DESARROLLO DE APLICACIONES EN ANDROID

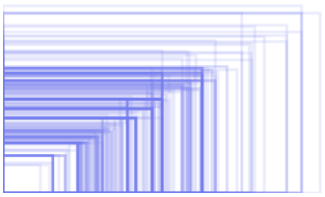
SAMSUNG

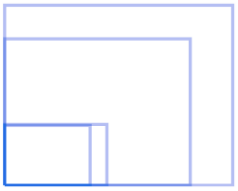
Abraham Gutiérrez Rodríguez

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños, Resoluciones, Proporciones y Densidades:
  - Cada fabricante diseña sus terminales a su gusto. No existen estándares de pantalla, de modo que a medida que aparecen distintos diseños en el mercado, se complica aún más el proceso de diseño.
  - Debemos empezar a pensar en porcentajes según las proporciones de pantalla o la dependencia de unos elementos con otros.
  - Este es el principal desafío a la hora de diseñar interfaces.





Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños, Resoluciones, Proporciones y Densidades:
  - El tamaño físico real de la pantalla es la medida de la diagonal de la pantalla y se da en pulgadas.
  - Las pantallas Android tienen diferentes densidades de píxeles, entendiendo esto como una proporción entre tamaño de pantalla y su resolución (el número total de pixeles físicos sobre una pantalla).
  - Los ***dpi*** o puntos de pantalla por pulgada, se refieren a la cantidad de píxeles que hay dentro de una pulgada de pantalla.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños, Resoluciones, Proporciones y Densidades:
  - Al añadir soporte para múltiples pantallas, las aplicaciones no deben trabajan directamente con la resolución (el número total de pixeles físicos sobre una pantalla).
  - las aplicaciones solo deben estar pensadas para que se vean correctamente según el tamaño de pantalla y la densidad, según lo especificado por los tamaños de pantallas generalizados y grupos de densidades.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños, Resoluciones, Proporciones y Densidades:
  - Al especificar el tamaño de un elemento en una interfaz de usuario de Android, hay que ser consciente de las siguientes unidades de medida:
    - **dp** - Densidad independiente del pixel. 1 dp es equivalente a un píxel en una pantalla de 160 dpi. Esta es la unidad recomendada de medición cuando se especifica la dimensión de componentes en el diseño.
      - La pantalla de 160 dpi es la densidad de la línea de base asumida por Android.
      - Se puede especificar "dp" o "dip" al referirse a un píxel independiente de la densidad.
    - **sp** – Escala independiente del pixel. Esto es similar a dp y se recomienda para especificar tamaños de fuente.
    - **pt** - Puntos. Un punto se define como 1/72 de una pulgada, basado en el tamaño de la pantalla física.
    - **px** - Píxeles. Corresponde a los píxeles reales en la pantalla. El uso de esta unidad no se recomienda, ya que la interfaz de usuario puede no visualizarse correctamente en los dispositivos con una resolución de pantalla diferente.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Resolución:
  - Densidad de píxeles Independiente (dp)
    - Una unidad de pixel virtual que se debe utilizar en la definición de diseño de la interfaz de usuario, para expresar las dimensiones de diseño o de la posición de una manera independiente de la densidad.
    - El ***dp*** es equivalente a un píxel físico en una pantalla de 160 dpi, que es la densidad de línea de base asumida por el sistema para una pantalla de densidad "media".
    - Durante la ejecución, el sistema de forma transparente maneja cualquier ampliación de las unidades ***dp***, según sea necesario, en base a la densidad real de la pantalla en uso.
    - La conversión de las unidades de ***dp*** a píxeles de la pantalla es simple:

      px = dp * (dpi / 160).
      - Por ejemplo, en una pantalla de 240 dpi, un 1dp es igual a 1,5 pixeles físicos.

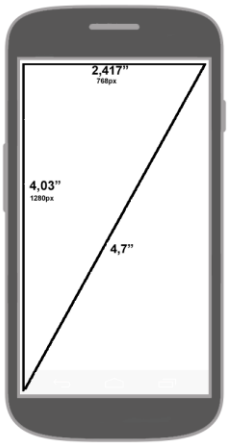
Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños, Resoluciones, Proporciones y Densidades:

En una pantalla de 4.7" de diagonal con una resolución de 1280x768 tiene una densidad de 318 DPIs.

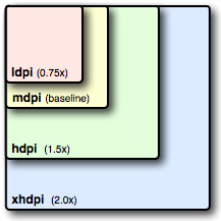
En este caso:
- En el caso de la anchura: 768/2,417" = 318 dpi
- En el caso de la altura: 1280/4,03" = 318 dpi



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños de pantalla y densidades:
  - Para simplificar, los grupos de todas las densidades de pantalla reales en Android se dividen en cuatro densidades generalizadas: low, medium, high, and extra high.
    - *Low density (ldpi)* — 120 dpi (0.75x)
    - *Medium density (mdpi)* — 160 dpi (1.00x)
    - *High density (hdpi)* — 240 dpi (1.50x)
    - *Extra High density (xhdpi)* — 320 dpi (2.00x)
  - Para simplificar, en Android se han dividido todos los tamaños de pantalla reales, en cuatro tamaños generalizados:
    - *xlarge* pantallas de al menos 960dp x 720dp
    - *large* pantallas de al menos 640dp x 480dp
    - *normal* pantallas de al menos 470dp x 320dp
    - *small* pantallas de al menos 426dp x 320dp
  - Las orientaciones son: landscape o portrait.
  - Las proporciones más habituales: 4:3 y 16:9



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

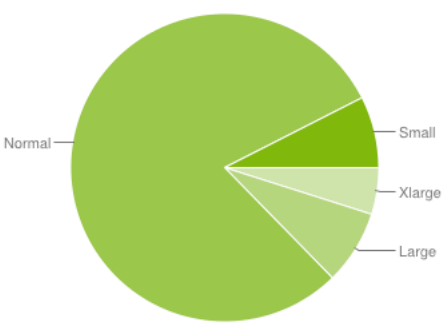
- Tamaños de pantalla y densidades:

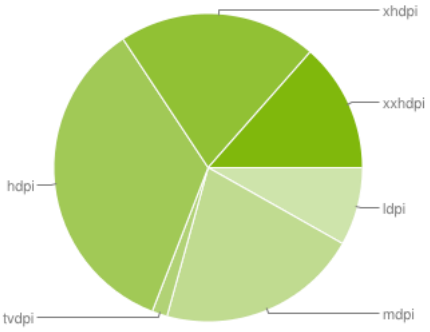
	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	**QVGA (240x320)**		480x640	
Normal screen	**WQVGA400 (240x 400)** **WQVGA (240x432)**	**HVGA (320x480)**	**WVGA (480x800)** **WVGA854 (480x854)** 600x1024	640x960
Large screen	**WVGA800 (480x800)** **WVGA854 (400x854)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024		
Extra large screen	1024x600	**WXGA (1280x800)** 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

- Tamaños de pantalla y densidades:





**datos recuperados durante un periodo de 7 días finalizado el 1 de mayo de 2014.*

Android: Conceptos Básicos
Conceptos previos

- Tamaños de pantalla y densidades:

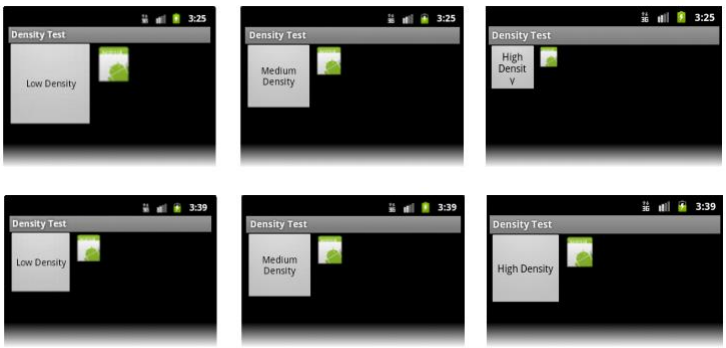
	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	7.5%						7.5%
Normal		12.5%		33.9%	19.9%	13.5%	79.8%
Large	0.6%	4.4%	1.6%	0.6%	0.6%		7.8%
Xlarge	0.1%	4.2%		0.3%	0.3%		4.9%
Total	8.2%	21.1%	1.6%	34.8%	20.8%	13.5%	

**datos recuperados durante un periodo de 7 días finalizado el 1 de mayo de 2014.*

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Conceptos previos

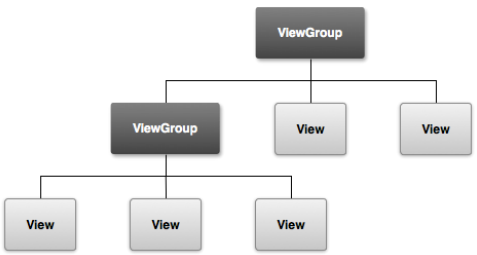
- Tamaños de pantalla y densidades:



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Todos los elementos de la interfaz de usuario en una aplicación Android se construyen utilizando las vistas (View) y los grupos o contenedores de vistas (ViewGroup).
- Una vista es un objeto que dibuja algo en la pantalla y que puede interactuar con el usuario.
- Un grupo de vistas es un objeto que contiene a su vez vistas (o grupos de vistas) con el fin de definir el diseño (layout) de la interfaz.



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Aunque la clase View representa el bloque básico para cuando queremos crear una interfaz gráfica. Android ofrece una colección de objetos subclases de vistas (en la API llamadas *widgets*) y contenedores de vistas (layouts) que ofrecen al desarrollador los controles de entrada más comunes (como botones y campos de texto) y diversos modelos de diseños.
  - vistas: Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, AutoCompleteTextView, ImageSwitcher, TextSwitcher…
  - contenedores de vistas: LinearLayout, FrameLayout, RelativeLayout…

Android: Conceptos Básicos
Interfaz de Usuario

- Para declarar un diseño, se pueden crear instancias de objetos de vista en el código y empezar a construir un árbol, pero la manera más fácil y efectiva para definir un diseño es mediante un archivo XML.
- El nombre de un elemento XML para una vista se corresponde con la clase Android que representa.
  - Así que un elemento <TextView> crea un componente TextView en la interfaz de usuario, y un elemento <LinearLayout> crea un contenedor de vistas LinearLayout.
- Cuando se carga un recurso de diseño en una aplicación, Android inicializa cada nodo del diseño en un objeto en tiempo de ejecución que se puede utilizar para definir comportamientos adicionales, consultar su estado, o modificar el diseño.

Android: Conceptos Básicos
Interfaz de Usuario

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

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- La ventaja de declarar la interfaz de usuario desde XML es que permite separar mejor la presentación de la aplicación del código que controla su comportamiento.
- Las descripciones de la interfaz de usuario son externas al código de la aplicación, lo que significa que se puede modificar o adaptar sin tener que modificar el código fuente y volver a compilar.
  - Por ejemplo, se pueden crear diseños de XML para diferentes orientaciones de la pantalla, diferentes tamaños de pantalla de dispositivos y diferentes idiomas.
- Además, declarar el diseño en XML hace que sea más fácil visualizar la estructura de la interfaz de usuario, por lo que es más fácil de depurar problemas.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

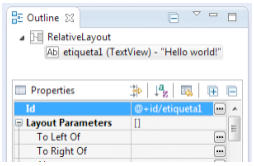
- Al compilar la aplicación, cada archivo de diseño XML (almacenado en el directorio del proyecto Android:*res/layout*) se compila en un recurso de tipo vista (View).
- Posteriormente el recurso de diseño se debe cargar en el código de la aplicación:
  - Esta operación se realiza dentro del cuerpo del método onCreate() de la actividad, utilizando la llamada setContentView(), a la que se pasa por referencia el recurso de diseño en forma de: *R.layout.layout_file_name*

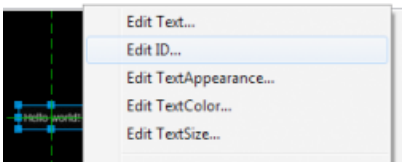
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Identificando elementos del diseño
  - Desde el código fuente de una actividad es muy común tener acceso a las propiedades y usar los métodos de los elementos de la interfaz (View)
  - Cualquier objeto vista puede tener un parámetro "ID" asociado con él, para identificarlo de forma única dentro de la jerarquía del diseño.
  - Cuando se compila una aplicación, este ID es referenciado como un entero, pero el ID se suele asignar en el archivo XML de diseño como una cadena, en el atributo id.





Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Identificando elementos del diseño
  - La sintaxis de una identificación, dentro de una etiqueta XML es:

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

  - El símbolo (@) al principio de la cadena indica que el analizador XML debe analizar y ampliar el resto de la cadena e identificarlo como un recurso ID.
  - El símbolo (+) significa que se trata de un nuevo nombre de recurso que debe ser creado y agregado a nuestros recursos (en el archivo *R.java*).
  - Al hacer referencia a un identificador de recurso de Android, no es necesario el signo (+), pero se debe agregar el espacio de nombres de paquete de *android*, así:

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

Android: Conceptos Básicos
Interfaz de Usuario

- Con el fin de crear vistas y referenciarlas desde la aplicación, un patrón común es:
  1. Definir una vista/widget en el archivo de diseño y asignarle un identificador único:

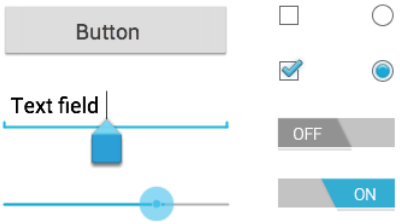
```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

  1. A continuación, creer una instancia del objeto vista y capturarlo desde el diseño (por lo general en el método onCreate()):

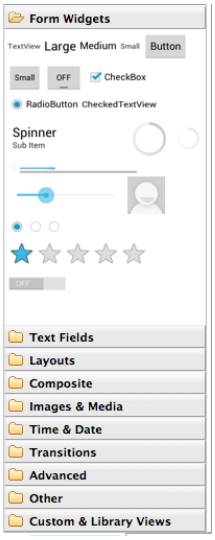
```
Button myButton = (Button) findViewById(R.id.my_button);
```

Android: Conceptos Básicos
Interfaz de Usuario

- Controles básicos de entrada.
  - Los controles de entrada son los componentes interactivos de la interfaz de usuario de su aplicación. Android ofrece una gran variedad de controles habituales de cualquier interfaz de usuario, tales como botones, campos de texto, barras de búsqueda, casillas de verificación, botones de zoom, botones de conmutación, y muchos más.
  - Añadir un control de entrada a la interfaz de usuario es tan simple como añadir un elemento XML a su fichero de configuración.



Android: Conceptos Básicos
Interfaz de Usuario



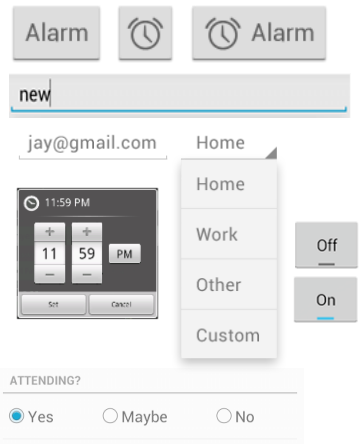
- Componentes de la interfaz de usuario.

http://developer.android.com/design/building-blocks/

Android: Conceptos Básicos
Interfaz de Usuario

- Controles básicos de entrada.

Control Type	Related Classes
Button	`Button`
Text field	`EditText`, `AutoCompleteTextView`
Checkbox	`CheckBox`
Radio button	`RadioGroup` `RadioButton`
Toggle button	`ToggleButton`
Spinner	`Spinner`
Pickers	`DatePicker`, `TimePicker`



Android: Conceptos Básicos
Interfaz de Usuario

- Eventos de entrada.
  - En Android, hay más de una manera de interceptar los eventos de la interacción del usuario con la aplicación. Al considerar los eventos dentro de la interfaz de usuario, el enfoque consiste en capturar los eventos desde la vista de objeto específico con la que el usuario interactúa.
  - La clase View proporciona diferentes medios para hacerlo. En especial están las interfaces de los escuchadores de eventos (*event listeners*).
  - Estas interfaces contienen un único método de devolución de llamada que es invocado por el sistema cuando la vista para que el escuchador se ha registrado es disparada por la interacción del usuario con el elemento de la interfaz.

Android: Conceptos Básicos
Interfaz de Usuario

- Incluidos en las interfaces escuchadoras de eventos están los siguientes métodos de devolución de llamada:
  - onClick(), desde View.OnClickListener. Se llama cuando el usuario o bien toca el ítem (en el modo de contacto) , o fija el foco en él con las teclas de navegación o la *rueda de desplazamiento* y presiona la tecla "enter" o la rueda.
  - onLongClick() desde View.OnLongClickListener. Se llama cuando el usuario o bien toca y mantiene el ítem (en el modo de contacto), o fija el foco en él con las teclas de navegación o *rueda de desplazamiento* y presiona y mantiene la tecla "enter" o la rueda (durante un segundo).
  - onFocusChange(), desde View.OnFocusChangeListener. Se llama cuando el usuario navega hacia un ítem o se aleja de él, utilizando las teclas de navegación o la *rueda de despalazamiento*.
  - onKey(), desde View.OnKeyListener. Se llama cuando el usuario focaliza en el elemento y presiona o libera una tecla de hardware en el dispositivo.
  - onTouch(), desde View.OnTouchListener. Se llama cuando el usuario realiza una acción calificada como un evento de contacto, incluyendo presionar, liberar, o cualquier gesto de movimiento en la pantalla (dentro de los límites del elemento).
  - onCreateContextMenu(), desde View.OnCreateContextMenuListener. Se llama cuando un menú contextual se está construyendo (como resultado de una pulsado sostenido).

Android: Conceptos Básicos
Interfaz de Usuario

- Interfaces escuchadoras de eventos:

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

```
public class ExampleActivity extends Activity implements OnClickListener {
    ...
}
```

Android: Conceptos Básicos
Interfaz de Usuario

- Diseño en marco (FrameLayout)
  - Éste es el más simple de todos los layouts de Android. Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia).
    - Por ello, suele utilizarse para mostrar un único control en su interior, a modo de contenedor (*placeholder*) sencillo para un sólo elemento sustituible, por ejemplo una imagen.
  - Los componentes incluidos en un FrameLayout podrán establecer sus propiedades *android:layout_width* y *android:layout_height*, que podrán tomar los valores "*match_parent*" (para que el control hijo tome la dimensión de su layout contenedor) o "*wrap_content*" (para que el control hijo tome la dimensión de su contenido).
    - NOTA: Si estás utilizando una versión de la API de Android inferior a la 8 (Android 2.2), en vez de "*match_parent*" deberás utilizar su equivalente "*fill_parent*".

Android: Conceptos Básicos
Interfaz de Usuario

- Diseño lineal (LinearLayout)
  - Es un grupo de vistas que alinea a todos sus componentes hijos en una sola dirección, vertical u horizontalmente. Se puede especificar la dirección del diseño con el atributo *android:orientation*.
  - Los elementos contenidos en un LinearLayout pueden establecer sus propiedades *android:layout_width* y *android:layout_height* para determinar sus dimensiones dentro del diseño.
  - Además tiene la propiedad *android:layout_weight* que permite dar a los elementos contenidos en el diseño unas dimensiones proporcionales entre ellas.

Android: Conceptos Básicos
Interfaz de Usuario

- Diseño en tabla (TableLayout)
  - Permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla. La estructura de la tabla se define indicando las filas que compondrán la tabla (objetos TableRow), y dentro de cada fila las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna sino que directamente insertaremos los controles necesarios dentro de cada fila. Cada componente insertado (que puede ser un control sencillo o incluso otro contenedor de vistas) corresponderá a una columna de la tabla.
    - De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.
  - Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:
    - *android:stretchColumns*. Indica las columnas que pueden expandir para absorber el espacio libre dejado por las demás columnas a la derecha de la pantalla.
    - *android:shrinkColumns*. Indica las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la pantalla.
    - *android:collapseColumns*. Indica las columnas de la tabla que se quieren ocultar completamente.
  - Otra propiedad importante es *android:layout_span* que permite que una celda pueda ocupar el espacio de varias columnas de la tabla.

Android: Conceptos Básicos
Interfaz de Usuario

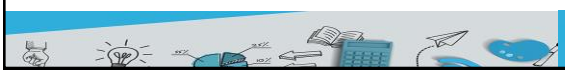
- Diseño en cuadrícula (GridLayout)
  - Introducido en la API 14 (Android 4.0) sus características son similares al TableLayout, ya que se utiliza para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas.
  - En este caso indicaremos el número de filas y columnas como propiedades del diseño, mediante las propiedades *android:rowCount* y *android:columnCount*.
    - Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento TableRow del TableLayout, sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad *android:orientation*) hasta completar el número de filas o columnas indicadas en los atributos anteriores.
  - También cuenta con las propiedades *android:layout_rowSpan* y *android:layout_columnSpan* para conseguir que una celda ocupe el lugar de varias filas o columnas.
  - Existe también una forma de indicar de forma explícita la fila y columna que debe ocupar un determinado elemento hijo contenido en el GridLayout, y se consigue utilizando los atributos **android:layout_row** y *android:layout_column*.

Android: Conceptos Básicos
Interfaz de Usuario

- Diseño relativo (RelativeLayout)
  - Este diseño permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio diseño. Para ello define propiedades como:
    - *android:layout_above.*
    - *android:layout_below.*
    - *android:layout_toLeftOf.*
    - *android:layout_toRightOf.*
    - *android:layout_alignLeft.*
    - *android:layout_alignRight.*
    - *android:layout_alignTop.*
    - *android:layout_alignBottom.*
    - *android:layout_alignBaseline.*
    - *android:layout_alignParentLeft.*
    - *android:layout_alignParentRight.*
    - *android:layout_alignParentTop.*
    - *android:layout_alignParentBottom.*
    - *android:layout_centerHorizontal.*
    - *android:layout_centerVertical.*
    - *android:layout_centerInParent.*

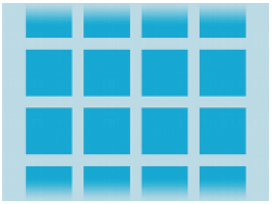
Android: Conceptos Básicos
Interfaz de Usuario

- Vista de lista (ListView)
  - Es un grupo de vistas que muestra una lista desplazable de elementos.
  - Los elementos de la lista se insertan automáticamente a la lista utilizando un adaptador que extrae el contenido de una fuente tal como una matriz o una consulta de base de datos y convierte cada resultado de elemento en una vista que se coloca en la lista.
  - Su uso es tan habitual que de han definido actividades que contienen vista de lista (ListActivity).



Android: Conceptos Básicos
Interfaz de Usuario

- Vista de cuadrícula (GridView)
  - Es un contenedor de vista que muestra los elementos en una, cuadrícula desplazable en dos dimensiones.
  - Los elementos de la cuadrícula se insertan automáticamente en el diseño mediante un ListAdapter.



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Adaptadores de vistas (AdapterView)
  - Cuando el contenido de un diseño es dinámico o no pre-determinado, se puede utilizar un diseño que herede de AdapterView para rellenar el diseño con vistas en tiempo de ejecución.
  - Una subclase de la clase AdapterView utiliza un adaptador (Adapter) para enlazar los datos al diseño. El adaptador se comporta como un intermediario entre la fuente de datos y el esquema de diseño.
  - El adaptador recupera los datos (de una fuente tal como una matriz o una consulta la base de datos) y convierte cada entrada en una vista que se puede agregar en el diseño fijado por el AdapterView .

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Adaptadores (Adapter)
  - Se puede rellenar un AdapterView mediante la unión de la instancia de un adaptador de vistas a un adaptador, que recupera datos de una fuente externa y crea una vista que representa cada entrada de datos.
  - Android proporciona varias subclases de adaptador que son útiles para la recuperación de los diferentes tipos de datos. Los dos adaptadores más comunes son:
    - ArrayAdapter. Utilice este adaptador cuando el origen de datos es una matriz. Por defecto, crea una vista para cada elemento de la matriz llamando al método toString() en cada ítem y colocar el contenido en un TextView.
    - SimpleCursorAdapter. Utilice este adaptador cuando los datos proviene de un cursor (Cursor). Se debe especificar un diseño que se utilizará para cada fila del cursor y qué columnas del cursor se deben insertar en qué vistas del diseño.

Android: Conceptos Básicos
Interfaz de Usuario

- Adaptadores (ArrayAdapter)

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, myStringArray);
```

  - Los argumentos del constructor son:
    - El contexto de la aplicación (Context).
    - El diseño que contiene un TextView para cada cadena de la matriz.
    - La matriz de cadenas.
  - Finalmente sólo hay que llamar al método setAdapter() en el ListView:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

  - Para personalizar la apariencia de cada elemento se puede sobrescribir el método toString() para los objetos de la matriz. O bien, para crear una vista para cada elemento, ampliar la clase ArrayAdapter y sobrescribir getView() para devolver el tipo de vista que desea para cada elemento .

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Toast)
  - Un Toast es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento).
  - Aunque son personalizables, aparecen por defecto en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido. Por sus propias características, este tipo de notificaciones son ideales para mostrar mensajes rápidos y sencillos al usuario, pero por el contrario, al no requerir confirmación por su parte, no deberían utilizarse para hacer notificaciones demasiado importantes.

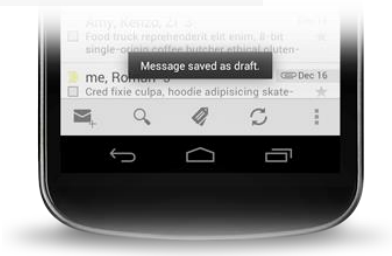
Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Toast)

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

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



Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Dialog)
  - Un diálogo es una pequeña ventana que pide al usuario para tomar una decisión o introducir información adicional.
  - Un cuadro de diálogo no llena la pantalla y se utiliza normalmente para los eventos modales que requieren de los usuarios tomar una acción antes de que puedan proceder.
  - La clase Dialog es la clase base para los diálogos, pero se debe evitar crear instancias de ella directamente. En su lugar, es preferible utilizar una de las siguientes subclases:
    - AlertDialog . Un cuadro de diálogo que puede mostrar un título, un máximo de tres botones, una lista de elementos seleccionables, o un diseño personalizado.
    - DatePickerDialog o TimePickerDialog . Un diálogo con una interfaz de usuario predefinida que permite al usuario seleccionar una fecha u hora.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Dialog)
  - También se puede utilizar DialogFragment como un contenedor para diálogos. Esta clase proporciona todos los controles que se necesitan para crear un diálogo y gestionar su apariencia, en vez de llamar los métodos en el objeto Dialog que llama.
    - DialogFragment asegura que se gestionan correctamente los eventos del ciclo de vida, como cuando el usuario pulsa el botón "Atrás" o gira la pantalla.
    - La clase DialogFragment también permite reutilizar la interfaz del cuadro de diálogo como un componente integrable de otras interfaces, al igual que un fragmento tradicional.
    - Nota: Debido a que la clase DialogFragment se añadió originalmente con Android 3.0 (nivel de API 11), para utilizarlo en versiones anteriores hay que hacer uso de la biblioteca de soporte: *android.support.v4.app.DialogFragment*

http://developer.android.com/design/building-blocks/dialogs.html

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Dialog)

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // FIRE ZE MISSILES!
                   }
               })
               .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog, int id) {
                       // User cancelled the dialog
                   }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

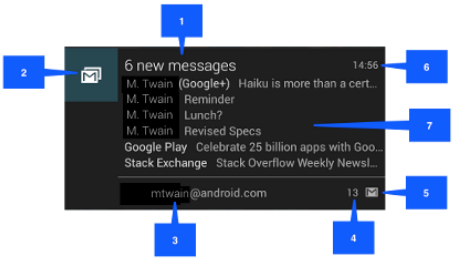

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Notification)
  - Una notificación es un mensaje que podemos mostrar al usuario fuera de la interfaz gráfica normal de nuestra aplicación. Para ello, Android cuenta con una área de notificación.
  - Además, podremos ver detalles de las notificaciones en el panel de notificaciones.
    - Ambas definiciones están siempre disponibles para que el usuario pueda verlos.
    - Nota: la clase Noltification.Builder se añadió en Android 3.0, para hacer suo en versiones anteriores hay que utilizar la librería de soporte: NotificationCompat.Builder

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Notification)

Vista normal



Vista ampliada
<Android 4.1>

http://developer.android.com/design/patterns/notifications.html

Android: Conceptos Básicos
Interfaz de Usuario

- Notificaciones (Notification)
  - Para crear una notificación simplemente hay que llamar al método build() del objeto Notification.Builder (añadido en Android 3.0) que devuelve un objeto de Notificación conteniendo sus especificaciones. Para emitir la notificación, se pasa el objeto de notificación al sistema llamando NotificationManager.notify().
    - En la versión 4 de la librería de soporte está disponible el objeto NotificationCompat.Builder para proporcionar compatibilidad en versiones anteriores a la 3.0.
  - Un objeto de notificación debe contener al menos lo siguiente:
    - Un pequeño icono, establecido por setSmallIcon()
    - Un título, establecido por setContentTitle()
    - Detalle de texto, fijado por setContentText()
  - Entre los contenidos adicionales están las acciones y la prioridad de la notificación.

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)
  - Los menús son un componente de interfaz de usuario común en muchos tipos de aplicaciones. Para proporcionar una experiencia de usuario familiar y consistente, se debe utilizar las API de menú para presentar las acciones del usuario y otras opciones en las actividades.
  - Existen tres tipos de menús:
    - Menús de opciones y barra de acción. Es la colección principal de elementos de menú para una actividad.
    - Menús contextuales y modos de acción contextual . Un menú contextual es un menú flotante que aparece cuando el usuario realiza una pulsación de larga duración en un elemento. Proporciona acciones que afectan al contenido seleccionado o al marco contextual.
    - Menús desplegables (Popup). Un menú desplegable muestra elementos en una lista vertical que se anclan a la vista que ha invocado el menú. Proporciona acciones que se relacionan con el contenido específico u opciones para segundadas parte de un comando.

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)
  - Nota: A partir de Android 3.0 (nivel de API 11), los dispositivos no necesitan proporcionar un botón de menú. Con este cambio, las aplicaciones de Android deben migrar del panel de menú de 6 ítems tradicional a la barra de acción para presentar acciones comunes a los usuarios.
  - Aunque el diseño y la experiencia del usuario para algunos elementos del menú han cambiado, la semántica para definir un conjunto de acciones y opciones se basan todavía en las API de menú.

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)
  - Para todos los tipos de menú, Android proporciona un formato estándar XML para definir los elementos del menú.
    - En lugar de construir un menú en el código de la actividad, se debe definir un menú y todos sus artículos en un recurso de menú XML.
    - A continuación, se debe inflar el recurso de menú y cargarlo como un objeto Menu en la actividad o fragmento.
  - El uso de un recurso de menú es una buena práctica por varias razones
    - Es más fácil de visualizar la estructura de menús en XML.
    - Separa el contenido del menú del comportamiento fijado por el código de la aplicación.
    - Permite crear configuraciones de menú alternativos para diferentes versiones de la plataforma, tamaños de pantalla y otras configuraciones, aprovechando el entorno de recursos de la aplicación.

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)
  - Para definir el menú hay que crear un archivo XML en el directorio */res/menu* del proyecto y definir el menú con los siguientes elementos:
    - <menu> Define el contenedor para los elementos del menú. Debe ser el nodo raíz del archivo y puede contener uno o más elementos <item> y <group>.
    - <item> Crea un MenuItem, que representa un solo elemento de un menú. Este elemento puede contener un elemento <menu> anidada con el fin de crear un submenú.
    - <group> Es un contenedor opcional e invisible para elementos <item>. Permite clasificar los elementos de menú para que compartan propiedades tales como estado activo y la visibilidad.

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Android: Conceptos Básicos
Interfaz de Usuario

- Menús (Menu)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

- Nota: Android 3.0 añade la capacidad para definir en XML el comportamiento on-click para una opción del menú, utilizando el atributo android:onClick. El valor del atributo debe ser el nombre de un método definido por la actividad que está utilizando el menú.

Android: Conceptos Básicos
Interfaz de Usuario

- Barra de acción (ActionBar)
  - La barra de acción es una función de ventana que identifica la ubicación del usuario, y proporciona acciones y modos de navegación.
    - Es la barra de título y herramientas. Normalmente muestra un icono, el título de la actividad en la que nos encontramos, una serie de botones de acción, y un menú desplegable (*menú de overflow*) donde se incluyen más acciones que no tienen espacio para mostrarse como botón o simplemente no se quieren mostrar como tal.

Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Barra de acción (ActionBar)
  - La barra de acción proporciona varias funciones clave:
    - Proporciona un espacio dedicado para dar a la aplicación una identidad y que indica la ubicación del usuario en la aplicación.
    - Permite hacer accesible de una manera prominente y predecible acciones importantes (como Buscar).
    - Soporta navegación consistente y vista de conmutación dentro de aplicaciones (con pestañas o listas desplegables).
    - Nota: La API ActionBar (*android.app.ActionBar*) se añadió por primera vez en Android 3.0 (nivel de API 11), pero también están disponibles en la biblioteca de soporte para la compatibilidad con Android 2.1 (nivel de API 7) y superiores (*android.support.v7.app.ActionBar*). Se puede acudir a librerías externas como *ActioBarSherlock* para hacer la aplicación compatible con cualquier versión de Android.

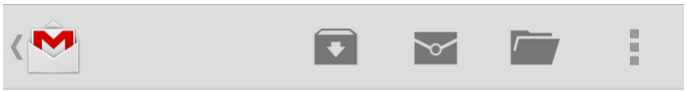
Samsung TECH INSTITUTE

Android: Conceptos Básicos
Interfaz de Usuario

- Barra de acción (ActionBar)
  - Nota: La API ActionBar se añadió por primera vez en Android 3.0 (nivel de API 11), pero también están disponibles en la biblioteca de soporte para la compatibilidad con Android 2.1 (nivel de API 7) y superiores.
    - Soporte de **niveles de API inferior a 11**: *android.support.v7.app.ActionBar.* Además hay que crear la actividad mediante la extensión de la clase ActionBarActivity y utilizar o extender uno de los temas de *Theme.AppCompat* para la actividad.
    - Soporte de **niveles de API 11 y superiores**: *android.app.ActionBar.* La barra de acción se incluye en todas las actividades que utilizan el tema *Theme.Holo* (o uno de sus descendientes), que es el tema por defecto cuando el atributo targetSdkVersion o minSdkVersion se establece en "11" o superior. Si no desea que la barra de acción en una actividad, hay que establecer el tema de la actividad a *Theme.Holo.NoActionBar*.
    - Se puede acudir a librerías externas como *ActioBarSherlock* para hacer la aplicación compatible con cualquier versión de Android.

Android: Conceptos Básicos
Interfaz de Usuario

- Barra de acción (ActionBar)



```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item
     android:id="@+id/menu_settings"
     android:title="@string/menu_settings"
     android:icon="@drawable/item_icon"
     android:showAsAction="never|ifRoom|withText|always|collapseActionView"
  </item>
</menu>
```

Android: Conceptos Básicos
Interfaz de Usuario

- Estilos y Temas
  - Un estilo es un conjunto de propiedades que especifican el aspecto y el formato de una vista o una ventana. Un estilo puede especificar propiedades como la altura, el relleno, color de fuente, tamaño de fuente, color de fondo, y mucho más.
    - Un estilo se define en un recurso XML, <style>, que es independiente del documento XML que especifica el diseño.
    - Los estilos de Android comparten una filosofía similar a las hojas de estilo en cascada en el diseño web, permitiendo separar el diseño del contenido.
  - Un tema es un estilo aplicado a toda una actividad <activity> o aplicación <application>, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible.
  - La plataforma Android ofrece una gran colección de estilos y temas que puede utilizar en las aplicaciones:http://developer.android.com/reference/android/R.style.html

Android: Conceptos Básicos
Interfaz de Usuario

- Estilos y Temas
  - El siguiente código:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="Un texto" />
```

  - Es equivalente a escribir:

```
<TextView
    style="@style/MiEstilo"
    android:text="Un texto" />
```

  - Habiendo definido en *res/values/styles.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="MiEstilo"
     parent="@android:style/TextAppearance.Medium">
       <item name="android:layout_width">match_parent</item>
       <item name="android:layout_height">wrap_content</item>
       <item name="android:textColor">#00FF00</item>
       <item name="android:typeface">monospace</item>
   </style>
</resources>
```

Android: Conceptos Básicos
Interfaz de Usuario

- Estilos y Temas
  - El atributo parent en el elemento <style> permite especificar un estilo del que se heredarán propiedades.
    - Se puede usar para heredar las propiedades de un estilo existente y luego definir sólo las propiedades que desea cambiar o agregar.
    - Se puede heredar de estilos que haya creado el programador o de los estilos que están integradas en la plataforma.

```
<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

    - Si desea heredar de estilos propios, no hay que utilizar el atributo parent. Sólo hay que indicar un prefijo con el nombre del estilo que desea heredar y el nombre del nuevo estilo, separados por un punto.

```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```


Android: Conceptos Básicos
Interfaz de Usuario

- Estilos y Temas
  - El atributo parent en el elemento <style> permite especificar un estilo del que se heredarán propiedades.
    - Se puede usar para heredar las propiedades de un estilo existente y luego definir sólo las propiedades que desea cambiar o agregar.
    - Se puede heredar de estilos que haya creado el programador o de los estilos que están integradas en la plataforma.

```
<style name="GreenText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

    - Si desea heredar de estilos propios, no hay que utilizar el atributo parent. Sólo hay que indicar un prefijo con el nombre del estilo que desea heredar y el nombre del nuevo estilo, separados por un punto.

```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

Samsung TECH INSTITUTE