Tema 2

Interfaces de usuario

Lo que abordaremos…

- Introducción
- Actividades
- Layouts
- Componentes de la interfaz
- Gestión de eventos de usuario

Introducción

- Características de la interfaz gráfica de Android:
  - Potente
  - Intuitiva para el usuario final
  - Fácil de implementar para el desarrollador
- Se incluyen una gran cantidad de componentes prediseñados y listos para ser usados
- Organización jerarquica de los componentes *(layouts* o *view groups* y controles o *views)*

Introducción

- Android nos brinda dos formas, compatibles entre sí, de programar interfaces de usuario
  - **Programación Java imperativa:** misma filosofía que programación en Java (incluso también en C#)
  - **Definición XML declarativa:** se define la estructura jerárquica de una pantalla y sus componentes mediante sencillos XML

Actividades

- Una actividad corresponde a una pantalla que en algún momento podrá ser visualizada por el usuario
  - Una app estará compuesta por 1 o más actividades
  - Se crean derivando una clase de *android.app.Activity*
- Se integran diferentes componentes (derivados de *android.view.View*)
  - Botones, cajas de texto, listas, etc
- Para organizar o agrupar dichos componentes se utilizan *Layouts* o *View Groups* (derivados de *android.view.ViewGroup*)
- Eclipse incluye un diseñador gráfico de actividades
- Explicar con un ejemplo como se crea una app y una acitividad

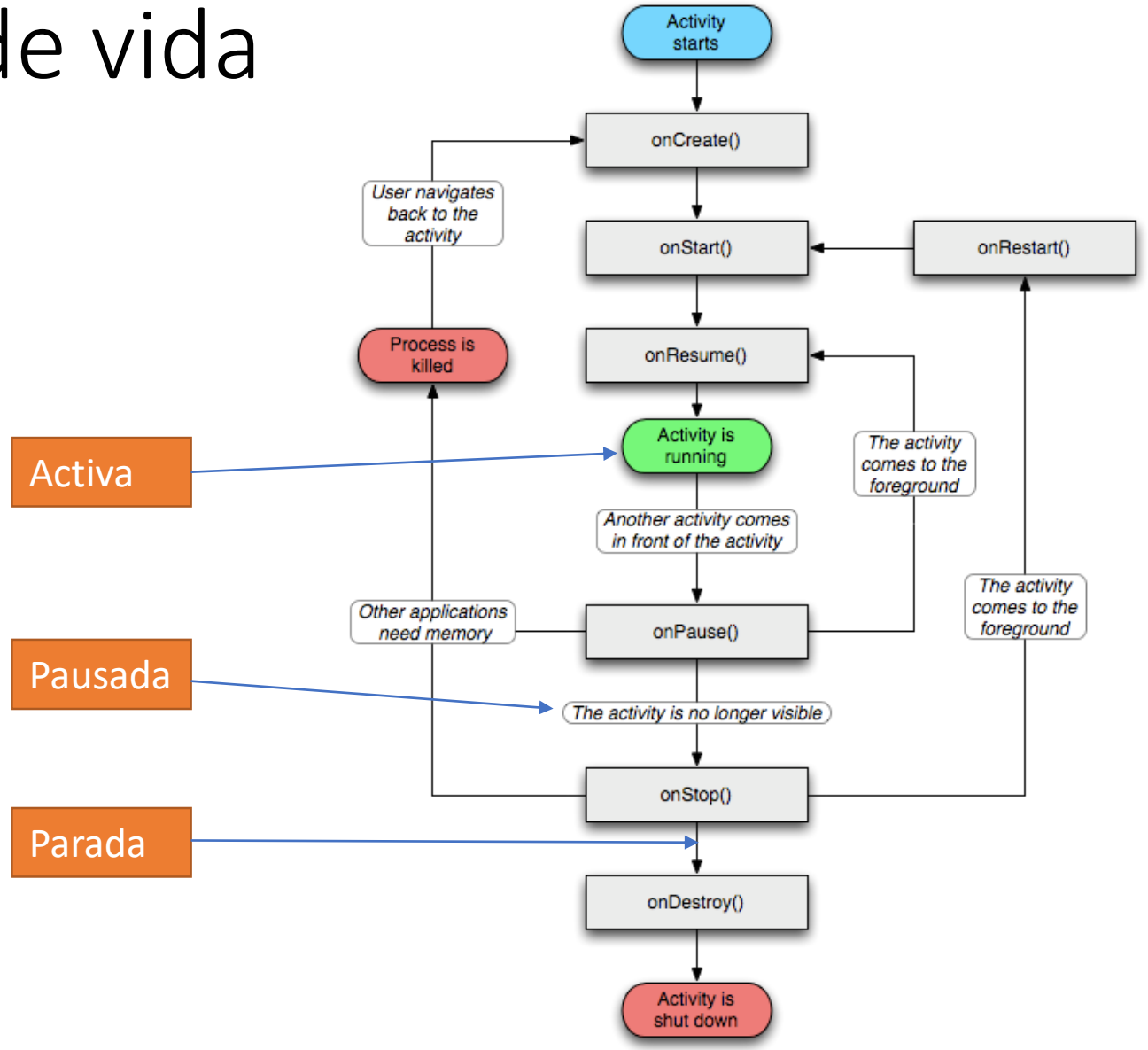
Actividades

- El archivo *AndroidManifest.xml*
  - Contiene aspectos comunes a toda la aplicación
  - Descriptor de la misma app
  - Eclipse presenta editor gráfico e intuitivo, también en código fuente
- Explicar componetes mediante un ejemplo
  - Manifest
  - uses-sdk
  - appication
  - Activity
  - Intent-filter

Actividades
Ciclo de vida

- Mostrar una nueva actividad implica detener otra actividad
  - La actividad anterior está disponible pulsando el botón *Back* del dispositivo
  - Dicha actividad no se destruye al pasar a segundo plano
  - El sistema destruye actividades cuando carece de memoria, no se puede predecir cuando sucederá
- Estados de una actividad
  - **Activa:** la actividad se encuentra visible por el usuario
  - **Pausada:** ha perdido el foco pero está visible por el usuario, la nueva actividad es parcialmente transparente o no ocupa la totalidad de la pantalla (diálogos o avisos)
  - **Parada:** la actividad no está visible. Está en memoria o fue eliminada por el sistema

Actividades
Ciclo de vida



Actividades
Ciclo de vida - métodos

- **onCreate:** llamado cuando la actividad se crea por primera vez. Se inicializan vistas y datos
  - Obligatorio sobreescribir
  - Recibe un parámetro *Bundle* que vale null si se ejecuta por primera vez, sino contiene info para restaurar la actividad
- **onRestart:** se invoca cuando la actividad había sido detenida previamente, pero se quiere volver a visualizar.
  - La app no ha sido destruida
- **onStart:** se ejecuta justo antes de que la actividad sea visualizada
  - Hay ocasiones en que la actividad no se visualiza finalmente
- **onResume:** se invoca justo antes de que la app empiece a interactuar con el usuario
  - La actividad ya se encuentra visualizada
  - Lugar idóneo para inicializar datos

Actividades
Ciclo de vida - métodos

- **onPause:** llamado cuando la app dejará el foco porque otra actividad será visualizada
  - Apropiado para salvar el estado de la app
  - Debe ser rápido, porque detiene la visualización de la nueva actividad
- **onStop:** invocado cuando la app. ha sido ocultada por completo
  - Puede que la app. sea destruida sin invocar este método por escases de recursos
- **onDestroy:** se invoca para destruir completamente la actividad .
  - Igual que el anterior, puede que no se llegue a invocar

Actividades
Guardar y restaurar estado

- Es muy importante guardar el estado en el que quedó la app.
  - Cuando el usr. vuelva a ella no es iniciada de cero
- Para este cometido pueden usarse *onPause* y *onResume*, pero también se ofrecen dos métodos que se pueden sobreescribir
  - **onSaveInstanceState (Bundle outState):** guarda todo lo necesario para recuperar posteriormente el estado de la actividad
  - **onRestoreInstanceState(Bundle savedInstanceState):** se invoca para recuperar el estado salvado anteriormente

Layouts

- Corresponden a las formas de organizar los elementos de la interfaz de usuario.
- Se trata del mismo concepto que apps. Java de escritorio
- Los más comunes son:
  - **LinearLayout:** es el más simple de todos. Dispone los elementos uno detrás de otro. Puede ser horizontal o vertical
  - **TableLayout:** divide la pantalla en filas y columnas
  - **RelativeLayout:** los elementos se ubican dada una posición relativa a otro elemento del mismo *layout*
  - **FrameLayout:** ubica los elementos alineados sobre la esquina superior izquierda de manera que unos controles quedarán encima de otros. Útil para cambio dinámico de elementos
  - **ScrollView:** es un tipo especial de *FrameLayout* utilizado cuando la cantidad de elementos desborda el espacio físico de la pantalla, siendo necesario un scroll vertical
- Derivados de la clase *View* o *ViewGroup*

Layouts
Atributos comunes

Atributo	Descripción
layout_width	Anchura
layout_height	Altura
layout_marginTop	Margen adicional en la parte superior
layout_marginBottom	Margen adicional en la parte inferior
layout_marginLeft	Margen adicional en la parte izquierda
layout_marginRight	Margen adicional en la parte derecha
layout_gravity	Orientaciones vertical y horizontal
layout_weight	Proporción del espacio disponible que ocupará el elemento
layout_x	Posición horizontal dada una coordenada sobre el eje x
layout_y	Posición vertical dada una coordenada sobre el eje y

Componentes de la interfaz

- Una actividad se compone de todo tipo de controles o *widgets* llamados *Views* en Android.
- La clase *View* es la clase base de todos estos Widgets
  - Button, EditText, TextView, etc

Componentes de la interfaz

CLASE	DESCRIPCIÓN
android.widget.AnalogClock	Reloj analógico con horas y minutos
android.widget.Button	Dibuja un botón de comando con texto
android.widget.CalendarView	Dibuja un calendario, configurable por XML
android.widget.CheckBox	Controles tipo check
android.widget.ChekedTextView	Texto con imagen Check
android.widget.DatePicker	Selector de fechas
android.widget.DigitalClock	Reloj digital
android.widget.EditText	Caja de texto editable
android.widget.ImageButton	Botones con imagen
android.widget.ImageView	Cuadro de imagen

Componentes de la interfaz

Clase	Descripción
android.widget.ProgressBar	Información de progreso
android.widget.RadioButton	Selector tipo radio
android.widget.SeekBar	Barra de progreso SeekBar
android.widget.Spinner	Lista desplegable Spinner (ComboBox)
android.widget.ListView	Lista scrollable de componentes
android.widget.TextView	Texto simple

Gestión de eventos de usuario

- **Eventos:** acción del usuario sobre el dispositivo o sobre uno de los controles de la pantalla.
  - Pulsación, escribir caracteres, arrastrar elementos, etc
  - En ocasiones son encapsulados en objetos descendientes de la clase *android.view.InputEvent*
- **Listeners o escuchadores:** componentes que se registran para recibir y tratar eventos
  - Generalmente existirá un listener para cada evento
  - Están representados como interfaces abstractas que obligan al desarrollador a implementar el método que recibe el evento a capturar
- **Método para registrar los listener:** la clase *View* contiene métodos para registrar escuchadores para los eventos que se produzcan sobre un control
  - Por ejemplo, existe el método *setOnClickListener* que permite registrar un listener de tipo *OnClickListener* para capturar los eventos generados al pulsar sobre el componente al que se aplique

Gestión de eventos de usuario
Eventos más comunes

Evento	Listener	Descripción
onClick()	OnClickListener	Pulsación sobre un componente
onLongClick()	OnLongClickListener	Pulsación durante un período de tiempo
onKey()	OnKeyListener	Se genera cuando un componente tiene el foco y se pulsa una tecla del dispositivo
onTouch()	OnTouchListener	Generado cuando el usuario ejecuta una “acción” touch sobre la pantala, por ejemplo presionar la pantalla, arrastrar elementos, gestures, etc
onDrag()	OnDragListener	Producido al realizar la acción de arrastrar un elemento
onFocusChange()	OnFocusChangeListener	Se ejecuta cuando un componente recibe o pierde el foco

Gestión de eventos de usuario
Eventos más comunes

- Esos eventos son los más comunes y genéricos, pero existen muchos más, incluso algunso específicos de un tipo de componente, por ejemplo
  - **onCheckedChanged():** específico de *CheckBox*. Controlado por *OnChekedChanged.* Gestiona el evento de *chekear* o eliminar la selección de este tipo de componentes
  - **onItemClick():** específico de *ListView.* Controlado por *OnItemClickListener.* Se lanza al pulsar sobre un elemento de este componente
  - **onGesturePerformed():** frecuentemente generdo en vistas tipo *GestureOverlayView* y controlado por *OnGesturePerformedListener*. Referente a gestos realizados sobre la pantalla táctil. Dependiendo del gesto realizado se podrán llevar a cabo unas acciones u otras

Gestión de eventos de usuario

- Algunas formas de cómo manejar eventos
  - **Construir escuchadores de eventos exteriores a la propia clase**
    - Crear una nueva clase que implemente al *listener* reescribiendo sus métodos heredados. En este caso implementaría a la clase *OnClickListener*
  - **Construir clases internas (*inner class*)**
    - Consiste en escribir la clase que implementa la interfaz del *listener* dentro de la clase que registra el evento
  - **Implementar el listener en la propia clase**
    - En este caso la propia clase que registra el evento, será quien implemente la funcionalidad del *listener*
  - **Declarar clases anónimas**
    - No necesita de la creación de una nueva clase, sino que se declara de forma anónima dentro de la propia asignación del *listener*
- Mostrar a través de un ejemplo

Gestión de eventos de usuario

- Usando eventos para abrir nuevas pantallas
  - Mostrar a través de un ejemplo