El futuro digital es de todos

MinTIC

Unidad 3

Aplicaciones móviles

Universidad Tecnológica de Pereira

MisiónTIC 2022

El futuro digital es de todos

MinTIC

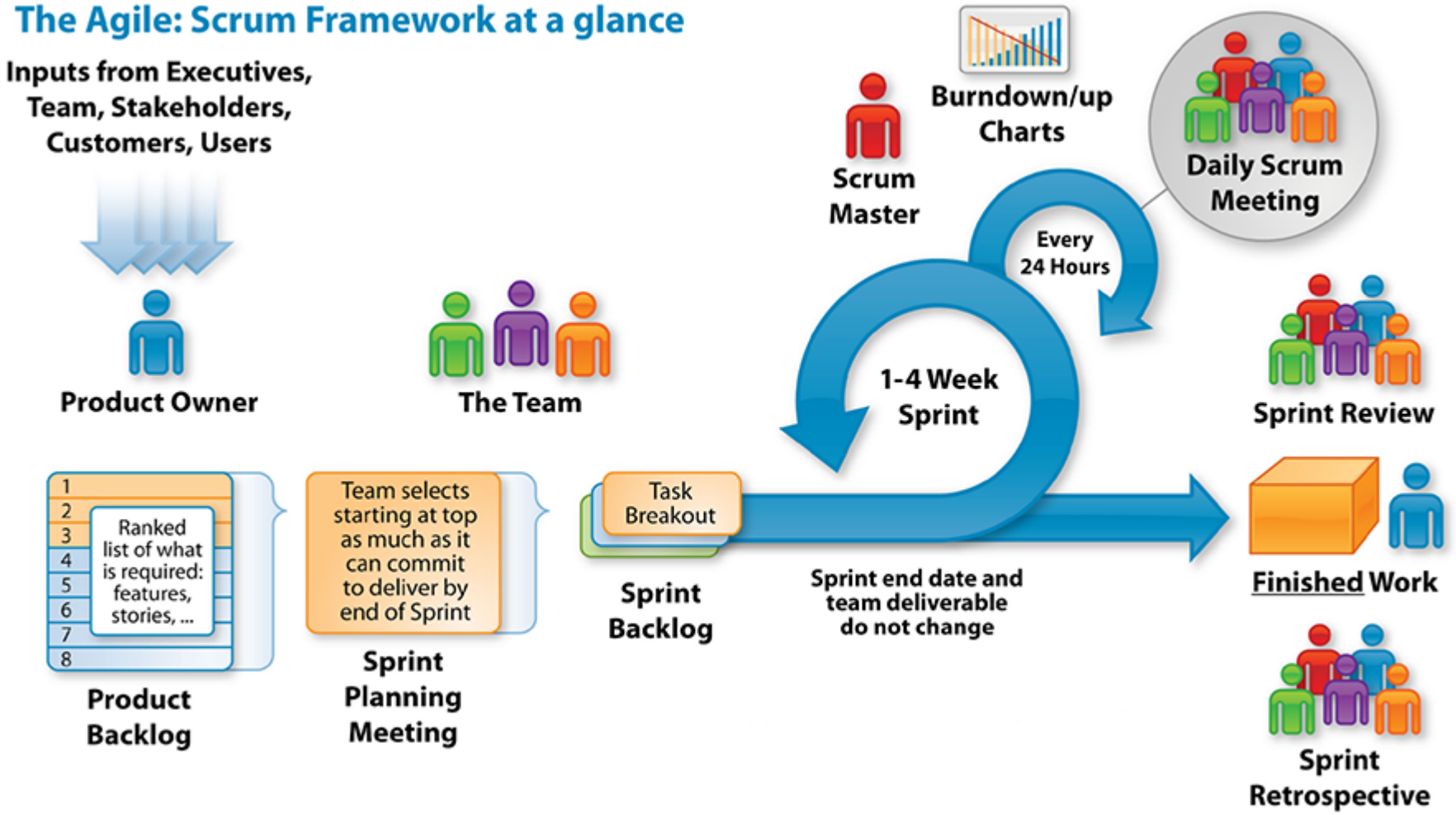
Tema 1

Retrospectiva Scrum

Universidad Tecnológica de Pereira

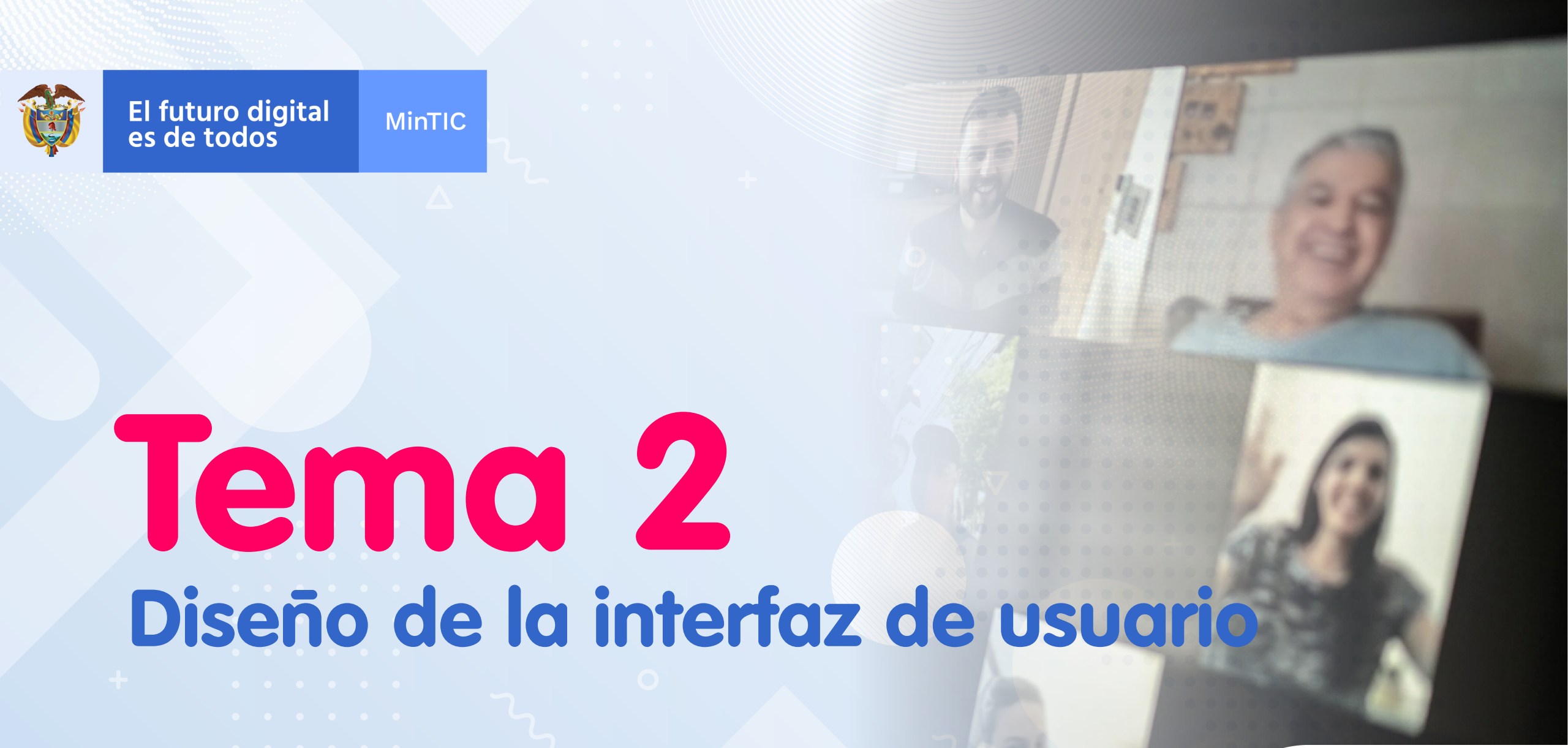
MisiónTIC 2022

Retrospectiva Scrum



El futuro digital es de todos

MinTIC



Tema 2

Diseño de la interfaz de usuario

Universidad Tecnológica de Pereira

MisiónTIC 2022

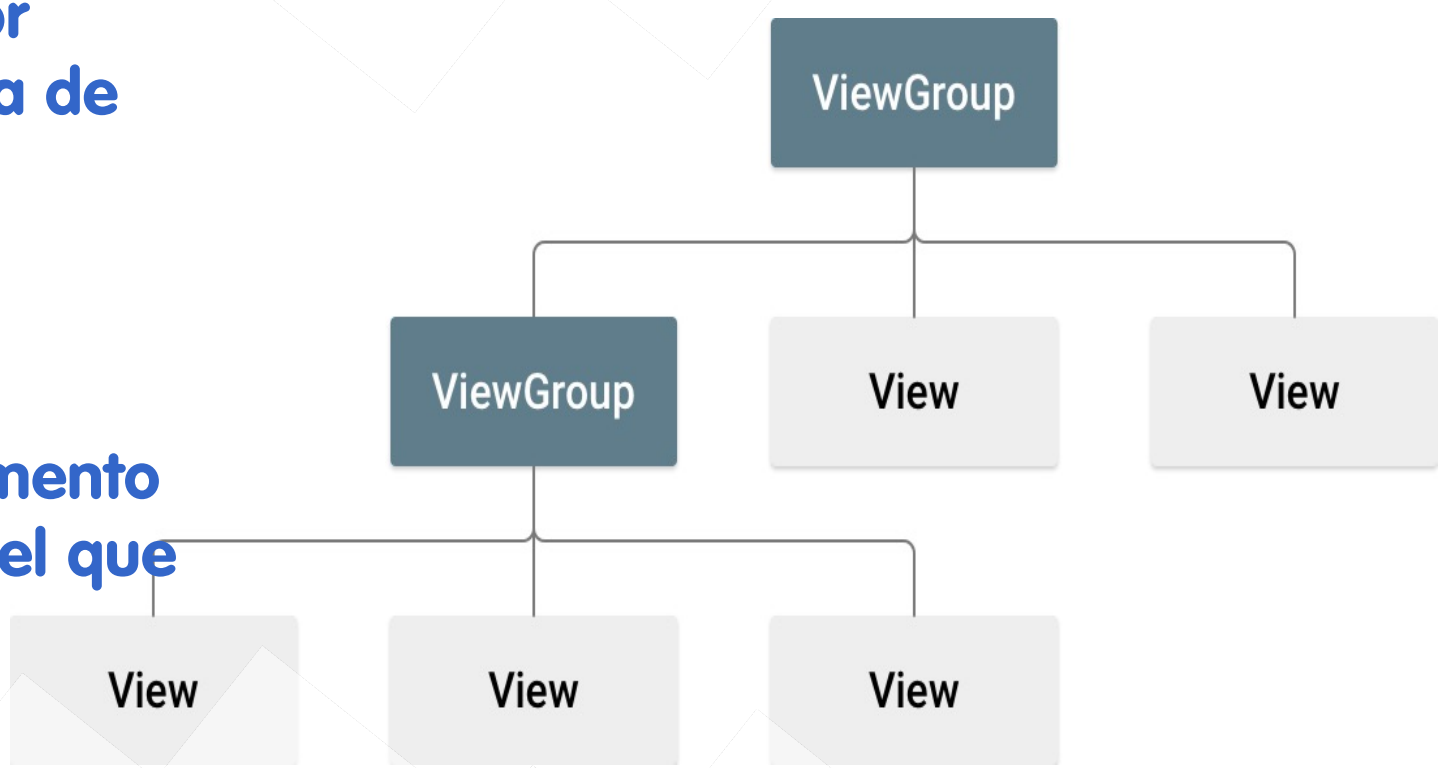
Diseño de la interfaz de suario

Un diseño define la estructura de una interfaz de usuario en tu aplicación, por ejemplo, en una actividad. Todos los elementos del diseño se crean usando una jerarquía de objetos View y ViewGroup.

Diseño de la interfaz de usuario

Un ViewGroup es un contenedor invisible que define la estructura de diseño de View y otros objetos ViewGroup, como se muestra a continuación.

Una View suele mostrar un elemento que el usuario puede ver y con el que puede interactuar.



Diseño de la interfaz de usuario

Los objetos View se denominan “widgets” y pueden ser una de muchas subclases, como Button o TextView.

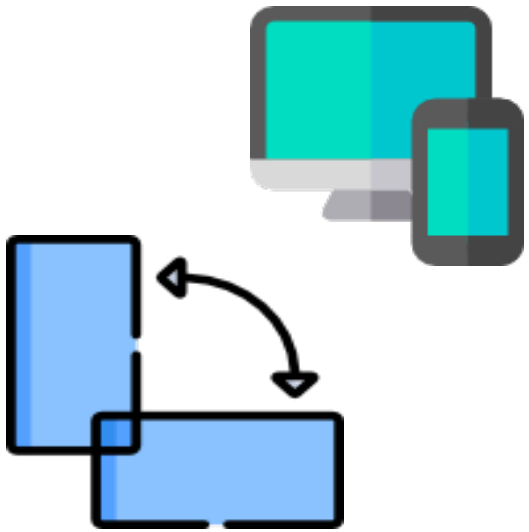
Los objetos ViewGroup se denominan “diseños” y pueden ser uno de los muchos tipos que proporcionan una estructura de diseño diferente, como LinearLayout o ConstraintLayout

Formas de declarar un diseño

1. Declarar elementos de la IU en XML. Android proporciona un vocabulario XML simple que coincide con las clases y subclases de vistas, como las que se usan para widgets y diseños.
También puedes utilizar la función editor de diseño de Android Studio para crear tu diseño XML mediante una interfaz de arrastrar y soltar.

2. Crear una instancia de elementos de diseño durante el tiempo de ejecución. Tu aplicación puede crear objetos View y ViewGroup (y manipular sus propiedades) de forma programática.

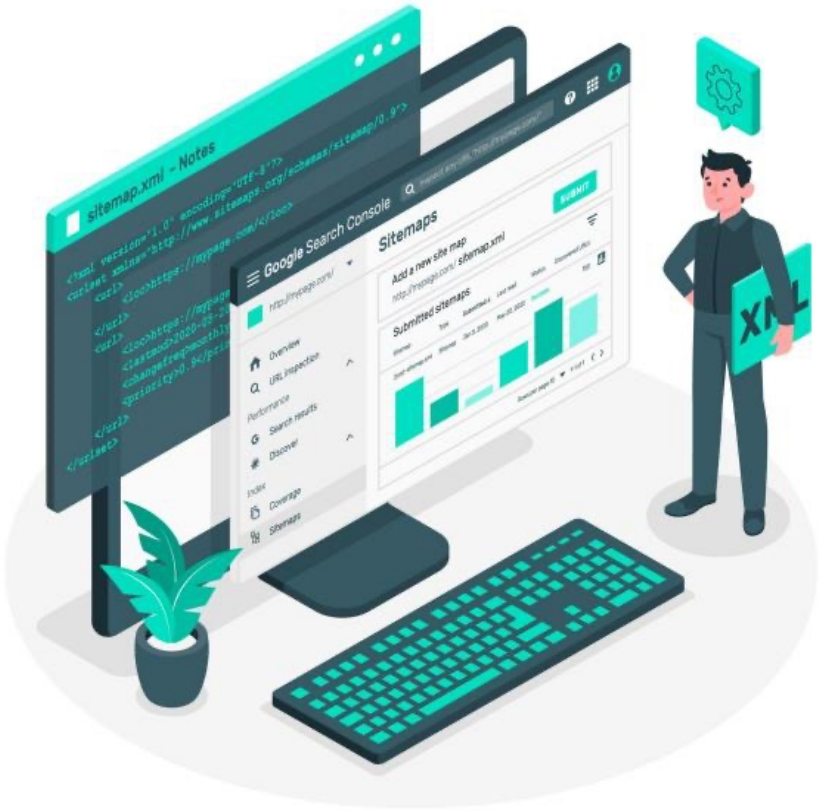
Formas de declarar un diseño



Declarar tu IU en XML te permite separar la presentación de tu app del código que controla su comportamiento. El uso de archivos XML también facilita la creación de distintos diseños para diferentes tamaños de pantalla y orientaciones.

El framework de Android te ofrece la flexibilidad de usar uno o ambos métodos para crear la IU de tu app. Por ejemplo, puedes declarar los diseños predeterminados de la app XML y, luego, modificar el diseño durante el tiempo de ejecución.

Escribe en XML



Al usar vocabulario XML de Android, puedes crear rápidamente diseños de IU y de los elementos de pantalla que contienen, de la misma manera que creas páginas web en HTML, con una serie de elementos anidados.

Cadaarchivo de diseño debe contener exactamente un elemento raíz, que debe ser un objeto View o ViewGroup. Una vez que hayas definido el elemento raíz, puedes agregar widgets u objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina el diseño.

Escribe en XML

Por ejemplo, aquí se muestra un diseño XMl que usa un LinearLayout vertical para incluir una TextView y un Button.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```


Después de declarar tu diseño en XML, guarda el archivo con la extensión .xml en el directorio res/layout/ de tu proyecto de Android para que pueda compilarse correctamente

Carga el recurso XML

Cuando compilas tu aplicación, cada archivo XML de diseño se compila en un recurso View. Debes cargar el recurso de diseño desde el código de tu aplicación, en la implementación de devolución de llamada Activity.onCreate(). Para eso, llama a setContentView() pasando la referencia a tu recurso de diseño en forma de R.layout.layout_file_name.

Por ejemplo, si tu diseño XML se guarda como main_layout.xml, los cargarás para tu actividad de la siguiente manera

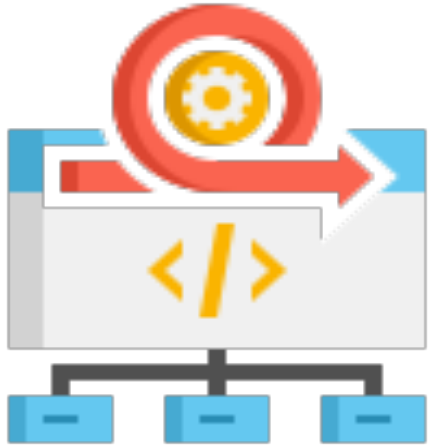
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Atributos

Cada objeto View y ViewGroup admite su propia variedad de atributos XML. Algunos atributos son específicos de un objeto View (por ejemplo, TextView admite el atributo textSize), aunque estos atributos también son heredados por cualquier objeto View que pueda extender esta clase

Algunos son comunes para todos los objetos View, porque se heredan de la clase raíz View (como el atributo id). Además, otros atributos se consideran "parámetros de diseño", que son atributos que describen ciertas orientaciones de diseño de View, tal como lo define el objeto ViewGroup superior de ese objeto.

Id



Cualquier objeto View puede tener un ID entero asociado para identificarse de forma única dentro del árbol. Cuando se compila la aplicación, se hace referencia a este ID como un número entero, pero normalmente se asigna el ID en el archivo XML de diseño como una string del atributo id. Este es un atributo XML común para todos los objetos View (definido por la clase View) y lo utilizarás muy a menudo. La sintaxis de un ID dentro de una etiqueta XML es la siguiente

```
android:id="@+id/my_button"
```

El símbolo arroba (@) al comienzo de la string indica que el analizador de XML debe analizar y expandir el resto de la string de ID, y luego identificarla como un recurso de ID. El símbolo más (+) significa que es un nuevo nombre de recurso que se debe crear y agregar a nuestros recursos (en el archivo R.java). El framework de Android ofrece otros recursos de ID. Al hacer referencia a un ID de recurso de Android, no necesitas el símbolo más, pero debes agregar el espacio de nombres de paquete Android de la siguiente manera

```
android:id="@android:id/empty"
```

Con el espacio de nombres de paquete android establecido, se hace referencia a un ID de la clase de recursos android.R, en lugar de la clase de recursos local

Id

1

Para crear vistas y hacer referencia a ellas desde la aplicación, puedes seguir este patrón común:

Definir una vista o un widget en el archivo de diseño y asignarle un ID único:

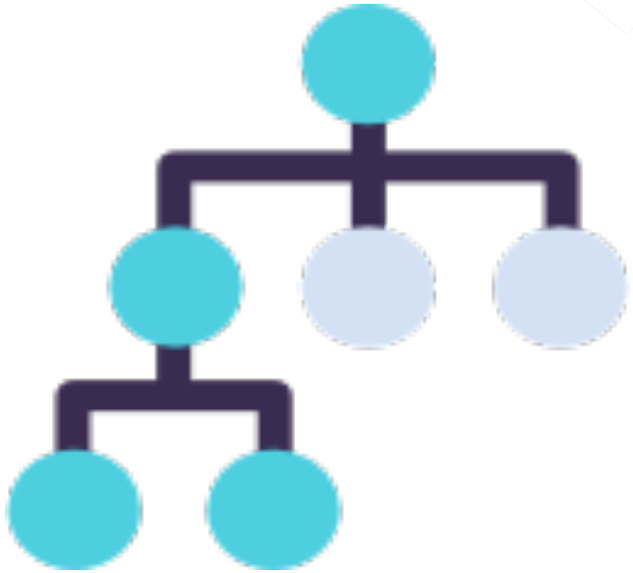
```
<Button android:id="@+id/my_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/my_button_text"/>
```

2

Luego, crear una instancia del objeto View y capturarla desde el diseño (generalmente en el método onCreate():

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

Id



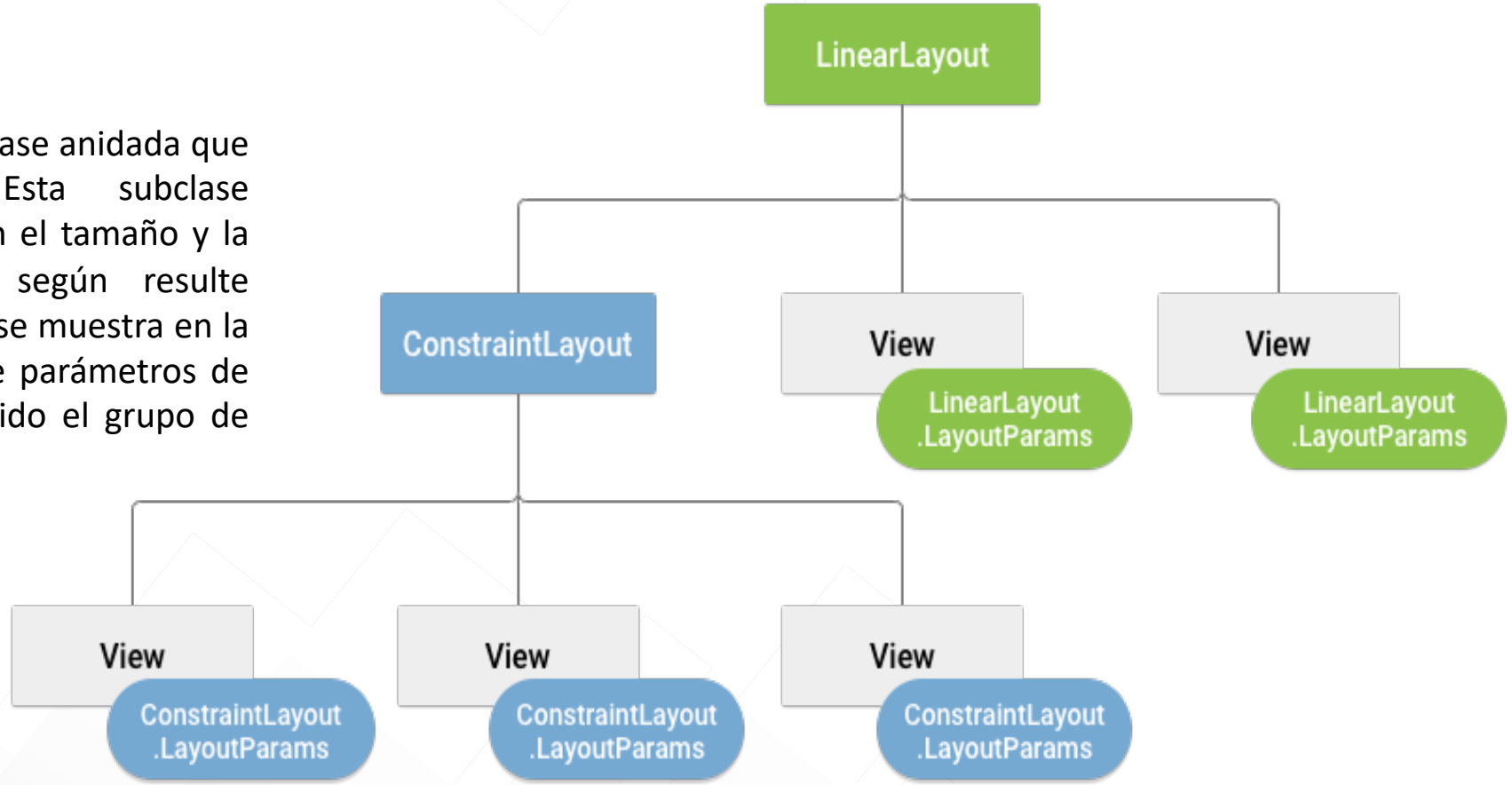
Definir ID para objetos View es importante cuando se crea un RelativeLayout. En un diseño relativo, las vistas del mismo nivel pueden definir su diseño en función de otra vista del mismo nivel, que se identifica con un ID único.

No es necesario que un ID sea único en todo el árbol, pero debe ser único dentro de la parte del árbol en la que estás buscando (que a menudo puede ser el árbol completo, por lo que es mejor que, en lo posible, sea siempre único).

Parámetros de diseño

Los atributos de diseño XML denominados layout_something definen parámetros de diseño para el objeto View que son adecuados para el objeto ViewGroup en el que reside.

Cada clase ViewGroup implementa una clase anidada que extiende ViewGroup.LayoutParams. Esta subclase contiene tipos de propiedad que definen el tamaño y la posición de cada vista secundaria, según resulte apropiado para el grupo de vistas. Como se muestra en la figura, el grupo de vistas superior define parámetros de diseño para cada vista secundaria (incluido el grupo de vistas secundario).



Parámetros de diseño

Ten en cuenta que cada subclase LayoutParams tiene su propia sintaxis para configurar valores. Cada elemento secundario debe definir LayoutParams adecuados para su elemento superior, aunque también puede definir diferentes LayoutParams para sus propios elementos secundarios.
Todos los grupos de vistas incluyen un ancho y una altura (layout_width y layout_height), y cada vista debe definirlos. Muchos LayoutParams también incluyen márgenes y bordes opcionales.

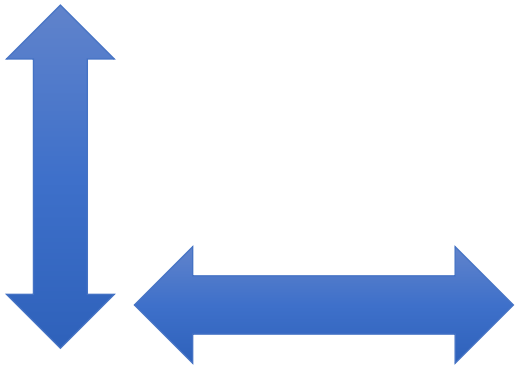
Puedes especificar el ancho y la altura con medidas exactas, aunque no es recomendable hacerlo con mucha frecuencia. Generalmente, se usa una de estas constantes para establecer el ancho o la altura:

- wrap_content le indica a tu vista que modifique su tamaño conforme a las dimensiones que requiere su contenido.
- match_parent le indica a tu vista que se agrande tanto como lo permita su grupo de vistas superior.

Parámetros de diseño

En general, no se recomienda especificar el ancho ni la altura de un diseño con unidades absolutas como píxeles. En cambio, el uso de medidas relativas, como unidades de píxeles independientes de la densidad (dp), wrap_content o match_parent, es un mejor enfoque, ya que ayuda a garantizar que la app se visualice correctamente en distintos tamaños de pantalla de dispositivos. Los tipos de medidas aceptados se definen en el documento ***Recursos disponibles***

Posición de diseño



La geometría de una vista es la de un rectángulo. Una vista tiene una ubicación, expresada como un par de coordenadas izquierda y superior, y dos dimensiones, expresadas como un ancho y una altura. La unidad para la ubicación y las dimensiones es el píxel.

Es posible recuperar la ubicación de una vista al invocar los métodos *getLeft() y getTop()*. El primero muestra la coordenada izquierda, o X, del rectángulo que representa la vista. El segundo muestra la coordenada superior, o Y, del rectángulo que representa la vista. Ambos métodos muestran la ubicación de la vista respecto al elemento superior.

Por ejemplo, cuando *getLeft()* muestra 20, significa que la vista se encuentra a 20 píxeles a la derecha del borde izquierdo de su elemento superior directo.
Además, se ofrecen varios métodos convenientes para evitar cálculos innecesarios*: getRight()* y *getBottom()*. Estos métodos muestran las coordenadas de los bordes inferior y derecho del rectángulo que representa la vista.
Por ejemplo, llamar a *getRight()* es similar al siguiente cálculo: *getLeft()* + *getWidth()*.

Cómo utilizar el LinearLayout

- ✓ Este componente te permite acomodar los componentes de la interfaz de usuario de manera lineal, puede ser tanto horizontal como vertical.
- ✓ Se puede usar para ser el principal layout de tu UI o bien para ser una parte de la misma, o ambos casos. Es decir, es muy útil y se puede usar cuando se desee, no solo al principio del código.
- ✓ Es a una de las maneras para acomodar los componentes de la UI. Y juega un papel muy importante junto con RelativeLayout y otros más.

Linear layout permite acomodar los componentes de tu UI, de manera lineal. Es decir, acomodar los componentes uno enseguida de otro, así que es muy fácil saber qué componente va enseguida de otro, a diferencia que el en Relative layout, en el cual se tiene que especificar en dónde irá cada componente. Aunque esto último es más difícil en el Relative layout, es una ventaja en cuanto a más maneras de acomodar todo.

El futuro digital es de todos

MinTIC

Tema 3

Orientación

Universidad Tecnológica de Pereira

MisiónTIC 2022

Orientación

Uno de los aspectos fundamentales del linear layout es la orientación. Por default usa orientación horizontal. Y solo se puede alternar entre Horizontal y Vertical.

Si se usa un linear layout, pero por mas que se añaden componentes, no aparecen en la vista previa del proyecto, entonces se debe revisar la orientación. Si no se tiene una orientación definida, se acomodarán de manera horizontal, es decir uno al lado de otro. Y si se quiere que aparezcan debajo, por obvias razones no se verán debajo.

Dependiendo de la orientación que se le dé, se pueden acomodar por debajo del anterior (si se usa orientación vertical) o bien al lado derecho del anterior (si se usa orientación horizontal)

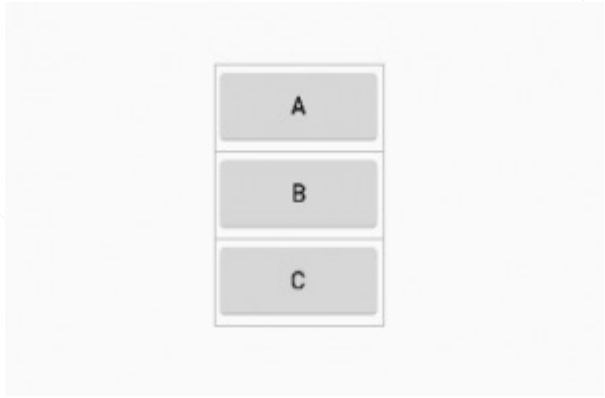
Orientación - ejemplo

Un ejemplo es el siguiente, donde se colocan 3 botones, y simplemente cambiando la orientación se verán diferentes resultados.

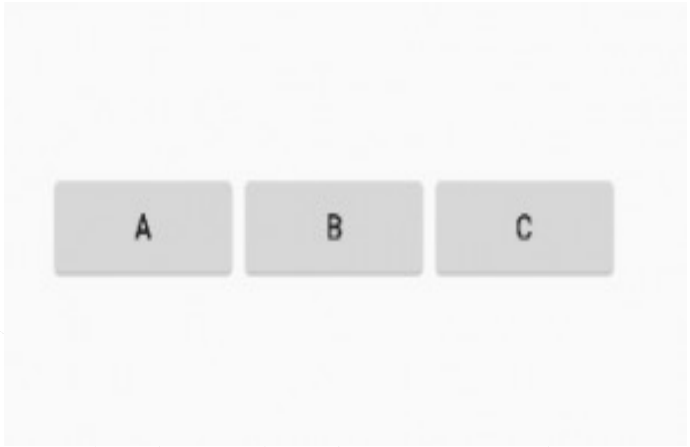
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical"
        tools:context=".MainActivity">
        <Button
        android:id="@+id/id_boton"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="A"
        />
        <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="B"
        />
        <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="C"
        />
</LinearLayout>
```

Orientación - ejemplo

Si se deja la orientación como vertical, se tendrá lo siguiente:



Si se cambia la orientación a horizontal entonces se verá así:



android:orientation="horizontal"

Ajustar tamaño

Al igual que en todos los componentes, Linear Layout tiene la opción de ajustar su tamaño (alto y ancho). Con los comandos:

```
android:layout_width="match_parent"

android:layout_height="wrap_content"
```

- ✓ Si se quiere que cubra toda la pantalla se debe usar "Match parent", así se ajusta al componente Padre, y si es el Layout principal entonces cubrirá toda la pantalla.
- ✓ Para que el tamaño sea en base al contenido entonces se debe usar: "Wrap content"
- ✓ Cuando se necesite un tamaño específico se puede colocar el valor seguido por "dp".

Color e imagen de fondo

Al igual que el resto de los componentes del xml en Android Studio. El linear layout puede tener un color de fondo o alguna imagen. Para elegir esto se hace exactamente igual que en cualquier otro componente, usando el siguiente comando:

```
android:background="@color/colorPrimaryDark"
```

Color e imagen de fondo

Puedes elegir cualquier color, hasta puedes usar código de colores por ejemplo, en el siguiente comando le damos un tono azul al fondo.

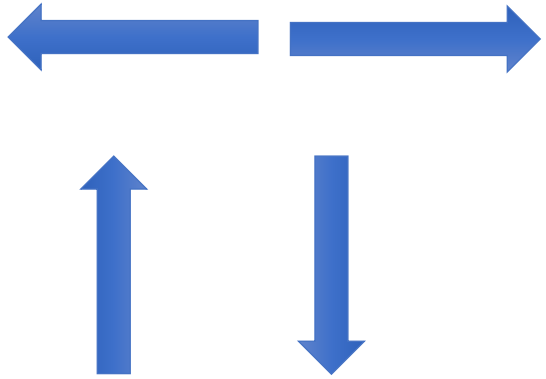
```
android:background="#00f"
```

Además puedes agregar imágenes en ves de colores, simplemente tomando la imagen de la carpeta donde este:

```
android:background="@drawable/ic_launcher_background"
```

Márgenes

Otro de los comandos importantes es el de añadir margen al layout. Se usa el comando, añadiendo un "dp" que es la unidad para el tamaño del margen

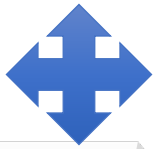


Margen derecha o izquierda 👉

```
android:layout_marginLeft="10dp"
android:layout_marginRight="10dp"
```


Margen superior o inferior 👉

```
android:layout_marginTop="20dp"
android:layout_marginBottom="20dp"
```



Mismo margen a todo el entorno 👉

```
android:padding="10dp"
```

Con este comando se añadirá un margen alrededor del layout.

Ejemplo LinearLayout

Paso 1

Ve a **res/layout** y crea un nuevo archivo llamado **ejemplo_linear_layout.xml** y agrega el siguiente código.

Crearemos un diseño de login, donde se muestren campos para digitar el usuario y el password. Además, se incorporará un botón de confirmación y un mensaje que pregunte por el olvido de la contraseña. Todos estarán alineados verticalmente sobre un linear layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="48dp">

    <TextView
        android:id="@+id/texto_conectar"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:text="Conectar“
android:textAppearance="?android:attr/textAppearanceLa
rge" />
```


Ejemplo LinearLayout

```
<EditText
    android:id="@+id/input_usuario"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:hint="Correo" />

<EditText
    android:id="@+id/input_contrasena"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:ems="10"
    android:hint="Contraseña"
    android:inputType="textPassword" />
```


Ejemplo LinearLayout

```
<Button
    android:id="@+id/boton_iniciar_sesion"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:text="Iniciar Sesion" />

<TextView
    android:id="@+id/texto_olvidaste_contrasena"
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:layout_gravity="center_horizontal"
    android:layout_weight="1"
    android:gravity="center_vertical"
    android:text="¿Olvidaste tu constraseña?"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:textColor="#0E8AEE" />
</LinearLayout>
```

Ejemplo LinearLayout

Ejemplo LinearLayout

Paso 2

Modifica el archivo **ActividadPrincipal.java** cambiando el layout que existe dentro del método `setContentView()` por `R.layout.ejemplo_linear_layout`.

```
@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.ejemplo_frame_layout);


}
```

Paso 3

Corre la aplicación y obtén el siguiente resultado.