Licenciatura en Sistemas - Taller de Aplicaciones Móviles – 2022

Bibliografía: https://developer.android.com/

Equipo docente: Lic. Romina Mansilla

Unidad 2

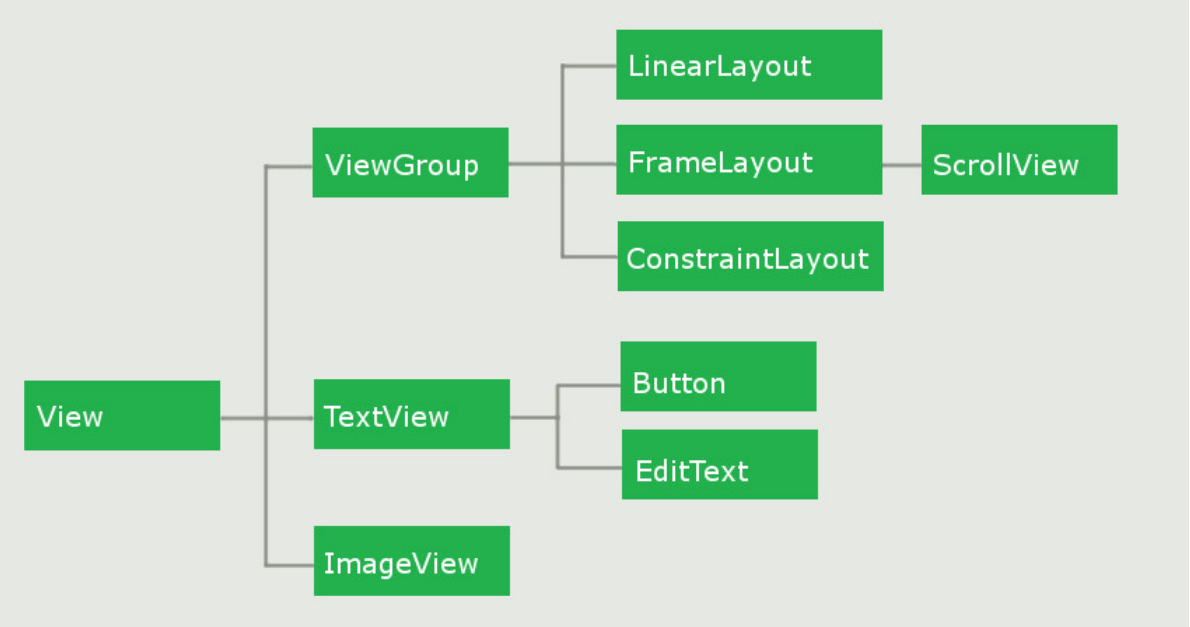
Views y ViewGroups

En android, todos los elementos visibles en la pantalla son vistas que son hijas de la clase View.

Esto hace que se compartan muchas propiedades, como por ejemplo, que todas las vistas tengan un width, height y un background. Dentro de ellas se pueden encontrar: TextView, Button, ImageView, EditText, Checkbox, RadioButton y muchas más.

Las Views que contienen otras Views se llaman ViewGroups, y estas, a su vez, pueden contener otras ViewGroups.

Jerarquía:



Width y Height

Cada View tiene dimensiones de alto (height) y ancho (width) que se deben definir de manera obligatoria. Para esto hay 3 maneras de definirlo:

- **Match_parent**: ajusta su tamaño lo máximo que puede hasta llegar a la vista padre.
- **Wrap_content**: ajusta su tamaño al propio, es decir, al de su contenido.
- **"xx"dp**: se puede definir una medida en particular. En Android no se utiliza la unidad de medida "px" (pixel), sino que se utiliza "dp" (densidad independiente de pixel).

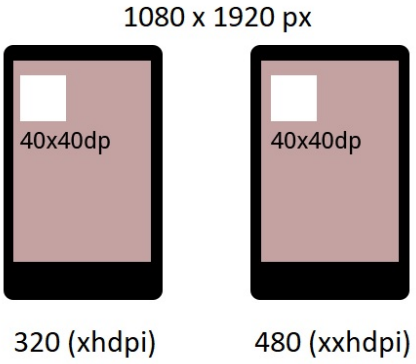
Densidad de pantalla

Es la cantidad de píxeles que se encuentran en un área específica.

¿Por qué utilizar DP y no PX?

Si se utilizara la medida “px” para definir las vistas, en pantallas del mismo tamaño pero con distinta resolución se verían diferentes. Esto es debido a que existen pantallas de 5” pulgadas que poseen una densidad de, por ejemplo, “XHDPI” y “XXHDPI".

Utilizar la medida “dp” permite que la vista se maneje de manera responsive. Esto quiere decir que el tamaño de la vista definida no cambiará en distintos dispositivos con distintas resoluciones. Internamente se realiza una operación que permite escalar la medida definida de la manera correcta.

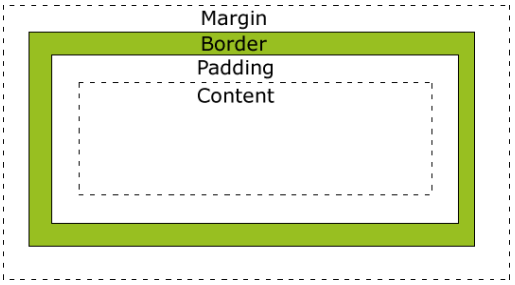


¿Qué utilizar para definir el tamaño del texto?

La medida que hay que utilizar es “sp” (pixel escalable-scalable pixel). Esto permite que el tamaño del texto se redimensione respecto a las preferencias que haya definido el usuario en la configuración del dispositivo.

Más información: Densidades de pantalla

Margin y Padding



A las vistas también se les puede definir un **margin** y un **padding**, ambos se definirán con la unidad de medida "dp".

El **margin** generará un espacio entre nuestra vista con respecto a lo que lo rodea.

El **padding** hará que nuestra vista se expanda según lo definido.

Atributos

Ambos atributos tienen la posibilidad de que se definan para un sólo lado de la vista, como también para todos a la vez:

- ***android:layout_margin*** = 16dp. Aplica para toda la vista.
- ***android:layout_marginTop*** = 16dp. Aplica para la parte de arriba de la vista.
- ***android:layout_marginBottom*** = 16dp. Aplica para la parte de abajo de la vista.
- ***android:layout_marginStart*** = 16dp. Aplica para la parte de la izquierda de la vista.
- ***android:layout_marginEnd*** = 16dp. Aplica para la parte de la derecha de la vista.

Para definir el paddíng se hace de la misma manera, solo que cambia la palabra "margin" por "padding" y no se utiliza el prefijo “layout”.

Gravity y Layout Gravity

Son atributos que tienen las vistas.

Gravity: indica de qué manera se va a disponer su contenido.

Layout Gravity: indica de qué manera se va a disponer la vista propia con respecto a la vista padre.

Ambos atributos comparten las mismas maneras de disposición, dentro de las cuales las más utilizadas son:

- Top
- Start (izquierda)
- End (derecha)
- Bottom
- Center
- Center_vertical
- Center_horizontal

Ejemplo Gravity

En el siguiente caso se puede visualizar que el ViewGroup contiene el atributo “gravity” = “center_horizontal”, ya que las vistas hijas están centradas de manera horizontal.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:padding="20dp"
    tools:context=".MainActivity">
```

Ejemplo Layout_gravity

En este caso es puede ver que el ViewGroup contiene el atributo “gravity” = “center_horizontal”, y que un TextView contiene el atributo “layout_gravity” = ”start”, haciendo que esta vista esté alineada a la izquierda.

```
<TextView
    android:layout_gravity="start"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ingresar usuario" />
```

Linear Layout

El LinearLayout es un ViewGroup donde se define la orientación (vertical u horizontal) sobre la cual serán dispuestas las Views hijas. Si este atributo no se define, las vistas se dispondrán de manera horizontal. Aunque no muestre un error de compilación al no definirlo, es buena práctica agregarlo.

WeightSum

Es un atributo que permite manejar el tamaño que tendrán las vistas hijas del LinearLayout a través de la utilización de porcentajes. Para ello se definirá el valor total de la suma del peso de las vistas hijas dentro de este atributo.

Para definir el valor que ocupará cada vista hija se utilizará el atributo “layout_weight”.

Algo que se debe tener en cuenta es que la vista al estar siendo dimensionada mediante un porcentaje y no por un "wrap_content", "match_parent" o un valor específico, se recomienda definir el atributo (width o height) en "0dp". El atributo que se definirá con este valor, dependerá de la orientación del LinearLayout. Si es Horizontal, el "layout_width" deberá ser igual a "0dp", si es vertical, el "layout_height" deberá ser igual a "0dp".

Ejemplo utilización del Weight

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:orientation="horizontal"
    android:weightSum="1">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight=".5"
        android:text="Iniciar sesion" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight=".5"
        android:text="Crear cuenta" />

</LinearLayout>
```

Relative Layout

El RelativeLayout es un ViewGroup que dispone a sus vistas hijas en posiciones relativas. La posición de cada vista se puede especificar de manera relativa a sus vistas hermanas o en posiciones relativas al padre.

Actualmente este ViewGroup está deprecado, por lo que se recomienda que no se utilice. Se puede encontrar en la pestaña de "Design", dentro de las vistas "Legacy".

Edit Text

El EditText es una View que extiende de TextView cuya funcionalidad es la de permitir el ingreso de texto, números, y fechas mediante el teclado. Uno de los atributos más útiles que posee es el del inputType, el cual permite definir el tipo de teclado que se mostrará al posicionarse sobre esta vista. Los valores más comunes que se definen para este parámetro son:

- text
- number
- numberDecimal
- datetime
- textPassword
- textEmailAddress

Otro atributo que se suele utilizar es el del hint. El mismo permite informarle al usuario mediante una descripción de lo que debe ingresar. Este texto se posicionará dentro del EditText y se verá únicamente si este componente no posee ningún texto.

Esta vista, al heredar de TextView, puede utilizar los métodos de getText() para obtener el texto ingresado, y setText("texto") para definir uno precargado en el componente.

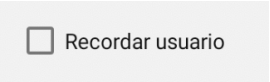
```
etUsuario.setText("Romina");
```

```
String usuario = etUsuario.getText().toString();
```

CheckBox

Es una View que permite tildar un casillero que resulta en valores de True ó False. Se suele utilizar cuando se desea que el usuario tome decisión acerca de algo en particular, como por ejemplo la opción de recordar una contraseña.

Para verificar si el casillero se encuentra tildado se utiliza el método isChecked() devolviendo un Boolean.



ScrollView.

Es un ViewGroup que contiene 1 solo hijo y permite realizar la acción de scrolling en la pantalla. Para agregar más vistas se debe agregar otro ViewGroup dentro del ScrollView (como un LinearLayout). Si lo que se necesita es scrollear horizontalmente, se debe usar la ViewGroup "HorizontalScrollView". El ScrollView posee un atributo muy importante que se llama "fillViewPort", el cual se le debe definir un booleano (true o false). Si a este atributo se lo define en "true", hará que el ScrollView amplíe su contenido (su vista hija) para ocupar toda la pantalla.

El ejemplo más común donde se utiliza el ScrollView es cuando nuestra pantalla se queda sin espacio para seguir agregando vistas. Otro ejemplo común es al momento de mostrar los términos y condiciones.

Framelayout.

Es un ViewGroup en el cual sus hijos se disponen mediante el atributo "layout_gravity". El FrameLayout hace que el eje Z tenga efecto en sus vistas hijas, es decir que las vistas se superpongan.

Se recomienda que este ViewGroup se utilice con una sola vista hija, ya que puede llegar a complejizar la organización de las vistas hijas en distintos tipos de pantallas debido a que estas podrían superponerse entre sí. Que una vista se encuentre sobre otra dependerá de cuál se agregó primero.

RecyclerView.

Es un ViewGroup que dispone una lista de vistas (ítems) scrolleables en modo: Lista vertical/horizontal ó Grilla.

Existe un componente que se llama ListView y otro GridView, los cuales se utilizaban previo al RecyclerView. Estos quedaron deprecados debido a que el RecyclerView los reemplazó gracias a su abanico de funcionalidades y facilidades que provee tales como:

- Reciclado de vistas
- Modos de disposición configurables de manera ágil y dinámica
- Animaciones
- Separadores de elementos
- Coordinación con otros elementos como el CoordinatorLayout

Disposición de vistas

Para elegir el modo a disponer las vistas utiliza el “LayoutManager”, el cual puede ser uno de los siguientes tipos:

- **LinearLayoutManager**: se debe definir si su orientación será vertical u horizontal.
- **GridLayoutManager**: se debe definir la cantidad de columnas que poseerá.

Estos modos se pueden definir tanto en el XML como en la clase:

XML:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rvExamenes"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"/>
```

Clase: el primer parámetro del LinearLayoutManager hace referencia al Context, el segundo a la orientación, y el tercero a la disposición de las vistas de manera ascendente/descendente.

```
RecyclerView.LayoutManager layoutManager = new LinearLayoutManager( context: this,RecyclerView.VERTICAL,  reverseLayout: false);
rvExamenes.setLayoutManayer(layoutManager);
```

View Holder y Adapter

Las vistas en la lista son representadas por objetos del tipo ViewHolder. Estas son instancias de la clase a crear que extiende a RecyclerView.ViewHolder. Es decir que cada view holder se encarga de mostrar cada ítem de la vista.

Los objetos del tipo ViewHolder son administrados por un adapter que se crea extendiendo a RecyclerView.Adapter. Esta clase será la encargada de crear los view holder, y de vincular la vista con la información a mostrar. A cada view holder se le asigna una posición para identificarlo. A su vez, al extender la clase RecyclerView.Adapter se le debe pasar un tipo que extienda a RecyclerView.ViewHolder, para esto se deberá crear una clase que extienda lo mencionado para luego poder pasarle al RecyclerView.Adapter el tipo correspondiente. Esto obligará a que la clase implemente 3 métodos que se verán a continuación.

Ejemplo: se hace referencia a ExamenAdapter.ExamenViewHolder debido a que la clase ExamenViewHolder se encuentra dentro de ExamenAdapter.

```
public class ExamenAdapter extends RecyclerView.Adapter<ExamenAdapter.ExamenViewHolder> {
```

Ejemplo ViewHolder: se debe crear un constructor haciendo referencia a “super” pasando una vista por parámetro que será el xml del ítem. A su vez en esta clase se deberán crear las instancias de la vista para poder vincularlas mediante el findViewById y luego poder manipularlas..

```
class ExamenViewHolder extends RecyclerView.ViewHolder {
    TextView txtMateria;
    TextView txtFecha;

    ExamenViewHolder(@NonNull View itemView) {
        super(itemView);
        txtMateria = itemView.findViewById(R.id.txtMateria);
        txtFecha = itemView.findViewById(R.id.txtFecha);
    }
}
```

El primer método que se sobreescribe llamado “onCreateViewHolder(ViewGroup, Int)” se utiliza para crear los viewholders. Este debe devolver un objeto del tipo RecyclerView.ViewHolder. Para esto, se usará la ayuda del LayoutInflater que permitirá “inflar” la vista (el xml) deseado, es decir, el ítem. El concepto inflar en Android se utiliza para crear una instancia del tipo View a partir de un XML y un contexto.

Ejemplo: retorna una instancia del tipo ExamenViewHolder, en donde por parámetro le pasa la vista inflada.

```
@NonNull
@Override
public ExamenViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View itemExamen = LayoutInflater
            .from(parent.getContext())
            .inflate(R.layout.item_examen,parent,  attachToRoot: false);
    return new ExamenViewHolder(itemExamen);
}
```

El segundo método que se sobreescribe, onBindViewHolder(ViewHolder, Int), permitirá manipular el contenido de cada ítem de la lista. Por parámetro vendrán dos objetos: uno es el viewholder, que dejará acceder las vistas del mismo (como TextView, ImageView, etc), y el otro es la posición que permitirá obtener la información, es decir, si se posee una lista de examenes (List<Examenes>), se accederá a la posición que viene por parámetro para obtener la información necesaria (examenes.get(position)).

```
@Override
public void onBindViewHolder(@NonNull ExamenViewHolder holder, int position) {
    holder.txtMateria.setText(examenes.get(position).getMateria());
    holder.txtFecha.setText(examenes.get(position).getFecha());
}
```

El último método a sobreescribir es el getItemCount(), que será el encargado de devolver la el total de la cantidad de ítems que posee nuestra lista.

```
@Override
public int getItemCount() {
    return examenes.size();
}
```

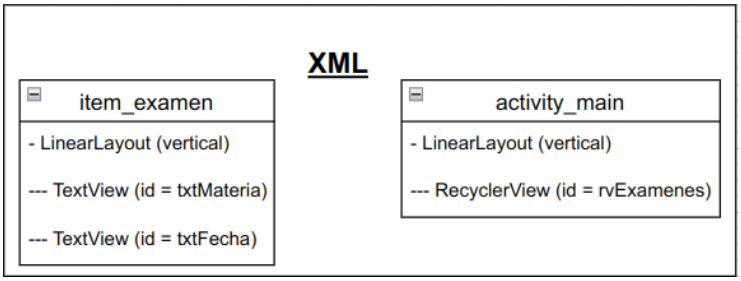
Finalmente, desde la Activity se deberá realizar la vinculación del objeto con la vista para luego crear el Adapter y asignárselo al RecyclerView:

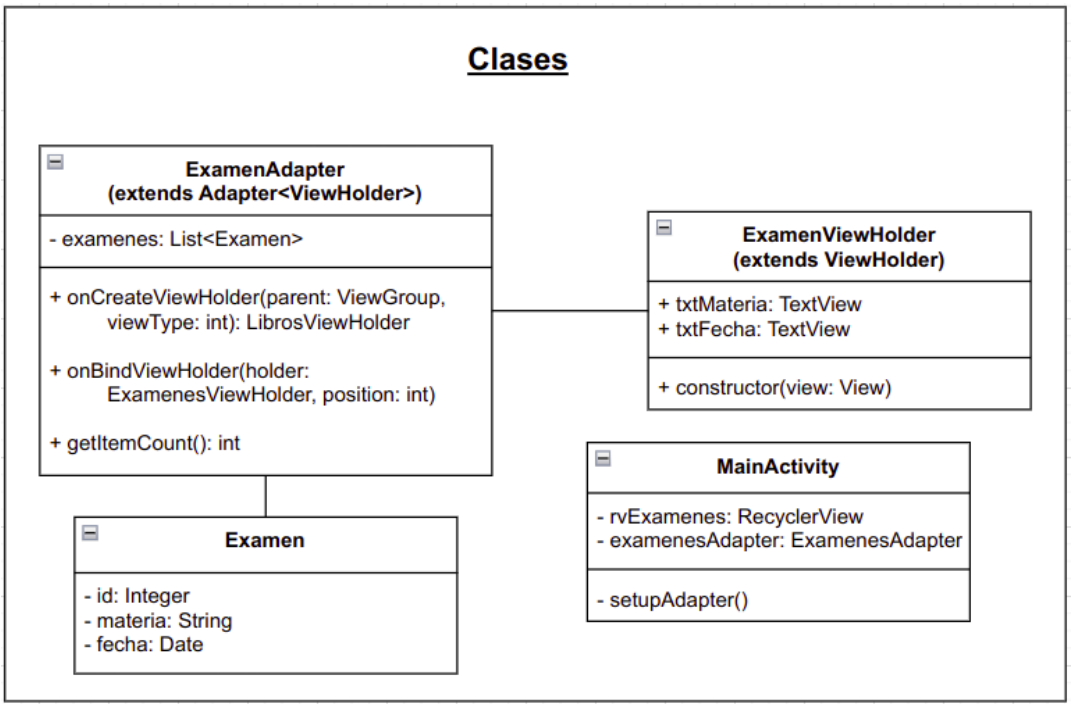
```
rvExamenes = findViewById(R.id.rvExamenes);
adapter = new ExamenAdapter(getExamenes());
rvExamenes.setAdapter(adapter);
```

Diagramas

Se utilizan 2 layouts:

- **activity_main**: es el que contendrá la vista del RecyclerView.
- **item_examen**: es el item de la vista que se replicará “n” veces dentro del RecyclerView.





Click sobre un ítem de la lista

Existen diversas maneras para poder escuchar el click sobre un ítem, o sobre un elemento en particular del mismo. Una solución es utilizar una interfaz que permita la comunicación con el Adapter desde la Vista, en este caso sería el ExamenesAdapter con el MainActivity.

Para implementar esta solución, se creará una interfaz “OnItemClickListener” dentro de ExamenesAdapter que poseerá el método onItemClick(Examen examen). Esta será una dependencia que se agregará al constructor de dicha clase. A su vez la implementará el MainActivity al momento de crear la instancia de ExamenesAdapter. Ejemplo:

Adapter

```
interface OnItemClickListener {
    void onItemClick(Examen examen);
}
```

```
holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        onItemClickListener.onItemClick(examenes.get(position));
    }
});
```

Activity

```
private void setupAdapter() {
    rvExamenes = findViewById(R.id.rvExamenes);
    adapter = new ExamenAdapter(getExamenes(), new ExamenAdapter.OnItemClickListener() {
                @Override
                public void onItemClick(Examen examen) {
                    Toast.makeText( context: MainActivity.this, examen.getMateria(), Toast.LENGTH_SHORT).show();

                }
    });
    rvExamenes.setAdapter(adapter);
```

Reciclado de vistas

Una de las razones por las cuales el RecyclerView es superior a la ListView/GridView, es debido a que por defecto este ViewGroup obliga a aplicar el concepto del reciclado de las vistas. Esto significa que al crearse todos los items y luego scrollear para ver el siguiente, no se creará una nueva vista (ítem), sino que se reutilizará el anterior para mostrarlo.

Más información: RecyclerView

Práctica 2. A

Se deberá maquetar la siguiente pantalla, recordando asignar los ids correspondientes y agregando el atributo inputType con el valor de “textPassword” dentro del EditText de la contraseña.

Luego se deberá:

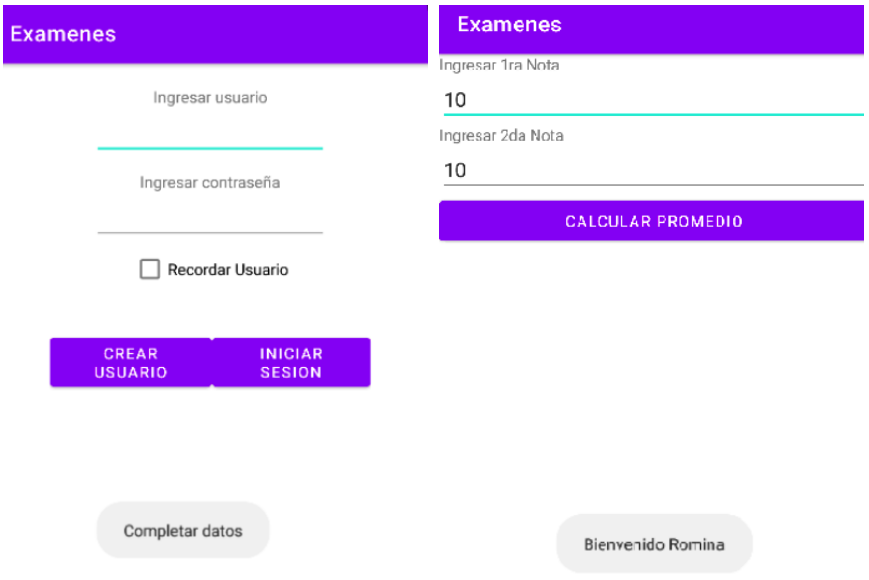
- Hacer los findViewById correspondientes
- Capturar el click del botón Iniciar sesión. Al presionarlo se deberá mostrar un Toast o mostrar un Log dentro del Logcat para validar que se encuentra funcionando.
- La acción real será aplicada en uno de los siguientes laboratorios. Para estos casos se suele indicar que la funcionalidad no está implementada mediante el comentario “TODO” en el código.

Práctica 2. B

El objetivo de este laboratorio es aplicar los conceptos vistos acerca de la navegación entre Activities mediante el Intent.

El usuario al completar los campos de usuario y contraseña podrá presionar el botón de “Iniciar Sesión”. Si al

clickear el botón los datos están vacíos, se deberá mostrar un mensaje de “Completar datos”, sino la aplicación lo llevará a la pantalla denominada MainActivity. Por parámetro se le enviará el nombre de usuario, y desde la MainActivity se deberá obtener el valor y mostrar un Toast en donde se lo saludara.

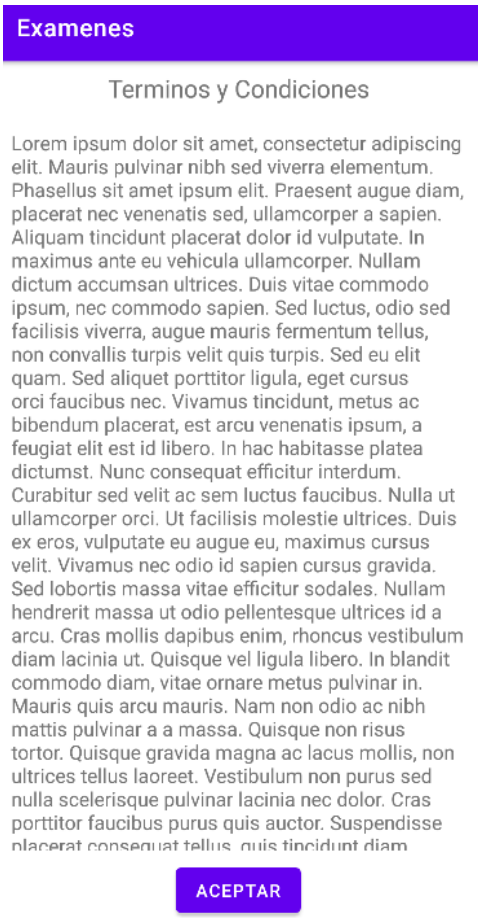


Práctica 2. C

El objetivo de este laboratorio es aplicar los conceptos vistos acerca del ScrollView.

Se deberá crear una pantalla nueva (TermsAndConditionsActivity) que contenga los Términos y condiciones de la aplicación.

El contenido del texto debe poder ser scrollable. Se accede a la misma al hacer click en “Crear Usuario” El diseño a maquetar es el siguiente:



Recursos de la aplicación.

Context

Es una clase abstracta que permite acceder a los recursos de la aplicación, como por ejemplo a Strings.xml, Colors.xml, etc. También se utiliza para realizar algunas acciones, como por ejemplo para instanciar un Intent, mostrar un Toast, etc.

La clase Activity (o AppCompatActivity) en su jerarquía extiende al Context. Esto quiere decir que desde la clase Activity se pueden acceder a sus funcionalidades.

R

R es una clase que se genera al momento de la compilación. Contiene los ids que hacen referencia a las vistas, colores, strings, etc.

Strings

Es un archivo xml que se encuentra en la carpeta "values" de la carpeta "res". Utilizarlo es una buena práctica ya que nos permite centralizar la utilización de textos y a su vez Android nos da la posibilidad de traducir estos textos a otro idioma. Dependiendo de la configuración del idioma del dispositivo, se mostrará el correspondiente.

Ejemplo XML

Al tener el siguiente botón con el atributo “text” hardcodeado, se debe crear este texto en el archivo “strings.xml"

```
<Button
    android:id="@+id/btnIniciarSesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Iniciar sesion" />
```

Archivo “strings.xml”:

```
<resources>
    <string name="app_name">My Application</string>
    <string name="iniciar_sesion">Iniciar sesion</string>
</resources>
```

Referencia desde el texto del botón:

```
<Button
    android:id="@+id/btnIniciarSesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/iniciar_sesion" />
```

Ejemplo Java

```
String texto = getResources().getString(R.string.iniciar_sesion);
```

Colors

Es un archivo xml que se encuentra en la carpeta "values" de la carpeta "res". Utilizarlo es una buena práctica ya que nos permite centralizar la utilización de los colores en un solo lugar. Si luego se debe cambiar un color que se utilizó en varios lados de manera directa, se lo podría modificar desde este lugar permitiendo así ahorrar un gran trabajo.

Además, si se define algún color específico, al momento de querer reutilizarlo, con recordar el nombre definido es suficiente. Esto ahorra el paso de la búsqueda puntual del color en hexadecimal que se requiere usar.

Ejemplo XML

Al tener el siguiente botón con un background por defecto, se cambiará por un color definido en el colors.xml:

Botón: 

XML:

```
<Button
    android:id="@+id/btnIniciarSesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:text="Iniciar sesion" />
```

Archivo colors.xml:

```
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="btnIniciarSesionColor">#1de9b6</color>
</resources>
```

Referencia desde el xml del botón:

```
<Button
    android:id="@+id/btnIniciarSesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@color/btnIniciarSesionColor"
    android:padding="10dp"
    android:text="Iniciar sesion" />
```

Resultado botón:

Ejemplo Java

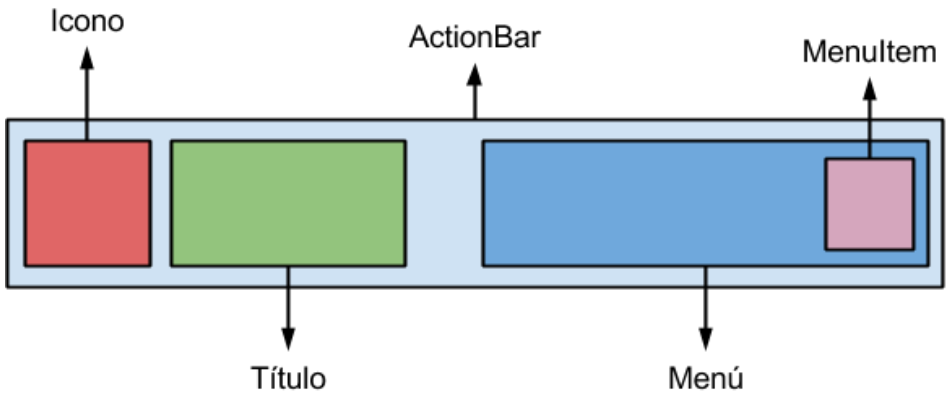
```
Button btnIniciarSesion = findViewById(R.id.btnIniciarSesion);
btnIniciarSesion.setBackgroundColor(ContextCompat.getColor(
        context: MainActivity.this, R.color.btnIniciarSesionColor));
```

Barra superior.

Actionbar vs Toolbar

Actionbar

Es una vista que se ubica en el margen superior de la pantalla, ocupa un alto fijo y brinda la posibilidad de agregarle título, subtítulo, íconos e items de menú en lugares predefinidos. Estos últimos permiten reaccionar a la interacción del usuario.



Por defecto, al crear un proyecto el ActionBar viene configurado para que se encuentre en todas las pantallas.

Toolbar

Es una generalización de la ActionBar, brinda la flexibilidad de disponer las vistas dentro de ella como sea más conveniente. Por ejemplo, si se posee un logo que es extenso, no habría inconvenientes en agregarlo dentro de un ImageView y que ocupe lo que necesite.

A su vez la Toolbar puede comportarse como un ActionBar. Por ejemplo, si por el momento no se necesita agregarle vistas, y sólo se desea que tenga un título fijo (como la ActionBar), se puede implementarlo mediante los métodos que proporciona.

Comparación

Ambas vistas permiten darle una identidad a la aplicación, ya que representan el encabezado de las pantallas, las cuales a su vez se les puede definir un color que represente la marca.

La Toolbar posee la ventaja de brindar mayor flexibilidad, ya que permite customizar al 100% dicha vista, y que también se comporte como la ActionBar.

Remoción de la Actionbar

Si se desea quitar el ActionBar para utilizar la Toolbar, se debe ir a la carpeta Res -> values -> themes.xml. Dentro de este se podrá ver que el parent del estilo de la aplicación contiene la palabra "DarkActionBar" (o similar), la cual se deberá cambiar por "NoActionBar".

Luego de hacer ese cambio, si se ejecuta nuevamente el proyecto, se podrá ver que la ActionBar no se encontrará más a la vista.

Cómo agregar una Toolbar y manipularla

Se debe asegurar que dentro del archivo themes.xml se posea configurado el Theme para que sea “NoActionBar”. Esto permitirá que la Toolbar y el ActionBar no se dupliquen. Ejemplo:

```
<style name="Theme.Examenes" parent="Theme.MaterialComponents.DayNight.NoActionBar">
```

Dentro del res -> layout se deberá agregar la vista primary_toolbar.xml:

```
<androidx.appcompat.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@color/purple_200"
    android:elevation="8dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

En el archivo .xml del Activity se deberá agregar:

```
<include
    android:id="@+id/toolbar"
    layout="@layout/primary_toolbar"/>
```

En la clase Java de la Activity se deberá:

1. Verificar que se extiende de AppCompatActivity.
2. Crear una variable del tipo Toolbar llamada toolbar:

```
private Toolbar toolbar;
```

Nota: revisar que la clase que se importa es la siguiente:

```
import android.appcompat.widget.Toolbar;
```

3. Dentro del método onCreate() hacer el findViewById correspondiente:

```
toolbar = findViewById(R.id.toolbar);
```

4. Decirle a la Toolbar que se comporte como un ActionBar para manipular los mismos métodos que esta posee, y agregarle un título en un sector específico. Para ello, se empleará el método setSupportActionBar(toolbar).
5. Y para acceder a dichos métodos se deberá llamar al getSupportActionBar() y no acceder directamente desde la instancia de la Toolbar.

Ejemplo:

```
setSupportActionBar(toolbar);
getSupportActionBar().setTitle("Mis Examenes");
```

Menú de opciones.

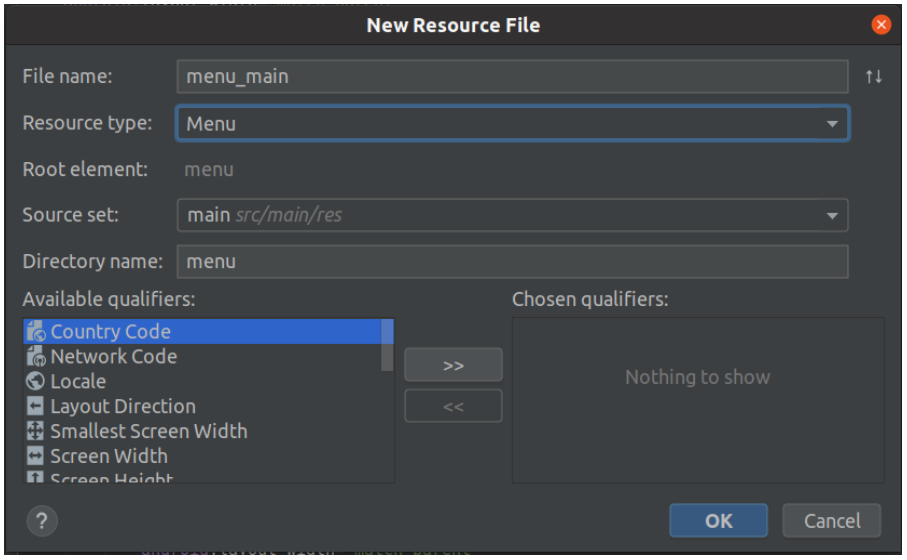
Items del menú

Tanto a la ActionBar como a la Toolbar se le puede agregar items en el menú en un lugar fijo que permitirá asociarlos con acciones. Estos ítems deben contener un id, un título y pueden tener como opcional un ícono. Se presentan en el margen derecho de la ActionBar/Toolbar.

Si el ítem no posee ícono, se mostrará por defecto un ícono con los 3 puntitos, que al presionarlo desplegará la lista de items que no contienen imagen, mostrando solamente el nombre definido.

Cómo agregar items del menú a la barra superior

Presionar click derecho sobre la carpeta “res”, elegir new -> AndroidResourceFile. Se eligirá un nombre representativo “menu_xx”, y se cambiará el Resource Type por “Menu”. Finalmente se presionará OK. Esto creará un archivo xml en la carpeta “menu”. Ejemplo:



El tag <item></item> permite crear un ítem en el menú. Mediante la propiedad “showAsAction” se podrá decidir si se quiere que el ícono del ítem se muestre siempre (always), si hay lugar (ifRoom), o nunca (never). Ejemplo:

```
<item
android:id="@+id/item_agregar"
android:title="Agregar"
app:showAsAction="never" />
```

Para agregar un ícono al ítem se debe utilizar la propiedad *icon*

```
android:icon="@drawable/ic_add"
```

Se deberán implementar dos métodos dentro de la Activity que contiene a la Toolbar:

1. Para crear el menú se deberá inflar el xml.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_home, menu);
        return super.onCreateOptionsMenu(menu);
}
```

2. Para obtener los clicks sobre los ítems del menú se comparará el id del item que viene por parámetro con el del menú:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.item_agregar){
        //Realizar acción
        }
        return super.onOptionsItemSelected(item);
}
```

Práctica 3. A

El objetivo de este laboratorio es aplicar los conceptos vistos acerca del FrameLayout.
Se deberá crear una pantalla nueva (SplashScreenActivity) la cual tendrá como raíz un FrameLayout, y un ImageView como vista hija.

Examenes

Para este ejercicio se deberá descargar una imagen con extensión jpg o png que será la que contendrá el ImageView. Esta imagen descargada deberá ser ubicada dentro de la carpeta res -> drawable. Se debe tener en cuenta que el nombre de la imagen no debe tener mayúsculas ni caracteres especiales (sólamente se permite el guión bajo).

Ejemplo : https://escuelapce.com/wp-content/uploads/2022/05/Examenes_01.jpg

Se utilizará el atributo src para hacer referencia a la imagen que mostrará el ImageView.

```
<ImageView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/examenes"/>
```

Por la parte del código Java del SplashScreenActivity se utilizará un Handler que permitirá ejecutar código luego de un tiempo definido.

```
private static final long DELAYED_TIME = 2000;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splash_screen);

    new Handler().postDelayed(new Runnable() {
        @Override
        public void run() {
            Intent intent = new Intent( packageContext: SplashScreenActivity.this, LoginActivity.class);
            startActivity(intent);
            finish();
        }
    }, DELAYED_TIME);
}
```

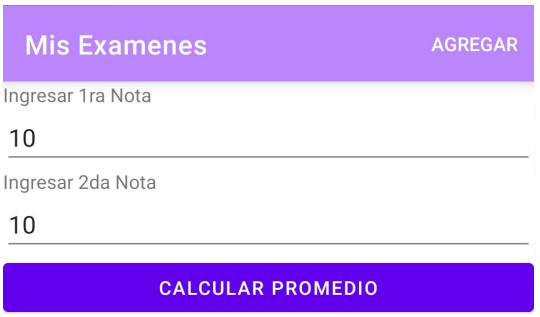
Finalmente se deberá mover el intent-filter del launcher de la aplicación hacia el SplashScreenActivity. De esta manera, la aplicación entenderá que va a ser la primera Activity en lanzarse al abrir la app.

Práctica 3. B

El objetivo de este laboratorio es aplicar los conceptos vistos acerca de la barra superior.

Se deberá agregar una Toolbar dentro de todas las pantallas (excepto SplashScreenActivity). El título de la misma deberá estar relacionado al tema del proyecto, como por ejemplo “Mis exámenes”.

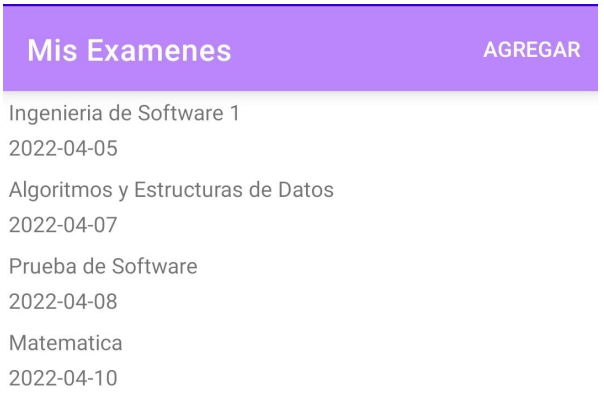
A su vez, en el toolbar de la pantalla MainActivity se deberá agregar un ítem del menú que permite navegar hacia otra pantalla para agregar Examenes. Al presionar sobre este ítem agregado, navegará hacia la pantalla de AgregarExamenActivity que por el momento no poseerá lógica, es decir que será una pantalla en blanco.



Práctica 3. C

El objetivo de este laboratorio es aplicar los conceptos vistos acerca del RecyclerView.

Se deberá implementar un Recycler View dentro del MainActivity. Pero primero, crearemos la CalcularPromedioActivity y pasaremos la lógica que está en MainActivity ahí (para luego acceder a la misma desde otro lado).



Para cargar el Recycler View y verificar que ande como se espera, se puede utilizar el siguiente método para proveer Examenes precargados:

```
private List<Examen> getExamenes() {
  return new ArrayList<Examen>() {{
    add(new Examen(1, "Ingenieria de Software 1", "2022-04-05"));
    add(new Examen(2, "Algoritmos y Estructuras de Datos", "2022-04-07"));
    add(new Examen(3, "Prueba de Software", "2022-04-08"));
    add(new Examen(4, "Matematica", "2022-04-10"));
}};
```

A su vez al presionar sobre los ítems se deberá mostrar mediante un Toast el nombre de la materia.