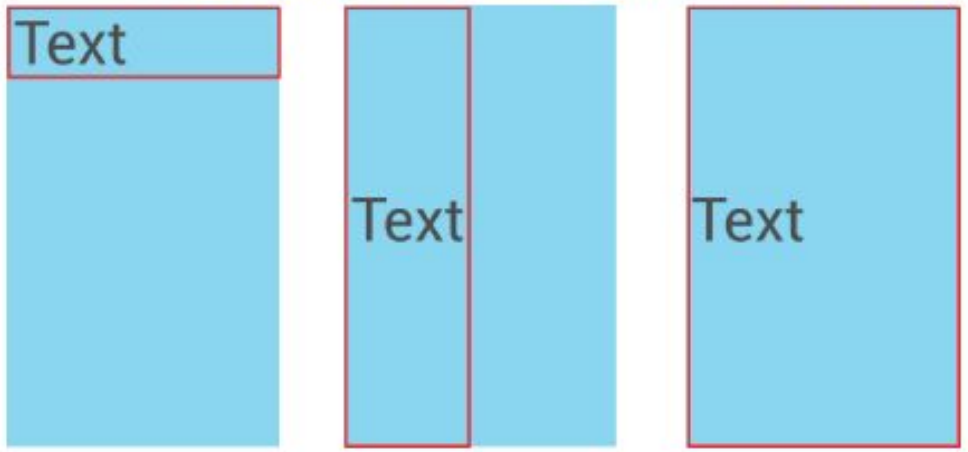
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    />
```

match_parent

```
android:layout_width="wrap_content"
android:layout_height="match_parent"
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
```



RelativeLayout

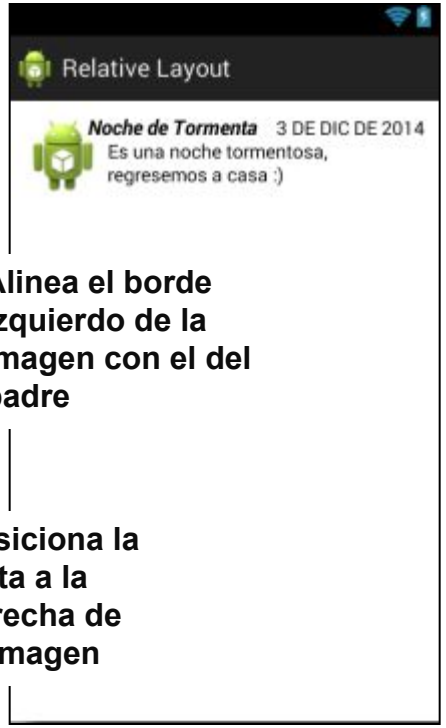
El **RelativeLayout** ubica sus vistas hijas en relación a sus hermanas y al padre. Usando **RelativeLayout** es posible posicionar una vista a la **izquierda**, **derecha**, **arriba** o **abajo** de sus vistas hermanas. También es posible posicionar una vista en relación a su padre alineado horizontalmente, verticalmente o según alguno de los bordes.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_margin="5dp"
   android:padding="5dp" >
   <ImageView
      android:id="@+id/pic"
      android:layout_width="60dp"
      android:layout_height="60dp"
      android:layout_alignParentLeft="true"
      android:layout_gravity="center"
      android:layout_marginRight="3dp"
      android:src="@drawable/ic_launcher“/>
  <TextView
      android:id="@+id/info"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/title"
      android:layout_marginRight="3dip"
      android:layout_toRightOf="@+id/pic"
      android:gravity="left"
      android:text="Es una noche tormentosa, regresemos a casa :)"
      android:textColor="#000000"
      tools:ignore="HardcodedText" />
```

ID: identifica a la vista

Alinea el borde izquierdo de la imagen con el del padre

Posiciona la vista a la derecha de la imagen

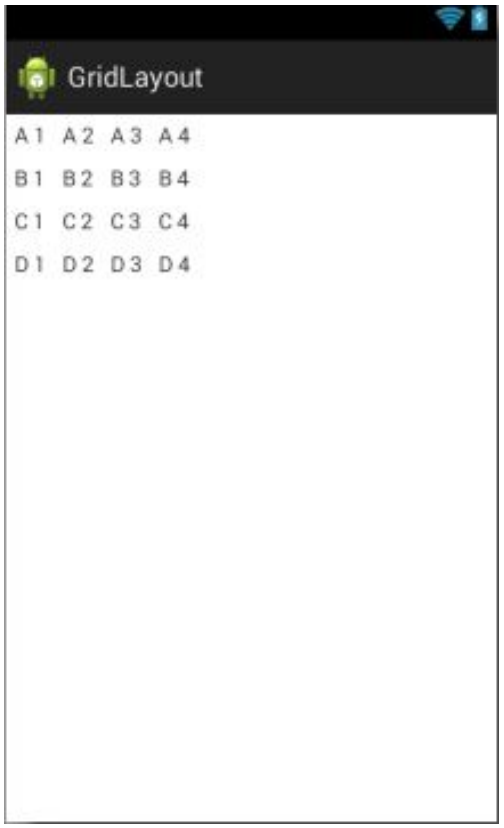



```
<TextView
      android:id="@+id/date"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_above="@+id/info"
      android:layout_alignParentRight="true"
      android:text="3 DE DIC DE 2014"
      android:textColor="#000000"
      tools:ignore="HardcodedText" />

  <TextView
      android:id="@+id/title"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignTop="@+id/pic"
      android:layout_marginRight="14dp"
      android:layout_toLeftOf="@+id/date"
      android:text="Noche de Tormenta"
      android:textColor="#040404"
      android:textSize="15sp"
      android:textStyle="bold|italic"
      android:typeface="sans"
      tools:ignore="HardcodedText" />

</RelativeLayout>
```

GridLayout

El **GridLayout** ubica sus hijos en una grilla rectangular. Este layout divide el área de dibujo en: filas, columnas y celdas. Soporta espaciado entre filas y columnas. Un widget puede ocupar un área rectangular de varias celdas contiguas.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:columnCount="4"
   android:orientation="horizontal"
   android:rowCount="4" >
   <TextView
      android:padding="5dp"
      android:text="A 1" />
   <TextView
      android:padding="5dp"
      android:text="A 2" />
   <TextView
      android:padding="5dp"
      android:text="A 3" />
   <TextView
      android:padding="5dp"
      android:text="A 4" />
   <TextView
      android:padding="5dp"
      android:text="B 1" />
   <TextView
      android:padding="5dp"
      android:text="B 2" />
 <TextView
      android:padding="5dp"
      android:text="B 3" />
```




```
<TextView
      android:padding="5dp"
      android:text="B 4" />
 <TextView
      android:padding="5dp"
      android:text="C 1" />
  <TextView
      android:padding="5dp"
      android:text="C 2" />
  <TextView
      android:padding="5dp"
      android:text="C 3" />
  <TextView
      android:padding="5dp"
      android:text="C 4" />
  <TextView
      android:padding="5dp"
      android:text="D 1" />
  <TextView
      android:padding="5dp"
      android:text="D 2" />
  <TextView
      android:padding="5dp"
      android:text="D 3" />
  <TextView
      android:padding="5dp"
      android:text="D 4" />

</GridLayout>
```

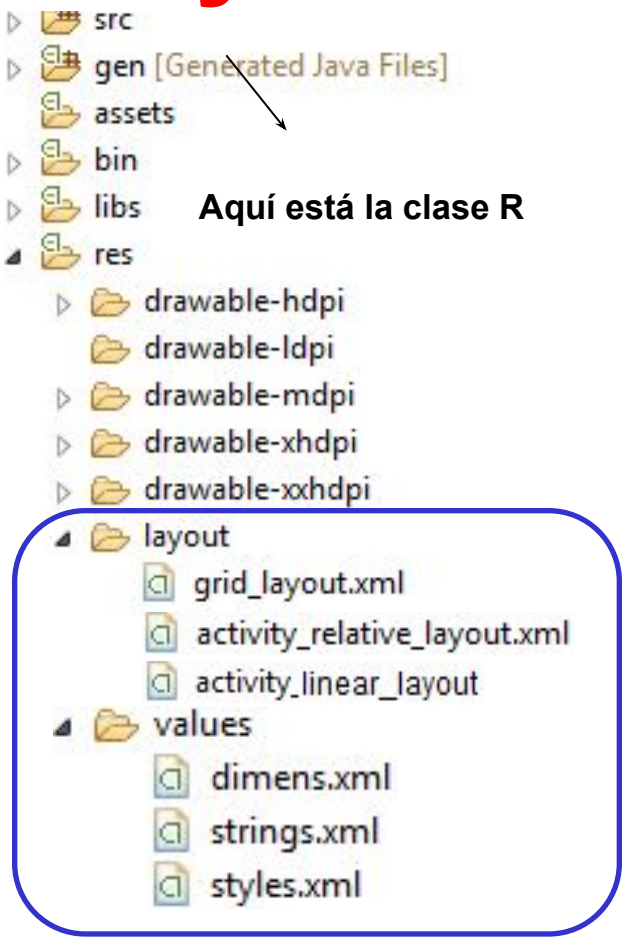
Layouts & Activity

```
public class GridLayoutActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.grid_layout);
    }
}
```

```
public class RelativeLayoutActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_relative_layout);
    }
}
```

```
public class LinearLayoutActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_linear_activity);
    }
}
```

inflating from XML



Aquí está la clase R

setContentView(): lee el archivo XML, lo parsea, crea todos los objetos JAVA correspondientes a los elementos XML, setea las propiedades de los objetos y (inflate) completa toda la vista. Luego la vista está lista para dibujarse.

¿Cómo referenciamos los widgets/vistas desde nuestra app?

ID: es un identificador entero único de un widget en un layout específico.
No todos los widgets necesitan tener un **ID.**
Los widgets que necesitan ser manipulados desde JAVA necesitan tener un **ID**.

El ID se define en XML, en el archivo de layout en el atributo **id.** Cuando la app se compila el ID es referenciado como un número entero y agregado a la clase R.

El formato del **ID** es el siguiente: @+id/unNombre
Ejemplo: @+id/botonActualizar

Código que crea una instancia del widget para manipular desde código JAVA:

```
Button myButton = (Button) findViewById(R.id.botonActualizar);
```

Ejemplo

```
public class StatusActivity1 extends Activity implements OnClickListener {
  private static final String TAG = "StatusActivity";
  EditText editText;
  Button updateButton;
  Twitter twitter;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.status);

    editText = (EditText) findViewById(R.id.editText);
    updateButton = (Button) findViewById(R.id.buttonUpdate);

    updateButton.setOnClickListener(this);

    twitter = new Twitter("student", "password");
    twitter.setAPIRootUrl("http://yamba.marakana.com/api");
  }

  public void onClick(View v) {
    twitter.setStatus(editText.getText().toString());
    Log.d(TAG, "onClicked");
  }
}
```

Listener de Botón

Crea instancia a partir de los widgets

Registra el listener de los eventos del botón

Método de la interface OnClickListener, será invocado cuando se clickea el botón

Fragmentos

Un **fragmento** es una porción de la GUI que puede agregarse o eliminarse de la GUI de forma independiente al resto de elementos del Activity y puede reutilizarse en otros Activities.

Un fragmento corre dentro del contexto de un Activity, pero tiene su propio ciclo de vida.

Los fragmentos nos permiten dividir nuestra interfaz en varias porciones de forma que podamos diseñar diversas configuraciones de pantalla, dependiendo de su tamaño y orientación, sin tener que duplicar código. Por ejemplo

Fragmentos

Ciclo de vida

El ciclo de vida del fragmento está conectado al ciclo de vida de el Activity.

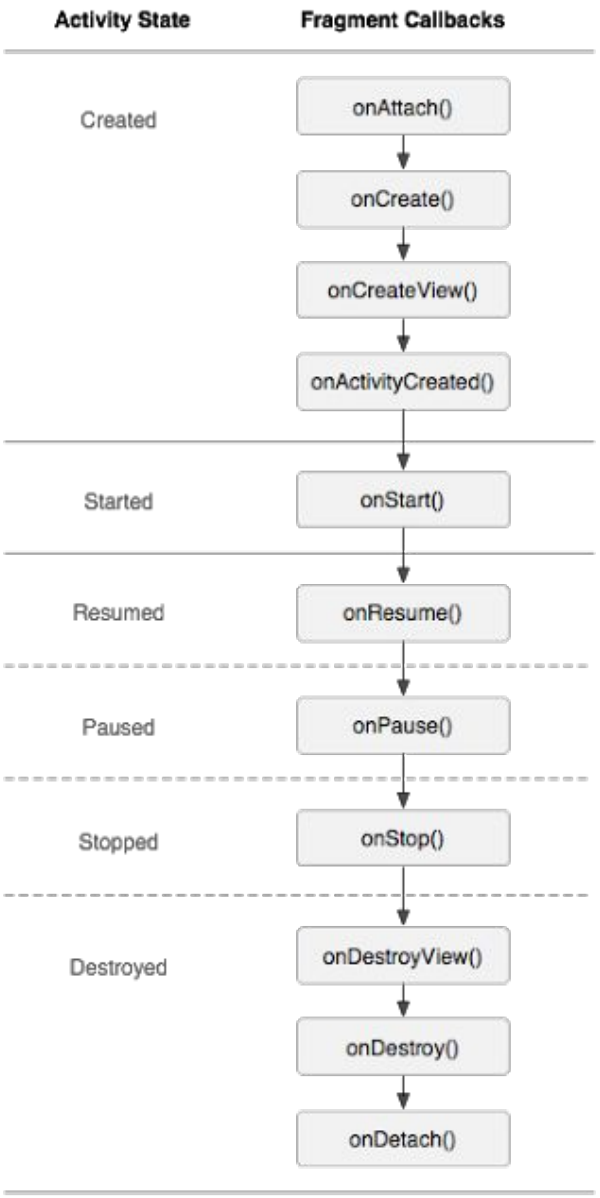
Si el Activity se detiene, sus fragmentos son detenidos; si el activity es destruido sus fragmentos también son destruidos

Generalmente se implementan los siguientes métodos:

onCreate() - El sistema llama a este método a la hora de crear el fragmento, normalmente iniciaremos los componentes esenciales del fragmento.

onCreateView() - El sistema llamará al método cuando sea la hora de crear la interface de usuario o vista, normalmente se devuelve la view del fragmento.

onPause() - El sistema llamará a este método en el momento que el usuario abandone el fragmento, por lo tanto es un buen momento para guardar información.



Fragmentos

Agregar un Fragmento a una Activity

A la hora de agregar un fragmento a una actividad lo podremos realizar de dos maneras:

1. Declarar el fragmento en el layout de la activity.
2. Agregar directamente el Fragment mediante programación Android.

Ejemplo declarativo de layout especificando un elemento fragment

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:name="com.fragments.FRAGMENTOS.FragmentUNO"
        android:id="@+id/fragment_uno"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

Fragmentos

Agregar un Fragmento a una Activity

Para agregar o eliminar un fragmento, se debe usar un FragmentManager con el cual es posible crear una FragmentTransaction. FragmentTransaction tiene la API para agregar, eliminar, reemplazar y llevar a cabo otras transacciones con fragmentos.

Ejemplo de agregado de un Fragmento programáticamente

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);


    setContentView(R.layout.mostrar_fragment_dos);


    //en versiones anteriores a 3.0 usar getSupportFragmentManager();
    FragmentManager FM = getFragmentManager();
    FragmentTransaction FT = FM.beginTransaction();

    Fragment fragment = new FragmentUNO();
    FT.add(R.id.fragment_container, fragment);
    FT.commit();
}
```

Ejemplo de uso de Fragmentos

Ejemplo: CAMBIAR FRAGMENTOS

Cada vez que el usuario toque el botón se reemplazará el fragmento.

Ejemplo de uso de Fragmentos

layout/**gestionar_fragments.xml**

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <RelativeLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@+id/boton"
        android:layout_below="@+id/texto"/>

    <Button
        android:id="@+id/boton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="CAMBIA FRAGMENT"/>

</RelativeLayout>
```

Ejemplo de uso de Fragmentos

```
public class GestionarFragments extends ActionBarActivity {
        private boolean bol = false;
     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gestionar_fragments);
        final Fragment fragmentUNO = new FragmentUNO();
        final Fragment fragmentDOS = new FragmentDOS();
        FragmentTransaction FT = getFragmentManager().beginTransaction();
        FT.add(R.id.fragment_container, fragmentUNO);
        FT.commit();
        Button boton = (Button) findViewById(R.id.boton);
        boton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                FragmentTransaction FT = getFragmentManager().beginTransaction();
                if (bol) { FT.replace(R.id.fragment_container, fragmentUNO);
                } else   { FT.replace(R.id.fragment_container, fragmentDOS);
                }
                FT.commit();
                bol = (bol) ? false : true;
            }
        });
    }
```

Ejemplo de uso de Fragmentos

```
package com.fragments.FRAGMENTOS;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.fragments.R;

public class FragmentUNO extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_uno, container, false);
    }

}
```

Idem para **FragmentDOS..**

Fragmentos

layout/**fragment_uno.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF8800" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textColor="@android:color/white"
        android:text="FRAGMENT UNO"/>


</RelativeLayout>
```

ActionBar

Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones o **ActionBar**.

Situada en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones.

Reúne varios elementos, los más habituales son:

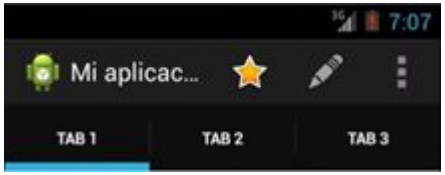
- el icono de la aplicación con su nombre
- los botones de acciones frecuentes.
- las acciones menos utilizadas (se sitúan en un menú desplegable, que se abrirá desde el botón Overflow).

ActionBar

Si la aplicación dispone de pestañas (tabs), estas podrán situarse en la barra de acciones.

También pueden añadirse otros elementos, como listas desplegables y otros tipos de widgets incrustados.

En caso de disponer de menos tamaño de pantalla el sistema puede redistribuir los elementos y pasar alguna acción al menú de «Overflow».



ActionBar

- Los dispositivos anteriores a la versión 3.0 requerían una ***tecla física*** para mostrar el menú de la actividad.

- Con esta versión la tecla física deja de ser un requisito para los dispositivos y ***los menús pasan a mostrarse en la barra de acciones***.

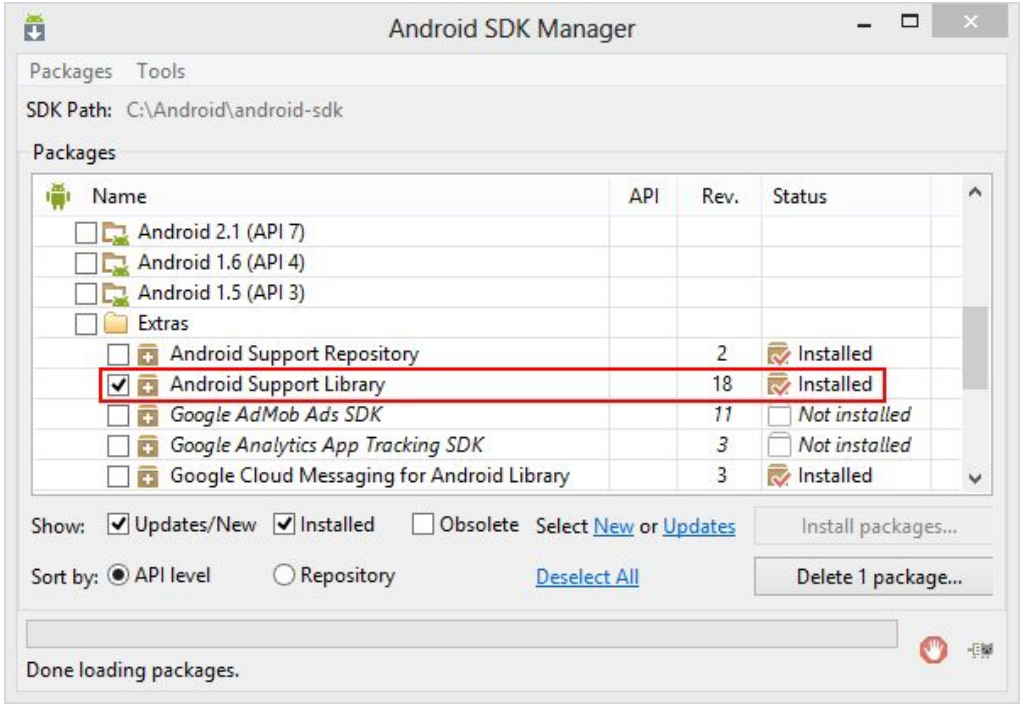
- En los dispositivos que dispongan del botón físico, es posible que los tres puntos que representan el menú de Overflow no se representen en la barra de acciones. En este caso hay que usar el botón físico para desplegar este menú.

- Desde finales de 2013 también podemos utilizar la ActionBar en versiones anteriores a la 3.0 descargando una librería de compatibilidad.

- A partir de la revisión 18 contamos con una versión del Action Bar conocida como **ActionBarCompat**, compatible con cualquier versión Android a partir de la 2.1 (API 7).

ActionBar

- Hay que instalar la última versión de la librería de compatibilidad: Android Support Library.



ActionBar

Ejemplo de menú que se muestra en la barra de acciones

res/menu/menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/ tools"
      tools:context=".MainActivity">
       <item android:title="@string/action_settings"
                android:id="@+id/action_settings"
                android:icon="@android:drawable/ic_menu_preferences"
                android:orderInCategory="5"
                android:showAsAction="never"/>
       <item android:title="Acerca de..."
                android:id="@+id/acerca_de"
                android:icon="@android:drawable/ic_menu_info_details"
                android:orderInCategory="10"
                android:showAsAction="ifRoom|withText"/>
       <item android:title="Buscar"
                android:id="@+id/menu_buscar"
                android:icon="@android:drawable/ic_menu_search"
                android:orderInCategory="115"
                android:showAsAction="always|collapseActionView"/>
</menu>
```

- Las acciones que indiquen en el atributo **showAsAction** la palabra **always** **serán mostrados siempre**, sin importar si caben o no.
- Las acciones que indiquen **ifRoom** serán mostradas en la barra de acciones **si hay espacio disponible**, y serán movidas al menú de Overflow si no lo hay.
- Si se indica **never** la acción **nunca se mostrará** en la barra de acciones, sin importar el espacio disponible.
- Las acciones son ordenadas de izquierda a derecha según lo indicado en **orderInCategory**, con las **acciones con un número más pequeño más a la izquierda**.

ActionBar

Una vez definido el menú desde su archivo .xml (res/menu/menu_main.xml) solo resta *inflarlo* desde nuestra activity.

```
public class MainActivity extends Activity {

    ...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
}
```

ActionBar

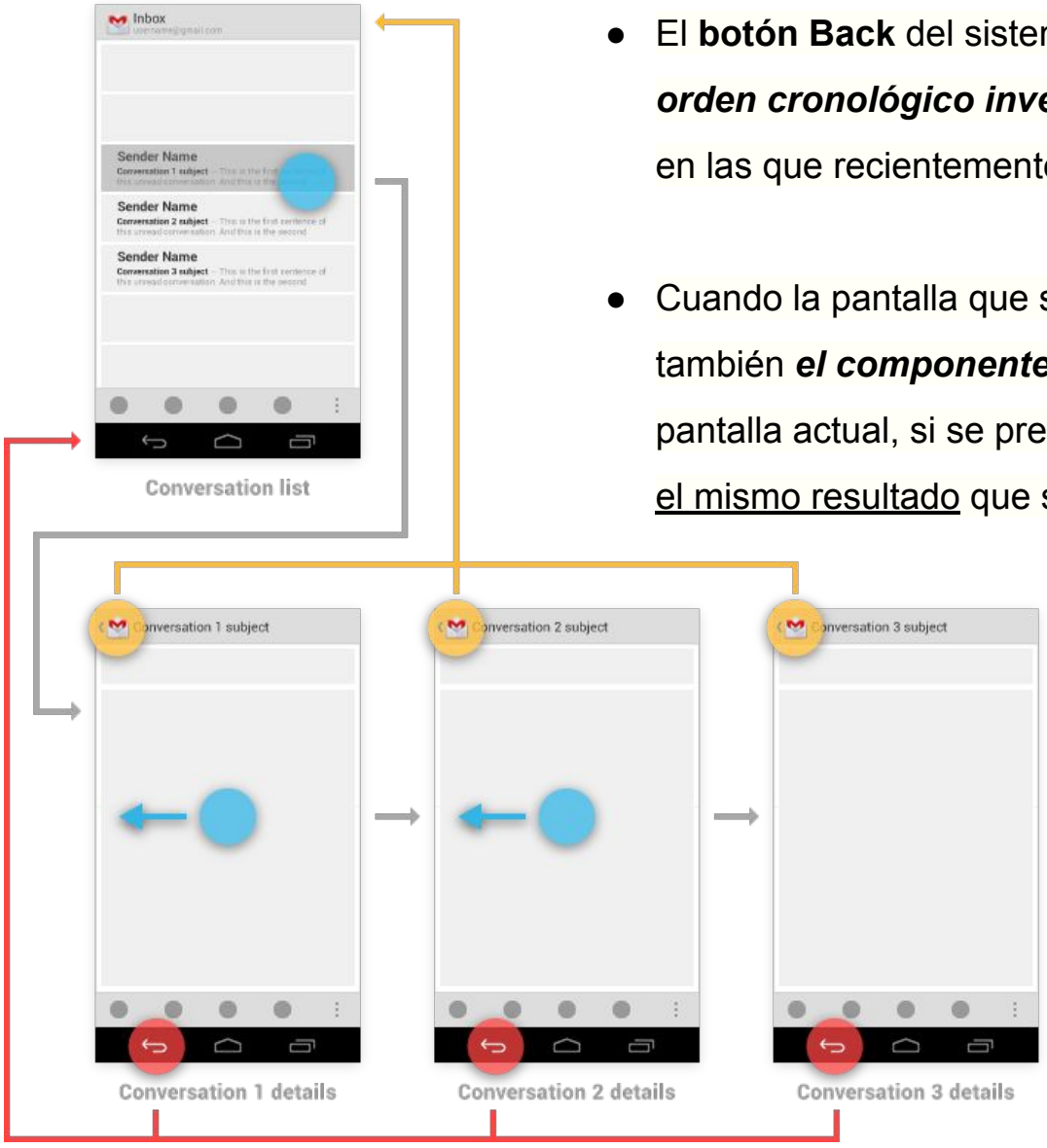
Luego cuando el usuario pulsa sobre el ActionBar, el activity recibirá el llamado a **onOptionsItemSelected**() con el MenuItem seleccionado

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            Log.i("ActionBar", "action_settings");
            return true;
        case R.id.acerca_de:
            Log.i("ActionBar", "acerca_de");;
            return true;
        case R.id.menu_buscar:
            Log.i("ActionBar", "menu_buscar");;
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Navegación con los botones Back y Up

- En Android 3.0, se introdujeron cambios significativos en el comportamiento global de la navegación.
- Utilizando las pautas de navegación con los botones Back y Up, la navegación en su aplicación será predecible y confiable para los usuarios.
- En Android 2.3 y versiones anteriores, se confió en el **botón Back** del sistema para respaldar la navegación dentro de una aplicación. Con la introducción de las barras de acciones en Android 3.0, apareció un segundo mecanismo de navegación: **el botón Up**, que consiste en el icono de la aplicación y una pequeña flecha a la izquierda.
- El **botón Up** se utiliza para navegar dentro de una aplicación sobre la base de las ***relaciones jerárquicas*** entre pantallas.
- Si una pantalla aparece en la parte superior de una aplicación (es decir, en el inicio de la aplicación), no debe incluir el botón Up.

Navegación con los botones Back y Up



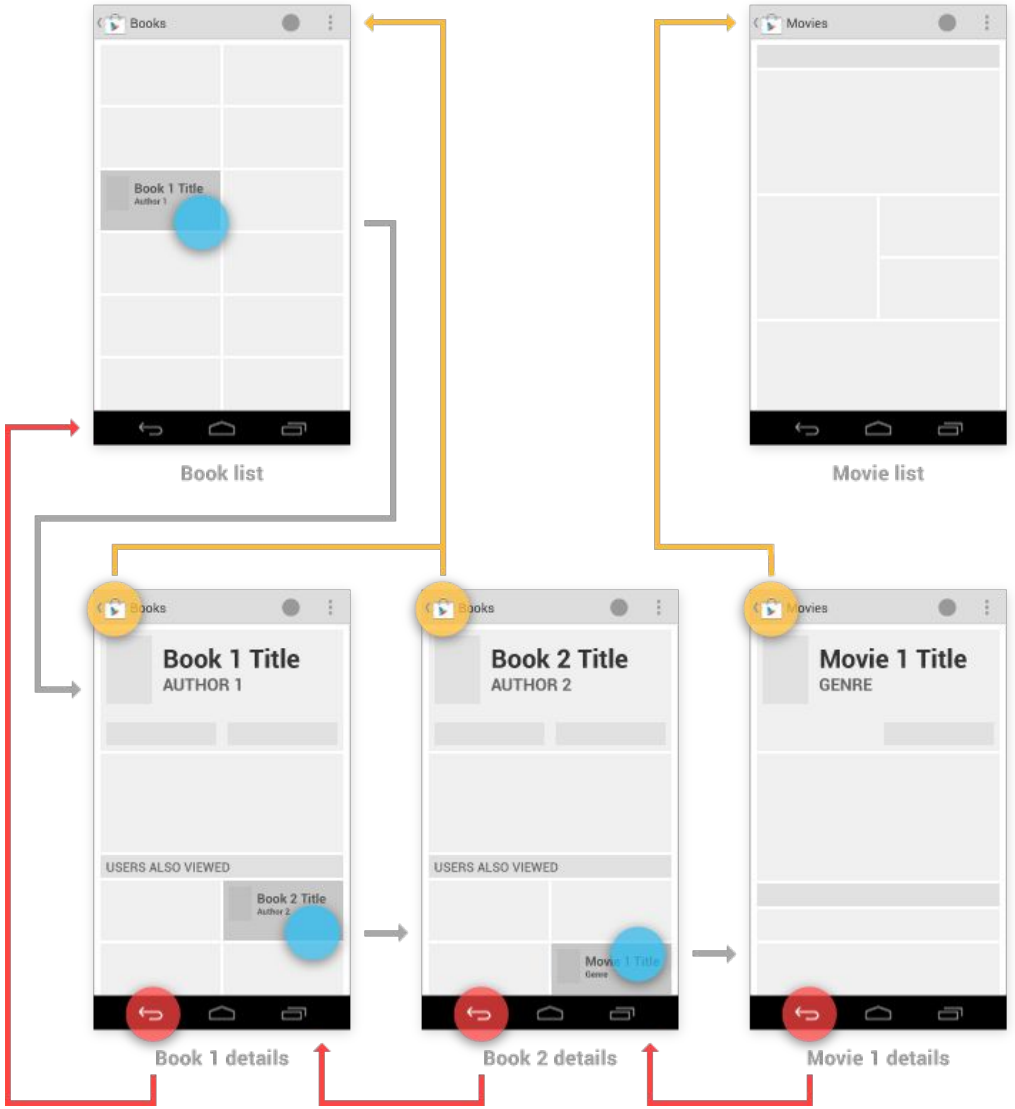
- El **botón Back** del sistema se utiliza para navegar, ***en orden cronológico inverso***, por el historial de pantallas en las que recientemente trabajó el usuario.

- Cuando la pantalla que se visitó anteriormente es también ***el componente jerárquico primario*** de la pantalla actual, si se presiona el botón Back, se obtendrá el mismo resultado que si se presiona el botón Up.

Navegación con los botones Back y Up

Se puede hacer que el botón Up sea más inteligente.

Por ejemplo en ese caso, mediante el botón Up ***podrá regresar a un contenedor*** *(Películas)* ***por el que el usuario no navegó anteriormente****.*



Implementando la navegación UP

- Para implementar la navegación UP, hay que declarar el parent de cada activity.
- Desde la versión de Android 4.1 (API level 16), se puede declarar el parent de cada activity especificando el atributo android:parentActivityName en el elemento <activity>.
- Si la aplicación es menor a la versión 4, hay que incluir la librería Support Library, especificando el parent del activity mediante android.support.PARENT_ACTIVITY y el atributo android:parentActivityName

```
<application ... >
    ...
    <!-- The main/home activity (it has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

Implementando la navegación UP

- Para habilitar el botón Up en el ActionBar

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

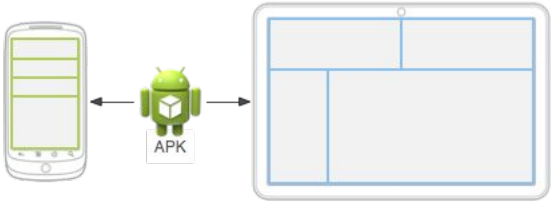
- El activity recibirá el llamado a onOptionsItemSelected() y el ID para la acción es android.R.id.home

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    // Respond to the action bar's Up/Home button
    case android.R.id.home:
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

- Se puede usar ***NavUtils.navigateUpFromSameTask()*** el cual finaliza el activity actual y arranca el activity parent

Provisión de recursos alternativos

Las aplicaciones pueden proporcionar recursos alternativos para adaptarse a distintas densidades de pantallas, utilizar strings alternativos para diferentes idiomas, etc



Para proveer conjunto de recursos alternativos:

crear en res/ un nuevo directorio cuyo nombre sea de la siguiente manera:
<resources_name>-<config_qualifier>

<resources_name> es el nombre del directorio de los recursos predeterminados

<qualifier> es un nombre que especifica una configuración individual para la cual se deben usar estos recursos

Se pueden agregar varios <qualifier> separados con guión.

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

recursos default

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

recurso drawable alternativo

Provisión de recursos alternativos

Nombres de calificadores de configuración

Estos son algunos calificadores de configuración válidos (la tabla es extensa), en orden de precedencia. Si se utilizan varios calificadores para un directorio se deben deben agregar en el siguiente orden.

Configuración	Valores de calificadores	
1) MCC y MNC	Ejemplos: `mcc310` `mcc310-mnc004` `mcc208-mnc00` etc.	Estos son códigos de país del móvil (MCC), opcionalmente seguido del código de red móvil (MNC) de la tarjeta SIM del dispositivo
2) Idioma y región	Ejemplos: `en` `fr` `en-rUS` `fr-rFR` `fr-rCA` etc.	Código de idioma ISO 639-1 de dos letras, opcionalmente seguido de un código de región ISO 3166-1-alfa-2 de dos letras (precedido por “r” en minúscula).
...		

7) Tamaño de pantalla	`small`
	`normal`
	`large`
	`xlarge`
		small: 320 x 426
		normal: 320 x 470
		large: 480 x 640
		xlarge: 720 x 960

...		

13) Densidad de píxeles de la pantalla (dpi)	`ldpi` `mdpi` `hdpi` `xhdpi`
	`xxhdpi` `xxxhdpi` `nodpi`
	`tvdpi` `anydpi`
		ldpi: 120 dpi. mdpi: 160 dpi.
		hdpi: 240 dpi. xhdpi: 320 dpi
		xxhdpi: 480 dpi xxxhdpi: 640 dpi
		nodpi: utilizado para no escalar
		anydpi: cualquier densidad, coincide siempre

...		
19) Versión de la plataforma (nivel de API)	Ejemplos: `v3` `v4` `v7` etc.	

Más informacioón en: https://developer.android.com/guide/topics/resources/providing-resources?hl=es-419

Provisión de recursos alternativos

Reglas de nombres de calificadores

- Se pueden especificar varios calificadores, separados por guiones, para un solo conjunto de recursos. Por ejemplo, **drawable-en-rUS-land** se aplica a dispositivos de **inglés** (Estados Unidos) en **orientación horizontal**.

- Los calificadores **deben estar en el orden estipulado**, ver slide anterior. Por ejemplo:
  Incorrecto: drawable-hdpi-en-rUS/
  Correcto: **drawable-en-rUS-hdpi/**

- Los directorios de recursos alternativos **no pueden estar anidados**.
  Incorrecto: res/drawable/drawable-en/

- Los valores **no distinguen mayúsculas de minúsculas**.

- Se admite un solo valor para cada tipo de calificador.
  Incorrecto: drawable-rES-rFR/
  Correcto: drawable-rES/ y drawable-rFR/, mismos archivos ó uso de alias para un recurso.

Recursos de strings

Existen tres tipos de recursos que pueden proporcionar strings:

- **String**
  Recurso XML que ofrece un solo string.
  Archivo XML guardado en `res/values/strings.xml`

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

El nombre del archivo es arbitrario. El name del elemento <string> se usará como ID del recurso. Referencia del recurso:

En Java: *R.string.string_name*

En XML:*@string/string_name*

- **Matriz de strings**
  Recurso XML que ofrece una matriz de strings.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
    </string-array>
</resources>
```

```java
Resources res = getResources();

String[] planets =
res.getStringArray(R.array.planets_array);
```

- **Strings de cantidad (plurales)**
  Recurso XML que conlleva diferentes strings para brindar pluralización.

Más informacioón en: https://developer.android.com/guide/topics/resources/string-resource?hl=es-419

Recursos de strings

Alias de Strings

Para crear un alias de un string existente se utiliza el ID de recurso del string deseado como el valor del nuevo string.

Por ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <string name="hello">Hello</string>

    <string name="hi">@string/hello</string>

</resources>
```

El recurso `R.string.hi` ahora es un alias para `R.string.hello`.

Internacionalización

¿Como se puede internacionalizar una aplicación proveyendo los textos como recursos alternativos?

Supongamos la siguiente situación:

El lenguaje default de la aplicación es Inglés. Además queremos proveer todos los textos en francés y casi todos los textos en Japonés salvo el título.

Para esta situación es posible crear tres archivos `strings.xml` ubicados de esta manera:

1. **`res/values/strings.xml`**

   contiene todos los textos en inglés
2. **`res/values-fr/strings.xml`**

   contiene todos los textos en francés
3. **`res/values-ja/strings.xml`**

   contiene todos los textos en japonés excepto el título (en este caso `title`)

Si desde nuestro código Java utilizamos **`R.string.title`**, en ejecución sucede lo siguiente:

- Si el dispositivo tiene configurado cualquier lenguaje que no sea francés, Android cargará `title` desde `res/values/strings.xml`
- Si el dispositivo tiene configurado el lenguaje francés, Android cargará `title` desde `res/values-fr/strings.xml`

Si el dispositivo tiene configurado el lenguaje japonés, Android buscará `title` en `res/values-ja/strings.xml`, pero como no existe `title` en ese archivo resolverá cargarlo de `res/values/strings.xml`.

Internacionalización

¿Como es la resolución de recursos cuando no coincide la configuración regional de manera exacta?

Supongamos la siguiente situación:

> El idioma predeterminado de la app es `en_US` (inglés de EE. UU.), y también tiene strings en español localizadas en archivos de recursos `es_ES`. ¿Qué sucedería si por ejemplo se fija un dispositivo en el valor `es_MX`?.

Versiones anteriores a Android 7 (Nivel de API 24)

Cuando Android no encuentra una coincidencia exacta, continúa buscando recursos y quitando el código del país de la configuración regional. Si aún no encuentra una coincidencia, el sistema regresa a la configuración predeterminada, que es `en_US`.

Otro ejemplo

Configuración del usuario	Recursos de la app	Resolución de recursos
fr_CH	Predeterminado (en) de_DE es_ES fr_FR it_IT	Intentar con fr_CH => Error Intentar con fr => Error Usar predeterminado (en)

En Android 7 o superior

Configuración del usuario	Recursos de la app	Resolución de recursos
1. fr_CH	Predeterminado (en) de_DE es_ES fr_FR it_IT	Intentar con fr_CH => Error Intentar con fr => Error Intentar con campo secundario de fr => fr_FR Usar fr_FR

Tips

- Siempre incluir **todas** las imágenes y **todos** los textos que necesita la aplicación en los directorios default: `res/drawable/` y `res/values/` . De otra manera la aplicación fallará si el dispositivo tiene configurado un idioma no contemplado.

- Proveer los strings utilizando calificadores de idioma en lugar de idioma + región, por ejemplo para francés utilizar el calificador **fr** en lugar de **fr_FR**. De esta manera la resolución es más rápida y predecible.

- Para obtener el locale primario configurado en el dispositivo se puede utilizar:

```
Locale primaryLocale = context.getResources().getConfiguration().getLocales().get(0);
String locale = primaryLocale.getDisplayName();
```