

# Programación para Dispositivos Móviles(T.E.)

Ciclo I - 2020

Clase:006

Coordinador:Ing. César González



**Android**  
Studio



OBJECTIVE-C

# Unidad II

## Interfaz Grafica (continuación ...)

Basado en clases de  
Ing. Carlos A. Aguilar

# Agenda

- **Desarrollo de Apps para Android**
  - **Eventos(continuacion)**
  - **Menus**
  - **Menus contextuales**
  - **Elementos de Aplicacion en Android**
    - **Services**
    - **Broadcast Receiver**
    - **Activity**
    - **Content Provider**
  - **Intent**





## Eventos(continuación ...)

Los eventos que posee un View y las respectivas interfaces son:

- `onClick()` - `View.OnClickListener`. Este evento es disparado cuando el usuario toca el View.
- `trackball.onLongClick()` - `View.OnLongClickListener`. Este evento es disparado cuando el usuario toca un View por un tiempo determinado.
- `onFocusChange()` - `View.OnFocusChangeListener`. Este evento es disparado cuando el usuario da el focus o lo quita de un View por medio de las teclas de navegación o el trackball



## Eventos(continuación ...)

- `onKey()` - `View.OnKeyListener`. Este evento es lanzado cuando un `View` tiene el focus y el usuario presiona una tecla en su teclado físico o virtual.
- `onTouch()` - `View.OnTouchListener`. Este evento es llamado cuando el usuario realiza una acción que es calificado como un toque o realizar un gesto.
- `onCreateContextMenu()` - `View.OnCreateContextMenuListener`. Es llamado cuando un menú contextual esta a punto de ser creada como reacción a un click prolongado.



# Menús

La mayoría de aplicaciones en Android implementan Menús y son importantes porque exponen las funciones de la aplicación al usuario

- Los Menús pueden ser de 3 tipos:
- Menús de opciones: Llamados cuando el usuario presiona la tecla Menú en su dispositivo.
- Menús Contextuales: Son llamados cuando el usuario mantiene presionado un elemento de la UI. Similar al click derecho en la pc
- Submenús: Un submenú que es mostrado cuando el usuario presiona un elemento del menú de opciones.



# Menús

- Al igual que los layouts y otros recursos, los menús son creados y definidos en Android por medio de XML.
- Los menús se agrupan en la carpeta `/res/menu/`



# Menús

- Un archivo XML con una definición de un menú luce así:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game" />
  <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```





# Menús

- Desde el código fuente de la activity es necesario implementar la apertura del menú por medio de un Inflater(Inflador)
- El siguiente ejemplo crea un menú de opciones

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //Callback de menu
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```



# Menús

- Para capturar la selección del usuario, Android notifica a la activity desde la cual se inflo el Menu con el item que el usuario selección a traves de un Callback

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Declara el comportamiento para cada ítem del menu.
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            // Si ninguno el id del menú seleccionado no es resuelto por nuestra actividad. Lo
            // pasamos a la clase padre. Si el padre es Activity regresara el valor false.
            return super.onOptionsItemSelected(item);
    }
}
```



# Menús contextuales

Los menús contextuales se añaden a una View y deben ser registrados en la Activity por medio del método `registerForContextMenu()`

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //Obtenemos las referencias a los controles
    lblMensaje = (TextView)findViewById(R.id.textView1);

    //Asociamos los menús contextuales a los controles
    registerForContextMenu(lblMensaje);
}

```



# Menús contextuales

- Cuando el usuario mantiene presionado un View, Android llama al callback onCreateContextMenu, así:

```
,  
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
                               ContextMenuInfo menuInfo)  
{  
    super.onCreateContextMenu(menu, v, menuInfo);  
  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.contextual, menu);  
}
```



# Menús contextuales

Luego cuando el usuario selecciona un elemento de un View, se llama al callback `onContextItemSelected`.

```
@Override
public boolean onContextItemSelected(Menuitem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```



# Dialogos

Un Dialogo es una pequeña ventana al frente de una actividad

- Para abrir un Dialogo este debe ser llamado desde una actividad por el método `showDialog(int)`; el entero que se pasa como parámetro al método es un numero que identifica al dialogo inequívocamente.
- Este método provoca que Android llame al Callback `onCreateDialog(int)` el cual es una implementación definida por el programador para cargar los diálogos.
- Para cerrar el dialogo llamamos al metodo `dismiss()` desde el dialogo o por medio de `dismissDialog(int)` para cerrarlo desde cualquier punto de la actividad.



# Dialogos

Ejemplo, si una activity posee 2 diálogos, cuya identificación es declarada así:

```
static final int DIALOG_PAUSED_ID = 0;  
static final int DIALOG_GAMEOVER_ID = 1;
```

La implementación del callback onCreateDialog, quedaría así:

```
protected Dialog onCreateDialog(int id) {  
    Dialog dialog;  
    switch(id) {  
        case DIALOG_PAUSED_ID:  
            // do the work to define the pause Dialog  
            break;  
        case DIALOG_GAMEOVER_ID:  
            // do the work to define the game over Dialog  
            break;  
        default:  
            dialog = null;  
    }  
    return dialog;  
}
```



# Dialogos

Los Diálogos pueden mostrar diferentes opciones y existen varias formas de personalizarlos.

- La forma mas simple es usando una implementación de AlertDialog.
- Un AlertDialog permite agregar botones, títulos, texto, uno, dos o tres botones, una lista, opciones, etc.
- Para crear un AlertDialog utilizamos la clase AlertDialog.Builder





# Dialogos

El Código Alert Dialog con Botones es como se muestra:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Esta seguro que desea salir?")
    .setCancelable(false)
    .setPositiveButton("Si", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            MyActivity.this.finish();
        }
    })
    .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
AlertDialog alert = builder.create();
```



# Dialogos

El Código Alert Dialog con una lista es como se muestra:

```
final CharSequence[] items = {"Rojo", "Verde", "Azul"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Seleccione un Color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
        Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```



# Dialogos

El Código Alert Dialog con una lista seleccionable es como se muestra:

```
final CharSequence[] items = {"Rojo", "Verde", "Azul"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Seleccione un Color");
builder.setSingleChoiceItems(items, -1, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```



# Elementos de Aplicacion: Services

Los servicios (*service*) son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son similares a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. actividades) si se necesita en algún momento la interacción con del usuario.



# Broadcast Receiver

Un *broadcast receiver* es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: “Batería baja”, “SMS recibido”, “Tarjeta SD insertada”, ...) o por otras aplicaciones (cualquier aplicación puede generar mensajes (*intents*, en terminología Android) broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).



# Activity

Las actividades (*activities*) representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana o pantalla en cualquier otro lenguaje visual.



# Intents

3 de los 4 componentes principales de una aplicación (Activities, Services y Broadcast receivers) son activados mediante mensajes

- Estos mensajes son llamados Intents
- El mecanismo de Intents permite que podamos abrir elementos en nuestra aplicación u elementos en otra aplicación.
- Un intent es una abstracción de una acción que se desea realizar junto con la descripción de información adicional.



# Intents

El mecanismo propio para enviar al Intent depende del tipo del componente.

- Por ejemplo, para abrir una activity podemos usar: `startActivity()` o `startActivityForResult()`
- Para iniciar un Servicio, utilizamos `Context.startService()` . Si lo que deseamos es conectarnos a un servicio para interactuar con el llamamos a `Context.bindService()`
- Para enviar un broadcast, utilizamos `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()` o `Context.sendStickyBroadcast()`





# Intents

- Android se encarga de entregar el intent a la actividad, servicio o broadcast que responde al intent.
- Los intents no se cruzan, es decir, siempre son entregados al tipo de componente que corresponde al mecanismo utilizado para enviarlo
- Por ejemplo, un intent enviado por `startActivity` jamas será enviado a un servicio o un broadcast reciever.



# ¿Qué contiene un Intent?

Los intents contienen varios tipos de información, que son útiles para que Android determine como realizar el envío de los intents a los receptores adecuados y también información que utilizara el receptor para procesar la mensaje

- Component Name:
  - Acá especificamos el nombre de la clase que queremos llamar. Podemos ocupar los objetos propios de la clase como `SecondActivity.class` o el nombre del paquete declarado en el manifiesto como `sv.ues.fia.Activities.SecondActivity`. Este parámetro puede ser nulo.



# ¿Qué contiene un Intent?

## Action:

- Una acción es algo que debe se necesita hacer(en el caso de activities) o la acción que sucedió(en el caso de broadcast receivers). Como por ejemplo:

Constante	Componente Target	Accion que realiza
ACTION_CALL	activity	Inciar una llamada
ACTION_EDIT	activity	Desplegar data para que sea editada por el usuario
ACTION_MAIN	activity	Iniciar la primera actividad de una tarea que no recibe parametros ni devuelve salidas
ACTION_SYNC	activity	Sincronizar data con un servidor que se encuentra en el dispositivo
ACTION_BATTERY_LOW	broadcast receiver	Adevertencia de que la bateria esta en un nivel bajo
ACTION_HEADSET_PLUG	broadcast receiver	Un set de manos libres o audifonos ha sido conectado o desconectado del dispositivo.
ACTION_SCREEN_ON	broadcast receiver	La pantalla ha sido encendida
ACTION_TIMEZONE_CHANGED	broadcast receiver	La zona horaria ha cambiado, por el usuario o la red.



# ¿Qué contiene un Intent?

- Se puede ver el listado completo de Actions predefinidos en el API de referencia de Android
- El programador puede definir sus propias Actions, sin embargo, solo podrán interactuar con los proyectos de los cuales conozcamos el nombre de los Actions.
- Las Actions permiten el remplazo de las apps del sistema.



# ¿Qué contiene un Intent?

Data:

- Es posible enviar data incluida en el mensaje para que pueda ser procesada por el receptor.
- La data puede contener valores, un URI y un MIME type
- El URI(Uniform Resource Identifier) especifica adonde esta la información almacenada. Los Uri pueden ser recursos web(http:), archivos(file:), Telefonos(tel:)
- El mime type identifica el tipo de contenido por ejemplo “text/\*” indicara que estas enviando un documento que contiene texto. “text/plain” texto plano, etc.
- Android verifica ambos datos a la hora de tomar la decisión de quien debe de responder a ese intent



# ¿Qué contiene un Intent?

## Category:

Es una cadena de texto que tipo de componente debe de responder a un Intent. Las 3 categorias mas utilizadas son:

- **CATEGORY\_LAUNCHER:** Indica que el receptor del intent debe de ser el Iniciador de una tarea. Las activities que tienen esta categoria en un intent-filter son mostrados en el listado de aplicaciones.
- **CATEGORY\_BROWSABLE:** Indica que la actividad puede recibir intents desde un navegador y procesar información de manera apropiada. Por ejemplo, correos electrónicos, imágenes, etc.
- **CATEGORY\_PREFERENCE:** El componente que recibira el intent es un panel de preferencias



# ¿Qué contiene un Intent?

Extras:

- Existen ciertas actions para las cuales es necesario proporcionar información adicional. Por ejemplo, para la action `ACTION_TIMEZONE_CHANGED` existe un extra que indica la nueva zona horaria.
- Los intents tienen varios metodos `putXXXX` y `getXXXX` para agregar diferentes tipos de datos al intent o recuperarlos. Los valores son almacenados por medio de una llave que identifica al recurso(Llave-Valor)



# Intent filters

Si se llama a un componente de manera explícita (por medio de su nombre de clase), Android sencillamente inicia el componente directamente sin importar el resto de la información del intent.

- En cambio, si lo que hemos especificado es un Action, Android debe de decidir cuales componentes cumplen con la información (si existe) que fue agregado en el Intent.
- Esta tarea es realizada analizando los Intent-Filters que han sido declarados para cada componente en sus Android Manifests respectivos.





# Intent filters

Los intent filters son los mecanismos para exponer las capacidades de una aplicación al sistema operativo. Con ellos se puede especificar que intents puede procesar nuestro componente.

- Una actividad, un servicio o un broadcast receiver puede tener varios intent-filters o ninguno.
- Si no posee ninguno, el componente solo puede ser llamado explícitamente.



# Intent filters

Los intent filters son los mecanismos para exponer las capacidades de una aplicación al sistema operativo. Con ellos se puede especificar que intents puede procesar nuestro componente.

- Una actividad, un servicio o un broadcast receiver puede tener varios intent-filters o ninguno.
- Si no posee ninguno, el componente solo puede ser llamado explícitamente



# Intent filters

Para entregar un intent a un componente, el intent debe de pasar todas las pruebas realizadas a todos los detalles especificados en el intent filter.

- Los intent filter pueden contener especificaciones acerca de las Action, Data y Categories que puede recibir.
- Como un componente puede tener varios intent-filter, si al analizar un intent este falla en pasar, otro puede pasar y el intent es entregado.



# Intent filters

- Un intent filter puede tener un solo action o varios. Pero no podemos dejar de especificar aunque sea uno, porque sino todos los intents son rechazados por el intent-filter.
- Para pasar la prueba de action, la action del intent debe de coincidir al menos con un action especificado en el intent-filter. Si el intent no tiene un action especificado, la prueba es pasada.
- Los action son declarados asi:

```
<intent-filter . . . >  
<action android:name="com.example.project.SHOW_CURRENT" />  
<action android:name="com.example.project.SHOW_RECENT" />  
<action android:name="com.example.project.SHOW_PENDING" /> . . .  
  
</intent-filter>
```



# Intent filters

Un intent-filter puede tener una o mas categorías establecidas y para que un intent pueda ser entregado, todas las categorias del Intent deben de existir en el Intent-Filter.

- Un intent sin categorías siempre pasa la prueba de categorías, a excepción de los que son llamados usando `startActivity()` ya que Android espera que una activity contenga al menos `android.intent.category.DEFAULT` para poder llamarla implícitamente.
- Lo anterior aplica para todas las activitys excepto aquellas que están marcadas para ser mostradas en el Launcher, ya que estas siempre inician una tarea nueva.



# Intent filters

- Las categorías son declaradas así:

```
<intent-filter . . . >
```

```
<category android:name="android.intent.category.DEFAULT" />
```

```
<category android:name="android.intent.category.BROWSABLE" />
```

```
. . .
```

```
</intent-filter>
```



# Intent filters

El otro elemento que se puede filtrar con un intent-filter es la data

- Este debe de coincidir con al menos una de las especificaciones data del intent-filter.

- Recordemos que la data es enviada a traves de Uris

```
<intent-filter . . . > <data android:mimeType="video/mpeg"
android:scheme="http" . . . /> <data
android:mimeType="audio/mpeg" android:scheme="http" . . . /> . .
. </intent-filter>
```