ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 - Materia: **Aplicaciones Android**

TEMA: User Interfaces I

Consideraciones

- Estas transparencias **no** tienen el objetivo de suplir las clases.
- Por tanto, serán **complementadas** con ejemplos, códigos, profundizaciones y comentarios por parte del docente.
- El **orden** de dictado de estos temas está sujeto a la consideración del docente.

Referencias

- Información general de IU:
  - *https://developer.android.com/guide/topics/ui/overview.html*
- Diseños (layouts):
  - *https://developer.android.com/guide/topics/ui/declaring-layout.html*
- Controles de entrada:
  - *https://developer.android.com/guide/topics/ui/controls.html*
- Eventos de entrada:
  - *https://developer.android.com/guide/topics/ui/ui-events.html*
- Menús:
  - *https://developer.android.com/guide/topics/ui/menus.html*
- Cuadros de diálogo:
  - *https://developer.android.com/guide/topics/ui/dialogs.html*

Agenda

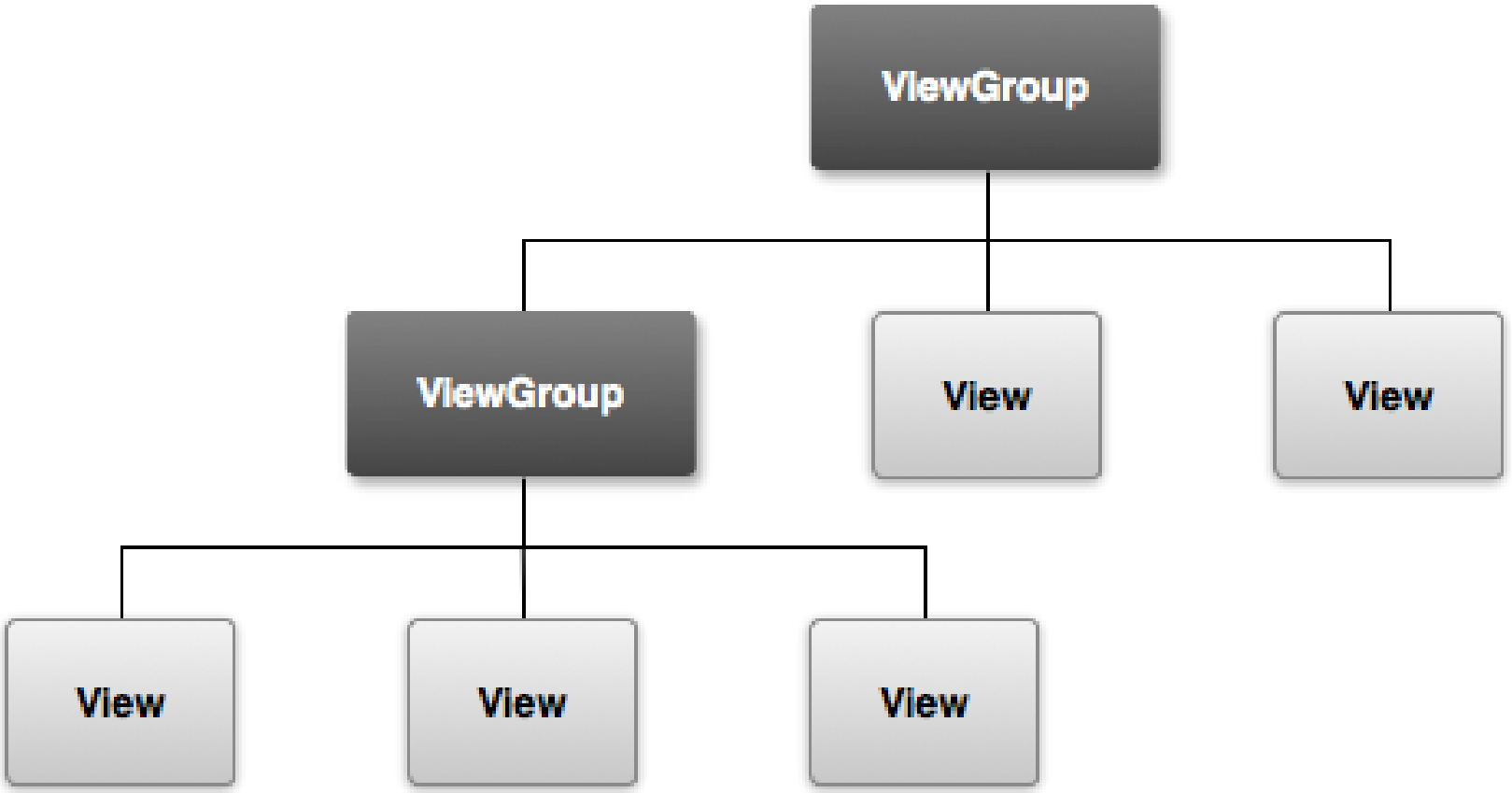
- Interfaz de Usuario
- Layouts
- Controles
- Menús
- Diálogos

Interface de Usuario (1)

- Se construye utilizando objetos **View** o **ViewGroup**.
  - *View*: todo elemento dibujable es subclase de View.
  - *ViewGroup*: contenedor de uno o mas objetos View.

- Creación de interfaces de usuario:
  - Instanciando los objetos View en código Java (agregarlo a sus respectivos contenedores, seteo de propiedades, etc).
  - Utilizando XML Layouts.

Interface de Usuario (2)

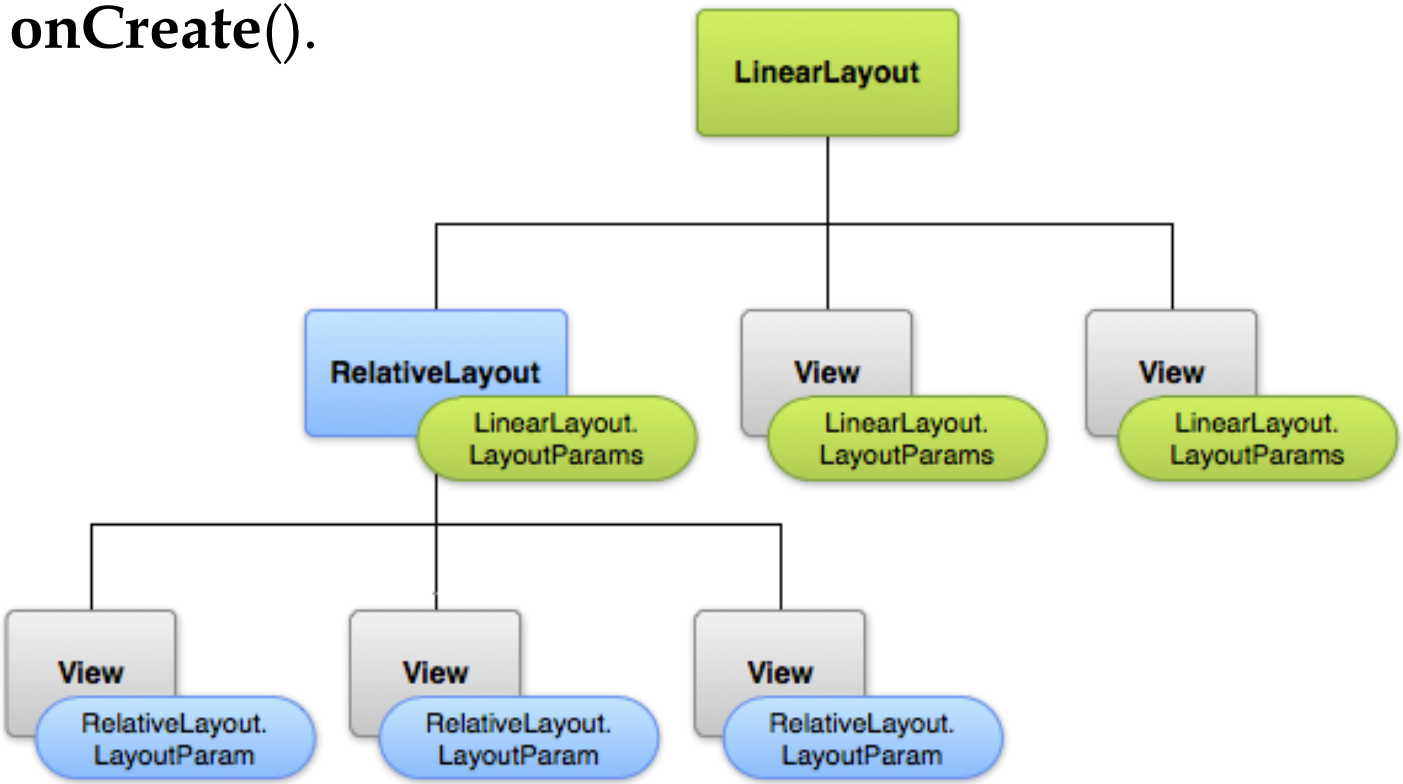


Layouts (1)

- Archivo XML donde se define la estructura de una interfaz de usuario.
- Puede ser de una Actividad o Widget / User Control.
- Cumple el rol de separar la definición de la Interfaz del código Java.
- Típicamente, con Java se maneja el comportamiento o interacciones con otros sistemas.
- Al especificar la capa de presentación en archivos externos, es posible escribir código independiente de la plataforma / pantalla y proveer varios Layout para cada caso específico.
- Obligatorio: un nodo *View* o *ViewGroup* como raíz del XML de Layout.

Layouts (2)

- El Layout de una aplicación se compila en un objeto de tipo *View*.
- Luego hay que cargarlo manualmente en el código en el método **onCreate**().



Layouts (3)

- Atributos de View
  - **ID**: identifica de forma única el View dentro de la jerarquía del Layout. Es utilizado para asignar (bind) a los objetos View correspondientes en clases Java.
  - **Atributos** de un control View que son "enviados" a su contendor (ViewGroup): posición dentro del layout / Tamaño / Padding / Márgenes
  - **match_parent**: la vista pide ser tan grande como su contenedor, menos los paddings del mismo si los tiene. Desde el API level 8.
  - **wrap_content**: la vista pide ser lo suficientemente grande para ajustar a su propio contenido. El propio control debe tomar Padding en consideración. Desde API level 1.

Layouts (4)

- Tipos:
  - **LinearLayout**:
    - Alinea a sus “hijos” de forma lineal.
    - Vertical u Horizontal (Atributo android:orientation)
    - Los hijos se ordenan de acuerdo al orden en que se agregan al Layout.
    - LinearLayout.LayoutParams

  - **RelativeLayout**:
    - Posiciona a sus “hijos” en posiciones relativas entre ellos o al contenedor.
    - Relativa entre “hijos” (toLeftOf / below / etc.)

Layouts (5)

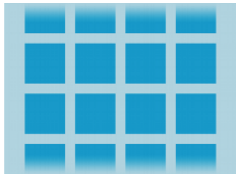
- Tipos (cont):
  - **RelativeLayout (cont.)** :
    - Relativa entre contenedores (alignParentbottom / centerVertical / etc.)
    - RelativeLayout.LayoutParams
    - **android:layout_alignParentTop**: con valor "true", hace coincidir el extremo superior de esta vista con el extremo superior del contenedor.
    - **android:layout_centerVertical**: con valor "true", centra esta vista verticalmente dentro de su contenedor.
    - **android:layout_below**: posiciona el extremo superior de esta vista debajo de la vista con un ID dado.
    - **android:layout_toRightOf**: posiciona el extremo izquierdo de esta vista a la derecha de la vista con un ID dado.

Layouts (6)

- Tipos (cont):
  - **ListView**:

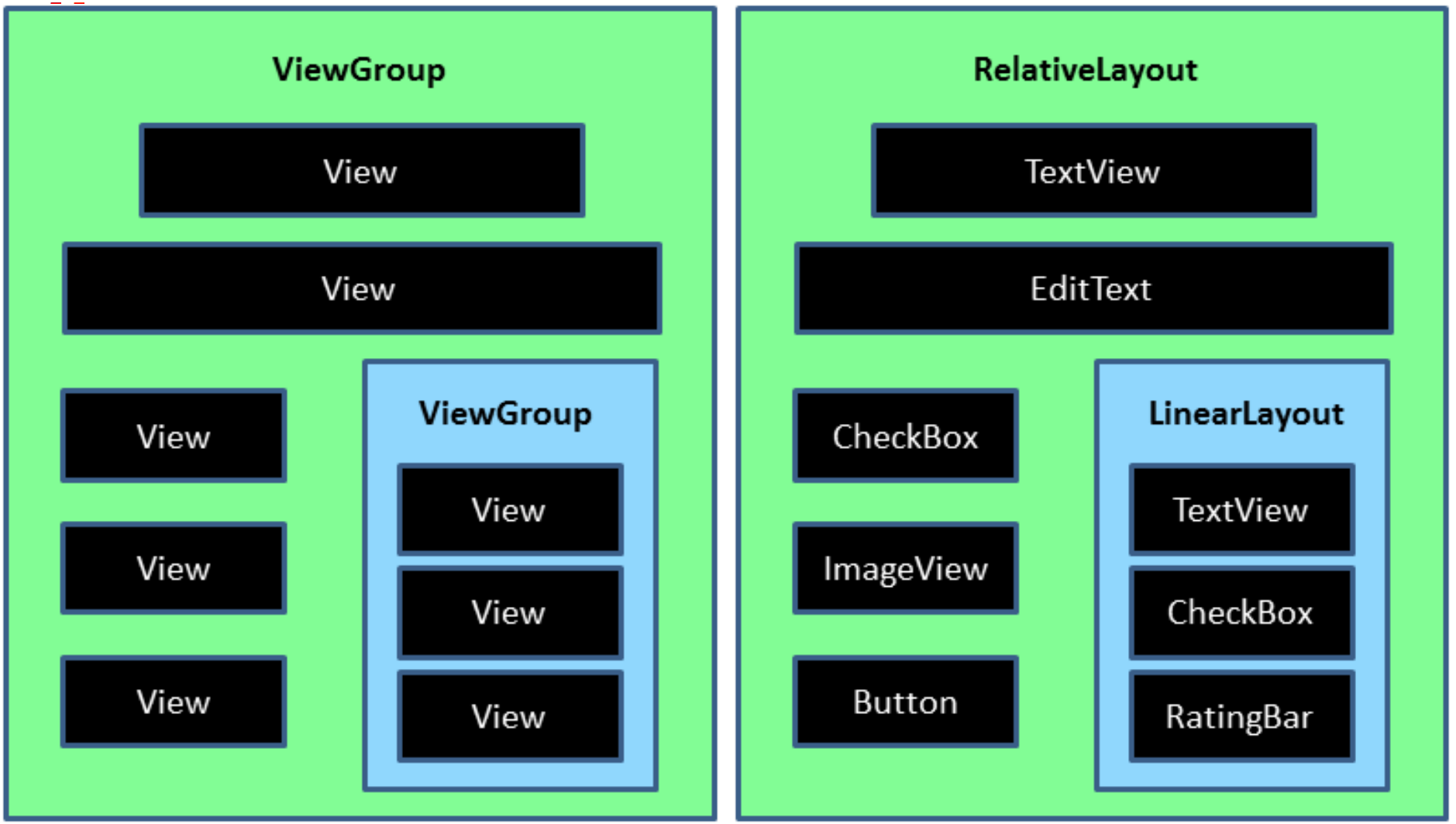
    - Grupo que despliega una lista navegable (scrolleable).
    - Los elementos son insertados automáticamente cuando se utiliza un Adapter para proveer los datos a la lista.

  - **GridView**:



    - Despliega elementos en una grilla navegable (scrolleable) de 2 dimensiones.
    - Los elementos son insertados automáticamente cuando se utiliza un Adapter para proveer los datos a la lista.

Layouts (7)


ViewGroup
View
View
View
View
View
ViewGroup
View
View
View
RelativeLayout
TextView
EditText
CheckBox
ImageView
Button
LinearLayout
TextView
CheckBox
RatingBar

Controles (1)

- **Button**
  - Clase: ***Button***
  - Responder a eventos de click:
    - Mediante onClick en el layout.
    - Mediante un manejador del evento
  - Estilar un botón dependiendo de su estado (presionado o no):
    - Crear una lista de estados : (selector) con bitmaps / (drawables).
    - Ubicarla en res/drawable.
    - Aplicar el estilo en el botón.

Controles (2)

- **Textfields**
  - Clases: *EditText* y *AutoCompleteTextView*
  - Permite al usuario ingresar texto.
  - Puede ser de una línea o multilínea.
  - Atributo para tipo de texto (inputType):
    - Permiten mostrar un teclado especifico.
    - Además, comportamiento dependiendo del uso del teclado.
    - **text**: teclado normal.
    - **textEmailAddress**: teclado normal con la @ ya visible.
    - **textUri**: teclado normal con la / ya visible.
    - **number**: teclado numérico.

Controles (3)

- **Textfields** (cont.)
  - Atributo para tipo de texto (inputType) (cont.):
    - **phone**: teclado numérico para discar.
    - **textCapSentence**: inicio en mayúscula cada nueva línea.
    - **textCapWord**: inicia en mayúscula cada palabra (por ej.: nombres).
    - **textAutoCorrect**: teclado normal con sugerencias de corrección en base a idioma.
    - **textPassword**: teclado normal pero mostrando caracteres con puntos.
    - **textMultiLine**: permite ingreso de varias líneas en la caja de texto.

Controles (4)

- **Checkbox**
  - Clase **CheckBox**
  - Selector de tipo si/no
  - Se presenta al usuario un grupo de opciones no mutuamente excluyentes
- **Radio button**
  - Clases **RadioGroup** y **RadioButton**
- **Toggle button**
  - Clase **ToggleButton**
  - Selector de tipo botón activado/desactivado
- **Spinner**
  - Clase **Spinner**
  - Lista desplegable de selección simple

Controles (5)

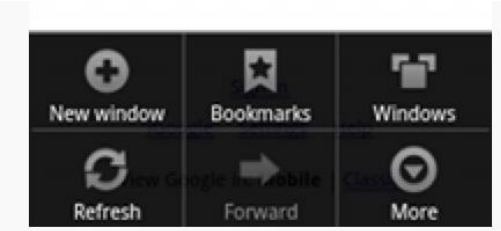
- **Pickers**
  - Clases **DatePicker** y **TimePicker**
  - Controles específicos para selección de fecha u hora.
  - Se recomienda la utilización de diálogos para contener los pickers (clase **DialogFragment**).
  - El diálogo maneja el ciclo de vida del control Picker.
  - Ajusta la visualización a diferentes escenarios (los muestra en dialogo en ciertas situaciones -teléfonos antiguos- o embebido en el layout principal en otras tablets).
  - Para buena compatibilidad hacia atrás (anterior a 3.0) utilizar la clase **DialogFragment** que se encuentra en la biblioteca de soporte.

Controles (6)

- Manejo del Foco
  - Android maneja movimientos de foco en respuesta a acciones del usuario.
  - **isFocusable**() - Los controles declaran al contenedor si toman foco.
  - **setFocusable**() / **setFocusableInTouchMode**() - Property / método que se deberá modificar para ajustar al comportamiento deseado
  - Navegación en ciclos entre controles 'top' y 'bottom'.
  - Navegar hacia arriba del primer botón no tiene ningún efecto.
  - Navegar hacia abajo del ultimo tampoco tiene ningún efecto.

Menu (1)

- Acciones relevantes en el contexto de la actividad.
- Su ubicación depende del API Level para el que se desarrolle.
  - 2.3.x (API Level 10) o inferior: aparece en la sección inferior luego de presionar la tecla de menú (hasta 6 opciones visibles, si hay más hay que presionar “Más”).
  - 3.0 (API Level 11) o superior: Aparece en la barra de acciones arriba a la derecha. Acciones (ítems) más relevantes: ***android:showAsAction=“ifRoom”.*** Acción de “Más” si no hay espacio suficiente.



Menu (2)

- Se pueden agregar acciones a un menú de opciones tanto desde una *Activity* como desde un *Fragment*.
- Se debe redefinir **onCreateOptionsMenu**:
  - **onCreateOptionsMenu** (**Menu**) en actividades.
  - **onCreateOptionsMenu** (**Menu**, **Inflater**) en fragmentos.
- Para manejar eventos del menú, redefinir ***onOptionsItemSelected*(*MenuItem*)**.
- Se debe identificar si la opción de menú seleccionada debe ser manejada. Si es manejada, retornar true. En caso contrario se debe llamar al súper método para no impedir que otro pueda manejarlo.

Menu (3)

- **Contextuales**
  - Acciones para un ítem o contexto específico.
  - Técnicamente, disponibles para cualquier tipo de *View*.
  - En la práctica, se utilizan en ítems de Listas, Grillas, o colecciones en general donde es posible realizar acciones sobre un ítem en particular.
  - Dos formas de proveer acciones:
    - **Menú flotante**: "clic largo" (presionar y sostener). Popup similar al "botón derecho" en aplicaciones de escritorio.
    - **Barra de acciones contextuales**: Al seleccionar un ítem, se despliega una barra de acciones contextual. Permite múltiple selección (si la aplicación lo implementa).

Menu (4)

- **Contextuales** (cont.)
  - Ya sea desde una actividad o un fragmento, se debe redefinir el método **onCreateContextMenu**() en el momento en que el usuario hace un clic largo.

Diálogos (1)

- Se utilizan cuando se requiere confirmación del usuario cuando se necesita información del usuario.
- No llenan la pantalla.
- Diálogos modales (acción previo a continuar).
- Varios tipos: **AlertDialog**, **DatePickerDialog**, **ProgressDialog**, **TimePickerDialog**.
- Deben ser utilizados con un ***DialogFragment*** como contenedor de los mismos.
  - Manejo del ciclo de vida del control.
  - Cambios en layout (rotar).
  - Decide como se despliega (popup en teléfono, panel embebido en tablets).

Diálogos (2)

- La clase **AlertDialog** nos permite construir varios tipos de diálogos comunes. Proporciona ayudas para agregar botones comunes (Ok, Cancel, etc.)
- Para setear Layouts definidos por el usuario a un Diálogo: **AlertDialog.Builder.setView**().
- Variantes al mostrar diálogos: a pantalla completa o embebido. Posible porque un **DialogFragment** extiende de ***Fragment***. Se requiere:
  - Layout definido en XML (no utilizando *ActionDialog.Builder*).
  - Cargar el Layout en el método **onCreateView**().
  - Manejar la variable por la cual se decidirá si mostrar a pantalla completa o embebido (recurso bool con alternativas, obviamente en base al tamaño del display).