Bloque 2. Desarrollo de Aplicaciones en Android

José A. Montenegro

Dpto. Lenguajes y Ciencias de la Computación
ETSI Informática. Universidad de Málaga
monte@lcc.uma.es twitter

22 de noviembre de 2011

Visión General de una Aplicación Android

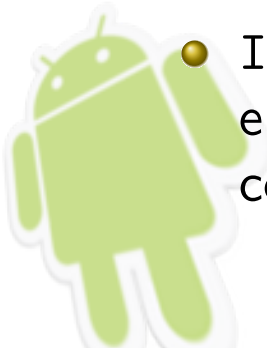
Cada aplicación Android es representada por un proyecto Android. En este bloque veremos una breve introducción de los elementos básicos de construcción de una aplicación, básicamente nos centraremos en el concepto de `activities` e `intents`.

- Una aplicación Android consta de varias funcionalidades. Por ejemplo algunas son editar una nota, reproducir un archivo de música, establecer una alarma o abrir un contacto de teléfono.
- Estas funcionalidades pueden ser clasificadas en cuatro componentes Android diferentes, mostrada en la siguiente tabla, cada una de las cuales es especificada por una clase base de Java.

Funcionalidad	**Clase Base Java**	**Ejemplo**
Acción que un usuario puede hacer	`Activity`	Editar una nota, jugar a un juego
Proceso en Segundo Plano	`Service`	Reproducir música, actualizar tiempo
Recibir mensajes	`BroadcastReceiver`	Lanzar alarma de un evento
Almacenar y obtener información	`ContentProvider`	Abrir un contacto de teléfono

Tabla 1: Los cuatro posibles componentes de una Aplicación Android

- Cada aplicación está formada de uno o más de los cuatro componentes, siendo instanciados por Android cuando se requieran. Además otras aplicaciones pueden utilizarlas, previo establecimiento de los correspondientes permisos.
- Debido que el S.Op. puede ejecutar múltiples funcionalidades a la vez (incluso no relativas con nuestra aplicación, como llamadas entrantes), cada componente tiene un ciclo de vida a la hora de ser creado, tener el foco, perder el foco y destruido.
- Menos `ContentProvider`, cada componente es activado por un mensaje asíncrono denominado `Intent`.
- `Intent` contiene un conjunto (`Bundle`) de información que describe el componente, además es un método para pasar información entre componentes.



Crear un Proyecto y una Activity

Volvemos a repasar la creación de un proyecto Android con Eclipse.

1. En Eclipse, escoger $File \rightarrow New \rightarrow AndroidProject$. Aparecerá una ventana para creación nuevo proyecto.
2. Rellenar el nombre del proyecto, como PrimerProyecto.
3. Seleccionar una de las versiones de las SDK instaladas.
4. Rellenar el nombre de la aplicación, como PrimerProyecto.
5. Rellenar el nombre del paquere, como es.uma.PrimerProyecto.
6. Crear la actividad (`Activity`) principal en el mismo paso, asegurarse que la opción `Create Activity` está marcada.

Todas las actividades extiende la clase abstracta `Activity` o alguna de sus subclases.

El punto de entrada de cada actividad es el método `onCreate()`.

Debemos sobreescribir el método para inicializar la actividad, para configurar el UI, crear los métodos de los botones, inicializar los parámetros y comenzar las hebras.

Si la actividad principal no es creada con el proyecto o debemos añadir otra actividad, los pasos para crear una actividad serán:

1. Crear una clase que extiende `Activity`. En Eclipse, sobre el proyecto con el botón derecho, escoger $New \rightarrow Class$, y especificar $android.app.Activity$ como clase padre.)
2. Sobreescribir el método `onCreate()`. En Eclipse, este puede ser realizando seleccionado la clase y con el botón derecho escoger $Source \rightarrow Override/ImplementMethods\ldots$, y seleccionar el método onCreate().)

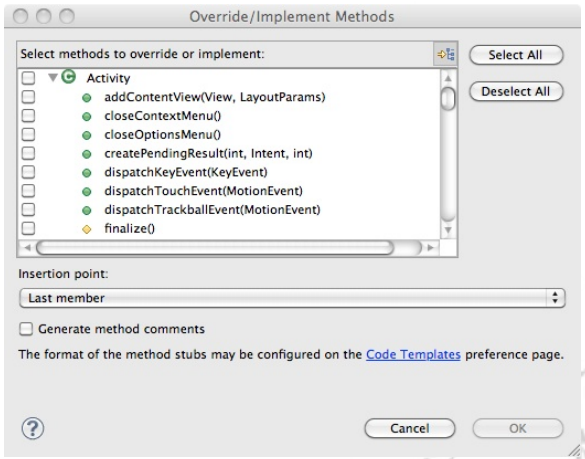


Figura 1: Sobreescribir el método.

- Como la mayoría de los métodos que sobreescribimos, debemos invocar al método de la clase padre, o una excepción es lanzada en tiempo de ejecución. En este caso `super.onCreate()` debe ser invocado primero para inicializar la actividad.

```
1  package es.uma.PrimerProyecto;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class PrimerProyectoActivity extends Activity {
7      /** Called when the activity is first created. */
8      @Override
9      public void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.main);
12     }
13 }
```

Código 1: src/es/uma/PrimerProyectoActivity.java

- Si utilizamos una UI, especificamos el diseño (layout) en un archivo XML en el directorio res/layout/.En este caso es denominado main.xml,tal y como muestra el siguiente listado.
- Establecemos el diseño de la actividad utilizando el método setContentView(), pasándole el ID del recurso del archivo XML. En este caso R.layout.main, como vimos en el listado anterior.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      >
7  <TextView
8      android:layout_width="fill_parent"
9      android:layout_height="wrap_content"
10     android:text="@string/hello"
11     />
12 </LinearLayout>
```

Código 2: res/layout/main.xml

- El siguiente paso es declarar las propiedades de la actividad en el archivo XML AndroidManifest XML.
- Los recursos son definidos en el archivo strings.xml en la carpeta res/values/ folder, como mostramos en el siguiente listado. Por tanto tenemos un lugar donde almacenamos todas las cadenas que necesitamos cambiar o reutilizar.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="hello">Hola Mundo, PrimerProyectoActivity!</string>
4     <string name="app_name">PrimerProyecto</string>
5 </resources>
```

Código 3: res/values/strings.xml

Una vez compilado el proyecto, las referencias a los recursos son referenciados a una clase denominada `R.java`. La aplicación `Asset Packaging Tool` (aapt) auto genera este archivo.

```
1  package es.uma;
2
3  public final class R {
4      public static final class attr {
5      }
6      public static final class drawable {
7          public static final int icon=0x7f020000;
8      }
9      public static final class layout {
10         public static final int main=0x7f030000;
11     }
12     public static final class string {
13         public static final int app_name=0x7f040001;
14         public static final int hello=0x7f040000;
15     }
16 }
```

Código 4: res/values/strings.xml

- En `R.java` cada recurso es mapeado a un único valor. De esta forma, la clase R.java proporciona una forma de referenciar recursos externos al código Java.
- Por ejemplo, para referencia el archivo `main.xml` en java, utilizamos el entero R.layout.main. Para referenciar el mismo elemento en XML, utilizaremos la cadena `@layout/main`.

Recurso	Referencia en Java	Referencia en XML
res/layout/main.xml	`R.layout.main`	@layout/main
res/drawable-hdpi/icon.png	`R.drawable.icon`	@drawable/icon
@+id/home_button	`R.id.home_button`	@id/home_button
⟨ string name=“hello” ⟩	`R.string.hello`	@string/hello

Tabla 2: Como diferentes recursos son referenciados desde Java y XML

Manifiesto Proyecto

- Para que el S. Op. accede a las aplicaciones, deben declarar sus componentes disponibles en un archivo XML `AndroidManifest`.
- Este archivo contiene los permisos necesarios y el comportamiento de la aplicación.

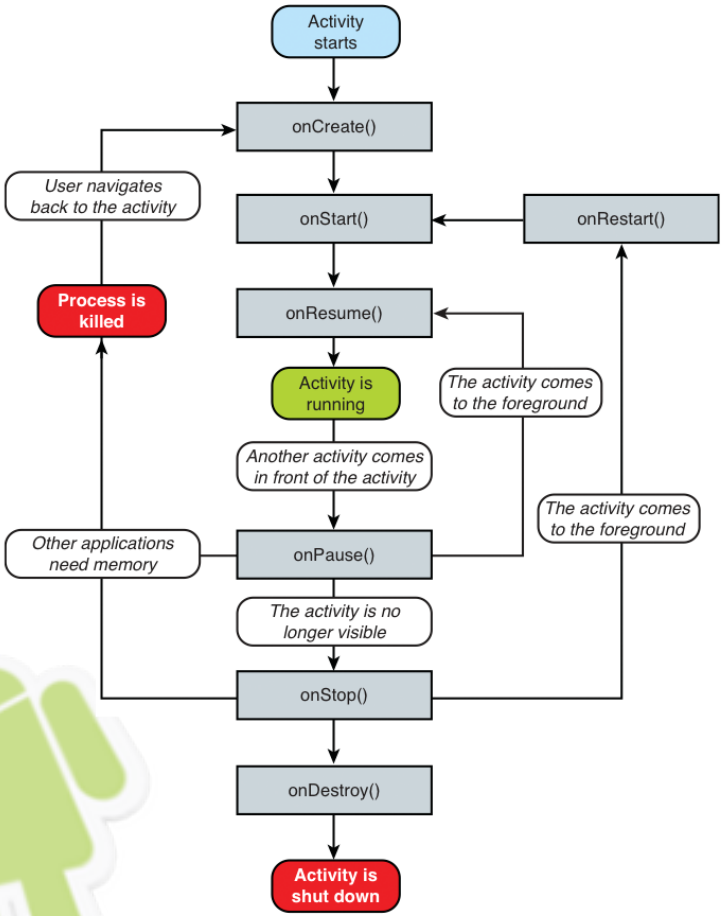
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        package="es.uma"
4        android:versionCode="1"  android:versionName="1.0">
5      <uses-sdk android:minSdkVersion="3" />
6      <application android:icon="@drawable/icon" android:label="@string/app_name">
7          <activity android:name=".PrimerProyectoActivity"
8                    android:label="@string/app_name">
9              <intent-filter>
10                 <action android:name="android.intent.action.MAIN" />
11                 <category android:name="android.intent.category.LAUNCHER" />
12             </intent-filter>
13         </activity>
14     </application>
15 </manifest>
```

- La primera línea es obligatoria en los archivos XML en Android y especifica la codificación.
- `manifest` define la versión y nombre del paquete Android. `versionCode` es un entero que será utilizado por los programas para las actualizaciones. `versionName` representa la misma información en formato textual.
- `application` define el icono y la etiqueta que el usuario verá en el menú del dispositivo. La etiqueta es una cadena con una dimensión suficiente para mostrar debajo del icono.
- `activity` define la actividad principal que es lanzada cuando la aplicación comienza y la el nombre que muestra en la barra del título cuando la actividad está activa.

- `intent-filter` informa a Android de las capacidades del componente, que pueden ser múltiples acciones, categorías o elementos.
- `uses-sdk` define el nivel de API necesaria para ejecutar la aplicación. En general, es especificada como sigue:

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

Ciclo de Vida Activity



- Cada actividad en una aplicación tiene su propio ciclo de vida.
- El método `onCreate()` es ejecutado cuando la actividad es creada. Por otro lado si la actividad finaliza, ejecutaremos la función `onDestroy()`.
- Entre los dos estados, varios eventos pueden darse y la actividad pasará por múltiples estados, como muestra la figura 2.

Práctica 1

Vamos a crear un método para cada una de los estados de la actividad y estudiar que ocurre cuando realizamos las siguientes acciones:

- *Cambiar la orientación de la pantalla destruye y recrea la actividad.*
- *Presionar el botón Home pausa la actividad pero no la destruye.*
- *Presionar el icono de Aplicación podría dar comienzo a una nueva instancia de la actividad, incluso si la antigua no se ha destruido.*
- *Dejar el equipo sin actividad pausa la actividad y cuando lo volvemos activar la tarea se reactiva. (Similar a una llamada entrante)*

Por ejemplo en el método onCreate:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.v(this.name, "Metodo Crear");
}
```

Modificando el comportamiento Ciclo Vida Actividad

Necesitamos realizar una serie de cambios en el AndroidManifest.xml. Concretamente son realizados en el elemento `activity` que contiene los `intent filters` `MAIN` y `LAUNCHER`.

- Normalmente una aplicación puede tener varias instancias ejecutándose en el dispositivo. Normalmente las instancias redundantes de una actividad son eliminadas para liberar memoria y ciertas situaciones no deseadas pueden ocurrir.
- Para asegurar que solamente una instancia de la actividad se ejecuta en el dispositivo, incluiremos:

android:launchMode="singleInstance"

- que mantiene una única instancia de cada actividad en una tarea y cualquier hijo es lanzado como su propia tarea. Para restringir aún más y solo tener una sola tarea para todas las actividades de una aplicación utilizaremos:

android:launchMode="singleTask"

- En ciertas ocasiones es deseable mantener el estado de una tarea. Por ejemplo, si el usuario deja una aplicación y la lanza más tarde, el comportamiento natural volvería a su estado inicial.

  android:alwaysRetainTaskState="true"

- Cuando el dispositivo es modificado de vertical a apaisado, la aplicación debe rotar para cambiar la vista acorde con el giro. Como vimos anteriormente la actividad es destruida y reiniciada, perdiendo el estado actual de la actividad.

  Podemos especificar la posición de la pantalla por defecto con:

  android:screenOrientation="portrait"
  android:screenOrientation="landscape"

Más información puede ser consultada en: Android Developer Web.

Relación Activities e Intent

- Sabemos que una actividad de Android es una unidad para la interacción con el usuario que normalmente ocupa la pantalla del dispositivo y una unidad de ejecución. Todo programa que necesite de la interacción con el usuario debe comenzar haciendo una subclase de la clase `Activity`.
- Ahora, necesitamos saber como una actividad puede invocar a otra, y pasar información sobre que quiere hacer el usuario.
- La unidad de comunicación es la clase `Intent`.

  Un `Intent` representa una descripción abstracta de una función que una actividad necesita que otra actividad realice, como por ejemplo tomar una foto.
- Cuando una aplicación ejecuta un `Intent`, es posible que varias actividades diferentes estén registradas para proporcionar una operación.

- Sabemos que una actividad de Android es una unidad para la interacción con el usuario que normalmente ocupa la pantalla del dispositivo y una unidad de ejecución. Todo programa que necesite de la interacción con el usuario debe comenzar haciendo una subclase de la clase `Activity`.
- Ahora, necesitamos saber como una actividad puede invocar a otra, y pasar información sobre que quiere hacer el usuario.
- La unidad de comunicación es la clase `Intent`.

  Un `Intent` representa una descripción abstracta de una función que una actividad necesita que otra actividad realice, como por ejemplo tomar una foto.
- Cuando una aplicación ejecuta un `Intent`, es posible que varias actividades diferentes estén registradas para proporcionar una operación.

- Por todo lo comentado anteriormente, el código que implementa una actividad no invoca directamente métodos que implementan otra actividad, tienen que hacer uso de `Intent`.
- El entorno de ejecución de Android, que crea y administra las actividades a menudo reclama la memoria que utilizan las actividades para restringir que las tareas utilicen una cantidad pequeña de memoria.
- En vez de utilizar un flujo de control basadas en las llamadas a los métodos, las aplicaciones describen mediante un Intent que quieren ejecuta y preguntan al sistema que encuentre quién puede realizarlo.

- La aplicación inicial comienza el programa utilizando estas descripciones, y cada aplicación puede hacer los mismo utilizando su propia selección de intents. El flujo resultante es una tarea: una cadena de actividades que a menudo es realizada en más de una aplicación (tabla 3).

Aplicación	**Actividad**	**Próxima Acción Usuario**
Mensaje	Ver lista de mensaje	Selecciona un mensaje de la lista
Mensaje	Ver un mensaje	Selecciona $Menu \rightarrow VerContacto$
Contactos	Ver un contacto	Selecciona llamar al contacto
Teléfono	Llamar al número del contacto	

Tabla 3: Ejemplo de una tarea, realizada mediante actividades en varias aplicaciones

Declarar Intent en la aplicación

- Anteriormente hemos comentado que una vez invocada una acción el sistema busca todas las aplicaciones que están subscritas.
- Las aplicaciones hacen su declaración mediante el Manifiesto que vimos anteriormente. Dentro de la sección `activity` en el atributo que mencionamos anteriormente `intent-filter`.

`intent-filter` Declaramos aquí un filtro que comunica Android cuando esta Actividad debe ser ejecutada. Cuando una aplicación pregunta al sistema para completar un Intent, el sistema busca entre todas las actividades y servicios. Debemos establecer dos atributos.

`action` Dice al sistema como lanzar esta aplicación una vez que se ha decido que esta es la aplicación que tenemos que ejecutar.Android buscará una actividad que declare adecuada para resolver la acción `MAIN`.

`category` El sistema utiliza este atributo para calificar el Intent que está buscando. Es posible tener una aplicación válida sin este atributo pero no es posible lanzarla desde el escritorio.

Objetos Intent

- Hemos comentado anteriormente que cada Actividad puede hacer una llamada a Intent para que realice una tarea sin saber quien exactamente recibe ese llamada.
- Intent por tanto son asignados a una componente que realiza la tarea en tiempo de ejecución. Esta situación permite desacoplar los componentes individuales y realizar un mantenimiento sin que afecte a todo el sistema.
- Por tanto inicialmente veremos como invocar un objeto Intent para que realice una tarea y como Android resuelve la petición con las clases IntentFilter.
- Android tiene por defecto Intents disponibles para que cualquiera los pueda utilizar.

Definición Intent

- Para la creación de un Intent necesitamos dos elementos básicos de información. Una *acción* que es un verbo describiendo que queremos hacer y *datos* que es un nombre que detalla que vamos hacer.

  Por ejemplo, si queremos cifrar un mensaje de correo, tendríamos como acción la función de cifrar y como datos el mensaje de correo.

Elemento Intent	Descripción
`Action`	Cadena con un formato especifica que indica la acción. `android.intent.action.CIPHER`
`Category`	Describe donde y como el Intent puede ser usado, menú principal o navegador
`Component`	Especifica un paquete o clase para usar el Intent, en vez de utilizar action, type y category
`Data`	Datos con los que trabajar, expresad como una URI.
`Extras`	Información adicional
`Type`	Tipo MIME, tal como `text/plain` o `vnd.android.cursor.item/email_v2`

Tabla 4: Los cuatro posibles componentes de una Aplicación Android

Invocación Explícita e Implícita

- Una invocación implícita ocurre cuando la plataforma determina que componente debe ejecutar el Intent.
- En otras ocasiones, queremos que una acción sea ejecutada por un código propietario y que el sistema no administre dicha acción. Es el caso de una invocación explícita.
  - La invocación es realizada de forma explícita si especificamos Clase (Class) o Nombre del Componente (ComponentName) del receptor.
  - El nombre del componente es un nombre de clase completa, que consiste en una cadena con el paquete y la clase.

```
Intent(Context ctx, Class cls)
```

  - Finalmente ejecutamos la Actividad.

```
startActivity(intent);
```

```java
/* Definicion varible */
Intent intent = null;

/* Abrir un enlace web */
intent = new Intent(Intent.ACTION_VIEW,Uri.parse(link));
startActivity(intent);

/* Buscar una localizacion */
intent=new Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=" + dirreccionCalle);
startActivity(intent);

/* Llamar por telefono */
intent = new Intent(Intent.ACTION_CALL,Uri.parse("tel:" + telefono));
startActivity(intent);
```

Código 6: Ejemplos de invocación explícita Intent

- Hemos visto anteriormente que la aplicación muestra que `Intent` puede ejecutar mediante el elemento <intentfilter> en el Manifiesto.
- Android convierte cada <intent-filter> en un objeto `IntentFilter`.
- Después de la instalación de una aplicación (archivo .apk), registra los componentes de la aplicación incluyendo los filtros Intent.
- Ahora el sistema ya sabe como realizar las acciones, para ello hace uso de las propiedades `action`, `data` y `categories`. Para realizar la coincidencia debe cumplir las siguientes condiciones:
  - Deben coincidir los atributos `action` y `category`.
  - Si especificamos el tipo de los datos deberá coincidir.

Actions y Categories

- Cada IntentFilter puede especificar cero o más acciones y cero o más categorías. Si no especificamos ninguna acción en el IntentFilter, coincidirá con cualquier Intent, de otro modo solamente coincidirá con el Intent que tiene la misma acción.
- Un IntentFilter sin categorías coincidirá solamente con Intent que no tiene categorías; de otro modo un IntentFilder debe tener al menos las que el Intent especifica.

```
<activity android:name="Ejemplo" android:label="@string/app_name">
<intent-filter>
<category android:name="android.intent.category.DEFAULT" />
<action android:name="es.uma.ejemplo.VIEW_LIST" />
</intent-filter>
</activity>
```

Para coincidir con el filtro declarado, utilizamos el siguiente código, donde Constants.INTENT_ACTION_VIEW_LIST es la cadena es.uma.ejemplo.VIEW_LIST:

```
Intent intent = new Intent(Constants.INTENT_ACTION_VIEW_LIST);
startActivity(intent);
```

Constantes	Acción
ACTION_CALL	Inicializa una llamada de teléfono.
ACTION_EDIT	Muestra datos del usuario para editar.
ACTION_MAIN	Inicia la actividad inicial de una tarea.
ACTION_SYNC	Sincronizar datos en un servidor.

Tabla 5: Ejemplos de Acciones Predefinidas

Constantes	Significado
CATEGORY_BROWSABLE	La actividad puede ser invocada por el navegador para mostrar datos de un enlace.
CATEGORY_GADGET	La actividad puede estar incluida en otra actividad.
CATEGORY_HOME	La actividad se mostrará en la pantalla inicial (HOME).
CATEGORY_LAUNCHER	La actividad será la actividad inicial de una tarea.
CATEGORY_PREFERENCE	La actividad es una panel preferente.

Tabla 6: Ejemplos de Categorías Predefinidas

Data

Una vez que el sistema ha determinado que las propiedades action y category coinciden, pasan a mirar el campo data.
Los datos pueden ser un tipo MIME explícito (audio/mpeg) o una combinación de esquema, autoridad y ruta.

weather:// com.msi.manning/loc?zip=12345
esquema: weather:// autoridad:com.msi.manning/ data:loc?zip=12345

Las clases IntentFilter describe que combinación de tipo, esquema, autoridad y ruta aceptan. El proceso para determinar la coincidencia con Intent es el siguiente:

1. Si un esquema es presente y el tipo no, Intents con cualquier tipo coincide.
2. Si un tipo está presente y el esquema no está presente, Intents con cualquier esquema coincidirá.
3. Si ni el esquema ni el tipo están presente, solamente Intents si esquema ni tipo coincidirá.
4. Si una autoridad es especificada, un esquema debe ser especificado.
5. Si la ruta es especificada, un esquema y una autoridad deben ser especificada.

Servicio

Un Servicio (Service) es un componente que permite realizar operaciones mas costosas computacionalemnte en segundo plano y sin proporcionar un interfaz de usuario.

Normalmente una Actividad comenzará un servicio que continuará ejecutándose en segundo plano, incluso si el usuario cambia a otra aplicación.

Además, un elemento puede vincularse a un servicio para interactuar con él. Por ejemplo, un servicio puede administrar comunicaciones de red, reproducir música, realizar operaciones de entrada/salida, todo desde el segundo plano.

Un servicio puede tomar esencialmente dos formas:

Started Este caso sucede cuando por ejemplo una actividad[1] invoca el método `startService( )`.

Una vez que comienza el servicio se ejecutará en segundo plano indefinidamente, incluso si la actividad que la comenzó ha sido destruida.

Normalmente, el servicio realiza una operación y cuando termina vuelve el resultado a la actividad y el servicio se detendrá el mismo.

[1] Realmente existen más elementos que se pueden vincular a una actividad, por motivos de claridad lo dejamos reducido a las actividades.

Bound Este caso sucede cuando una aplicación se vincula a el llamando `bindService( )`. Un servicio vinculado ofrece un interfaz cliente-servidor que permite que interactuen con él.

Un servicio vinculado solamente se ejecuta cuando existe un elemento (actividad) vinculado a él.

Varias actividades pueden ser establecidas al mismo servicio, pero una vez que todas se desvinculen, el servicio finalizará.

Ciclo Vida Servicio

```
1  public class ServicioEjemplo extends Service {
2      int mStartMode;       // indica como comportarse si el servicio es eliminado
3      IBinder mBinder;      // interfaz para los clientes vinculados
4      boolean mAllowRebind; // indica si onRebind puede ser usado
5
6      public void onCreate() { // El servicio es creado
7      }
8      public int onStartCommand(Intent intent, int flags, int startId) {
9          // El servicio es creado debido a una llamada a startService()
10         return mStartMode;
11     }
12     public IBinder onBind(Intent intent) {
13         // Un cliente es vinculado al servicio con bindService()
14         return mBinder;
15     }
16     public boolean onUnbind(Intent intent) {
17         // Todos los clientes se han desvinculado con unbindService()
18         return mAllowRebind;
19     }
20      public void onRebind(Intent intent) {
21          // Un cliente esta vinculado al servicio con bindService(),
22     }
23    public void onDestroy() {  // El servicio no sera utilizado y ser destruido
24     }
25 }
```

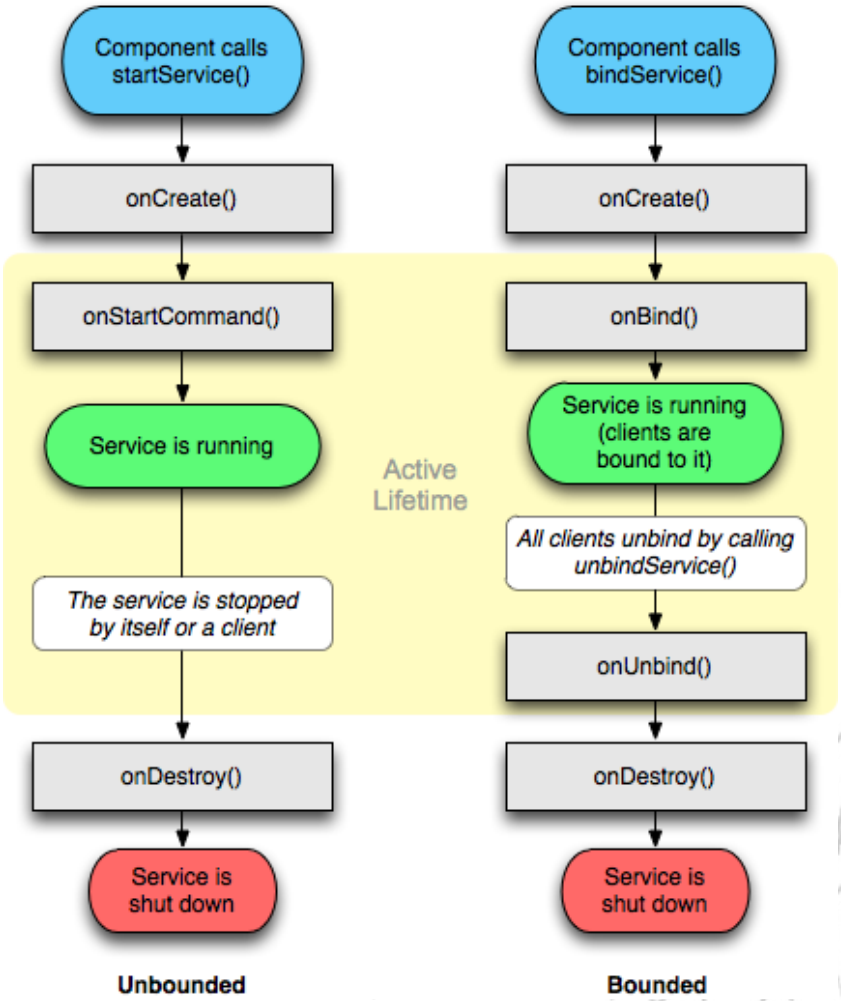


Figura 3: Ciclo Vida Servicio

Pasos para crear un Servicio "Started"

Paso 1: Comenzamos creando un clase servicio que hereda de `android.app.Service`. Mostraremos un mensaje cuando comienza y cuando finaliza. Para ello utilizaremos los métodos onStart() y onDestroy().

```
1  public class ServicioEjemplo extends Service {
2       public IBinder onBind(Intent arg0) {
3             return null;
4       }
5     public void onCreate() {
6             super.onCreate();
7             Toast.makeText(this,"Servicio creado ...", Toast.LENGTH_LONG).show();
8       }
9       public void onDestroy() {
10            super.onDestroy();
11            Toast.makeText(this, "Servicio eliminado ...", Toast.LENGTH_LONG).show();
12      }
13 public int onStartCommand(Intent intent, int flags, int startId) {
14       Toast.makeText(this, "Servicio iniciado ...", Toast.LENGTH_LONG).show();
15  return mStartMode;
16     }
17 }
```

Paso 2: Debemos incluir una entrada en el manifiesto (`AndroidManifest.xml`):

```
<service android:name=".ServicioEjemplo">
</service>
```

Paso 3: Ahora podemos ejecutar el servicio. Para ello escribimos una actividad que tenga dos botones para ejecutar y parar el servicio:

Utilizaremos esta sentencias para comenzar el servicio

```
startService(new Intent(this,ServicioEjemplo.class));
```

Y esta sentencia para parar el servicio

```
stopService(new Intent(this, ServicioEjemplo.class));
```

Introducción

- Esta sección proporcionalas técnicas para implementar un interfaz gráfico en Android. Explica la arquitectura de UI de Android así como incluir los elementos básicos y la interacción con ellos.
- El marco de trabajo UI de Android, es como otros Java UI, organizados mediante el patrón Modelo-Vista-Controlador. Básicamente proporciona las estructuras y las herramientas para construir un Controlador que administra las entradas del usuario y una Vista que muestra la información de forma gráfica.

Modelo : Es la base de la aplicación, lo que realmente hace. Mientras que una la Vista y el Controlador de una aplicación concreta necesitan reflejar el Modelo que utilizan, un Modelo puede ser usados por varias aplicaciones.

Vista : Es la visualización del Modelo. Más generalmente, una vista es la porción de la aplicación responsable para mostrar la pantalla, enviar audio a los altavoces ... etc. La UI dibuja la pantalla mediante la exploración del árbol de los elementos de la vista, dibujando cada elemento y solicitando que los hijos de cada uno realicen la misma opción.

Controlador : Es la porción de la aplicación que responde a acciones externas: como pulsación de teclas, llamada entrante ... etc. Es implementado como una cola FIFO de eventos.

Class View

- Las Vistas son los bloques de construcción de la UI de una aplicación Android. Las Actividades contienen vistas y las clases View representan elementos en la pantalla y son los responsables para la interacción con los usuarios mediante los eventos.
- Cada pantalla Android contiene una jerarquía de elementos View. Estas vistas poseen su propias formas y tamaños.
- Mucho de los elementos básicos son proporcionados por la plataforma como elementos de texto, de entrada de datos, imágenes, botones ..etc. Además es posible crear vistas compuestas y a medida tal como son requeridas.
- Podemos establecer las vistas en una Actividad mediante código Java o utilizando código XML.

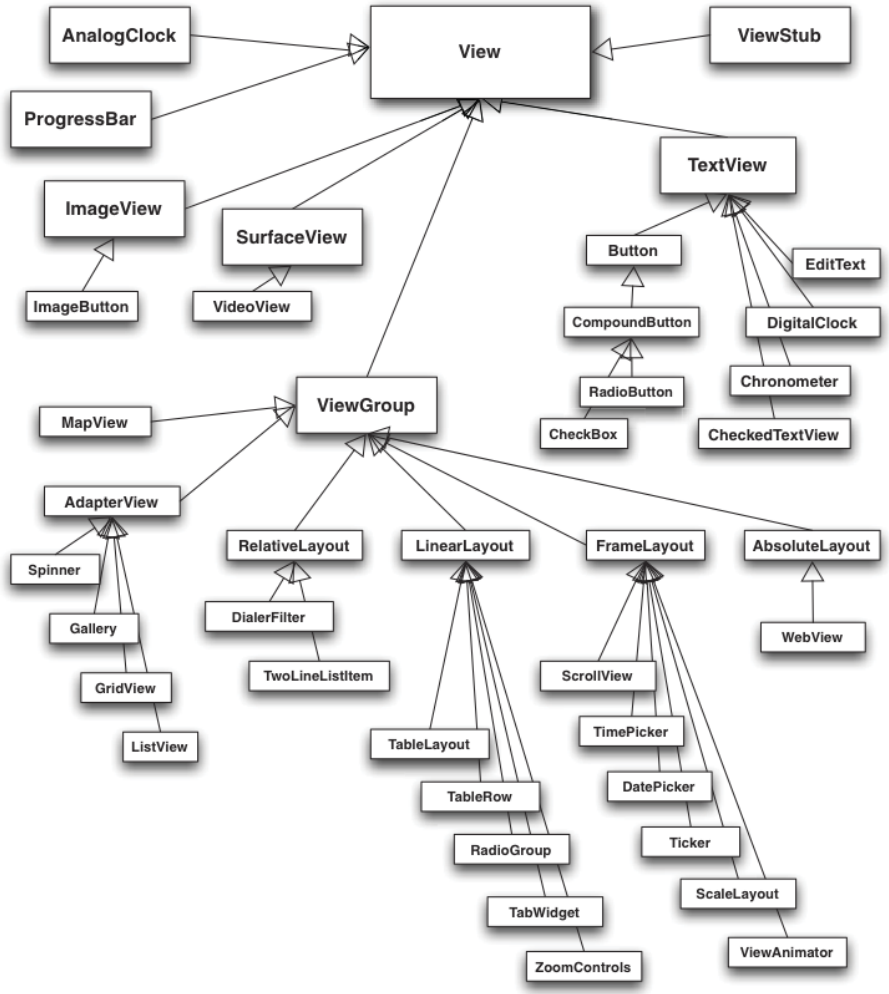


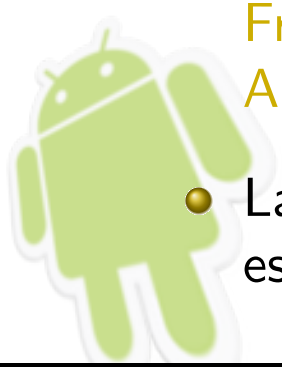
Figura 4: Jerarquía de Clases View

ViewGroup y Layout

- En la figura 4 observamos que entras las distintas subclases de View tenemos ViewGroup.
- Es una vista especial que contiene otras vistas hijas y además es la clase base los layouts y vistas contenedoras. También define la clase ViewGroup.LayoutParams.
- Los tipos de layout disponibles son:

  LinearLayout: dispone los hijos horizontal o verticalmente
  RelativeLayout: dispone unos elementos relativos a los otros
  FrameLayout: permite cambiar dinámicamente los controles
  AbsoluteLayout: dispone los elementos en coordenadas exactas

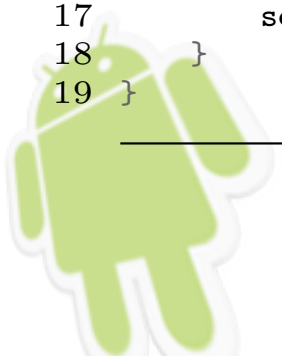
- La clase Layout contiene una clase LayoutParams que es especializada por cada tipo de Layout.



Creación Botón mediante Código Java

```
1  public class PrimerProyectoActivity extends Activity {
2      /** Called when the activity is first created. */
3      @Override
4      public void onCreate(Bundle savedInstanceState) {
5          super.onCreate(savedInstanceState);
6
7          LinearLayout.LayoutParams widgetParams = new LinearLayout.LayoutParams(
8                  ViewGroup.LayoutParams.FILL_PARENT,
9                  ViewGroup.LayoutParams.FILL_PARENT,
10                 1.0F);
11
12         Button b = new Button(this);
13         b.setText("PrimerBoton");
14         b.setTextColor(Color.RED);
15         b.setLayoutParams(widgetParams);
16
17         setContentView(b);
18     }
19 }
```

Código 7: Utilización Clase Button



Creación Botón mediante Código XML

```xml
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button1"
    android:text="@string/labelRed"
    android:textColor="@drawable/red"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"/>
```

Código 8: Inclusión Botón en main.xml

```java
package es.uma;

public class PrimerProyectoActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

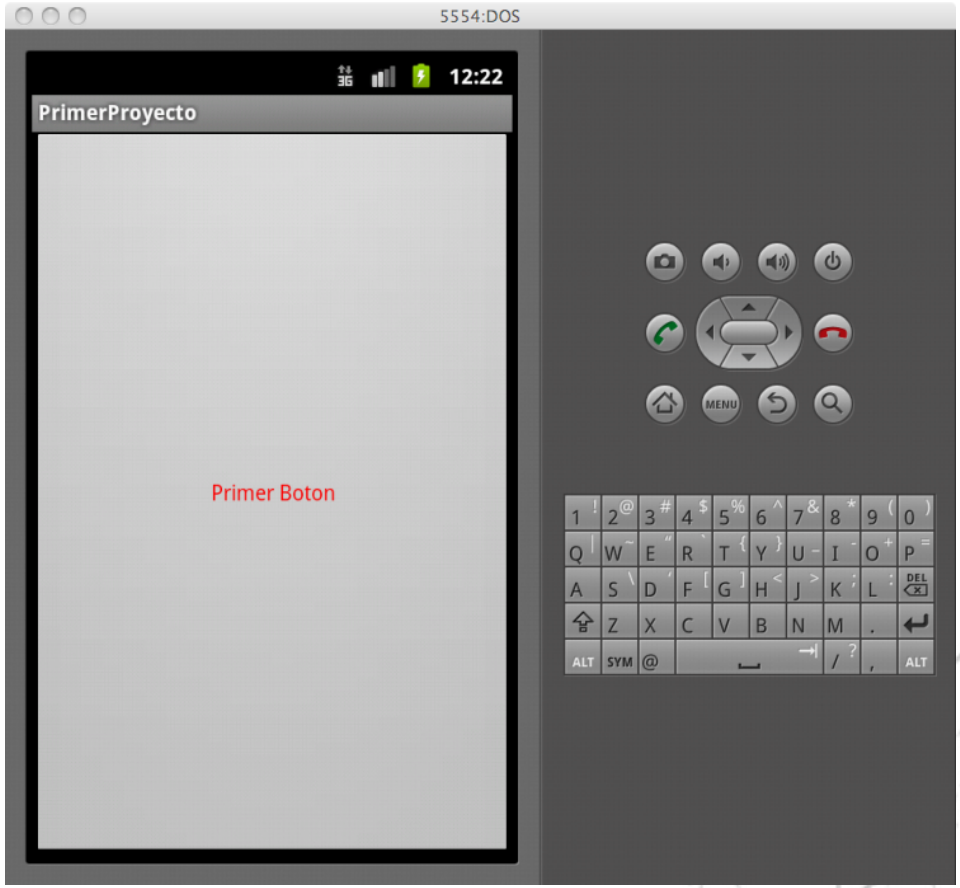


Figura 5: Ejecución de la clase Button

LinearLayout

En la figura 5 hemos visto la inclusión de un botón directamente sin utilizar un Layout. El siguiente código realiza la representación utilizando LinearLayout y dos botones. El resultado es mostrado en la figura 6 .

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      android:id="@+id/root" android:orientation="vertical"
3      android:layout_width="fill_parent"
4      android:layout_height="wrap_content">
5
6    <LinearLayout android:orientation="horizontal"
7      android:layout_width="fill_parent" android:layout_height="wrap_content">
8
9  <Button android:id="@+id/button1" android:text="@string/labelButton1"
10     android:layout_width="fill_parent" android:layout_height="fill_parent"
11     android:layout_weight="1"/>
12
13 <Button android:id="@+id/button2"    android:layout_width="fill_parent"
14       android:layout_height="fill_parent" android:layout_weight="1"
15       android:text="@string/labelButton2"/>
16   </LinearLayout>
17 </LinearLayout>
```

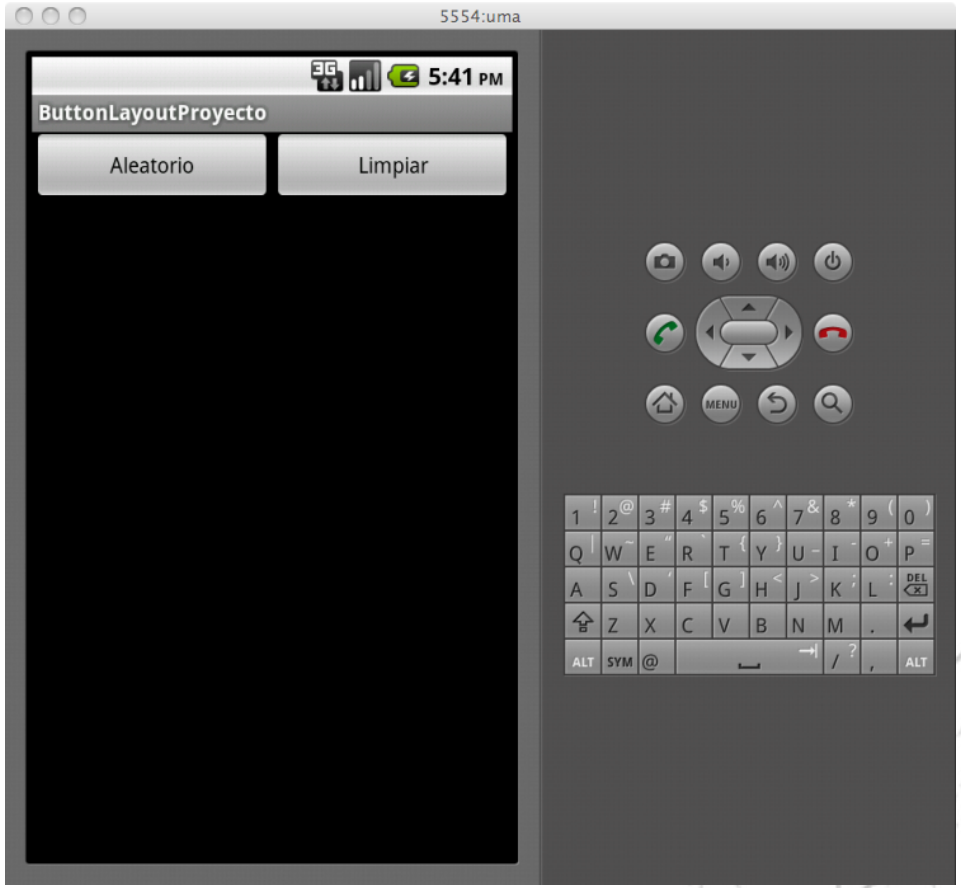


Figura 6: LinearLayout

EditText

Otro elemento visual que nos será de utilidad en el desarrollo del interfaz gráfico es `EditText`. Tanto la clase `Button` utilizada anteriormente como `EditText` heredan de la clase `TextView`.

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      android:id="@+id/root" android:orientation="vertical"
3      android:layout_width="fill_parent"  android:layout_height="wrap_content">
4    <LinearLayout android:orientation="horizontal"
5        android:layout_width="fill_parent" android:layout_height="wrap_content">
6       <EditText android:id="@+id/text"
7       android:focusable="false" android:layout_width="fill_parent"
8       android:layout_height="fill_parent" android:layout_weight="1"/>
9   </LinearLayout>
10  <LinearLayout android:orientation="horizontal"
11          android:layout_width="fill_parent" android:layout_height="wrap_content">
12 <Button android:id="@+id/button1" android:text="@string/labelButton1"
13      android:layout_width="fill_parent" android:layout_height="fill_parent"
14      android:layout_weight="1"/>
15 <Button android:id="@+id/button2"     android:layout_width="fill_parent"
16       android:layout_height="fill_parent" android:layout_weight="1"
17       android:text="@string/labelButton2"/>
18    </LinearLayout>
19 </LinearLayout>
```

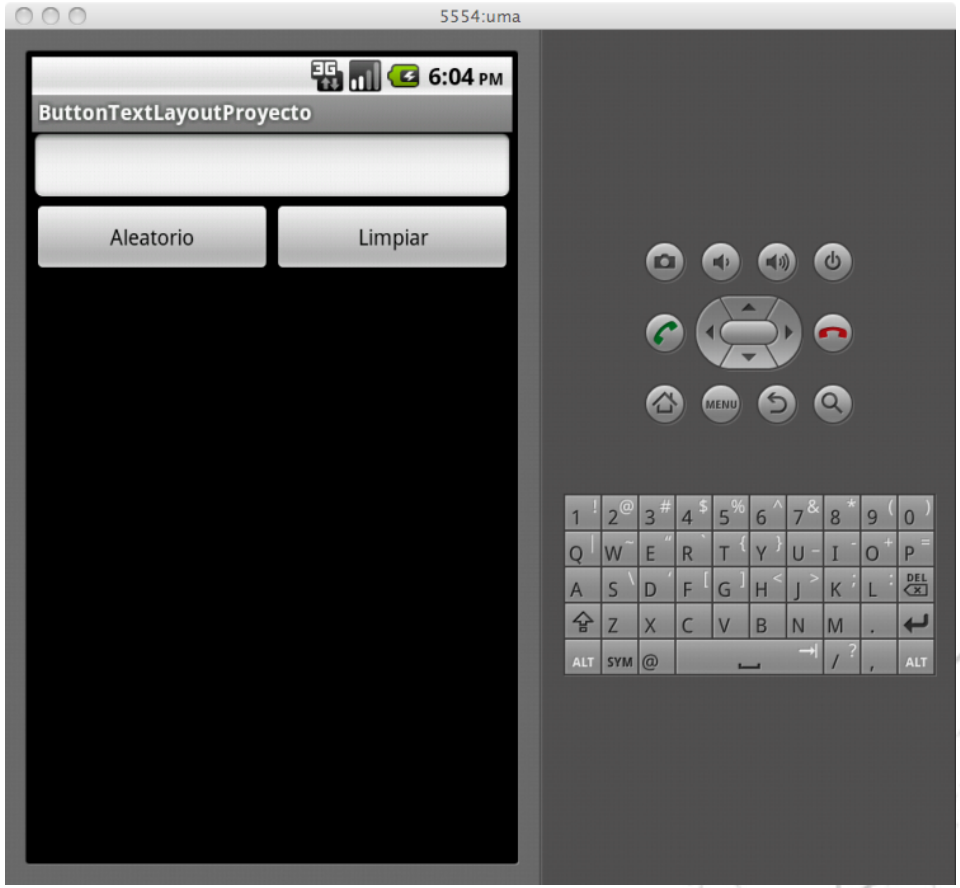


Figura 7: Inclusión EditText

Capturando Eventos

- Una vez que tenemos establecidos dos elementos básicos visuales, vamos a controlar los eventos del botón.
- El control se realizará desde el código Java, por lo que necesitamos una referencia al elemento XML, mediante el método `findViewById`.

```
Button aletorio= findViewById(R.id.button1)
```

- Una vez recuperado el objeto es necesario establecer el Listener del evento, en este caso el evento `onClick`.

```
 aletorio.setOnClickListener(new Button.OnClickListener() {
public void onClick(View v) {
      /*Codigo aqui*/
    }
 });
```

```
1       public void onCreate(Bundle savedInstanceState) {
2         super.onCreate(savedInstanceState);
3         setContentView(R.layout.main);
4
5         final EditText tb1 = (EditText) findViewById(R.id.editText1);
6         Button aleatorio= (Button) findViewById(R.id.button1);
7         Button limpiar= (Button) findViewById(R.id.button2);
8
9         aleatorio.setOnClickListener(new Button.OnClickListener() {
10         public void onClick(View v) {
11         Random generator = new Random();
12         int rand= generator.nextInt();
13                String rands = Integer.toString(rand);
14                tb1.setText(rands);
15 }
16        });
17
18        limpiar.setOnClickListener(new Button.OnClickListener() {
19         public void onClick(View v) {
20         tb1.setText("");
21 }
22        });
23      }
```

Código 9: Eventos en ProyectoEventoActivity.java

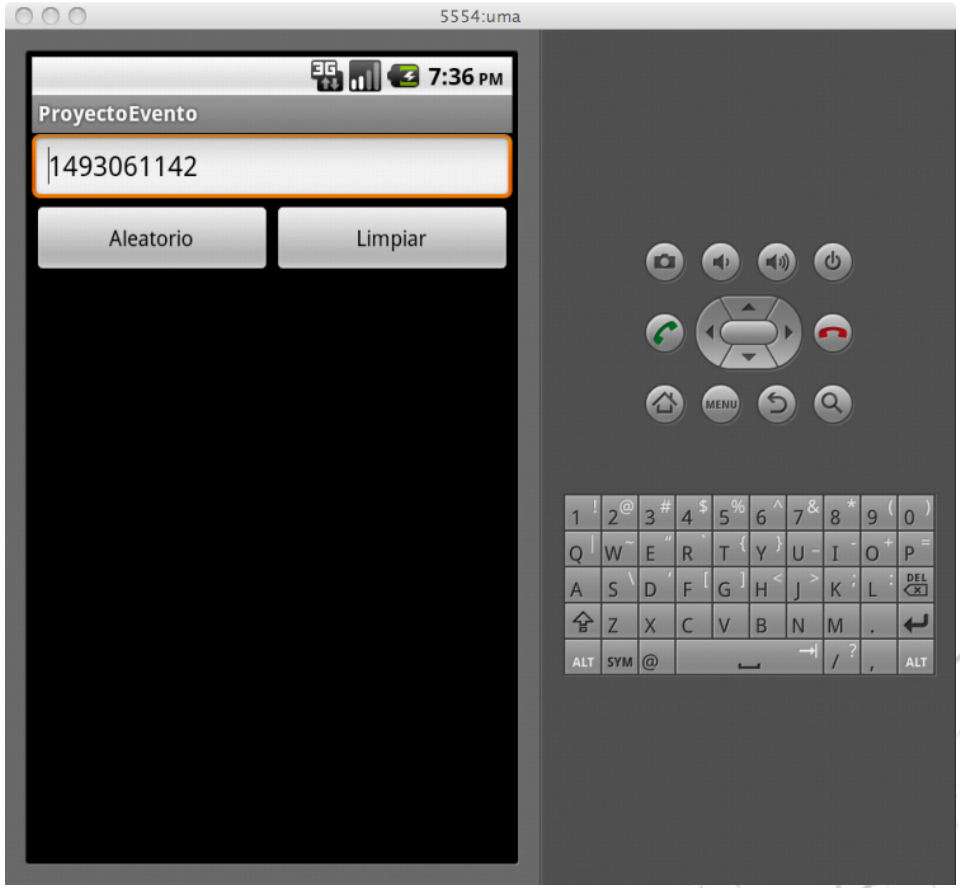


Figura 8: Tratamiento Eventos

Menu

Figura 9: Inclusion Menu

- Para incluir el menú en la aplicación solamente hay que sobreescribir dos métodos de la actividad principal:
- `onCreateOptionsMenu`: Este método permite añadir elementos al menú.
- `onOptionsItemSelected`: Este método declara las acciones a realizar cuando el menú es seleccionado.
- El código 10 muestra en detalle como crear dos elementos del menú, uno que limpia las casillas y otro que muestra el nombre del curso.

```
1   public boolean onCreateOptionsMenu(Menu menu) {
2       menu.add(Menu.NONE, CLEAR_MENU_ID, Menu.NONE, "Limpiar");
3       menu.add(Menu.NONE, ABOUT_MENU_ID, Menu.NONE, "About");
4       return super.onCreateOptionsMenu(menu);
5       }
6
7   public boolean onOptionsItemSelected(MenuItem item) {
8        final EditText tb1 = (EditText) findViewById(R.id.editText1);
9        switch (item.getItemId()) {
10          case CLEAR_MENU_ID: tb1.setText("");  return true;
11          case ABOUT_MENU_ID:
12                Toast.makeText(this, "Curso Criptografia en Android", Toast.LENGTH_LONG).show();
13                return true;
14        default: ; return false;
15    }
16  }
```

Código 10: Creación Menu

Barra de Estado

Este tipo de notificación añade un icono y un mensaje a la barra de estado del sistema. El proceso de creación consta de cuatro pasos que son mostrados en detalle por el código 11 :

1. Obtener una referencia al administrador de notificaciones `NotificationManager`.
2. ICrear un objeto Notificación.
3. Definir el mensaje de la notificación y establecer `PendingIntent`.
4. Enviar la notificación al administrador de notificaciones.

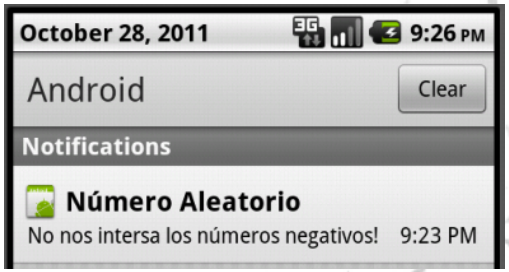


Figura 10: Notificación que el número aleatorio es negativo

```
1  private void numeroNegativoEvento(){
2
3        CharSequence title = "Numero Aleatorio";
4        CharSequence message = "No nos interesa los numeros negativos!";
5
6        NotificationManager notificationManager =
7         (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
8
9        Notification notification =
10       new Notification(R.drawable.ic_launcher, "Numero Negativo!", System.currentTimeMillis());
11                              notification.defaults |= Notification.DEFAULT_SOUND;
12                              notification.defaults |= Notification.DEFAULT_VIBRATE;
13                              notification.defaults |= Notification.DEFAULT_LIGHTS;
14
15       Intent notificationIntent = new Intent(this, ProyectoEventoActivity.class);
16       PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
17
18       notification.setLatestEventInfo(ProyectoEventoActivity.this, title, message, pendingIntent);
19       notificationManager.notify(NOTIFICATION_ID, notification);
20   }
```

Código 11: Notificación Barra Estado

Toast

- Es un mensaje que muestra una notificación superpuesta en la ventana principal.
- La ventana tendrá el espacio requerido por el mensaje.
- La notificación aparece y desaparece automáticamente y no acepta eventos, para tal caso utilizamos Barra de estado.

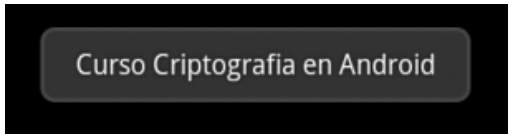


Figura 11: Toast que muestra información del sistema

```
Context context = getApplicationContext();
CharSequence text = "Curso Criptografia en Android";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Galería de Componentes Visuales

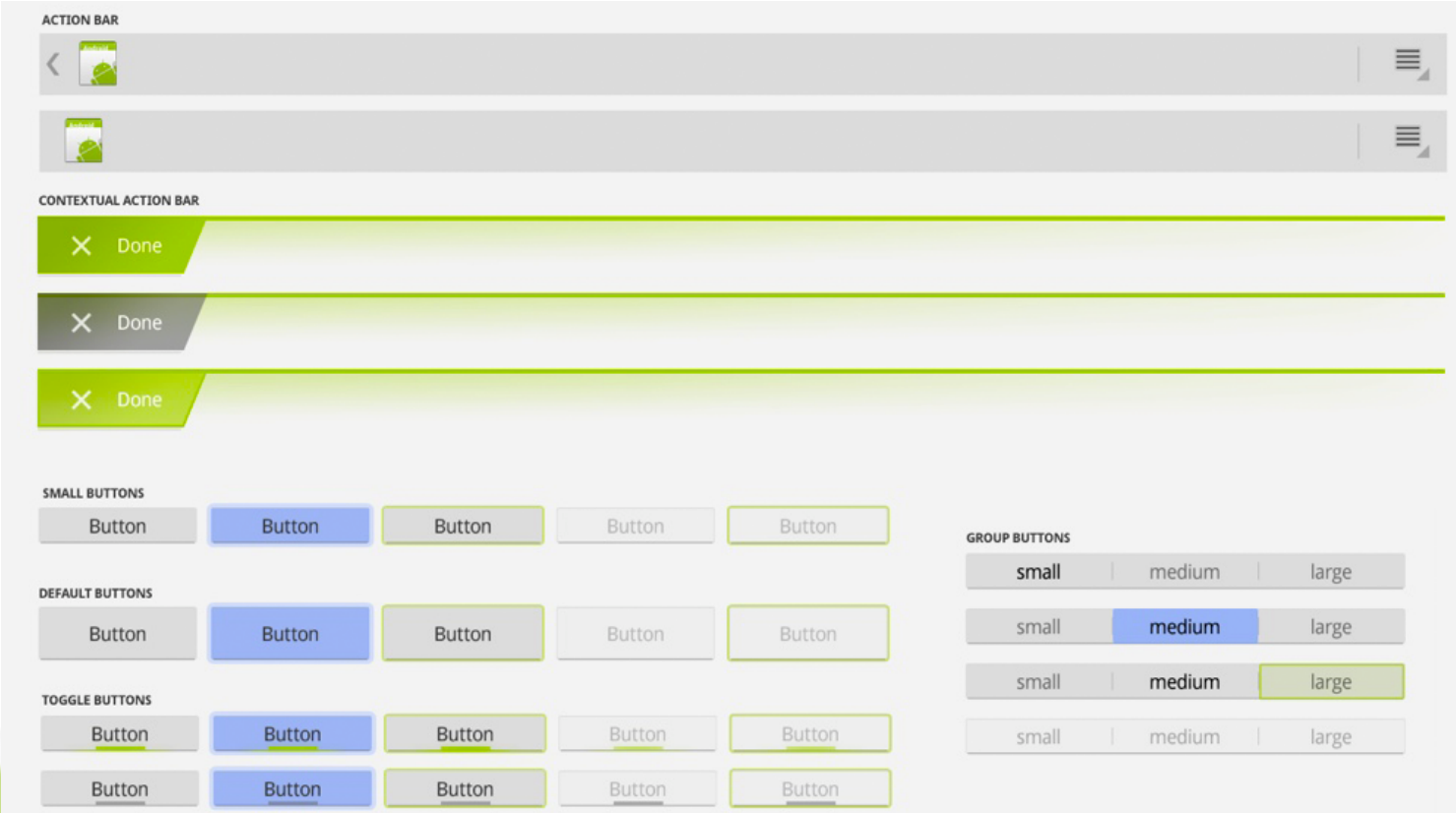


Figura 12: Distintos Componentes Visuales HoneyComb

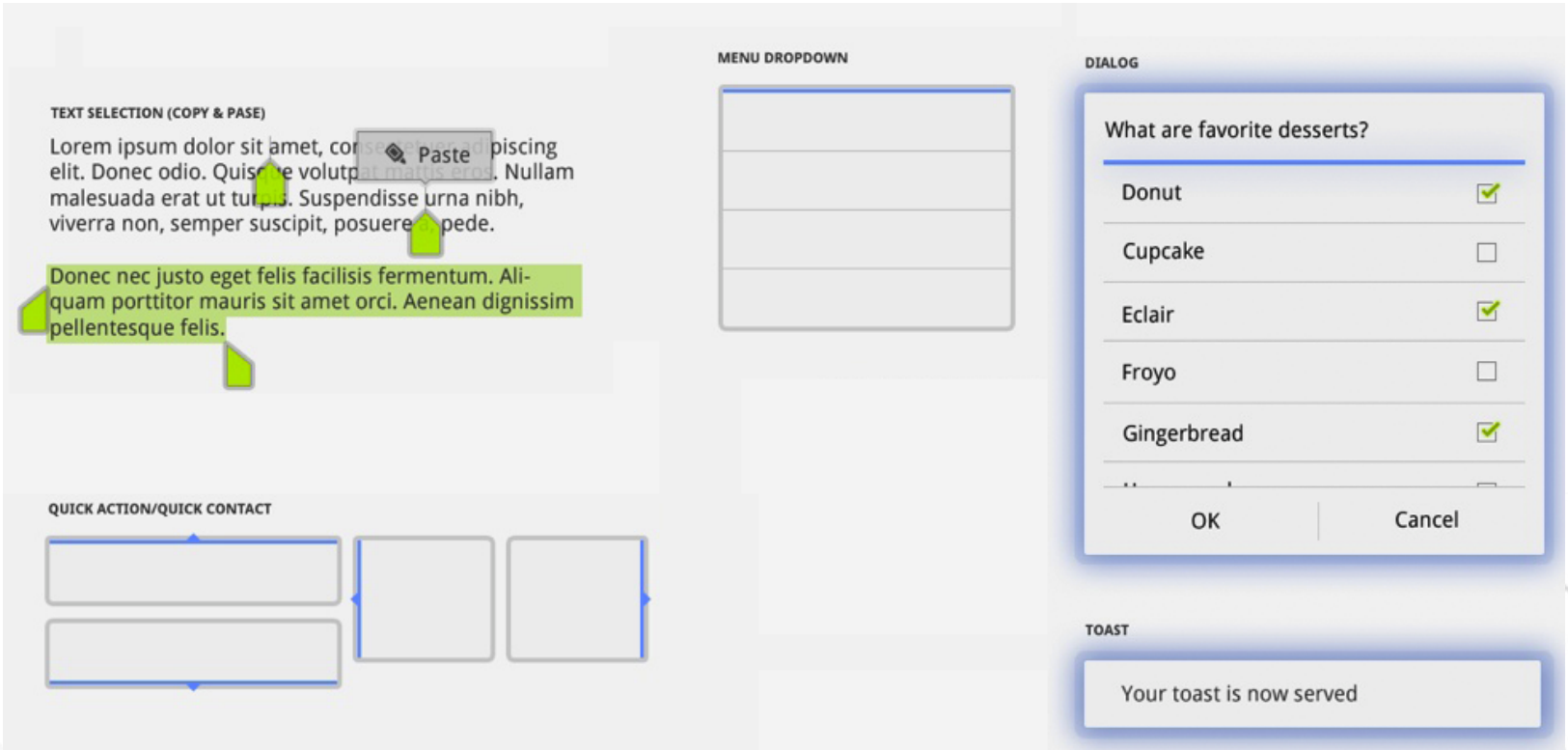


Figura 13: Distintos Componentes Visuales HoneyComb

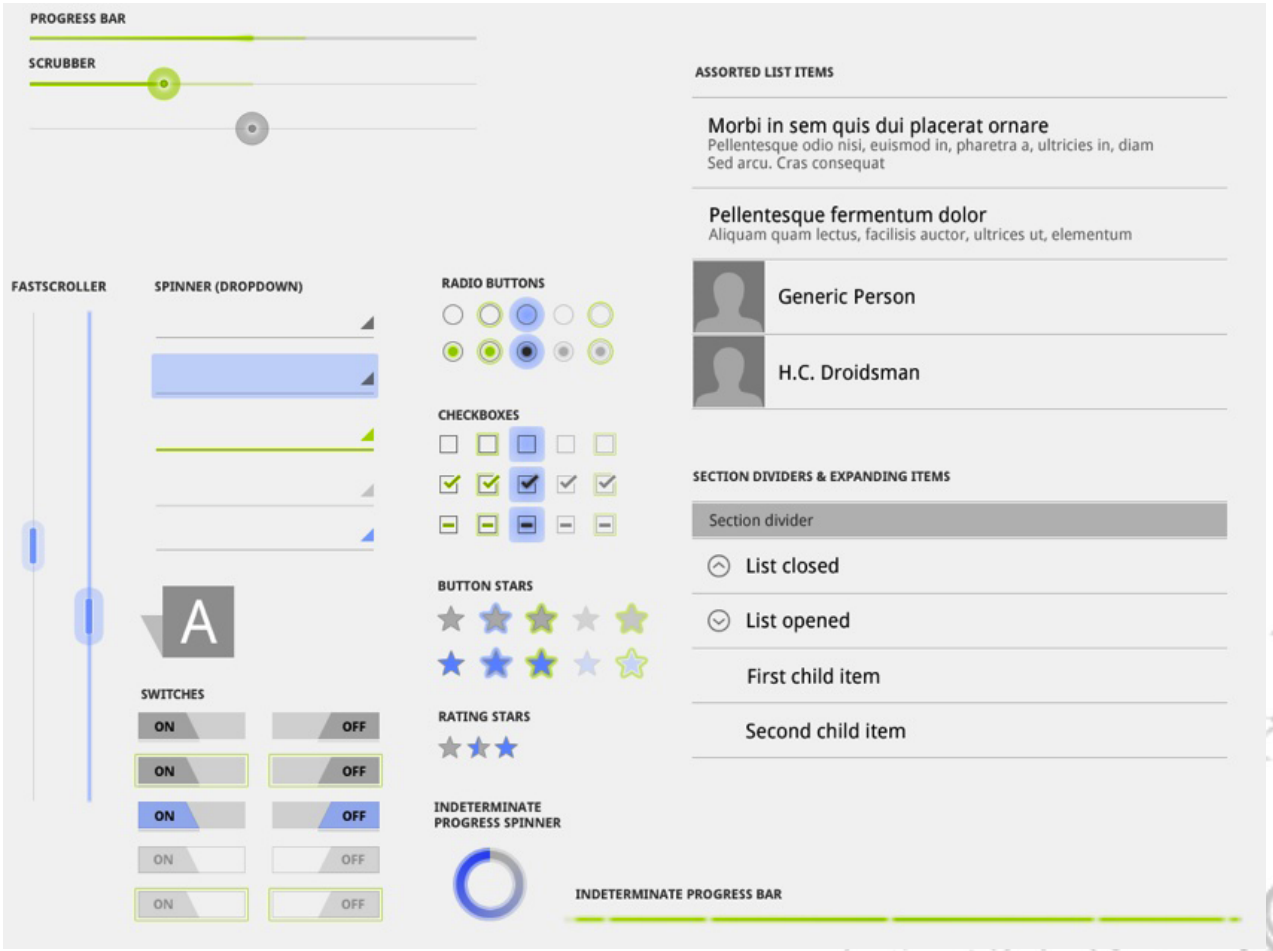


Figura 14: Distintos Componentes Visuales HoneyComb

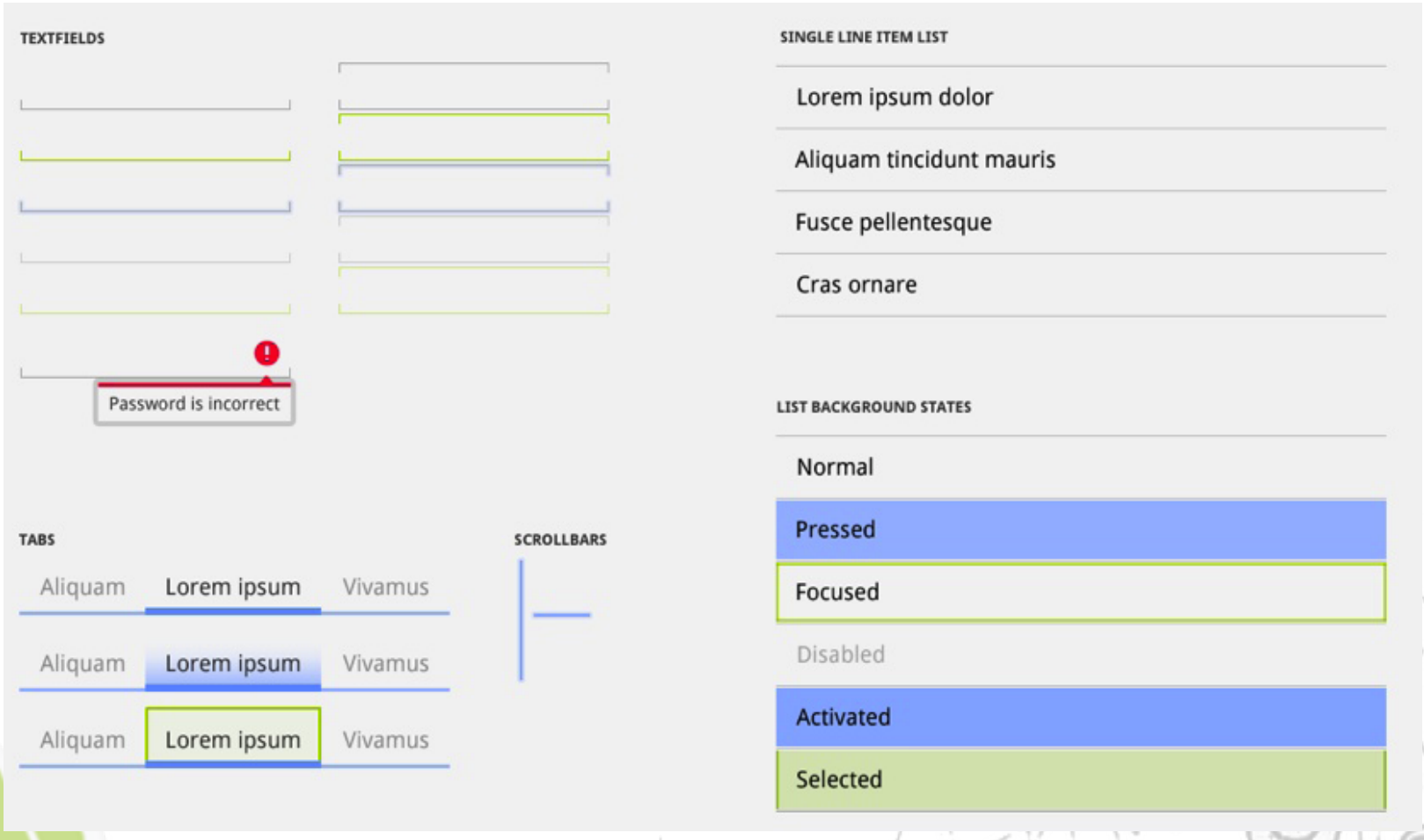


Figura 15: Distintos Componentes Visuales HoneyComb

Simplificando el Diseño

- Hasta ahora hemos realizado el diseño utilizando el archivo main.xml de la carpeta layout del proyecto.
- Además Eclipse nos permite realizar el diseño del interfaz de forma visual y crear de forma automática el archivo.xml así como los datos necesarios.
- La figura 16 nos muestra como utilizar la paleta visual, con todos los elementos visuales.
- Otra aplicación que podemos utilizar para realizarlo de forma gráfica es http://www.droiddraw.org/

Creación Interfaz mediante Método Visual

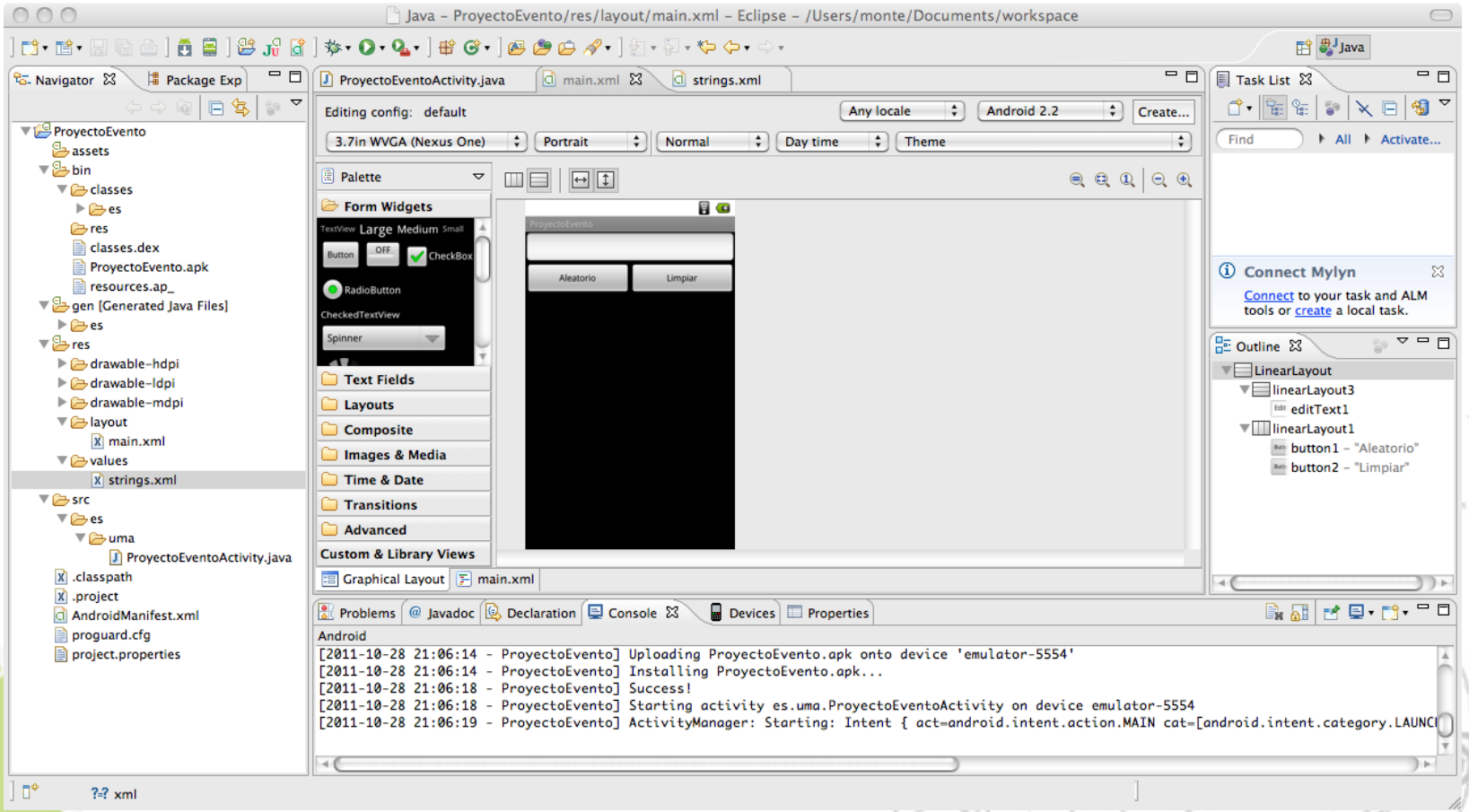


Figura 16: Creación Interfaz mediante Método Visual

Práctica 2

Vamos a realizar una calculadora con las operaciones básicas tal y como se muestra en la figura 17.

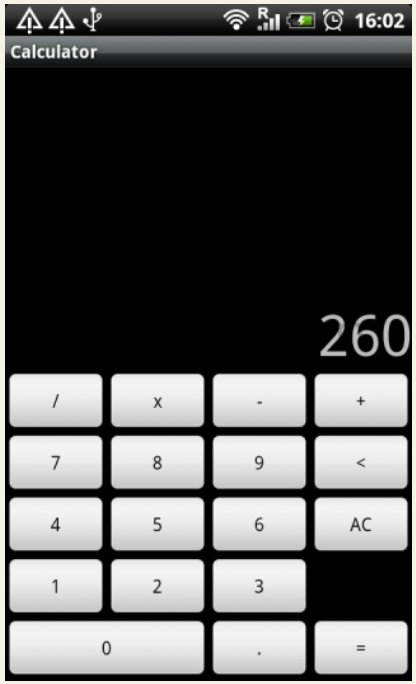


Figura 17: Ejemplo Calculadora Básica

José A. Montenegro Montes
Dpto. Lenguajes y Ciencias de la Computación
ETSI Informática. Universidad de Málaga
monte@lcc.uma.es
twitter