[**INTRODUCCIÓN**](#_4ct8xzcivj14) **2**

[Antecedentes](#_1l63tdmeg95c) 2

[Historia de versiones](#_g531gh5ykss9) 2

[Arquitectura](#_lspmckeya0a7) 3

[**AMBIENTE DE TRABAJO**](#_h2ihdetdk4eh) **4**

[Android studio](#_yry159bwxq06) 4

[Herramientas de desarrollo](#_gwdsjtykazid) 4

[**PRIMER PROYECTO EN ANDROID**](#_o2zx9id2be5h) **5**

[Manifest.xml](#_ksw8pvx8fbq5) 5

[Estructura de un proyecto](#_2juyakbx81xa) 6

[Ciclo de vida de una aplicación](#_x6crgh9mksiz) 8

[Elementos de una aplicación](#_thaw2lwvdgup) 10

[View](#_rw3fj1s0zkd0) 10

[Activity](#_yni5cclnt1wc) 10

[Intent](#_rgclelcpqvra) 11

[Service](#_quwdnicc5pfw) 11

[Content provider](#_mbxt4qry9hf4) 11

[**ACTIVIDADES**](#_qxbnpe6g2q78) **11**

[Invocación de componentes (intent)](#_izv7eucuak3v) 12

[Manejo de mensajes](#_pk2rbybd0x5d) 13

[Toast](#_ma4y5i2l222e) 14

[Cuadros de dialogo](#_yck47mo39nxy) 14

[Snackbar](#_fsa9l6wi0s9o) 16

[**DISEÑO DE INTERFACES GRÁFICAS**](#_2ki83297r0w1) **17**

[Esquema de distribución de componontes](#_46srr83uh212) 17

[Linearlayout](#_qwzjz71yd59q) 17

[Relativelayout](#_44myfu2xtjgf) 18

[Framelayout](#_thozy4gzg7zv) 19

[Coordinatorlayout](#_lo40ur6ac6u1) 21

ConstraintLayout

[Controles básicos](#_yo030rk4zubu) 23

[Button](#_tnral5l51q7f) 23

[ImageView](#_uh3boftczlly) 24

[TextView](#_js6ma05x9jth) 24

[EditText](#_8nq95zxaxkk4) 25

[CheckBox](#_q93ysaqhg3gt) 25

[RadioButton](#_im90jaobmn7d) 26

[FloatingActionButton](#_m7l23pnsueob) 27

[Implementación de un formulario de registro](#_tbjh9aw6iqne) 28

[Controles de selección](#_i9x6khhnfkn) 40

[Listas](#_cts2ekfst2sw) 40

[Listas personalizadas](#_f2gomw42trll) 43

[Recyclerview y cardview](#_e518euqkvq2k) 49

[Controles de navegación](#_l2mz7lkt66y9) 56

[Appbarlayout y toolbar](#_78komcesqgru) 56

[Menú](#_imo3vqd3ms37) 58

[**FRAGMENTS**](#_pzndzzt8w9hu) **61**

[Ciclo de vida](#_p6vmwwingbri) 61

[Integración en una actividad](#_dc5x8zes1t2v) 62

[**ALMACENAMIENTO DE DATOS EN EL DISPOSITIVO**](#_p0cnluf16qzo) **73**

[Introducción a SQLITE](#_t3lstal5vlc5) 73

[Insertar, actualizar y eliminar registros](#_5ui5quj6v45r) 73

[Recuperar información desde la base de datos](#_1ar0aqcemy1w) 77

[Manejo de preferencias](#_yzps4iebjrr5) 78

[**DISTRIBUCIÓN DE UNA APLICACIÓN**](#_oe2lsrrets2s) **79**

[Consideraciones antes de la versión final de la aplicación](#_qo92ofb5vhv7) 79

[Estableciendo la versión de la aplicación](#_j9xu6yxdkcgi) 80

[Firmando la aplicación](#_ogqorz2kyey8) 80

# **INTRODUCCIÓN**

## **Antecedentes**

Android, inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y en 2005, fue adquirida por la misma. Fue hasta 2007 que fue presentado junto a la fundación del Open Handset Alliance para avanzar en los estándares abiertos de los dispositivos móviles. El primer dispositivo móvil fue el HTC Dream en el 2008.

## **Historia de versiones**

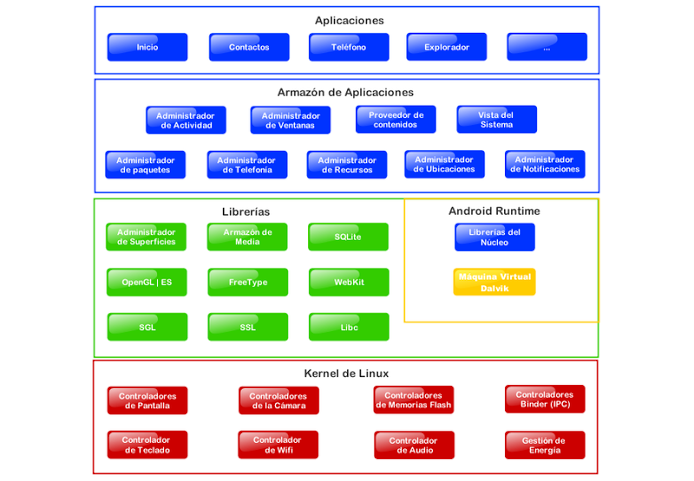
A partir de la primera liberación que fue a partir de Noviembre del 2007 en una versión beta, sin aún tener una versión pública, fue hasta Septiembre de 2008 que presento su primera versión al público el cual fue nombrada “Apple Pie”(1.0). Una peculiaridad del nombramiento de versiones para cada una de ellas, es llevar el orden alfabético y seleccionar el nombre de un dulce para definir el nombre de ella.

A continuación se muestra una tabla con las versiones de Android:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Versión | Fecha | Número de API |
| Apple Pie | 1.0 | 23 de Sept de 2008 | 1 |
| Banana Bread | 1.1 | 9 de Feb de 2009 | 2 |
| Cupcake | 1.5 | 27 de Abr de 2009 | 3 |
| Donut | 1.6 | 15 de Sept de 2009 | 4 |
| Eclair | 2.0 - 2.1 | 26 de Oct de 2009 | 5 - 7 |
| Froyo | 2.2 - 2.2.3 | 20 de May de 2010 | 8 |
| Gingerbread | 2.3 - 2.3.7 | 6 de Dic de 2010 | 9 - 10 |
| Honeycomb | 3.0 - 3.2.6 | 22 de Feb de 2011 | 11 - 13 |
| Ice Cream Sandwich | 4.0 - 4.0.4 | 18 de Oct de 2011 | 14 - 15 |
| Jelly Bean | 4.1 - 4.3.1 | 9 de Jul de 2012 | 16 - 18 |
| KitKat | 4.4–4.4.4, 4.4W–4.4W.2 | 31 de Oct de 2013 | 19 - 20 |
| Lollipop | 5.0 - 5.1.1 | 12 de Nov de 2014 | 21 - 22 |
| Marshmallow | 6.0 - 6.0.1 | 5 de Oct de 2015 | 23 |
| Nougat | 7.0 - 7.1.2 | 22 de Ago de 2016 | 24 - 25 |
| O (Oreo) | 8.0 | TBA | 26 |

**Tabla 1.- Historial de versiones de Android**

## **Arquitectura**



**Ilustración 1.- Arquitectura de Android**

# **AMBIENTE DE TRABAJO**

## **Android studio**

Toda aplicación requiere de una plataforma por la cual se desarrolla e integra sus funcionalidades, en este caso, la base de Android está basado en el lenguaje de programación JAVA, apoyándose en la utilización de XML. Toda aplicación se desarrolla bajo la herramienta Android Studio que proporciona Google, la cual contiene plugins como módulos que nos permiten desarrollar de una manera eficiente y de calidad las aplicaciones que deseemos crear.

Es un IDE basado en IntelliJ IDEA, tiene un potente editor de código y herramientas de desarrollo integradas que aumentan la productividad y facilitan la creación de aplicaciones para Android.

[](https://commons.wikimedia.org/wiki/File:Android_Studio_icon.svg)

**Ilustración 2.- Logo de Android Studio**

## 

## **Herramientas de desarrollo**

Las herramientas de desarrollo que se tiene para desarrollar aplicaciones utilizando Android Studio, son las siguientes:

* **Android Emulator**

Esta herramienta permite tener varios dispositivos móviles ejecutándose de manera que permite tener un dispositivo emulando todas las funciones de él y permite desarrollar pruebas directas sobre la aplicación, aunque estos emuladores pueden tener características de varios dispositivos del mercado, lo ideal es realizar pruebas sobre un dispositivo real.

* **Ddms**

Esta herramienta permite realizar pruebas de funcionalidad en la aplicación que se está probando y permite tener reportes como rendimiento, consumo de datos, muchos más; nos puede dar informes para tomar decisiones de optimización o solución de problemas en la aplicación. Está viene incluida en Android Studio.

* **Logcat**

Esta herramienta es la consola de mensajes que envía el dispositivo conectado, y que permite al desarrollador enviar mensajes desde el dispositivo y lograr rastrear funcionalidades o problemas en la aplicación.

* **Logging**

Este proceso es el envío de mensajes desde la aplicación a la consola de mensajes lo cual va permitir llevar un rastreo de las funcionalidades o problemas en la aplicación.

# **PRIMER PROYECTO EN ANDROID**

## **Manifest.xml**

Toda aplicación desarrollada está conjuntamente creada a través del lenguaje de programación JAVA y XML, este último se utiliza para generar el archivo “Manifest.xml”, el cual es requerido para la configuración de la aplicación, este archivo permite entregar información a la tienda de aplicaciones como al sistema Android para que conozca las características y permisos de nuestra aplicación.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="erick.com.miscontactos">   <uses-permission android:name="android.permission.CALL\_PHONE" />  <uses-permission android:name="android.permission.INTERNET" />   <application  android:allowBackup="true"  android:icon="@mipmap/ic\_launcher"  android:label="@string/app\_name"  android:roundIcon="@mipmap/ic\_launcher\_round"  android:supportsRtl="true"  android:theme="@style/AppTheme">  <activity android:name=".MainActivity">  <intent-filter>  <action android:name="android.intent.action.MAIN" />   <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  <activity android:name=".DetalleContacto" />  <activity android:name=".acerca" />  <activity android:name=".insert"></activity>  </application>  </manifest> |

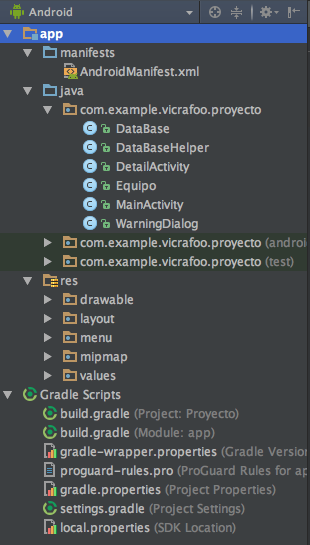
## **Estructura de un proyecto**

Actualmente la estructura de un proyecto en Android, es conformado por una jerarquía de archivos y directorios estructurados al momento de crear el proyecto, esta estructura que genera el Android Studio es básica para que nuestra aplicación pueda ejecutarse, aún hay directorios o archivos que no agrega, los cuales se pueden ir añadiendo conforme sean necesarios para la aplicación.

Las clases que se deben generar se establecen en el paquete declarado al generar el proyecto, este paquete se repite, debido a que la herramienta utiliza un paquete duplicado para configurar clases que se requieren para la aplicación, el siguiente paquete nos puede servir para desarrollar pruebas sobre nuestra aplicación.

Además la herramienta genera módulo de archivos de configuración los cuales son utilizados por Gradle, para identificar el proyecto y establecer las propiedades necesarias para establecer librerias y propiedades para la aplicación.

La estructura del proyecto se muestra en la siguiente imagen:



**Ilustración 3.- Estructura de un proyecto**

La carpeta “res” se guardan directorios correspondientes a archivos xml que se establecen para generar las vistas de la aplicación como recursos de imágenes, así como archivos para resguardar valores de dimensiones o cadenas que se utilizan en la aplicación.

## **Ciclo de vida de una aplicación**



**Ilustración 4.- Ciclo de vida de una aplicación**

Toda aplicación generalmente consiste en un conjunto de actividades, cada una de ella puede realizar diferentes funcionalidades dependiendo de los eventos realizados por parte del usuario, por lo tanto, la aplicación requiere realizar ciertas acciones a cada una de ellas.

Debido a que la aplicación gestiona a las actividades como una pila, cuando se ejecuta una de ellas, está se coloca al principio de la pila y está en ejecución, si se ejecuta una segunda actividad está reemplaza la actividad anterior en la primera posición en la pila, y está regresara hasta que la actividad se haya cerrado.

Una actividad tiene esencialmente cuatro estados:

* Si una actividad está en el primer plano de la pantalla (en la parte superior de la pila), está activa o en ejecución.
* Si una actividad ha perdido el enfoque pero sigue siendo visible (es decir, una actividad nueva o no transparente se centra en la parte superior de su actividad), se detiene. Una actividad en pausa está completamente viva (mantiene toda la información de estado y de miembro y permanece unida al administrador de ventanas), pero puede ser eliminada por el sistema en situaciones de baja memoria extrema.
* Si una actividad es completamente oscurecida por otra actividad, se detiene. Todavía conserva toda la información de estado y de miembro, sin embargo, ya no es visible para el usuario y su ventana está oculta y que a menudo será asesinado por el sistema cuando la memoria se necesita en otra parte.
* Si una actividad se detiene o se detiene, el sistema puede dejar caer la actividad de la memoria ya sea pidiéndole que finalice o simplemente mate su proceso. Cuando se vuelve a mostrar al usuario, debe reiniciarse completamente y volver a su estado anterior.

En la siguiente tabla se describe en resumen los principales estados de la actividad

|  |  |  |  |
| --- | --- | --- | --- |
| **Método** | **Descripción** | **¿Se puede eliminar después?** | **A continuación** |
| [onCreate()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onCreate(android.os.Bundle)) | Recibe una llamada cuando se crea la actividad por primera vez. Aquí es donde debes realizar todas las configuraciones estáticas normales: crear vistas, enlazar datos con listas, etc. A este método se le pasa un objeto agrupado que contiene el estado anterior de la actividad, si se hubiera capturado ese estado (lee [Guardar el estado de una actividad](https://developer.android.com/guide/components/activities.html?hl=es#actstate) más adelante).  Siempre seguido por onStart(). | No | onStart() |
| [onRestart()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onRestart()) | Recibe una llamada después de que se detiene la actividad, junto antes de que vuelva a iniciarse.  Siempre seguido por onStart(). | No | onStart() |
| [onStart()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onStart()) | Recibe una llamada justo antes de que la actividad se vuelva visible para el usuario.  Seguido por onResume() si la actividad pasa a primer plano, o por onStop()si se oculta. | No | onResume()  o  onStop() |
| [onResume()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onResume()) | Recibe una llamada justo antes de que la actividad comience a interactuar con el usuario. En este momento la actividad se encuentra en la parte superior de la pila de actividades, y recibe las entradas del usuario.  Siempre seguido por onPause(). | No | onPause() |
| [onPause()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onPause()) | Recibe una llamada cuando el sistema está a punto de reanudar otra actividad. Este método generalmente se usa para confirmar los cambios sin guardar como datos persistentes, para detener animaciones y otras tareas que podrían estar consumiendo CPU, etc. Lo que sea que haga, debe hacerlo muy rápido porque la siguiente actividad no se reanudará hasta que el método regrese.  Seguido por onResume() si la actividad vuelve al primer plano, o por onStop() si se vuelve invisible para el usuario. | Sí | onResume()  o  onStop() |
| [onStop()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onStop()) | Recibe una llamada cuando la actividad ya no es visible para el usuario. Esto puede ocurrir porque se la destruyó o porque se reanudó otra actividad (ya sea una actividad existente o una nueva) y la está cubriendo.  Seguido por onRestart() si la actividad vuelve a interactuar con el usuario, o por onDestroy() si la actividad desaparece. | Sí | onRestart()  o  onDestroy() |
| [onDestroy()](https://developer.android.com/reference/android/app/Activity.html?hl=es#onDestroy()) | Recibe una llamada antes de que se destruya la actividad. Esta es la última llamada que recibirá la actividad. Se lo puede llamar porque la actividad está finalizando (alguien llamó a [finish()](https://developer.android.com/reference/android/app/Activity.html?hl=es#finish()) para esa actividad), o porque el sistema destruye temporalmente esa instancia de la actividad para ahorrar espacio. Puedes diferenciar estos dos escenarios con el método [isFinishing()](https://developer.android.com/reference/android/app/Activity.html?hl=es#isFinishing()). | Sí | nada |

**Tabla 2.- Resumen de los estados del ciclo de vida de una actividad.**

## **Elementos de una aplicación**

### ***View***

Este elemento agrupa el conjunto de vistas y controles de una interfaz gráfica, la cual es responsable de pintar y capturar eventos que se realizan sobre el área del elemento, permitiendo implementar a través de contenedores propios que otorgan la posibilidad de distribuir y organizar elementos para representar nuestra interfaz.

### ***Activity***

Este elemento es una pantalla con la que un usuario puede interactuar para realizar una acción, por ejemplo, marcar un teléfono o enviar un correo electrónico. Dentro de ella contiene la vista de la interfaz. Este elemento contiene un conjunto de variedades dependiendo de las funcionalidades y de la versión de Android, actualmente, se cuenta con una librería de compatibilidad que permite incorporar la mayoría de las funcionalidades en gran parte de las versiones de Android.

### ***Intent***

Este elemento permite tener comunicación entre el conjunto de actividades de la aplicación, así como el llamado de ellas a través de este, puede contener valores que se requieran enviar a la siguiente actividad, además, es posible enlazar otras aplicaciones externas a la aplicación.

### ***Service***

Este componente de aplicación permite realizar operaciones durante un tiempo prolongado detrás de la ejecución de la aplicación y en la cual no contiene interfaz gráfica con la que el usuario pueda interactuar con el. Este componente puede iniciarlo desde otra aplicación asi como estarse ejecutando sin la necesidad de que la aplicación se este ejecutando.

### ***Content provider***

Este componente permite el administrar el acceso a un conjunto estructurado de datos, los cuales son encapsulados y proporcionan mecanismos para definir la seguridad de los datos. Este componente proporciona una interfaz estandar que conecta datos en un proceso que se ejecuta desde otro proceso.

# **ACTIVIDADES**

Como anteriormente se menciono, una actividad principalmente se puede definir como una pantalla en la que puede interactuar un usuario a través de una interfaz gráfica, este elemento ttienen muchas funcionalidades que permiten ofrecer una mejor experiencia de usuario y es un elemento vital en cualquier aplicación.

Estas elemento pueden interactuar entre si mismo o con otros elementos a través del paso de parametros o de conectores, en los cuales se explicaran en los siguientes temas de este capitulo.

A continuación se le muestra una estructura base de una actividad:

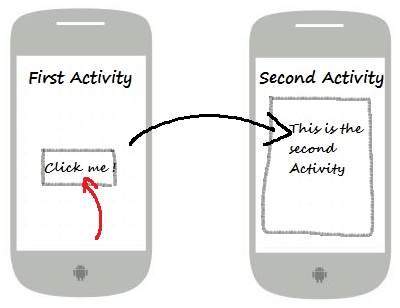
Código Java

|  |
| --- |
| package com.curso.tazmadroid.activityapp;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle;  public class MainActivity extends AppCompatActivity {   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  }   @Override  protected void onResume() {  super.onResume();  }   @Override  protected void onStart() {  super.onStart();  }   @Override  protected void onPause() {  super.onPause();  }   @Override  protected void onStop() {  super.onStop();  }   @Override  protected void onDestroy() {  super.onDestroy();  } } |

## **Invocación de componentes (intent)**

En toda aplicación esta conformada por varios elementos entre ellos actividades que interactuan, desde que se inicia la aplicación actua un componente esencial que permite invocar la actividad e implementar la secuencia entre las mismas, este es un “Intent”. El cual su principal funcionamiento es invocar una actividad de nuestra aplicación asi como compartir información entre ellas.

Este componente permite invocar actividades contenidas en nuestra aplicación o como otra aplicación o servicio fuera de ella compartiendo información para que sea puedan ejecutar. Tambien es utilizado para enlazar las notificaciones hacia nuestra aplicación y permita realizar operaciones con la información que contiene.



**Ilustración 5.- Paso de parametros (bundle)**

Cuando en una aplicación se requiere tener comunicación entre varias aplicaciones se debe utilizar un contenedor de valores que es denotado como “Bundle”, este elemento permite resguardar valores que se requieran transmitir a una actividad para que puedan ser usados para el control y flujo de las actividades.

Los tipos de datos que se pueden manejar dentro de este contenedor son datos primitivos(Int, Float, Double, Char, Short Byte), cadenas, arrreglos, colecciones y objetos serializados o parcelable son los que se pueden permitir debido a un control de funcionalidad para evitar un retraso en la comunicación entre las actividades.

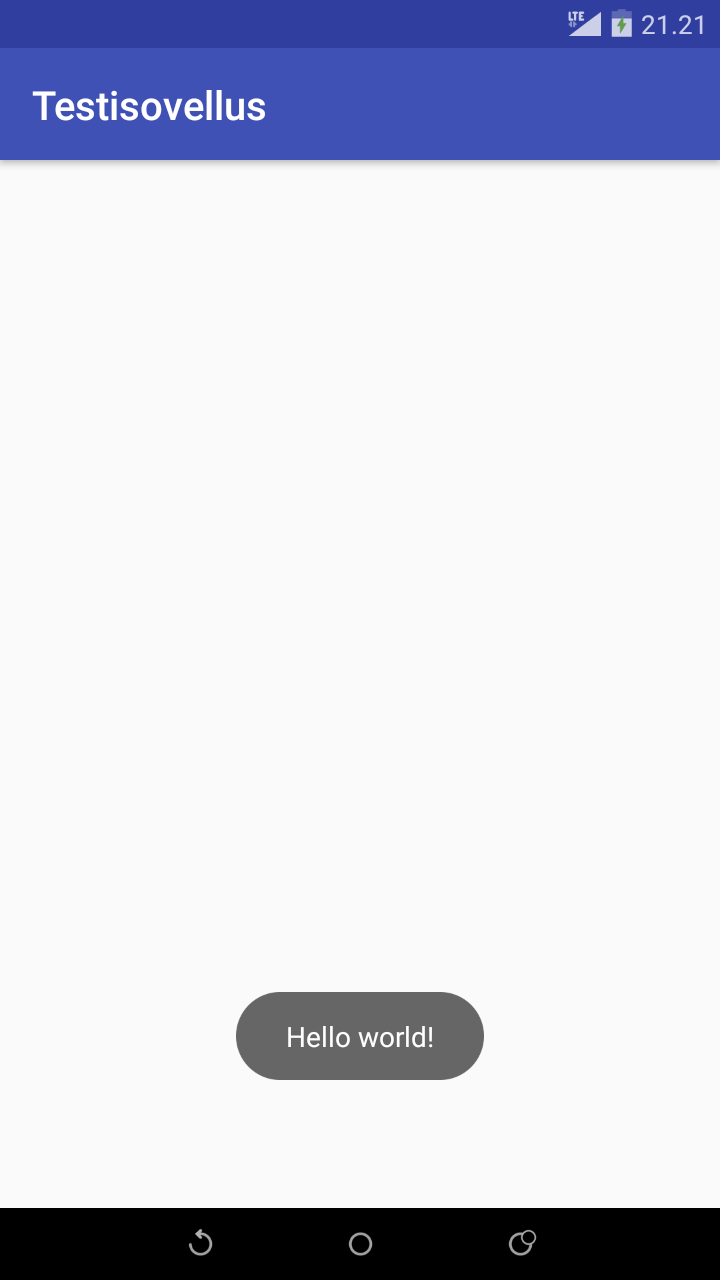
## **Manejo de mensajes**

Dentro de una aplicación es necesario mostrar mensajes dentro de ella, estos contienen información del usuario o de la aplicación, es por ello que en este capitulo hablaremos de los elementos que nos permiten mostrar mensajes dentro de una aplicación.

Actualmente, se tienen varios elementos para mostrar un mensaje en una aplicación de diferentes maneras, a continuación se explicaran algunos de ellos que se pueden utilizar en nuestra aplicación, estos son:

### ***Toast***

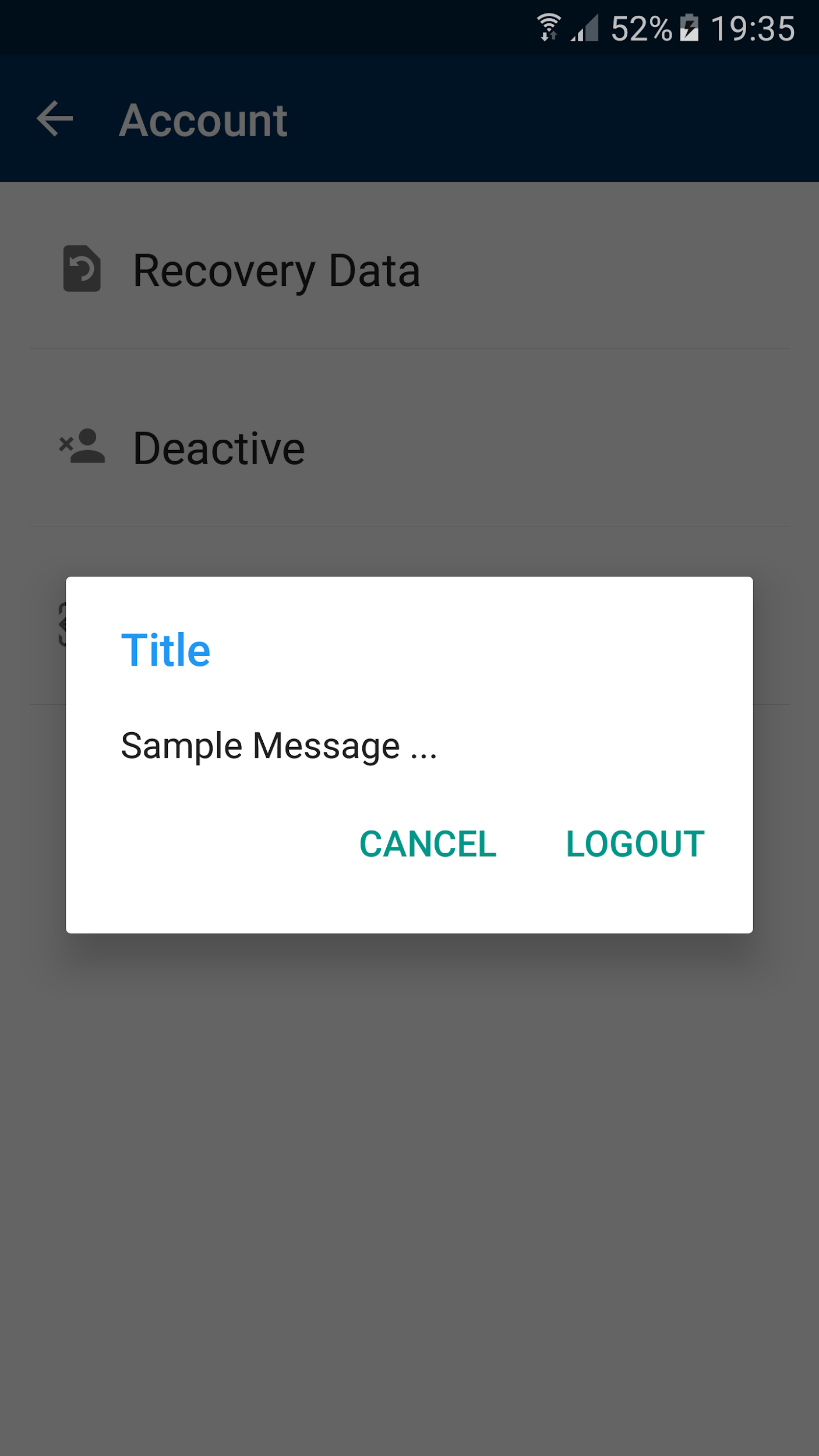
Este elemento se utiliza principalmente para mostrar mensajes al usuario de aviso o información en el cual no requiere ninguna acción de parte del usuario y el mensaje dura solo un tiempo para mostrarse.



**Ilustración 6.- Toast**

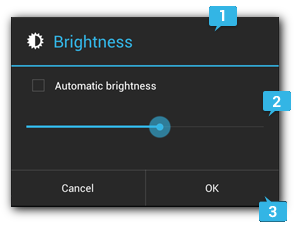
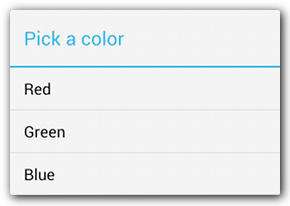
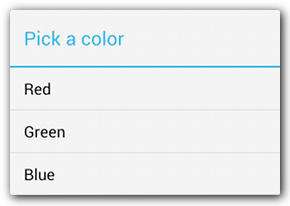
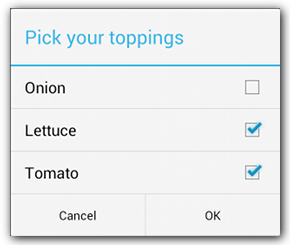
### ***Cuadros de dialogo***

Este elemento se utiliza para mostrar mensajes de aviso en el cual se requiere que el usuario realice una acción a partir del mensaje y esté se puede cancelar para ocultarlo.



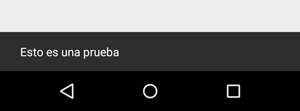
**Ilustración 7.- Cuadros de dialogo**

Este elemento permite implementar diferentes tipos de cuadros de dialogo que incluyan otros elementos como listas, cajas de texto, listas de selección multiple o selección simple, como se muestran en las siguientes imagenes:

**Ilustración 8.- Tipos de cuadros de dialogo**

### ***Snackbar***

Este elemento se desarrollo a partir de la versión 5.0 de Android, en el cual es una nueva forma de mostrar mensajes ofreciendo una mejor experiencia y aprovechando la interfaz de la aplicación, debido a que este elemento no ocupa mucho espacio para mostrarse.



**Ilustración 9.- Snackbar**

Este elemento es posible asignarle una acción que pueda realizar al usuario en el momento que se muestra el mensaje.

# **DISEÑO DE INTERFACES GRÁFICAS**

## **Esquema de distribución de componontes**

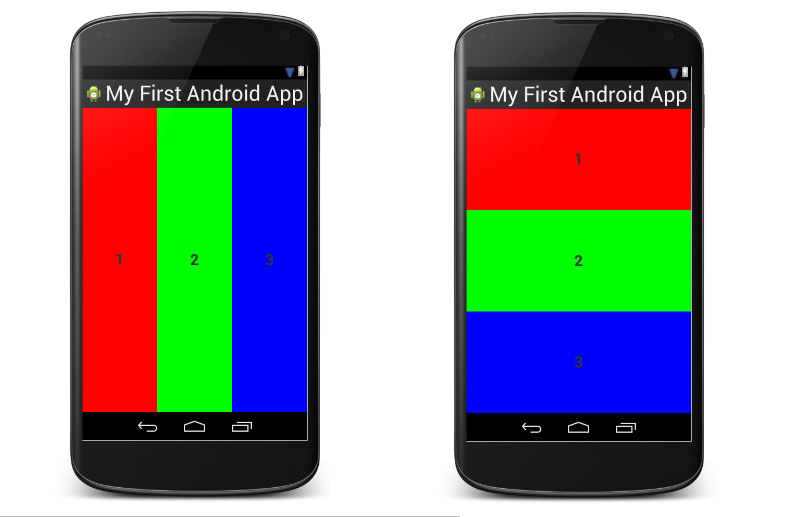
Como se ha descrito anteriormente, uno de los componentes principales de una aplicación es la interfaz gráfica, la cual se genera a partir del elemento “View” que se tiene para generarla a traves de sus subcomponentes que contiene este elemento que permiten tener varios controles y contenedores que permiten dsitribuir los elementos de una forma ordenada.

Estos contenedores permiten distribuir los elementos de nuestra interfaz mediante un patron o un ordenamiento que se especifican, la mayoria de estos elementos como los contenedores se establecen mediante archivos xml a traves de etiquetas que colaboran conjuntamente. Además, es posible generarlos desde código de programación.

Los contenedores son los siguientes:

### ***Linearlayout***

Este contenedor permite distribuir los elementos que contiene de manera vertical u horizontal entre los elementos que contenga. Este distribuye por default en forma vertical todos sus elementos.



**Ilustración 10.- Linearlayout**

Ejemplo de código:

**Linearlayout vertical**

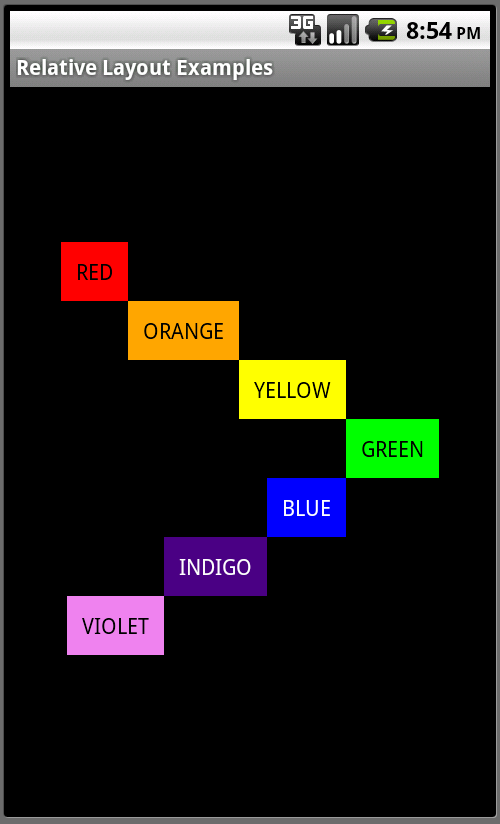
|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  </LinearLayout> |

**Linearlayout horizontal**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  </LinearLayout> |

### ***Relativelayout***

Este contenedor permite distribuir los elementos basandose en las relacion de las posiciones de los elementos que lo contengan como al contenedor, esto permite tener una mejor distribución para los diferentes tamaños de pantalla.



**Ilustración 11.- Relativelayout**

Ejemplo de codigo

|  |
| --- |
| <?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">  </RelativeLayout> |

### ***Framelayout***

Este contenedor distribuye los elementos en capas, debido a que sobrepone cada uno de ellos.



**Ilustración 12.- Framelayout**

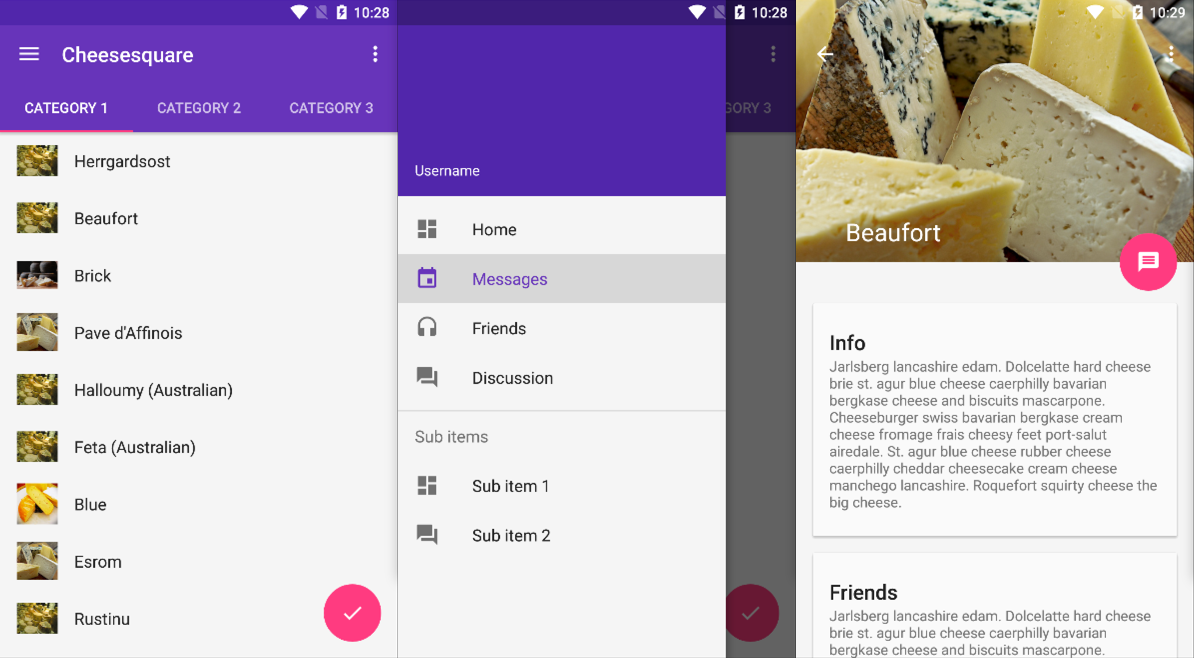
Ejemplo de código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  </FrameLayout> |

### ***Coordinatorlayout***

A partir de la versión 5.0 de Android fue liberado nuevos elementos que conformaban una guia de diseño que permitia tener una mejor experiencia de usuario, esta fue denominada “Material Design”, en el cual se enfoca de proporcionar nuevas funcionalidades e interacciones en sus elementos.

Este contenedor permite integrar nuevas interacciones y animaciones entre los elementos, este contenedor no tiene un orden para los elementos, puesto que trabaja junto con otros contenedores y elementos, es por ello que permite generar animaciones entre ellos.



**Ilustración 15.- Coordinatorlayout**

Ejemplo de código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.design.widget.CoordinatorLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.coordinatorexample.MainActivity">   <android.support.design.widget.AppBarLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:theme="@style/AppTheme.AppBarOverlay">   <android.support.v7.widget.Toolbar  android:id="@+id/toolbar"  android:layout\_width="match\_parent"  android:layout\_height="?attr/actionBarSize"  android:background="?attr/colorPrimary"  app:popupTheme="@style/AppTheme.PopupOverlay"/>   </android.support.design.widget.AppBarLayout>   <include layout="@layout/content\_main"/>   <android.support.design.widget.FloatingActionButton  android:id="@+id/fab"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="bottom|end"  android:layout\_margin="@dimen/fab\_margin"  app:srcCompat="@android:drawable/ic\_dialog\_email"/>  </android.support.design.widget.CoordinatorLayout> |

### ***Constraintlayout***

## 

## **Controles básicos**

### ***Button***

Este control es el más común, puesto que esté es muy utilizado en diferentes lenguajes. Este control permite generar eventos al momento de oprimir esté, lo encontramos como Button e ImageButton, estos controles heredan de la clase View, y la diferencia entre ellos es que uno utiliza texto y el otro utiliza una imagen y ambos implementan los eventos

|  |  |
| --- | --- |
| **Ilustración 16.- Boton**three-buttons.png | KeSHw.jpg  **Ilustración 17.- Botones** |

Código:

|  |
| --- |
| <Button  android:id="@+id/button"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Button"/> |

|  |
| --- |
| <ImageButton  android:id="@+id/imageButton"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  app:srcCompat="@mipmap/ic\_launcher"/> |

Ejemplo de implementación de un evento:

|  |
| --- |
| registerButton.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Log.d("Evento","Se registro la información");  } }); |

### ***ImageView***

Este control al igual que la mayoria hereda del elemento View, su principal funcionalidad es el manejo de imagenes, mediante el cual permite mostrar una imagen y permitiendo tener un control sobre de ella, como por ejemplo escalarla, etc.

Código:

|  |
| --- |
| <ImageView  android:id="@+id/imageView"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  app:srcCompat="@mipmap/ic\_launcher"/> |

### ***TextView***

Este control permite mostrar etiquetas de texto sin importar el número de caracteres que contenga el texto.

Código:

|  |
| --- |
| <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Hello World!"/> |

### ***EditText***

Este control permite introducir valores por el usuario y esté permite acceder a la información a través de eventos que reaccionan dependiendo de la acción del usuario, ademas; este elemento permite asignar validadores de texto para que el teclado del dispositivo muestre números, letras y simbolos .

Código:

|  |
| --- |
| <EditText  android:id="@+id/editText"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:ems="10"  android:inputType="textPersonName"  android:text="Name"/> |

Ejemplo de una implementación de un evento:

|  |
| --- |
| nameEditText.addTextChangedListener(new TextWatcher() {  @Override  public void beforeTextChanged(CharSequence s, int start, int count, int after) {   }   @Override  public void onTextChanged(CharSequence s, int start, int before, int count) {  String inputString = String.valueOf(s.subSequence(start, start + count));  Log.v("Prueba",inputString);  }   @Override  public void afterTextChanged(Editable s) {   } }); |

### ***CheckBox***

Este es un elemento que tambien es muy utilizado en otros lenguajes, esté es utilizado para la selección múltiple entre opciones, y al igual tiene eventos para el control de la selección por parte del usuario.

Código:

|  |
| --- |
| <CheckBox  android:id="@+id/checkBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="CheckBox"/> |

Ejemplo de implementación de un evento:

|  |
| --- |
| spanishCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked) {  Log.d("Materia seleccionada", "Español");  }  } }); |

### ***RadioButton***

Este elemento es muy similar que el CheckBox, solo que es utilizado para la selección única entre varias opciones, para ello es apoyado con otro elemento que se llama “RadioGroup” en el cual es el encargado de controlar un conjunto de esté elemento y mediante el cual implementa el evento para conocer la selección del usuario.

Código:

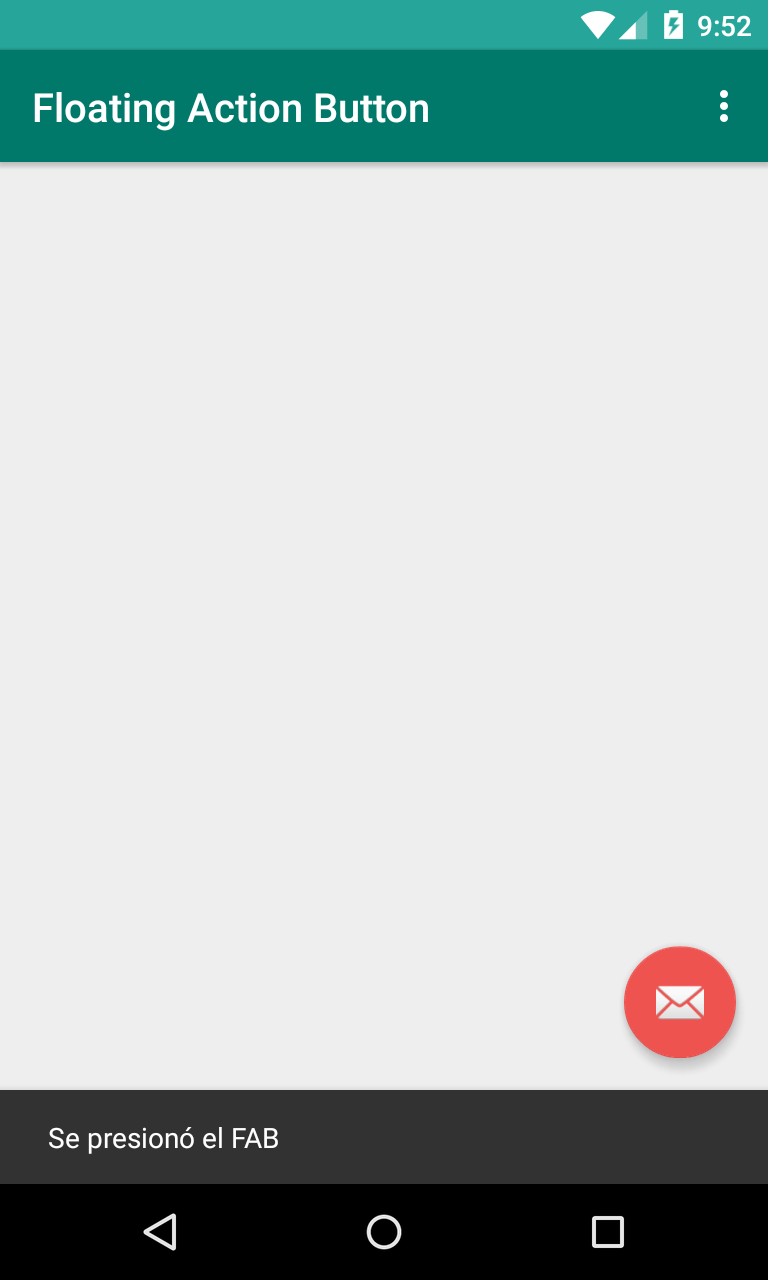
|  |
| --- |
| <RadioGroup  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content">  <RadioButton  android:id="@+id/radioButton"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="RadioButton"/>  </RadioGroup> |

Ejemplo de implementación de un evento:

|  |
| --- |
| sexRadioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(RadioGroup group, @IdRes int checkedId) {  switch (checkedId){  case R.id.menRadioButton:  Log.d("Sexo seleccionado","Masculino");  break;  case R.id.womenRadioButton:  Log.d("Sexo seleccionado","Femenino");  break;  default:  Log.d("Sexo seleccionado","Masculino");  break;  }  } }); |

### ***FloatingActionButton***

Este control es nuevo dentro de las gamas de controles en Android, este aparecio a partir de la versión 5 de Android conocida como Lolipop, esté elemento nacio para ofrecerla una mejor experiencia de usuario, este elemento trabaja conjuntamente con un contenedor de tipo CoordinatorLayout, este elemento se superpone a las vistas y ofrece una interacción a eventos de una manera más sencilla. Esté elemento es muy similar a un boton, solo cambia su presentación y la manera como se incluye en nuestra vista.



**Ilustración 18.- FloatingActionButton**

Código:

|  |
| --- |
| <android.support.design.widget.FloatingActionButton  android:id="@+id/fab"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="bottom|end"  android:layout\_margin="@dimen/fab\_margin"  app:srcCompat="@android:drawable/ic\_dialog\_email"/> |

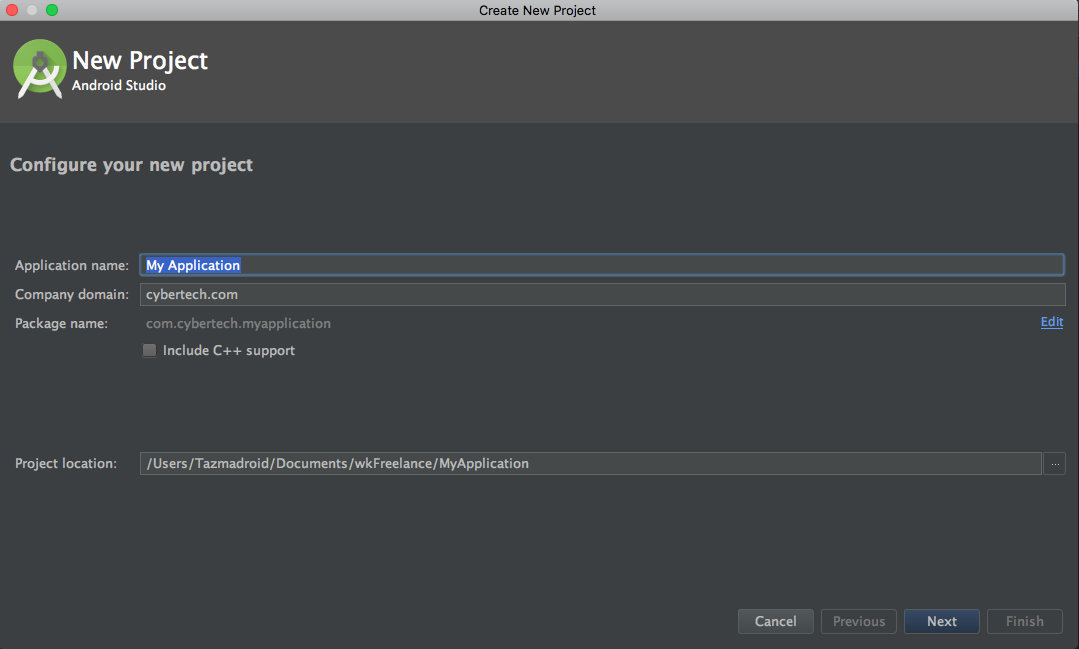
Ejemplo de implementación de un evento:

|  |
| --- |
| fab.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH\_LONG)  .setAction("Action", null).show();  } }); |

### ***Implementación de un formulario de registro***

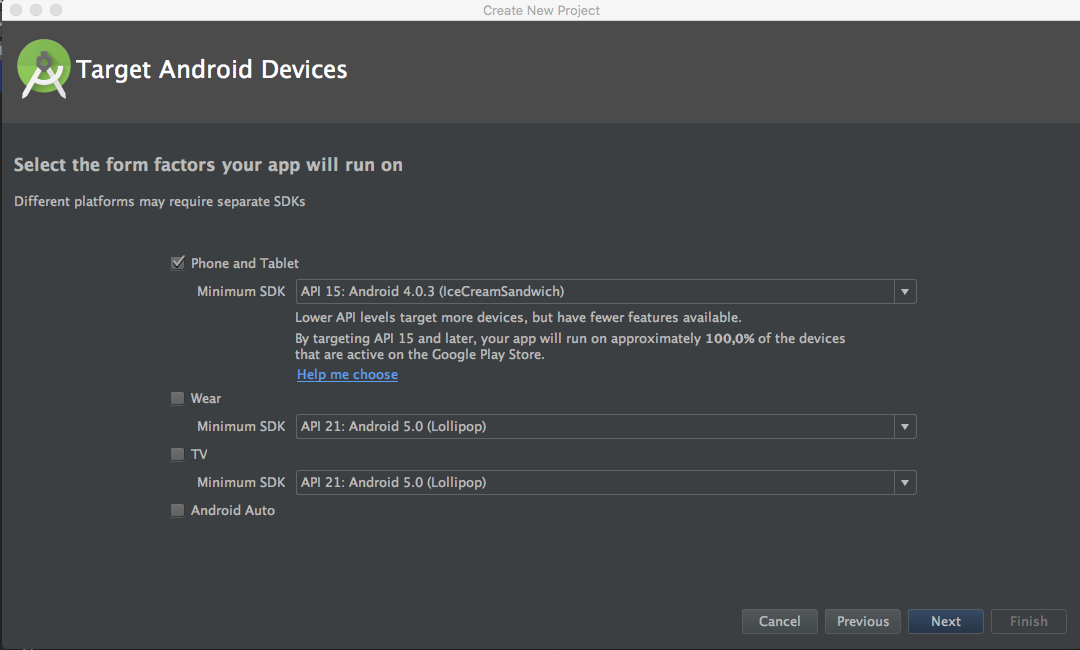
A continuación se desarrollara un ejemplo basico de contenedores y controles básicos, mediante un formulario de registro de un usuario, en este se desarrollarar los temas antes vistos implementando eventos de cada control.

Lo primero que se va a realizar es la creación de un proyecto en blanco, en este paso elegiremos crear nuevo proyecto, en este abrira una ventana para empezar la configuración del proyecto en el cual debemos especificar el nombre de la aplicación y especificar el identficador de la aplicación el cual se establece como se define un paquete en Java o .NET, este campo lo denomina “Company Domain” en el cual se establece el dominio de la compañia o sitio web.



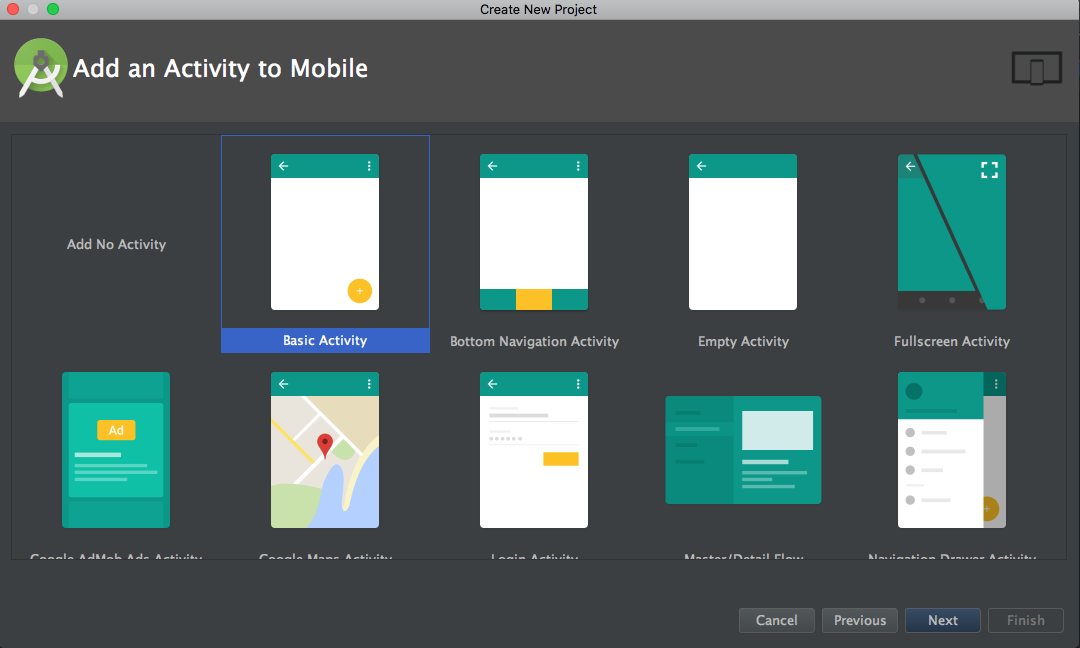
**Ilustración 19.- Configuración de proyecto en Android Studio**

Una vez que se establece estos datos, el siguiente paso se debe especificar la versión de android y la plataforma en la cual se va a trabajar que en este caso es “Phone and Tablet”



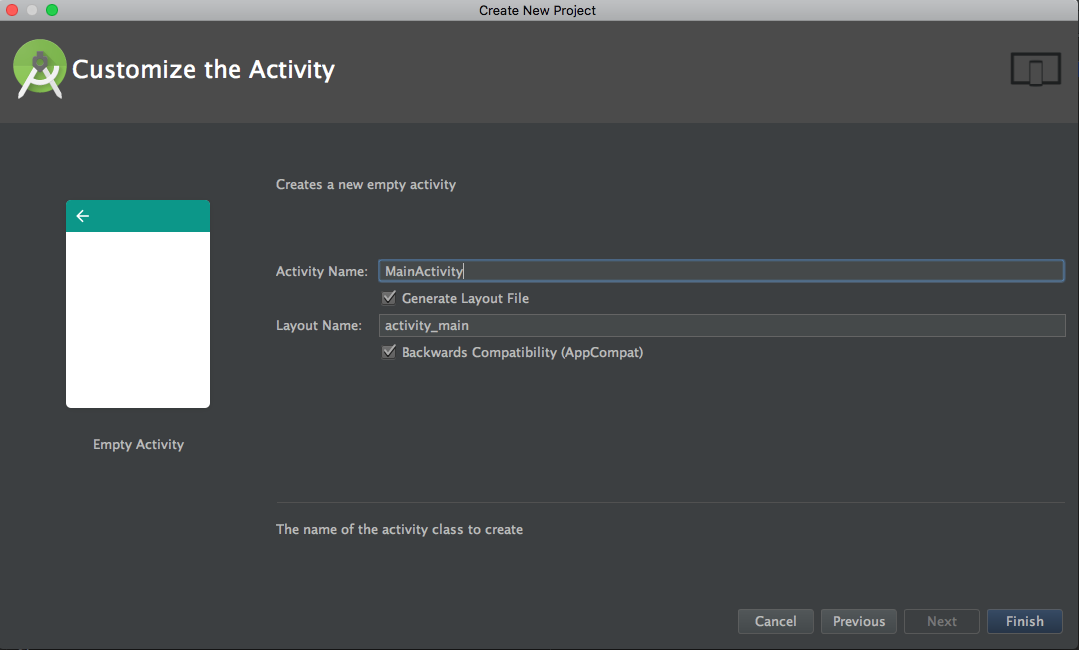
**Ilustración 20.- Definición de proyecto**

A continuación, esta ventana nos muestra plantillas de vistas preconfiguradas que permite tener la integración básica de controles o contenedores, en este caso se seleccionara la plantilla “Basic Activity”



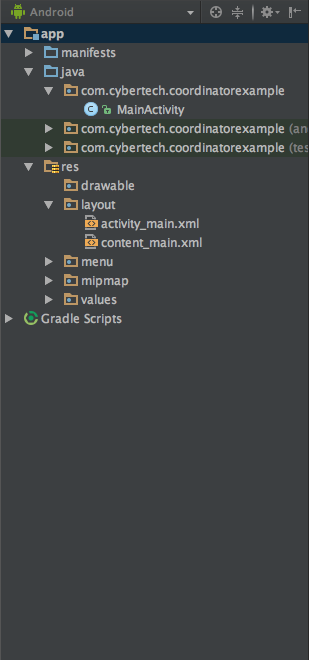
**Ilustración 21.- Plantillas de proyecto**

Una vez seleccionada la plantilla, la siguiente ventana nos pedira nombrar la clase y la vista de la actividad que se va a crear, el nombre de la clase que va actuar de controlador se puede especificar mediante la nomenclatura de clases de Java denominada CamelCase. Con respecto a la nomenclatura de la vista, los archivos xml que se utilicen deben ser nombrados con letras minusculas, en caso de que este compuesto por dos o más palabras se unen a través del uso de guíon bajo



**Ilustración 22.- Declaración de actividades en un proyecto**

Al finalizar la creación del proyecto nos generara una jeraquia de directorios, los cuales mediante una herramienta del IDE nos mostrara la lista de directorios y archivos que contiene nuestro proyecto como se muestra a continuación:



**Ilustración 23.- Estructura del proyecto**

En la vista “activity\_main” se genero la estructura necesaria para manejar una interfaz utilizando los elementos básicos, como el contenedor CoordinatorLayout y un FloatActionButton, ademas de un elemento para incluir la vista de nuestro formulario, asi como se muestra a continuación:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.design.widget.CoordinatorLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.coordinatorexample.MainActivity">   <android.support.design.widget.AppBarLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:theme="@style/AppTheme.AppBarOverlay">   <android.support.v7.widget.Toolbar  android:id="@+id/toolbar"  android:layout\_width="match\_parent"  android:layout\_height="?attr/actionBarSize"  android:background="?attr/colorPrimary"  app:popupTheme="@style/AppTheme.PopupOverlay"/>   </android.support.design.widget.AppBarLayout>   <include layout="@layout/content\_main"/>   <android.support.design.widget.FloatingActionButton  android:id="@+id/fab"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="bottom|end"  android:layout\_margin="@dimen/fab\_margin"  app:srcCompat="@android:drawable/ic\_dialog\_email"/>  </android.support.design.widget.CoordinatorLayout> |

En el segundo archivo nombrado “content\_main” es en el cual vamos a trabajar nuestro formulario de registro, en este se agregara los controles básicos y nos apoyaremos con otro contenedor para distribuirlos, como se muestra en el código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <RelativeLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"  tools:context="com.cybertech.coordinatorexample.MainActivity"  tools:showIn="@layout/activity\_main">   <TextView  android:id="@+id/textView"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentLeft="true"  android:layout\_alignParentStart="true"  android:layout\_alignParentTop="true"  android:layout\_marginLeft="21dp"  android:layout\_marginStart="21dp"  android:layout\_marginTop="13dp"  android:text="@string/label\_name"/>   <EditText  android:id="@+id/nameEditText"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView"  android:layout\_alignStart="@+id/textView"  android:layout\_below="@+id/textView"  android:layout\_marginTop="13dp"  android:ems="10"  android:inputType="textPersonName"  android:text="Name"/>   <TextView  android:id="@+id/textView2"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/nameEditText"  android:layout\_alignStart="@+id/nameEditText"  android:layout\_below="@+id/nameEditText"  android:layout\_marginTop="26dp"  android:text="@string/label\_age"/>   <EditText  android:id="@+id/ageEditText"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/nameEditText"  android:layout\_marginTop="15dp"  android:layout\_toEndOf="@+id/textView"  android:layout\_toRightOf="@+id/textView"  android:ems="10"  android:inputType="number"/>   <TextView  android:id="@+id/textView3"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginTop="15dp"  android:text="@string/label\_sex"  android:layout\_below="@+id/ageEditText"  android:layout\_alignRight="@+id/textView2"  android:layout\_alignEnd="@+id/textView2"/>   <RadioGroup  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:orientation="horizontal"  android:id="@+id/sexRadioGroup"  android:layout\_alignBottom="@+id/textView3"  android:layout\_alignLeft="@+id/ageEditText"  android:layout\_alignStart="@+id/ageEditText">   <RadioButton  android:id="@+id/menRadioButton"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="@string/label\_men"/>   <RadioButton  android:id="@+id/womenRadioButton"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="@string/label\_women"  android:layout\_marginLeft="10dp"  android:layout\_marginStart="10dp"/>   </RadioGroup>   <TextView  android:id="@+id/textView4"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView2"  android:layout\_alignStart="@+id/textView2"  android:layout\_below="@+id/textView3"  android:layout\_marginTop="25dp"  android:text="@string/label\_topics"/>   <CheckBox  android:id="@+id/spanishCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/textView3"  android:layout\_alignStart="@+id/textView3"  android:layout\_centerVertical="true"  android:text="Español"/>   <CheckBox  android:id="@+id/mathematicsCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentEnd="true"  android:layout\_alignParentRight="true"  android:layout\_alignTop="@+id/spanishCheckBox"  android:layout\_marginEnd="18dp"  android:layout\_marginRight="18dp"  android:text="Matemáticas"/>   <CheckBox  android:id="@+id/historyCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignTop="@+id/spanishCheckBox"  android:layout\_centerHorizontal="true"  android:text="Historia"/>   <CheckBox  android:id="@+id/englishCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/spanishCheckBox"  android:layout\_alignStart="@+id/spanishCheckBox"  android:layout\_below="@+id/spanishCheckBox"  android:layout\_marginTop="16dp"  android:text="Ingles"/>   <CheckBox  android:id="@+id/ecologyCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Ecologia"  android:layout\_alignBaseline="@+id/englishCheckBox"  android:layout\_alignBottom="@+id/englishCheckBox"  android:layout\_alignLeft="@+id/historyCheckBox"  android:layout\_alignStart="@+id/historyCheckBox"/>   <CheckBox  android:id="@+id/politicCheckBox"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignLeft="@+id/mathematicsCheckBox"  android:layout\_alignStart="@+id/mathematicsCheckBox"  android:layout\_alignTop="@+id/ecologyCheckBox"  android:text="Politica"/>   <TextView  android:id="@+id/resultTextView"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@+id/ecologyCheckBox"  android:layout\_centerHorizontal="true"  android:layout\_marginTop="42dp"  android:text="Resultado"/>  </RelativeLayout> |

Una vez que se tenga las vista para nuestra aplicación, ahora el siguiente paso es generar nuestro controlador para nuestra vista, en la cual implementaremos los eventos necesarios para que muestre el formulario de registro que acabamos de definir en los archivos xml

|  |
| --- |
| package com.cybertech.coordinatorexample;  import android.os.Bundle; import android.support.annotation.IdRes; import android.support.design.widget.FloatingActionButton; import android.support.design.widget.Snackbar; import android.support.v7.app.AppCompatActivity; import android.support.v7.widget.Toolbar; import android.text.Editable; import android.text.TextWatcher; import android.util.Log; import android.view.View; import android.view.Menu; import android.view.MenuItem; import android.widget.Button; import android.widget.CheckBox; import android.widget.CompoundButton; import android.widget.EditText; import android.widget.RadioGroup; import android.widget.TextView;  import java.util.ArrayList;  public class MainActivity extends AppCompatActivity {   private EditText nameEditText=null;  private EditText ageEditText=null;  private RadioGroup sexRadioGroup=null;  private CheckBox spanishCheckBox=null;  private CheckBox historyCheckBox=null;  private CheckBox mathematicsCheckBox=null;  private CheckBox englishCheckBox=null;  private CheckBox ecologyCheckBox=null;  private CheckBox politicsCheckBox=null;  private FloatingActionButton registerButton=null;  private TextView resultTextView=null;   private String name=null;  private int age=0;  private char sex=' ';  private ArrayList<String> topics=null;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  setSupportActionBar(toolbar);   nameEditText = (EditText) findViewById(R.id.nameEditText);  ageEditText = (EditText) findViewById(R.id.ageEditText);  sexRadioGroup = (RadioGroup) findViewById(R.id.sexRadioGroup);  spanishCheckBox = (CheckBox) findViewById(R.id.spanishCheckBox);  historyCheckBox = (CheckBox) findViewById(R.id.historyCheckBox);  mathematicsCheckBox = (CheckBox) findViewById(R.id.mathematicsCheckBox);  englishCheckBox = (CheckBox) findViewById(R.id.englishCheckBox);  ecologyCheckBox = (CheckBox) findViewById(R.id.ecologyCheckBox);  politicsCheckBox = (CheckBox) findViewById(R.id.politicCheckBox);  resultTextView = (TextView) findViewById(R.id.resultTextView);   registerButton = (FloatingActionButton) findViewById(R.id.registerButton);  registerButton.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  name=nameEditText.getText().toString();  age=Integer.parseInt(ageEditText.getText().toString());    resultTextView.setText(name+','+age+","+sex+","+topics.toString());  }  });   nameEditText.addTextChangedListener(new TextWatcher() {  @Override  public void beforeTextChanged(CharSequence s, int start, int count, int after) {   }   @Override  public void onTextChanged(CharSequence s, int start, int before, int count) {  String inputString = String.valueOf(s.subSequence(start, start + count));  Log.v("Prueba",inputString);  }   @Override  public void afterTextChanged(Editable s) {   }  });   sexRadioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(RadioGroup group, @IdRes int checkedId) {  switch (checkedId){  case R.id.menRadioButton:  Log.d("Sexo seleccionado","Masculino");  sex='M';  break;  case R.id.womenRadioButton:  Log.d("Sexo seleccionado","Femenino");  sex='F';  break;  default:  Log.d("Sexo seleccionado","Masculino");  break;  }  }  });   spanishCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked) {  topics.add("Español");  Log.d("Materia seleccionada", "Español");  }else{  if(topics.contains("Español")){  topics.remove("Español");  }  }  }  });   mathematicsCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked){  topics.add("Matematicas");  }else{  if(topics.contains("Matematicas")){  topics.remove("Matematicas");  }  }  }  });   historyCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked){  topics.add("Historia");  }else{  if(topics.contains("Historia")){  topics.remove("Historia");  }  }  }  });   englishCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked){  topics.add("Ingles");  }else{  if(topics.contains("Ingles")){  topics.remove("Ingles");  }  }  }  });   ecologyCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked){  topics.add("Ecologia");  }else{  if(topics.contains("Ecologia")){  topics.remove("Ecologia");  }  }  }  });   politicsCheckBox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  @Override  public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  if(isChecked){  topics.add("Politica");  }else{  if(topics.contains("Politica")){  topics.remove("Politica");  }  }  }  });  } } |

En esta clase, se implementan cada evento de cada control básico para obtener los valores en ellos, una vez que se tenga la implementación procederemos a ejecutar la aplicación en un emulador o dispositivo para probar su funcionamiento.

## **Controles de selección**

Actualmente, en la mayoria de las aplicaciones se requiere mostrar información para que el usuario pueda consultarla, en determinadas ocasiones debe realizar una selección de está, es por ello, que esta información se debe distribuir mediante listas, tablas, etc. Es por ello que en Android existen elementos que permiten distribuir información a través de listas o tablas que permiten seleccionarla para consultarla.

En Android tenemos varios controles de seleccion en los cuales nos permite distribuir la información ofreciendo una mejor experiencia para el usuario, estos controles de igual manera tiene sus propios eventos, ademas de ello, requieren de un contenedor de la información que se nombran como “adaptadores”, en los cuales se encarga de procesar la información y presentarla en cada elemento del control.

### ***Listas***

Este es el primer control de selección en el cual los elementos se muestran consecutivamente en forma lineal, en el cual son mostrados de acuerdo a su contenedor de información, este control es muy utilizado para información estructurada y que contenga gran cantidad de ella.

A continuación vamos a generar una ejemplo utilizando este control de selección, generando una aplicación en nuestro Android Studio a partir de una plantilla vacia, y en nuestra vista xml agregamos el siguiente código:

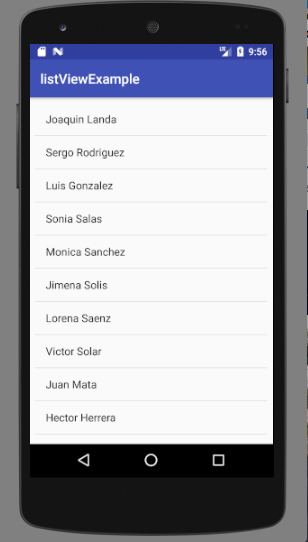
|  |
| --- |
| <ListView  android:id="@+id/peopleListView"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"/> |

Ahora lo siguiente es implementar esta lista del lado de nuestro controlador para que muestre la información, como se menciono anteriormente este requiere de un contenedor de información el cual es conocido como ‘Adaptador’, para ello vamos a implementar el siguiente código:

|  |
| --- |
| package com.cybertech.listviewexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.widget.ArrayAdapter; import android.widget.ListView;  public class MainActivity extends AppCompatActivity {   private ListView peopleListView = null;  private final String[] people= new String[]{  "Joaquin Landa","Sergo Rodriguez","Luis Gonzalez", "Sonia Salas",  "Monica Sanchez","Jimena Solis","Lorena Saenz", "Victor Solar",  "Juan Mata","Hector Herrera"  };   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  peopleListView = (ListView) findViewById(R.id.peopleListView);  ArrayAdapter<String> peopleAdapter = new ArrayAdapter<String>(this,  android.R.layout.simple\_list\_item\_1,people);  peopleListView.setAdapter(peopleAdapter);  } } |

En el código como se puede establecer se esta asignando un adaptador a la lista, este elemento para que funcione requiere de ciertos elementos para su funcionamiento, los cuales en este caso son el contexto donde va a trabajar, una vista donde va mostrar la información de cada elemento, y los elementos que va a contener y controlar para la lista, este es un controlador auxiliar para la lista apoyando en el manejo de la información.

Una vez que se tenga el código implementado procedemos a ejecutar este ejemplo, el cual tenemos un resultado similar como se muestra a continuación:



**Ilustración 24.- Listas**

Hasta aqui ya tenemos nuestro control implementando, solo falta agregar el principal evento correspondiente para seleccionar un elemento de la lista, para ello vamos a implementar el siguiente código:

|  |
| --- |
| private AdapterView.OnItemClickListener peopleOnItemClickListener = new  AdapterView.OnItemClickListener() {  @Override  public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {  Toast.makeText(getBaseContext(),people[i],Toast.LENGTH\_LONG).show();  }  }; |

Este atributo implementa el evento en el cual cuando se selecciona un elemento va a mostrar un mensaje en un Toast del nombre de la persona seleccionada, el siguiente paso es asignar este evento a nuestra lista para que pueda funcionar correctamente:

|  |
| --- |
| peopleListView.setOnItemClickListener(peopleOnItemClickListener); |

Con esto obtenemos un control de selección en forma de lista funcionando correctamente, en el siguiente capitulo veremos como implementar una lista en el cual cada elemento mostrara mas información en cada elemento.

### ***Listas personalizadas***

En este capitulo reforzaremos el control de selección de tipo “Lista”, en el cual mediante un ejemplo se desarrollara una aplicación en el cual se implementara una lista en la cual los elementos tendran información distribuida.

Creamos nuestra aplicacion de igual manera que el ejemplo anterior, lo primero que vamos a generar es nuestra vista para nuestros elementos como en el siguiente código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.constraint.ConstraintLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">   <ImageView  android:id="@+id/contactImageView"  android:layout\_width="96dp"  android:layout\_height="93dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"  app:srcCompat="@mipmap/contact"  android:layout\_marginLeft="8dp"/>   <TextView  android:id="@+id/nameTextView"  android:layout\_width="247dp"  android:layout\_height="21dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:text="TextView"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toEndOf="@+id/contactImageView"  app:layout\_constraintTop\_toTopOf="parent"/>   <TextView  android:id="@+id/lastnameTextView"  android:layout\_width="150dp"  android:layout\_height="21dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="52dp"  android:text="TextView"  app:layout\_constraintStart\_toEndOf="@+id/contactImageView"  app:layout\_constraintTop\_toBottomOf="@+id/nameTextView"  android:layout\_marginLeft="8dp"/>   <TextView  android:id="@+id/birthdayTextView"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="52dp"  android:text="TextView"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintHorizontal\_bias="0.725"  app:layout\_constraintStart\_toEndOf="@+id/lastnameTextView"  app:layout\_constraintTop\_toBottomOf="@+id/nameTextView"/>  </android.support.constraint.ConstraintLayout> |

Despues de esto, el siguiente paso es crear nuestro propio adaptador para que esté sea el que debe administrar la información y se muestre como deseamos en cada uno de los elementos de nuestra lista, debemos apoyarnos en generar una clase modelo que nos permitirá almacenar la información y que permitirá implementar nuestra lista, esta clase es la siguiente:

|  |
| --- |
| package com.cybertech.listviewexample;  */\*\* \* Created by Tazmadroid on 03/11/17. \*/*  public class PeopleModel {   private int id=0;  private String name=null;  private String lastname=null;  private String birthday=null;   public PeopleModel() {  }   public PeopleModel(int id, String name, String lastname, String birthday) {  this.id = id;  this.name = name;  this.lastname = lastname;  this.birthday = birthday;  }   public int getId() {  return id;  }   public void setId(int id) {  this.id = id;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public String getLastname() {  return lastname;  }   public void setLastname(String lastname) {  this.lastname = lastname;  }   public String getBirthday() {  return birthday;  }   public void setBirthday(String birthday) {  this.birthday = birthday;  } } |

Una vez que se tenga esta clase, procederemos a integrar nuestro adaptador como la siguiente clase:

|  |
| --- |
| package com.cybertech.listviewexample;  import android.content.Context; import android.support.annotation.NonNull; import android.support.annotation.Nullable; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.ArrayAdapter; import android.widget.TextView;  import java.util.List;  */\*\* \* Created by Tazmadroid on 03/11/17. \*/*    public class PeopleAdapter extends ArrayAdapter<PeopleModel> {   private Context context=null;  private List<PeopleModel> peopleModelList;   public PeopleAdapter(@NonNull Context context, @NonNull List<PeopleModel> objects) {  super(context, R.layout.item\_people, objects);  this.context=context;  this.peopleModelList=objects;  }   @NonNull  @Override  public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {  View item = convertView;  PeopleViewHolder peopleViewHolder;   if(item == null)  {  LayoutInflater inflater = (LayoutInflater) context.  getSystemService(context.LAYOUT\_INFLATER\_SERVICE);  item = inflater.inflate(R.layout.item\_people, null);  peopleViewHolder = new PeopleViewHolder();  peopleViewHolder.nameTextView=(TextView)item.  findViewById(R.id.nameTextView);  peopleViewHolder.lastnameTextView=(TextView)item.  findViewById(R.id.lastnameTextView);  peopleViewHolder.birthdayTextView=(TextView)item.  findViewById(R.id.birthdayTextView);  item.setTag(peopleViewHolder);  }else{  peopleViewHolder=(PeopleViewHolder) item.getTag();  }    peopleViewHolder.nameTextView.setText(peopleModelList.get(position).getName());  peopleViewHolder.lastnameTextView.setText(peopleModelList.get(position).getLastname());  peopleViewHolder.birthdayTextView.setText(peopleModelList.get(position).getBirthday());   return(item);  }   @Override  public int getCount() {  if(peopleModelList!=null){  return peopleModelList.size();  }else{  return 0;  }  }   static class PeopleViewHolder{  TextView nameTextView;  TextView lastnameTextView;  TextView birthdayTextView;  } } |

En esta clase se utiliza un concepto en programacion orientado a objetos, el cual es “Herencia”, aqui es un ejemplo claro para implementar el adaptador, para ello, esta clase debe de heredar de una clase de tipo adaptador, en este caso fue de la clase ArrayAdapter, nuestra clase al heredar de ella, debe implementar un constructor por lo menos y métodos que permitiran establecer la funcionalidad deseada.

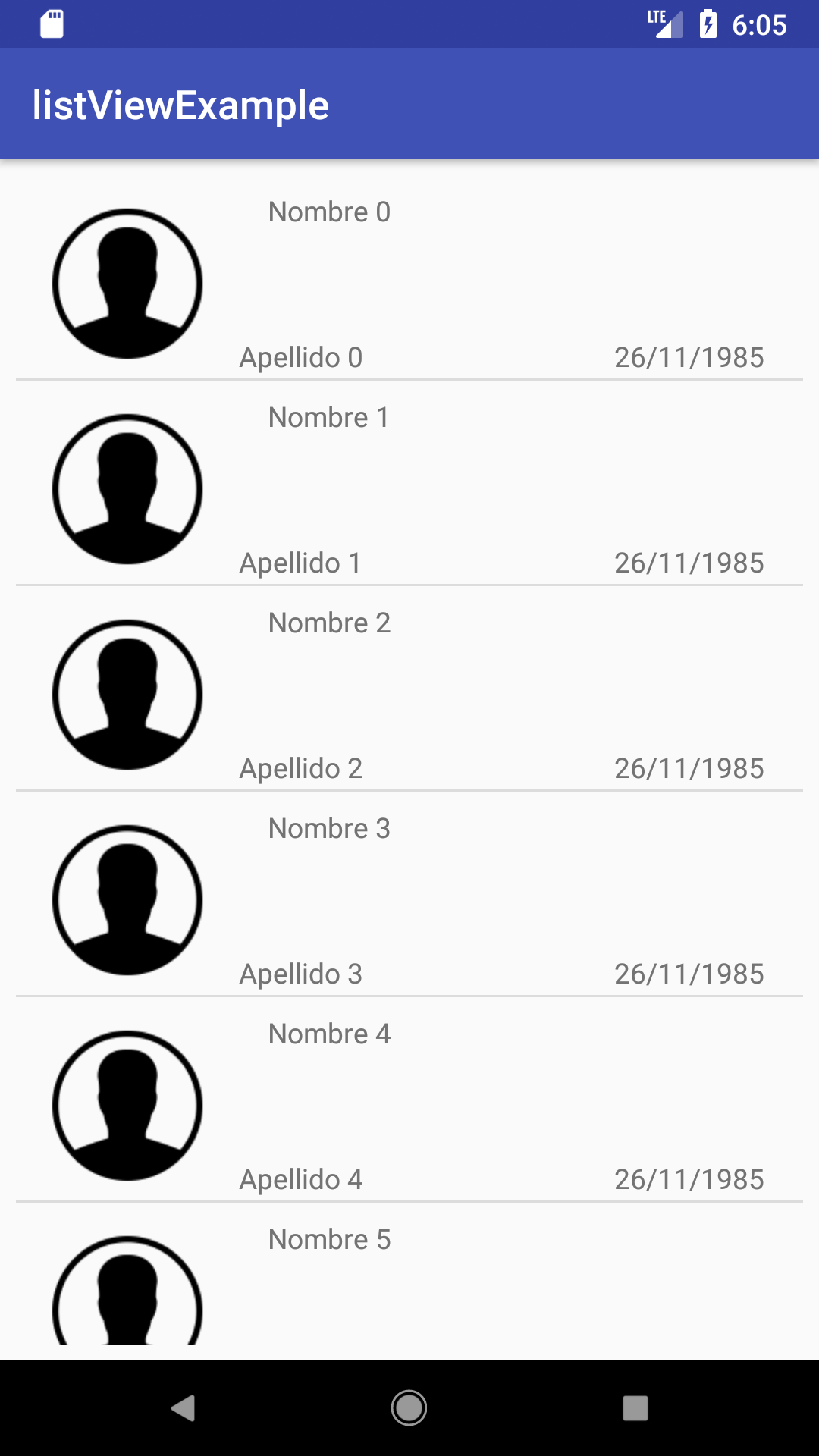
El constructor es propio de la clase, en el cual recibe el contexto y una coleccion de objetos de nuestro modelo que declaramos anteriormente, pero dentro de él, se implementa el metodo super, el cual permite implementar el constructor de la clase padre, en esté se comparten parametros entre ellos son los que recibe nuestro constructor y el identificador de la vista que se va a mostrar en cada elemento.

El método getView es el principal de nuestra clase, puesto es el encargado de mostrar cada elemento de la lista con la información contenida en el adaptador personalizado, en el cual se realiza un proceso en el que se asigna los valores en cada uno de los elementos de nuestra vista.

Una vez que se tenga este adaptador, procederemos a integrarlo en la lista para que empiece a mostrar nuestra información como deseamos, en nuestra actividad principal haremos unos cambios, en la cual quedara como en el siguiente código:

|  |
| --- |
| package com.cybertech.listviewexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.view.View; import android.widget.AdapterView; import android.widget.ArrayAdapter; import android.widget.ListView; import android.widget.Toast;  import java.util.ArrayList; import java.util.List;  public class MainActivity extends AppCompatActivity {   private ListView peopleListView = null;  private List<PeopleModel> people= null;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  initializePeople();  peopleListView = (ListView) findViewById(R.id.peopleListView);  PeopleAdapter peopleAdapter = new PeopleAdapter(this,  people);  peopleListView.setAdapter(peopleAdapter);  peopleListView.setOnItemClickListener(peopleOnItemClickListener);  }   public void initializePeople(){  people=new ArrayList<PeopleModel>();  for (int indice=0;indice<=10;indice++){  people.add(new PeopleModel(indice,"Nombre "+indice,"Apellido "+indice,"26/11/1985"));  }  }   private AdapterView.OnItemClickListener peopleOnItemClickListener = new  AdapterView.OnItemClickListener() {  @Override  public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {  Toast.makeText(getBaseContext(),people.get(i).getName(),Toast.LENGTH\_LONG).show();  }  }; } |

Este ejemplo se visualizaria como en la siguiente imagen:



**Ilustración 25.- Lista de contactos**

### ***Recyclerview y cardview***

A partir del lanzamiento del patron de diseño Material Design, se ha ido implementando nuevos elementos que permiten tener una mejor experiencia de usuario y proporcionar nuevas experiencias de usuario, uno de estos elementos se nombra como “RecyclerView”, este elemento esta basado en el funcionamiento de una lista, su principal diferencia es que se puede implementar diferentes distribuciones de los elementos como diferentes vistas de los mismos.

Este elemento dentro de sus funcionalidades es reciclar las vistas que contiene, asi tiene un mejor manejo de memoria y mejor experiencia de usuario, este elemento se puede implementar patrones de distribución de los elementos a través administradores de vistas conocidos como “LayoutaManager”, son los encargados de distribuir nuestros elementos a través de la utilización de adaptadores como en las listas.

Para implementar las vistas se puede utilizar los contenedores que ya conocemos o se puede utilizar un elemento nuevo que se nombra como “Cardview”, este fue lanzado al mismo tiempo que el patron de diseño “Material Design”, en el cual es utilizado para encapsular vistas que se muestran como una tarjeta en nuestra lista, ademas de integrar muchos elementos que se manejan en el patron de diseño como elevación, etc.

Este elemento permite integrar mejor la reutilización de la vista en el Recyclerview, permitiendo optimizar nuestra aplicación y tener una mejor experiencia de usuario. A continuación realizaremos un ejemplo utilizando el que realizamos en el tema anterior, lo primero que vamos hacer es un cambio en la vista de nuestra actividad:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.constraint.ConstraintLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.recyclerviewexample.MainActivity">    <android.support.v7.widget.RecyclerView  android:id="@+id/contactsRecyclerView"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"/>  </android.support.constraint.ConstraintLayout> |

El siguiente paso es realizar cambios sobre la vista de los elementos de la lista, es donde se implementara nuestro cardview del contacto, esta vista es la siguiente:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.v7.widget.CardView  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:cardview="http://schemas.android.com/apk/res-auto"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  cardview:cardBackgroundColor="#61B7FF"  cardview:cardCornerRadius="12dp"  cardview:cardElevation="8dp"  cardview:cardUseCompatPadding="true"  cardview:contentPadding="4dp">   <android.support.constraint.ConstraintLayout  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">   <ImageView  android:id="@+id/contactImageView"  android:layout\_width="96dp"  android:layout\_height="93dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  cardview:layout\_constraintStart\_toStartOf="parent"  cardview:layout\_constraintTop\_toTopOf="parent"  cardview:srcCompat="@mipmap/contact"  android:layout\_marginLeft="8dp"/>   <TextView  android:id="@+id/nameTextView"  android:layout\_width="247dp"  android:layout\_height="21dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:text="TextView"  cardview:layout\_constraintEnd\_toEndOf="parent"  cardview:layout\_constraintStart\_toEndOf="@+id/contactImageView"  cardview:layout\_constraintTop\_toTopOf="parent"/>   <TextView  android:id="@+id/lastnameTextView"  android:layout\_width="150dp"  android:layout\_height="21dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="52dp"  android:text="TextView"  cardview:layout\_constraintStart\_toEndOf="@+id/contactImageView"  cardview:layout\_constraintTop\_toBottomOf="@+id/nameTextView"  android:layout\_marginLeft="8dp"/>   <TextView  android:id="@+id/birthdayTextView"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="52dp"  android:text="TextView"  cardview:layout\_constraintEnd\_toEndOf="parent"  cardview:layout\_constraintHorizontal\_bias="0.725"  cardview:layout\_constraintStart\_toEndOf="@+id/lastnameTextView"  cardview:layout\_constraintTop\_toBottomOf="@+id/nameTextView"/>   </android.support.constraint.ConstraintLayout>  </android.support.v7.widget.CardView> |

A continuación generaremos nuestro adaptador que administre nuestro elementos para mostrarlos en la vista, he aqui la principal diferencia con las listas, pues se debe de ocupar un adaptador en especifico que permite implementar nuestros elementos, este a su vez requiere utilizar un elemento conocido como ViewHolder, el cual va permitir una consistencia y optimización de los elementos, este adaptador es el siguiente:

|  |
| --- |
| package com.cybertech.recyclerviewexample;  import android.support.v7.widget.RecyclerView; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.TextView; import android.widget.Toast;  import java.util.List;  */\*\* \* Created by Tazmadroid on 20/11/17. \*/*  public class PeopleAdapter extends RecyclerView.Adapter<PeopleAdapter.PeopleViewHolder>{   private List<PeopleModel> peopleModels=null;   public PeopleAdapter(List<PeopleModel> peopleModels) {  this.peopleModels = peopleModels;  }   @Override  public PeopleViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  View item = LayoutInflater.from(parent.getContext()).inflate(R.layout.  item\_people, parent,false);  return new PeopleViewHolder(item);  }   @Override  public void onBindViewHolder(PeopleViewHolder holder, int position) {  holder.nameTextView.setText(peopleModels.get(position).getName());  holder.lastnameTextView.setText(peopleModels.get(position).getLastname());  holder.birthdayTextView.setText(peopleModels.get(position).getBirthday());  holder.setPeople(peopleModels.get(position));  }   @Override  public int getItemCount() {  if(peopleModels!=null){  return peopleModels.size();  }else{  return 0;  }  }   static class PeopleViewHolder extends RecyclerView.ViewHolder{   protected TextView nameTextView;  protected TextView lastnameTextView;  protected TextView birthdayTextView;   private PeopleModel people=null;   public PeopleViewHolder(final View itemView) {  super(itemView);  nameTextView = (TextView) itemView.findViewById(R.id.nameTextView);  lastnameTextView = (TextView) itemView.findViewById(R.id.lastnameTextView);  birthdayTextView = (TextView) itemView.findViewById(R.id.birthdayTextView);    itemView.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  if(people!=null){  Toast.makeText(itemView.getContext(),people.getName(),Toast.  LENGTH\_LONG).show();  }  }  });   }   public void setPeople(PeopleModel people) {  this.people = people;  }  } } |

Hay que observar bien en esta clase, puesto que a través de una subclase estatica se implementa el ViewHolder para que nuestro adaptador pueda implementar, en la clase del adaptador se muestra la clase de cual debe heredar para implementarlo asi como la clase que debe heredar el ViewHolder que implementamos.

El ultimo paso es implementar nuestro RecyclerView en nuestra actividad para que se pueda mostrar los elementos que requerimos, para ello se actualizara el controlador de la vista principal con el siguiente código:

|  |
| --- |
| package com.cybertech.recyclerviewexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.support.v7.widget.GridLayoutManager; import android.support.v7.widget.LinearLayoutManager; import android.support.v7.widget.RecyclerView; import android.widget.ListView;  import java.util.ArrayList; import java.util.List;  public class MainActivity extends AppCompatActivity {   private List<PeopleModel> people= null;  private RecyclerView contactRecyclerView=null;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  initializePeople();  contactRecyclerView = (RecyclerView) findViewById(R.id.contactsRecyclerView);  LinearLayoutManager layoutManager= new LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false);  contactRecyclerView.setHasFixedSize(true);  contactRecyclerView.setLayoutManager(layoutManager);  PeopleAdapter peopleAdapter = new PeopleAdapter(people);  contactRecyclerView.setAdapter(peopleAdapter);   }   public void initializePeople(){  people=new ArrayList<PeopleModel>();  for (int indice=0;indice<=10;indice++){  people.add(new PeopleModel(indice,"Nombre "+indice,"Apellido "+indice,"26/11/1985"));  }  } } |

Una vez finalizado con los cambios, ejecutamos nuestro ejemplo y el resultado debe ser similar a la siguiente imagen:



**Ilustración 26.- Listas con RecyclerView y CardView**

En este elemento tenemos diferentes vistas que se pueden utilizar para distribuir nuestros elementos como se muestran en las siguientes imagenes:

|  |  |
| --- | --- |
| **Ilustración 27.- GridLayout** | **Ilustración 28.- StaggeredLayout** |

## **Controles de navegación**

### ***Appbarlayout y toolbar***

Este nuevo elemento que nacio junto con el patron de diseño anteriormente mencionada, nos va permitir implementar una pieza clave en nuestra app, que es tener una barra de navegación que permita interactuar con ciertas transiciones y eventos en ella, además trabaja conjuntamente con un elemento que ha tenido varias transformaciónes conocido como “Toolbar”, este vino a reemplazar el uso del elemento “ActionBar”; este elemento administra un menú de opciones con la navegación entre actividades y otros elementos.

A continuación realizaremos un ejercicio utilizando el ejemplo de RecyclerView anteriormente desarrollado, para ello realizaremos unos cambios en la vista principal la cual sera como en el siguiente código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.design.widget.CoordinatorLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  xmlns:app="http://schemas.android.com/apk/res-auto"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">   <android.support.design.widget.AppBarLayout  android:id="@+id/appbar"  android:layout\_height="wrap\_content"  android:layout\_width="match\_parent">   <android.support.v7.widget.Toolbar  android:id="@+id/toolbar"  android:layout\_height="?attr/actionBarSize"  android:layout\_width="match\_parent"  app:title="RecyclerView Example">  </android.support.v7.widget.Toolbar>   </android.support.design.widget.AppBarLayout>   <android.support.v7.widget.RecyclerView  android:id="@+id/contactsRecyclerView"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_behavior="android.support.design.widget.AppBarLayout$ScrollingViewBehavior"/>  </android.support.design.widget.CoordinatorLayout> |

En la vista podemos ver que se esta utilizando el elemento “CoordinatorLayout” para implementar los nuevos elementos, ahora el siguiente proceso es implementar nuestra barra de título como en el siguiente código:

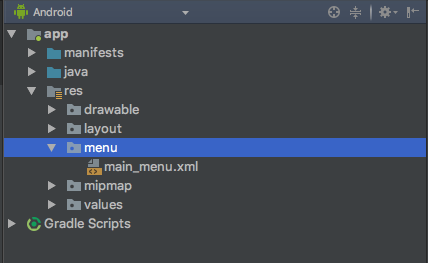
|  |
| --- |
| package com.cybertech.recyclerviewexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.support.v7.widget.LinearLayoutManager; import android.support.v7.widget.RecyclerView; import android.support.v7.widget.Toolbar;  import java.util.ArrayList; import java.util.List;  public class MainActivity extends AppCompatActivity {   private List<PeopleModel> people= null;  private RecyclerView contactRecyclerView=null;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  setSupportActionBar(toolbar);  initializePeople();  contactRecyclerView = (RecyclerView) findViewById(R.id.contactsRecyclerView);  LinearLayoutManager layoutManager= new LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false);  contactRecyclerView.setHasFixedSize(true);  contactRecyclerView.setLayoutManager(layoutManager);  PeopleAdapter peopleAdapter = new PeopleAdapter(people);  contactRecyclerView.setAdapter(peopleAdapter);   }   public void initializePeople(){  people=new ArrayList<PeopleModel>();  for (int indice=0;indice<=10;indice++){  people.add(new PeopleModel(indice,"Nombre "+indice,"Apellido "+indice,"26/11/1985"));  }  } } |

Con estos cambios ya tenemos implementado estos nuevos elementos, conjuntamente en el siguiente tema veremos elementos que trabajan con estos elementos que permiten integrar una navegación entre nuestras actividades.

### ***Menú***

Este elemento se integra dentro de nuestra “Toolbar”, en la cual se configuran acciones que puede mostrar a traves de iconos o listas, estas opciones pueden realizar acciones o mostrar actividades dentro de nuestra app.

Esto se integra generando un archivo xml que permitirá reconocer estas opciones, para ello vamos a utilizar en el mismo ejemplo que hemos manejado; se debera crear el directorio “menú” dentro de la carpeta de recursos de la app:



**Ilustración 29.- Directorio de menú**

Dentro de ella generamos un archivo xml de menú como se muestra en la imagen, y agregamos el siguiente código:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <menu xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto">   <item  android:id="@+id/registerContact"  android:icon="@android:drawable/ic\_input\_add"  android:orderInCategory="100"  android:title="Registrar contacto"  app:showAsAction="always"/>   <item  android:id="@+id/deleteContact"  android:icon="@android:drawable/ic\_input\_delete"  android:orderInCategory="100"  android:title="Eliminar contacto"  app:showAsAction="always"/>   <item  android:id="@+id/searchContact"  android:icon="@android:drawable/ic\_menu\_search"  android:orderInCategory="100"  android:title="Buscar contacto"  app:showAsAction="ifRoom"/>   <item  android:id="@+id/favoritesContact"  android:icon="@android:drawable/btn\_star"  android:orderInCategory="100"  android:title="Favritos"  app:showAsAction="ifRoom"/>  </menu> |

Este archivo solo configura el menu dentro de nuestro toolbar, ademas cada opción tiene una caracteristica muy importante, la cual es la etiqueta “showAsAction” a través de ella permite realizar la visualización de la opción dependiendo del valor, en este caso; si la opción es visible en el espacio de la barra esta se va a mostrar, en caso contrario; se mostrara en una lista desplegable. Procederemos a implementarlo para que realice las acciones correspondientes, como el siguiente códizgo:

|  |
| --- |
| package com.cybertech.recyclerviewexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.support.v7.widget.LinearLayoutManager; import android.support.v7.widget.RecyclerView; import android.support.v7.widget.Toolbar; import android.view.Menu; import android.view.MenuItem; import android.widget.Toast;  import java.util.ArrayList; import java.util.List;  public class MainActivity extends AppCompatActivity {   private List<PeopleModel> people= null;  private RecyclerView contactRecyclerView=null;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  setSupportActionBar(toolbar);  initializePeople();  contactRecyclerView = (RecyclerView) findViewById(R.id.contactsRecyclerView);  LinearLayoutManager layoutManager= new LinearLayoutManager(this,LinearLayoutManager.VERTICAL,false);  contactRecyclerView.setHasFixedSize(true);  contactRecyclerView.setLayoutManager(layoutManager);  PeopleAdapter peopleAdapter = new PeopleAdapter(people);  contactRecyclerView.setAdapter(peopleAdapter);   }   @Override  public boolean onCreateOptionsMenu(Menu menu) {  getMenuInflater().inflate(R.menu.main\_menu,menu);  return true;  }   @Override  public boolean onOptionsItemSelected(MenuItem item) {  switch (item.getItemId()){  case R.id.registerContact:  Toast.makeText(getBaseContext(),"Selecciono registrar",  Toast.LENGTH\_LONG).show();  return true;  case R.id.deleteContact:  Toast.makeText(getBaseContext(),"Selecciono eliminar",  Toast.LENGTH\_LONG).show();  return true;  case R.id.searchContact:  Toast.makeText(getBaseContext(),"Selecciono buscar",  Toast.LENGTH\_LONG).show();  return true;  case R.id.favoritesContact:  Toast.makeText(getBaseContext(),"Selecciono favoritos",  Toast.LENGTH\_LONG).show();  return true;  default:  return super.onOptionsItemSelected(item);  }  }   public void initializePeople(){  people=new ArrayList<PeopleModel>();  for (int indice=0;indice<=10;indice++){  people.add(new PeopleModel(indice,"Nombre "+indice,"Apellido "+indice,"26/11/1985"));  }  } } |

Con el método “onCreateOptionsMenu” se le indica a nuestra actividad que implemente nuestra archivo de configuración del menú para que se muestren las opciones y administre la forma de mostrarse, y a través del método “onOptionsItemSelected” se implementan las acciones de cada opción del menú.

En cada caso que se requiera se puede mostrar una actividad nueva o realizar una acción en cada una de las opciones.

# **FRAGMENTS**

Este nuevo elemento nacío a partir de la generación de nuevos dispositivos como tabletas, debido a que se requeria adaptar las vistas que se diseñaban para celulares dentro de una pantalla mas amplia como la de las tabletas, anteriormente, en Android se desarrollaba dos aplicaciones para cada tipo de dispositivo, lo que conllevaba a que el programador desarrollara en mas tiempo y de manera independiente, esto acabo a partir de la versión 4.0 de Android conocida “IceCreamSandwich”.

Aqui es donde se crea un elemento que permitiera desarrollar una sola aplicación, y que esta fuera capaz de adaptarse al tipo de dispositivo. Este elemento se puede denominar como un fragmento o porción de la vista en la cual tiene su propio ciclo de vida en la cual conlleva sus propios procesos e interactuando con la vista se podria llevar a cabo funciones que pemitan tener una mejor experiencia de usuario.

## **Ciclo de vida**

El ciclo de vida de un fragment, es muy similar al de la actividad; como se puede mostrar en la siguiente imagen:



**Ilustración 30.- Ciclo de vida de un fragment**

En la imagen se muestra el flujo entre los ciclos de vida de un fragmento, a continuación se describen algunos de ellos:

* **onAttach()**

Este ciclo se ejecuta cuando el fragmento se asocia a la actividad.

* **onCreateView()**

Se llama en el momento que se crean las jerarquias de las vistas que estan incluidas en el fragmento.

* **onActivityCreated()**

Se llama cuando la actividad donde se integra es creada.

* **onDestroyView()**

Se llama cuando la jerarquia de vistas es removida del fragmento.

* **onDetach()**

Se ejecuta cuando el fragmento es removido de la actividad asociada.

## **Integración en una actividad**

En este ejercicio se realizara la integración de dos fragmentos en una aplicación para dos diferentes dispositivos, en el cual se vera la importancia de contar con ellos en una aplicación, debido a que muchos elementos se valen de ellos para su funcionamiento.

Se creara una nueva aplicación para ello, lo primero que vamos a realizar es agregar dos directorios dentro de la carpeta de recursos, estas se nombraran como “values-large” y “values-xlarge”, una vez creados los directorios, vamos a generar un archivo xml de valores que se nombrara “bools.xml” y contendra los siguientes valores:

* **res/values/bools.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <resources>  <bool name="is\_tablet">false</bool> </resources> |

* **res/values-large/bools.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <resources>  <bool name="is\_tablet">true</bool> </resources> |

* **res/values-xlarge/bools.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <resources>  <bool name="is\_tablet">true</bool> </resources> |

Una vez que se tenga estos archivos, realizaremos un proceso similar en la carpeta layout, generaremos dos directorios “layout-large” y “layout-xlarge”, estas carpetas son muy importantes para la implementación de nuestros fragmentos, ya que en ellos se tendran archivos similares en los cuales se van a implementar nuestra vistas, la vista en comun va ser la de la actividad principal, como se muestra a continuación:

* **res/layout/activity\_main.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <FrameLayout  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"  android:id="@+id/main\_container"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.fragmentsexample.MainActivity"> </FrameLayout> |

* **res/large/activity\_main.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.constraint.ConstraintLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.fragmentsexample.MainActivity">    <FrameLayout  android:id="@+id/main\_container"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toTopOf="@+id/detail\_container"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent">   </FrameLayout>   <FrameLayout  android:id="@+id/detail\_container"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/main\_container">   </FrameLayout> </android.support.constraint.ConstraintLayout> |

* **res/xlarge/activity\_main.xml**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <android.support.constraint.ConstraintLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.fragmentsexample.MainActivity">   <FrameLayout  android:id="@+id/main\_container"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toTopOf="@+id/detail\_container"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent">   </FrameLayout>   <FrameLayout  android:id="@+id/detail\_container"  android:layout\_width="0dp"  android:layout\_height="0dp"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/main\_container">   </FrameLayout> </android.support.constraint.ConstraintLayout> |

Una vez que se tengan estas vistas, procederemos a crear dos fragmentos en el cual en uno de ellos se va a implementar una lista de contactos y en la otra el detalle de este, a continuación se generara la interfaz de la vista para la lista

|  |
| --- |
| <FrameLayout 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"  tools:context="com.cybertech.fragmentsexample.ListFragment">    <android.support.v7.widget.RecyclerView  android:id="@+id/listContactRecyclerView"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"/> </FrameLayout> |

Ahora procederemos a implementar el controlador, que en este caso sera un fragmento en el cual manejara el control de la lista, en este ejemplo utilizaremos las clases que se utilizaron en el tema de listas con RecyclerView, como se muestra en el siguiente código:

|  |
| --- |
| package com.cybertech.fragmentsexample;  import android.content.Intent; import android.os.Bundle; import android.support.annotation.Nullable; import android.support.v4.app.Fragment; import android.support.v7.widget.LinearLayoutManager; import android.support.v7.widget.RecyclerView; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup;  import java.util.ArrayList; import java.util.List;   public class ListFragment extends Fragment implements OnContactItemClickListener{   private List<PeopleModel> people= null;  private RecyclerView listContactRecyclerView=null;   private boolean isTablet=false;   public ListFragment() {  *// Required empty public constructor*  }   public static ListFragment newInstance() {  ListFragment fragment = new ListFragment();  return fragment;  }   @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);   }   @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  *// Inflate the layout for this fragment*  return inflater.inflate(R.layout.fragment\_list, container, false);  }   @Override  public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {  super.onViewCreated(view, savedInstanceState);  isTablet=getResources().getBoolean(R.bool.is\_tablet);  initializePeople();  listContactRecyclerView = (RecyclerView) view  .findViewById(R.id.listContactRecyclerView);  LinearLayoutManager layoutManager= new LinearLayoutManager(getActivity(),  LinearLayoutManager.VERTICAL,false);  listContactRecyclerView.setHasFixedSize(true);  listContactRecyclerView.setLayoutManager(layoutManager);  PeopleAdapter peopleAdapter = new PeopleAdapter(people);  peopleAdapter.setOnContactItemClickListener(this);  listContactRecyclerView.setAdapter(peopleAdapter);  }   public void initializePeople(){  people=new ArrayList<PeopleModel>();  for (int indice=0;indice<=10;indice++){  people.add(new PeopleModel(indice,"Nombre "+indice,"Apellido "+indice,"26/11/1985"));  }  }   @Override  public void onContactClick(PeopleModel peopleModel) {  if(isTablet){  getActivity().getSupportFragmentManager().beginTransaction()  .replace(R.id.detail\_container,DetailFragment  .newInstance(peopleModel),"Detail-container").commit();  }else{  Intent detailIntent = new Intent(getActivity(),DetailActivity.class);  detailIntent.putExtra(DetailFragment.ARG\_PEOPLE,peopleModel);  startActivity(detailIntent);  }  } } |

Ahora el siguiente proceso se realizara la implementación de un fragmento del detalle del contacto, en el cual mostrara la información del contacto, lo primero es que vamos a generar la vista del detalle como en el siguiente código:

|  |
| --- |
| <android.support.constraint.ConstraintLayout  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"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.fragmentsexample.DetailFragment">    <TextView  android:id="@+id/nameContactTextView"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:text="@string/hello\_blank\_fragment"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"/>   <TextView  android:id="@+id/birthdayContactTextView"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_marginBottom="8dp"  android:layout\_marginEnd="8dp"  android:layout\_marginStart="8dp"  android:layout\_marginTop="8dp"  android:text="TextView"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintEnd\_toEndOf="parent"  app:layout\_constraintHorizontal\_bias="0.025"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toBottomOf="@+id/nameContactTextView"  app:layout\_constraintVertical\_bias="0.0"/>  </android.support.constraint.ConstraintLayout> |

Con respecto a la implementación del controlador este sera como en el siguiente código:

|  |
| --- |
| package com.cybertech.fragmentsexample;  import android.os.Bundle; import android.support.annotation.NonNull; import android.support.annotation.Nullable; import android.support.v4.app.Fragment; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.TextView;  public class DetailFragment extends Fragment {   private static final String ARG\_PEOPLE = "people";   private PeopleModel people;   private TextView nameContactTextView=null;  private TextView birthdayContactTextView=null;   public DetailFragment() {  *// Required empty public constructor*  }   public static DetailFragment newInstance(PeopleModel people) {  DetailFragment fragment = new DetailFragment();  Bundle args = new Bundle();  args.putSerializable(ARG\_PEOPLE,people);  fragment.setArguments(args);  return fragment;  }   @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  if (getArguments() != null) {  people =(PeopleModel) getArguments().getSerializable(ARG\_PEOPLE);  }  }   @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  *// Inflate the layout for this fragment*  return inflater.inflate(R.layout.fragment\_detail, container,  false);  }   @Override  public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  super.onViewCreated(view, savedInstanceState);  nameContactTextView = view.findViewById(R.id.nameContactTextView);  birthdayContactTextView = view.findViewById(R.id.birthdayContactTextView);  if(people!=null){  nameContactTextView.setText(people.getName()+" "+people.getLastname());  birthdayContactTextView.setText(people.getBirthday());  }else{  nameContactTextView.setText("");  birthdayContactTextView.setText("");  }  } } |

Con esto tendremos finalizado los fragmentos necesarios, el siguiente proceso es el implementar la logica para que dependiendo del dispositivo, nuestra app tenga la capacidad de mostrarse correctamente, para ello vamos a apoyarnos en generar una actividad que unicamente se mostrara si el dispositivo es un celular.

Ahora para permitir que nuestra app permita interactuar entre fragmentos, generaremos una interface que nos permite implementar la accion cuando el usuario quiera mostrar el detalle de un contacto, el código es el siguiente:

|  |
| --- |
| package com.cybertech.fragmentsexample;  */\*\* \* Created by Tazmadroid on 27/12/17. \*/*  public interface OnContactItemClickListener {   public void onContactClick(PeopleModel peopleModel);  } |

Esta interface se implementa en nuestro adaptador, por lo cual queda con el siguiente código:

|  |
| --- |
| package com.cybertech.fragmentsexample;  import android.support.v7.widget.RecyclerView; import android.view.LayoutInflater; import android.view.View; import android.view.ViewGroup; import android.widget.TextView; import android.widget.Toast;  import java.util.List;  */\*\* \* Created by Tazmadroid on 20/11/17. \*/*  public class PeopleAdapter extends RecyclerView.Adapter<PeopleAdapter.PeopleViewHolder>{   private List<PeopleModel> peopleModels=null;  private boolean isTablet=false;   private OnContactItemClickListener onContactItemClickListener=null;   public PeopleAdapter(List<PeopleModel> peopleModels) {  this.peopleModels = peopleModels;  }   @Override  public PeopleViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  View item = LayoutInflater.from(parent.getContext()).inflate(R.layout.  item\_people, parent,false);  return new PeopleViewHolder(item);  }   @Override  public void onBindViewHolder(PeopleViewHolder holder, int position) {  holder.nameTextView.setText(peopleModels.get(position).getName());  holder.lastnameTextView.setText(peopleModels.get(position).getLastname());  holder.birthdayTextView.setText(peopleModels.get(position).getBirthday());  holder.setPeople(peopleModels.get(position));  holder.setOnContactItemClickListener(onContactItemClickListener);  }   @Override  public int getItemCount() {  if(peopleModels!=null){  return peopleModels.size();  }else{  return 0;  }  }   public void setOnContactItemClickListener(OnContactItemClickListener onContactItemClickListener) {  this.onContactItemClickListener = onContactItemClickListener;  }   static class PeopleViewHolder extends RecyclerView.ViewHolder{   protected TextView nameTextView;  protected TextView lastnameTextView;  protected TextView birthdayTextView;   private OnContactItemClickListener onContactItemClickListener=null;   private PeopleModel people=null;   public PeopleViewHolder(final View itemView) {  super(itemView);  nameTextView = (TextView) itemView.findViewById(R.id.nameTextView);  lastnameTextView = (TextView) itemView.findViewById(R.id.lastnameTextView);  birthdayTextView = (TextView) itemView.findViewById(R.id.birthdayTextView);  itemView.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View view) {  if(onContactItemClickListener!=null){  onContactItemClickListener.onContactClick(people);  }  }  });   }   public void setPeople(PeopleModel people) {  this.people = people;  }   public void setOnContactItemClickListener(OnContactItemClickListener onContactItemClickListener) {  this.onContactItemClickListener = onContactItemClickListener;  }  } } |

Para finalizar nuestra aplicación, para que el detalle se pueda visualizar en un celular debemos de apoyarnos en una actividad que incluya el fragmento del detalle de nuestro contacto, para ello vamos a generar una nueva actividad y esta es la vista:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <FrameLayout  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"  android:id="@+id/detail\_container"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="com.cybertech.fragmentsexample.DetailActivity">  </FrameLayout> |

Y a continuación el controlador quedara de la siguiente manera:

|  |
| --- |
| package com.cybertech.fragmentsexample;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle;  public class DetailActivity extends AppCompatActivity {   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_detail);  PeopleModel peopleModel = (PeopleModel) getIntent().getSerializableExtra(DetailFragment  .ARG\_PEOPLE);  getSupportFragmentManager().beginTransaction().replace(R  .id.detail\_container,DetailFragment.newInstance(peopleModel),  "Detail-container").commit();  } } |

Con esto finalizamos el ejemplo de fragmentos, como se puede ver, estos permiten integrar diferentes vistas ademas que diferentes elementos los utilizan para su funcionamiento como por ejemplo un menú desplegable.

# **ALMACENAMIENTO DE DATOS EN EL DISPOSITIVO**

## **Introducción a SQLITE**

Actualmente en la mayoria de aplicaciones como sistemas, es muy importante el guardar información que se administra mediante una base de datos, las cuales en su mayoria son relacionables, pero en el caso de los dispositivos moviles para almacenar información existen varias formas, entre ellas es SQLITE.

Es un manejador de bases de datos no relacionables, que nos permite crear tablas y realizar consultas y operaciones sobre ellas, muy similar a MySQL o algún otro manejador de base de datos. Dentro de un archivo se encapsula la información para tener un mejor acceso a ella, aunque esta herramienta es limitada para realizar operaciones en la base de datos, nos da la oportunidad de guardar gran información estructurada.

## **Insertar, actualizar y eliminar registros**

Lo primero que debemos de realizar es generar una clase que herede de la clase SQLiteOpenHelper, la cual nos permitirá crear nuestras tablas dentro de nuestra app como actualizarla en el caso de requerirlo.

Como se muestra en el siguiente código:

|  |
| --- |
| package com.cybertech.basedatosexample.database;  import android.content.Context; import android.database.SQLException; import android.database.sqlite.SQLiteDatabase; import android.database.sqlite.SQLiteOpenHelper; import android.util.Log;  */\*\* \* Created by jaivetorrespineda on 04/01/18. \*/*  public class ContactsDatabase extends SQLiteOpenHelper {  private static final String CREATE\_USER\_TABLE="CREATE TABLE `user` (" +  " `us\_id` INT NOT NULL," +  " `us\_name` VARCHAR(50) NULL," +  " `us\_lastname` VARCHAR(100) NULL," +  " `us\_number\_phone` INT(12) NULL," +  " `us\_email` VARCHAR(150) NULL," +  " PRIMARY KEY (`us\_id`))";  private static final String DATABASE\_NAME="users"; private static final int DATABASE\_VERSION=1;  public ContactsDatabase(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {  super(context, DATABASE\_NAME, factory, DATABASE\_VERSION); }  @Override public void onCreate(SQLiteDatabase sqLiteDatabase) {  try{  sqLiteDatabase.execSQL(CREATE\_USER\_TABLE);  }catch (SQLException e){  Log.e("[ERROR]", e.getMessage());  }catch (Exception f){  Log.e("[ERROR]", f.getMessage());  }  }  @Override public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {  try{  sqLiteDatabase.execSQL(CREATE\_USER\_TABLE);  }catch (SQLException e){  Log.e("[ERROR]", e.getMessage());  }catch (Exception f){  Log.e("[ERROR]", f.getMessage());  } } } |

Esta clase permitirá crear nuestra base de datos cuando sea llamada por primera vez, en caso contrario, permitirá el acceso a ella para realizar operaciones sobre ella, lo importante en ella es que se deben implementar los dos metodos que se muestran, el primero se ejecuta cuando se crea la base de datos por primera vez y ejecuta sentencias para crearla, en el segundo metodo se ejecuta en el caso de actualizar la versión de base de datos que contenga la aplicación, en este se debera llevar a cabo el proceso de migración de la información en el caso de que sea necesario.

Es importante implementar un constructor de la clase padre, este permitirá definir el nombre de la base de datos y su versión, en este caso se estan definiendo desde la propia y se especifica en el metodo super para que realize todo el proceso para crear la base de datos y ejecute los metodos antes mencionados.

Ahora debemos generar una clase que permita realizar operaciones sobre la base de datos para ello debemos de generar un constructor como el siguiente:

|  |
| --- |
| public ContactsDatabaseController(Context context, ACCESS\_MODE accessMode) {  this.context = context;  contactsDatabase=new ContactsDatabase(context);  if(accessMode==ACCESS\_MODE.READ){  sqLiteDatabase=contactsDatabase.getReadableDatabase();  }else{  sqLiteDatabase=contactsDatabase.getWritableDatabase();  } } |

En este caso se tienen dos objetos, uno de ellos es nuestro objeto ContactsDatabase para crear o actualizar la base de datos, el segundo es un objeto tipo SQLiteDatabase en el cual permite realizar operaciones sobre una base de datos, pero para ello se debe de obtener el objeto que esta referenciando a nuestra base de datos definiendo el tipo de acceso que se desea realizar, en el cual los metodos encargados son getReadablaDatabase() en el cual devuelve el objeto SQLiteDatabase de nuestra base de datos de tal manera que solo pueda realizar operaciones de lectura sin alterarla, en el caso de getWritableDatabase() este devuelve el mismo objeto de tal manera que permite la escritura sobre la base de datos referenciada.

Ahora para poder registrar en nuestra base de datos podemos realizar la siguiente implementación en nuestra clase:

|  |
| --- |
| public boolean insertContact (Contact contact) {  boolean insert = false;  ContentValues insertValues = new ContentValues();  insertValues.put("us\_id", contact.getId());  insertValues.put("us\_name", contact.getName());  insertValues.put("us\_lastname", contact.getLastname());  insertValues.put("us\_email", contact.getEmail());  insertValues.put("us\_number\_phone",contact.getNumberPhone());  long inserted = sqLiteDatabase.insert("user",null, insertValues);  if (inserted > 0)  insert = true;   return insert; } |

En este método insert() nos apoyamos en el objeto ContentValues el cual se encarga de almacenar los valores registrado mediante su llave y su valor correspondiente, despues se utiliza el objeto SQLiteDatabase a través de su método insert, el cual recibe tres parametros, el primero es el nombre de la tabla, el segundo es en el caso de registrar campos vacios, en el cual lo establecemos como null para omitirlo, y el tercer parametro son los valores guardados en el contenedor para guardarlos.

Este método devuelve el id del registro guardado si fue un exito, en caso contrario devuelve 0 o un número negativo en caso de un error al momento de la inserción.

El siguiente método que se implementa en la clase, muestra como se elimina un contacto en la base de datos:

|  |
| --- |
| public boolean deleteContact (int id) {  String[] args = new String[]{Integer.toString(id)};  long deleted = sqLiteDatabase.delete("user","us\_id=?", args);  if (deleted > 0)  return true;  else  return false; } |

Este método delete() es muy similar al de registrar, debido a que requiere tres parametros, el primero es la tabla, el segundo es la sentencia de validación para eliminar el registro, en la cual es una cadena y la principal caracteristica es que para especificar el valor de comparación se usa el caracter ‘?’ por cada uno, y estos valores se deben de establecer en un arreglo de cadenas de caracteres, la cual se establece en el tercer parametro, de acuerdo al orden que se guardaron los valores va reemplazando el caracter comodin.

Ahora a continuación se muestra el método que se encarga de actualizar un registro, es el siguiente:

|  |
| --- |
| public boolean updateContact(Contact contact){  boolean updated=false;  String[] args = new String[]{Integer.toString(contact.getId())};  ContentValues updatedValues = new ContentValues();  updatedValues.put("us\_id", contact.getId());  updatedValues.put("us\_name", contact.getName());  updatedValues.put("us\_lastname", contact.getLastname());  updatedValues.put("us\_email", contact.getEmail());  updatedValues.put("us\_number\_phone",contact.getNumberPhone());  long update=sqLiteDatabase.update("user",updatedValues,"us\_id=?",args);  if(update>0)  updated=true;   return updated; } |

Este método update() es una combinación de los anteriores, puesto que tambien se utiliza un objeto ContentValues para guardar los valores mediante llave y valor, para actualizar los campos, en los siguientes parametros son los mismos que en el método delete.

Como pueden ver estos métodos nos permiten implementar las operaciones básicas en nuestra base de datos y poder administrar la información.

## **Recuperar información desde la base de datos**

Una de las principales operaciones en una base de datos es recuperar la información a traves de sentencias de consultas a nuestras tablas de información y llevar a cabo una combinación de la información. En el caso de SQLite es posible realizar consultas a través de las tablas, como se vera en el siguiente código:

|  |
| --- |
| public List<Contact> getContacts(){  List<Contact> contacts=null;  String[] fields=new String[]{  "us\_id",  "us\_name",  "us\_lastname",  "us\_email",  "us\_number\_phone"  };  Cursor cursor = sqLiteDatabase.query("user",fields,null,null,null,null,null);  if(cursor.moveToFirst()){  contacts=new LinkedList<>();  do{  int id=cursor.getInt(cursor.getColumnIndex("us\_id"));  String name=cursor.getString(cursor.getColumnIndex("us\_name"));  String lastname=cursor.getString(cursor.getColumnIndex("us\_lastname"));  String email=cursor.getString(cursor.getColumnIndex("us\_email"));  int numberPhone=cursor.getInt(cursor.getColumnIndex("us\_number\_phone"));  contacts.add(new Contact(id,name,lastname,numberPhone,email));  }while (cursor.moveToNext());  }  cursor.close();  return contacts; } |

Este método permite realizar una consulta a la tabla de usuarios mediante el método query() se puede realizar consultas, este método devuelve un objeto Cursor en el cual en caso de que la consulta se realizo exitosamente nos va permitir obtener el resultado de ella.

Este método requiere los siguientes atributos:

* Nombre de la tabla
* Arreglo de cadenas de los nombres de los campos que se van a consultar
* Sentencia de selección para la consulta mediante el comodin de valor(Opcional)
* Arreglo de valores de argumentos de la sentencia de selección(Opcional)
* Nombre del campo que se desea agrupar la consulta(Opcional)
* Sentencia de referencia para la consulta(Opcional)
* Nombre del campo para ordenar la consulta(Opcional)

En caso de no requerir algún atributo, solo basta especificarlo como null para que el metodo pueda funcionar, una vez que se obtiene el cursor este tiene metodos con los cuales permite recorrer cada uno de los registros, puesto que su funcionalidad es muy similar a un puntero. Con el método moveToFirst() se asegura que se posicione al primer elemento de la colección de registros, si fue satisfactorio con apoyo del ciclo do-while recorremos cada una de las posiciones de la colección, siempre y cuando el metodo moveToNext() devuelva un valor verdadero.

Para acceder a los valores de cada campo de los registros, se utilizan métodos que se encuentran en el cursor para leer los valores como su tipo de dato correspondiente, por ejemplo el metodo getInt() lee un campo de tipo entero, se le debe indicar el indice del campo que se desea leer, para ello nos apoyamos en el metodo getColumnIndex() en el cual se indica el nombre del campo, para que devuelva el indice correspondiente.

Con estos metodos básicos podremos obtener la información de una tabla de bases de datos mediante consultas y que sea de manera segura, accesible y ordenada en cualquier momento.

## **Manejo de preferencias**

En android existe la oportunidad de almacenar información en un archivo xml, el cual es llamado “SharedPreferences”, este archivo permite almacenar información mediante la utilización de llaves y valores asociados. Este tipo de archivo es muy utilizado para resguardar configuraciones de la app o valores no sensibles para el usuario debido a que la información es guardada secuencialmente y es posible acceder a la información que se almacena.

Este tipo de archivo tienen tres modos de acceder a él, los cuales son los siguientes:

* MODE\_PRIVATE: Solo la aplicación puede tener acceso a él.
* MODE\_WORLD\_READABLE: Todas las aplicaciones pueden tener acceso a la información, pero solo nuestra aplicación puede modificarla.
* MODE\_WORLD\_WRITABLE: Todas las aplicaciones puende acceder a la información y editarla.

De acuerdo a estos modos, se puede ver claramente que los dos ultimos podemos llevar un riesgo de seguridad en la información es por ello que se debe tener un cuidado al respecto, es por ello que a partir de android 4.2 (API 17) fueron despreciados estos modos de acceso.

Para definir este archivo debemos de realizar lo siguiente en nuestra actividad:

|  |
| --- |
| SharedPreferences settings = getSharedPreferences("Settings", Context.MODE\_PRIVATE); |

A través del metodo getSharedPreferences se crea o se obtiene el archivo xml de nuestras preferencias, el primer parametro es el nombre de tal archivo y el segundo es el modo de acceso a él.

A continuación, se muestra como se debe de obtener un valor a través de este archivo:

|  |
| --- |
| int id=settings.getInt("id",0); String name=settings.getString("name",null); boolean isLong=settings.getBoolean("long",false); |

Como se puede ver a través del objeto que se genera de nuestro SharedPreferences podemos obtener los valores guardados, los cuales solo se aceptan los siguientes tipos:

* Números Enteros (Int)
* Números Flotantes (Float)
* Números Largos (Long)
* Valores boleanos (Boolean)
* Cadenas de caracteres (String)
* Colección de cadenas de caracteres (Set<String>)

Estos son los unicos valores permitidos dentro del archivo de preferencias, y cada metodo se le debe indicar la llave del valor registrado y un segundo valor en el cual si el metodo no encuentra la llave se devolvera este parametro.

Ahora para poder almacenar información sobre nuestro archivo de preferencias debemos de realizar lo siguiente:

|  |
| --- |
| SharedPreferences.Editor editor=settings.edit(); editor.putInt("id",1); editor.putString("name","Juan"); editor.putBoolean("long",true); |

A través de un objeto Editor que pertenece a la clase SharedPreferences nos va permitir guardar información en nuestro archivo, este objeto se obtiene a través del método getEdit(), y a través de los metodos put de cada tipo de dato se especifica la llave y valor que se va almacenar dentro del archivo.

# **DISTRIBUCIÓN DE UNA APLICACIÓN**

## **Consideraciones antes de la versión final de la aplicación**

El proceso final para poder publicar una aplicación, es recomendable tener las siguientes consideraciones:

* Definir un identificador de paquete para nuestras aplicaciones
* Definir un código de versionado
* Depurar el proyecto de código o librerias que no se utiliza en la app
* Los iconos o imagenes que se deben utilizar contar con cada una en sus diferentes escalas y medidas
* Establecer la compatibilidad de versiones de Android
* Generar la toma de pantallas de la aplicacion en un celular o tablet si es que se requieren
* Definir la clasificación de la aplicación
* Definir la categoria de la aplicación

Estas son consideraciones para que agilizar el proceso al publicar una aplicación.

## **Estableciendo la versión de la aplicación**

Para establecer una versión de una aplicación son dos campos que se deben de utilizar, el primero de ellos es el nombre de la version(versionName) este consta de un codigo que se define, por lo regular empieza con tres o dos digitos separados por un punto, por ejemplo 1.0 o 1.0.0, esto es a consideración de uno mismo.

El código de versión es un campo importante al momento de publicar la aplicación, el cual se consta de un valor númerico entero que permite a la tienda corroborar un cambio en la versión y permitir que la aplicación se actualice o se publique depende cual sea el caso.

## **Firmando la aplicación**

En el proceso de una publicación de una aplicación en la tienda, se debe realizar un proceso de firma en ella, este es asociar una llave encriptada mediante la cual la tienda verificara la información dentro de ella para autorizar la publicación.

Este proceso requiere generar un certificado con cierta información que va encriptar dentro de la llave, este proceso se puede llevar a cabo desde Android Studio, en el cual nos podemos apoyar desde la documentación en Android Developers en el tema firmando tu app.