Navigation Drawer

Este componente nos permitirá generar un contenido que aparece desde la izquierda de la pantalla mediante un gesto de arrastre desde el borde izquierdo hacia la derecha o presionando el botón Home en la ToolBar. Generalmente el contenido que se muestra es una lista de opciones creada con un archivo XML de menú.

Para utilizar este componente, deberemos agregar un “DrawerLayout” en el archivo xml de layout de la Activity donde queremos que exista el drawer. Ejemplo:

```
<android.support.v4.widget.DrawerLayout
      android:id="@+id/drawer_layout"
      android:background="#FF000000"
      android:layout_width="match_parent"
      android:layout_height="match_parent">

      <!-- The main content view -->
      <LinearLayout
            android:id="@+id/contenedor"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical" >
            <ImageView
                android:layout_width="match_parent"
                android:src="@drawable/sistema_solar"
                android:layout_height="wrap_content" />
        </LinearLayout>

        <!-- Menú Deslizante -->
        <android.support.design.widget.NavigationView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/navigation_view"
            android:layout_gravity="start"
            android:fitsSystemWindows="true"
            app:headerLayout="@layout/nav_header"
            app:menu="@menu/nav_menu" />

    </android.support.v4.widget.DrawerLayout>
```

Contenido de la Activity.

Menú que aparece a la izquierda.

En este caso, el contenido de la pantalla es un LinearLayout con un ImageView dentro, este Layout (con id “contenedor”) y el DrawerLayout, deben tener las propiedades “match_parent” tanto en el alto como en el ancho.
Por otro lado, debajo, y siempre dentro del tag “DrawerLayout”, tenemos la View que se mostrará a la izquierda de la pantalla cuando el usuario haga el gesto. En este ejemplo, se utilizó una NavigationView provista por la biblioteca design support.

NavigationView

Esta View está especialmente diseñada para componer el menú lateral del Navigation Drawer. Los dos atributos principales de esta View son:

- **app:headerLayout** Aquí indicamos un archivo XML de layout que se colocará en el header de la barra lateral.
- **app:menu** Aquí indicamos un archivo XML de menú mediante el cual se mostrarán los ítems en la barra lateral.

Para utilizar esta View, deberemos incluir la biblioteca design support en el archivo build.gradle de nuestro módulo app:

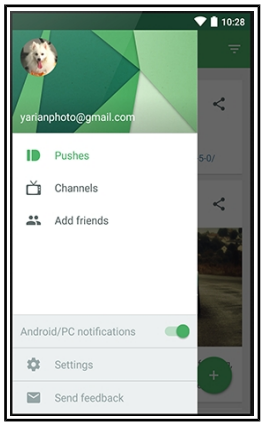
```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.android.support:design:22.2.1'
}
```

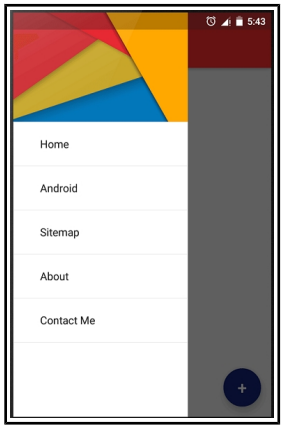
Creando el Header

Crearemos un archivo XML de layout llamado nav_header.xml en el cual pondremos como header una imagen:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <ImageView
        android:layout_width="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/sistema_solar"
        android:layout_height="100dp" />
</LinearLayout>
```

Algunos headers típicos para Navigation Drawer:





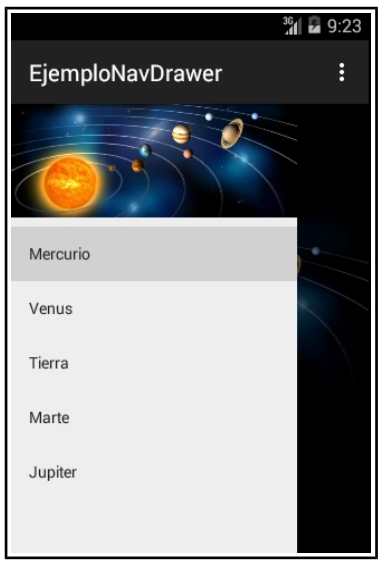
Creando los items

Para crear los ítems que formarán la lista, deberemos crear un archivo XML de menú. Cada ítem podrá contener un ícono si se desea, al igual que un ítem de menú.
Creamos el archivo nav_menu.xml con el siguiente contenido:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group
        android:checkableBehavior="single">
        <item
            android:id="@+id/drawer_item_mercurio"
            android:checked="true"
            android:title="Mercurio"/>
        <item
            android:id="@+id/drawer_item_venus"
            android:title="Venus"/>
        <item
            android:id="@+id/drawer_item_tierra"
            android:title="Tierra"/>
        <item
            android:id="@+id/drawer_item_marte"
            android:title="Marte"/>
        <item
            android:id="@+id/drawer_item_jupiter"
            android:title="Jupiter"/>
    </group>
</menu>
```

Agregando botón menú para lanzar el drawer

Si ejecutamos este ejemplo, notaremos que el drawer aparece al realizar el gesto, pero no presionando el botón en la ToolBar, ya que para ésto necesitaremos utilizar un objeto del tipo "ActionBarDrawerToogle"



Agreguemos en el onCreate de nuestra Activity, el siguiente código:

```
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
mDrawerToggle = new ActionBarDrawerToggle(
                this,
                mDrawerLayout,
                R.string.drawer_open,
                R.string.drawer_close
                );
mDrawerLayout.setDrawerListener(mDrawerToggle);
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
getSupportActionBar().setHomeButtonEnabled(true);
```

En este programa, obtenemos el DrawerLayout que está en pantalla y le seteamos un “DrawerListener” mediante el método “setDrawerListener”. También habilitamos el uso del botón Home de la Action Bar mediante “setDisplayHomeAsUpEnabled” y “setHomeButtonEnabled”

La porción más importante de este ejemplo es la creación del objeto ActionBarDrawerToogle, este objeto es el que le pasamos al DrawerLayout como listener.

El primer argumento en el contructor de este objeto es la Activity que contine al Drawer, luego el objeto DrawerLayout que está en pantalla, y por último el constructor recibe 2 recursos del tipo string, que es la descripción del drawer cuando está abierto y cerrado.

Ahora deberemos capturar el evento de click sobre el botón home, para ello en nuestra Activity sobreescribimos el método “onOptionsItemSelected” (el mismo evento que para los ítems de menú)

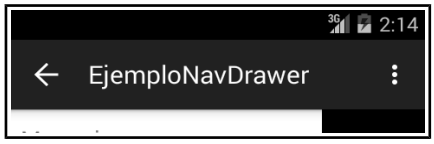
```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (mDrawerToggle.onOptionsItemSelected(item)) {
          return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

Al recibir el evento, llamamos al método “onOptionsItemSelected del objeto “ActionBarDrawerToogle” el cual devolverá true si el evento era el botón de home. De esta manera el drawer aparecerá mediante una animación, al presionarse el botón.

También deberemos sobreescribir dos métodos más en la Activity para notificar cambios de estado de la Activity, al objeto “ActionBarDrawerToogle” :

```
    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);
        // Sync the toggle state after onRestoreInstanceState has occurred.
        mDrawerToggle.syncState();
    }
    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        mDrawerToggle.onConfigurationChanged(newConfig);
    }
```

De esta manera, ahora aparecerá habilitado el botón Home.



Selección de ítems

Para detectar el ítem de la lista que se seleccionó, utilizaremos un listener sobre el NavigationView. En este ejemplo, haremos que el listener sea nuestra Activity:

```
navigationView = (NavigationView) findViewById(R.id.navigation_view);
navigationView.setNavigationItemSelectedListener(this);
```

En la Activity, deberemos implementar la interface OnNavigationItemSelectedListener, la cual nos obligará a implementar el método onNavigationItemSelected en el cuál podremos detectar qué ítem fue seleccionado:

```
@Override
public boolean onNavigationItemSelected(MenuItem menuItem) {
    menuItem.setChecked(true);
    switch(menuItem.getItemId()){
        case R.id.drawer_item_mercurio:
            Toast.makeText(this,"Mercurio",Toast.LENGTH_SHORT).show();
            break;
        //...
  mDrawerLayout.closeDrawers();
```

Mediante setChecked, hacemos que el ítem quede selecionado en la lista.

Cambiar contenido de la pantalla principal según opción del drawer: Fragments

Muchas veces necesitamos cambiar completamente el contenido de la pantalla de nuestra Activity según la opción que el usuario presiona en nuestro menú generado con drawer.

Para ello, utilizaremos Fragments, definiremos una clase que herede de Fragment y un archivo xml de layout por cada contenido que queramos mostrar, cada Fragment hará un inflate su archivo xml de layout. Recordemos que un Fragment puede pensarse como una Activity, con la característica de que la Activity que lo contiene, lo puede mostrar y ocultar bajo cierta lógica.

Definimos 2 archivos de layout para cada **tipo de contenido** que queremos mostrar en la pantalla principal:

layout 1

layout2

Como se observa en las figuras, en el archivo layout_contenido1.xml, colocamos un título, un texto, y luego una imagen:

```
<?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:background="#FF000000"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/titulo_mercurio"
        android:id="@+id/txtTitulo"
        android:textColor="#FFFFFFFF"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:layout_margin="20dp"
        android:layout_gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/info_mercurio"
        android:textColor="#FFFFFFFF"
        android:id="@+id/txtInfo" />

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/imgPlaneta"
        android:src="@drawable/mercurio"
        android:layout_gravity="center_horizontal" />
</LinearLayout>
```

Mientras que en el archivo layout_contenido2.xml, colocamos primero el título, luego la imagen y por último el texto:

```
<?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:background="#FF000000"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/titulo_mercurio"
        android:id="@+id/txtTitulo"
        android:textColor="#FFFFFFFF"
        android:layout_gravity="center_horizontal" />

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:id="@+id/imgPlaneta"
        android:src="@drawable/mercurio"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:layout_margin="20dp"
        android:layout_gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="@string/info_mercurio"
        android:textColor="#FFFFFFFF"
        android:id="@+id/txtInfo" />
</LinearLayout>
```

Utilizaremos los dos layouts como “modelos” para cargarlos con la información de cada planeta según corresponda. Esta lógica la resolveremos en dos clases que serán los Fragments que muestren la información utilizando un layout y el otro respectivamente.

Definimos dos clases: “Contenido1” y “Contenido2” ambas deben heredar de fragment y reescribir el método “onCreateView” donde se hace un inflate del archivo xml de layout correspondiente y se devuelve la view obtenida. Puede pensarse en este método como el “onCreate” de la Activity.

Ejemplo de clase “Contenido1”

```
public class Contenido1 extends Fragment{

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View v = inflater.inflate(R.layout.layout_contenido1, container,false);

        ImageView img = (ImageView)v.findViewById(R.id.imgPlaneta);
        TextView txtTitulo = (TextView)v.findViewById(R.id.txtTitulo);
        TextView txtInfo = (TextView)v.findViewById(R.id.txtInfo);

        Bundle args = getArguments();
        int idPlaneta = args.getInt("idPlaneta");
        switch(idPlaneta)
        {
            case 0:
                img.setImageResource(R.drawable.mercurio);
                txtTitulo.setText(getText(R.string.titulo_mercurio));
                txtInfo.setText(getText(R.string.info_mercurio));
                break;
            case 1:
                img.setImageResource(R.drawable.venus);
                txtTitulo.setText(getText(R.string.titulo_venus));
                txtInfo.setText(getText(R.string.info_venus));
                break;
            case 2:
                img.setImageResource(R.drawable.tierra);
                txtTitulo.setText(getText(R.string.titulo_tierra));
                txtInfo.setText(getText(R.string.info_tierra));
                break;
            case 3:
                img.setImageResource(R.drawable.marte);
                txtTitulo.setText(getText(R.string.titulo_marte));
                txtInfo.setText(getText(R.string.info_marte));
                break;
            case 4:
                img.setImageResource(R.drawable.jupiter);
                txtTitulo.setText(getText(R.string.titulo_jupiter));
                txtInfo.setText(getText(R.string.info_jupiter));
                break;
        }

        return v;
    }
}
```

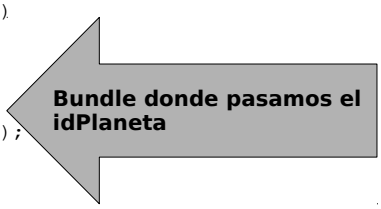
Es importante remarcar el uso del método getArguments el cual devuelve un objeto Bundle, que adentro contiene, con la clave “idPlaneta” un número que le indica al fragment qué información mostrar. Este Bundle lo generaremos en la Activity, en el evento de click sobre un item de la lista, al generar el Fragment y lanzarlo.

Para la clase “Contenido2” se hace lo mismo y se reemplaza “layout_contenido1” por “layout_contenido2”.

En nuestra Activity, no debemos cambiar demasiado con respecto al ejemplo anterior, solamente debemos llamar al método setContenido cuando se presiona un ítem, en el lugar donde mostrabamos un Toast.

Por último escribimos el método “setContenido” en nuestra Activity:

```
private void setContenido(int numeroContenido)
{
    Fragment fragment;

    Bundle bundle = new Bundle();
    bundle.putInt("idPlaneta",numeroContenido);

    if(numeroContenido%2==0)
        fragment = new Contenido1();
    else
        fragment = new Contenido2();

    fragment.setArguments(bundle);

    FragmentManager fragmentManager = getSupportFragmentManager();
    fragmentManager.beginTransaction()
            .replace(R.id.contenedor, fragment)
            .commit();

}
```



Como se observa en el código, según el argumento se genera el Fragment Contenido1 o Contenido2 (según si es par o impar), y luego mediante el objeto FragmentManager, se hace un “replace” del fragment contenido en la view “R.id.contenedor” por el que creamos.

Si observamos el archivo xml de layout de la Activity, el contenedor es el LinearLayout utilizado como contenedor de lo que se muestra en la pantalla, el cual deberá estar vacío, de modo que debemos comentar el ImageView que habíamos colocado con anterioridad:

```
<!-- The main content view -->
<LinearLayout
    android:id="@+id/contenedor"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <!-- <ImageView
        android:layout_width="match_parent"
        android:src="@drawable/sistema_solar"
        android:layout_height="wrap_content" /> -->

</LinearLayout>
```

Se aclara que de esta forma, el fragment se esta creando nuevamente cada vez que el usuario elige una opción de la ListView, se podrían tratar dichos fragments como singletons para reutilizar las mismas instancias ya generadas.

Conclusiones

En este ejemplo se pretende demostrar que cada contenido, que se muestra en pantalla al hacer click sobre un ítem de la lista del Navigation Drawer, puede ser un fragment diferente, o puede ser el mismo fragment, así como también que un mismo fragment puede trabajar con muchos archivos de layout, decidiendo cuál se carga, o cada fragment puede trabajar con un archivo de layout propio para ese fragment.

La anterior aclaración surge de la posibilidad de resolver el mismo ejemplo utilizando una sola clase “Contenido” que herede de fragment, y que seleccione qué layout cargar según el argumento idPlaneta.

La idea principal de un fragment es que tenga un solo archivo de layout (en la mayoría de los casos, los fragments serán completamente diferentes entre sí) y que maneje la lógica de la representación en dicho layout.