Desarrollo Android

Arkaitz Garro

Master en desarrollo de Apps Móviles

Action Bar

Introducción

Action Bar

La Action Bar ha sido ideada para ofrecer una UI consistente a lo largo de la aplicación (y entre aplicaciones).

Es posible incluir una imagen de marca, navegación, menús y acciones directas.

La versión mínima de SDK necesaria para utilizar Action Bar es la 11.

Personalizar Action Bar

Es posible personalizar el Action Bar, pero de una manera muy limitada, ya que se espera una consistencia.

Podemos modificar el color de fondo, el icono, el texto, añadir acciones y un menú.

Personalizar Action Bar

Por defecto, se muestra el icono de la aplicación, pero es posible personalizarlo para cada actividad, utilizando el atributo `android:logo`.

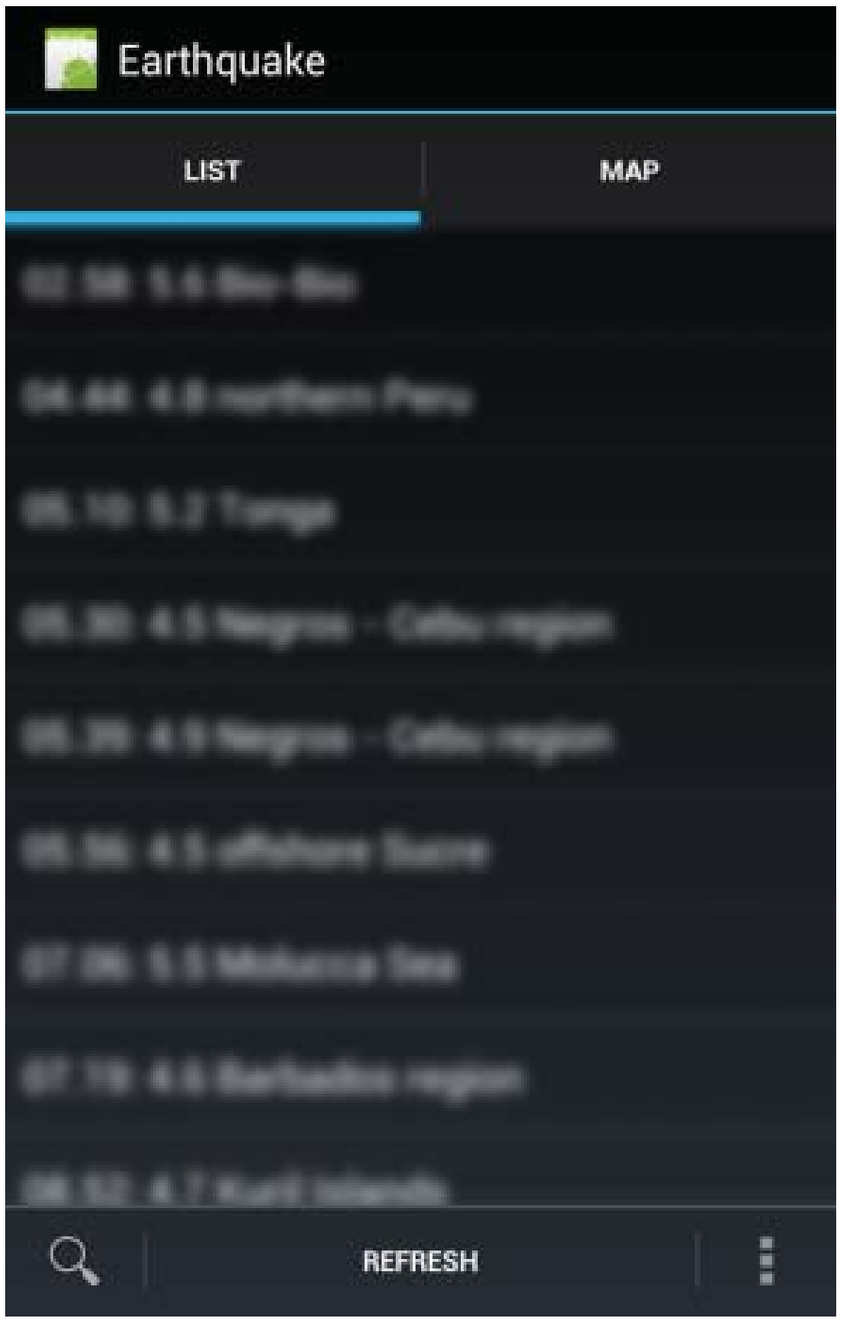
Por lo general, se utiliza este logo como acceso directo a la pantalla principal, por lo que podemos ocultar el título de la aplicación si no es necesario.

```
ActionBar actionBar = getActionBar();
actionBar.setDisplayShowTitleEnabled(false);
```

Sin embargo, puede resultar interesante utilizar este título para hacer referencia a la navegación.

```
actionBar.setSubtitle(“Inbox”);
actionBar.setTitle(“Label:important”);
```

Personalizar Action Bar



Tenemos la posibilidad de dividir la barra de acción.

En pantallas estrechas, Android la dividirá en diferentes secciones.

Es calculado en tiempo de ejecución, y se muestra la mejor configuración en base al espacio disponible.

```
android:uiOptions=”splitActionBarWhenNarrow”
```

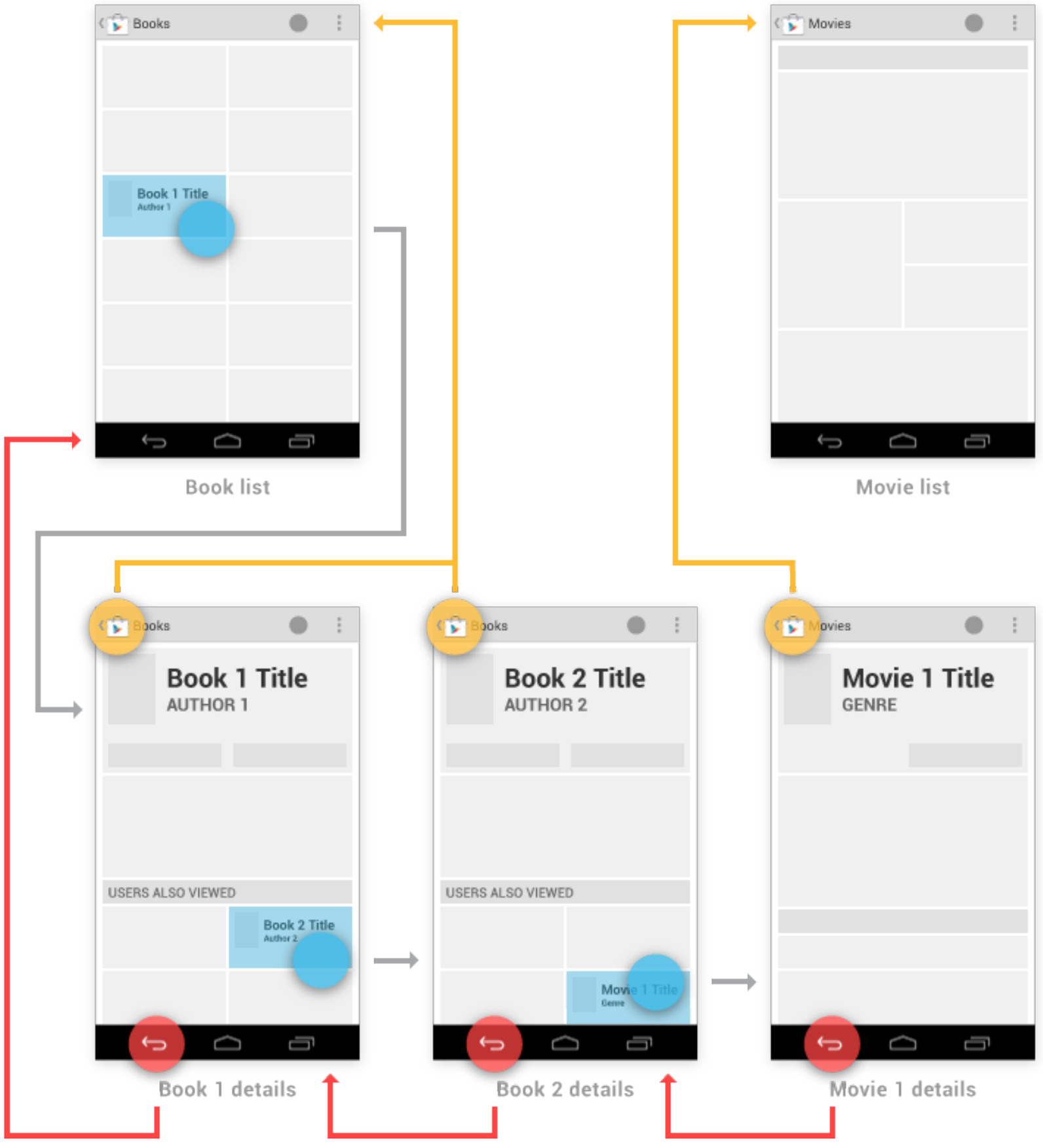
Navegación en Action Bar

La barra de acción ofrece la posibilidad de añadir elementos de navegación.

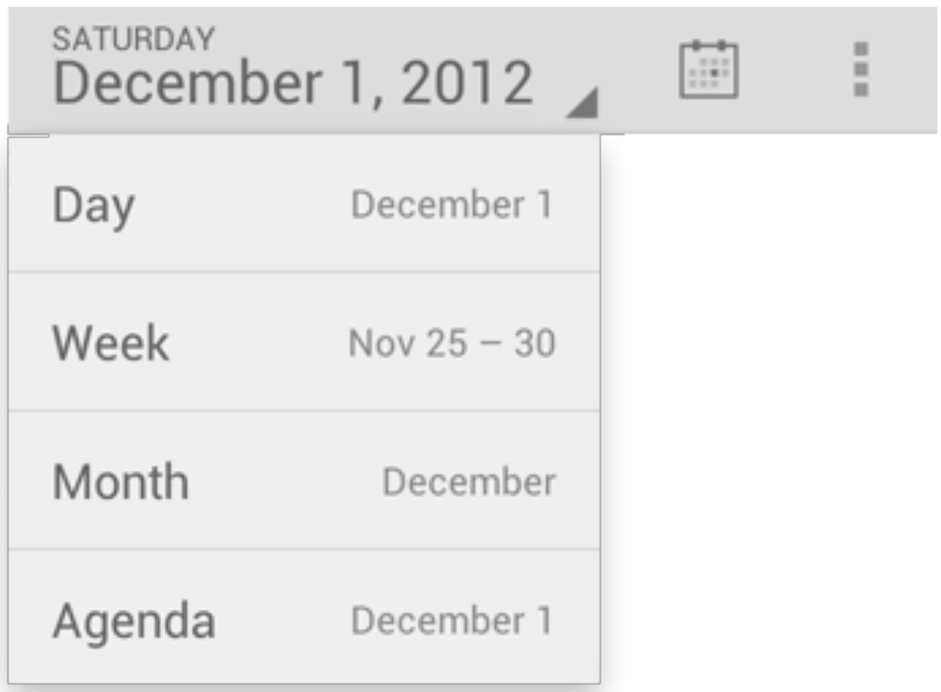
Podemos distinguir dos categorías:

Icono de aplicación: dirigido a ofrecer navegación con respecto a la estructura de la aplicación.

Pestañas y desplegables: dirigidos a cambiar la información a mostrar dentro de la actividad, pero sin cambiar su contexto.


Books
Book 1 Title
Author 1
Book list
Movies
Movie list
Books
Book 1 Title
AUTHOR 1
USERS ALSO VIEWED
Book 2 Title
Author 2
Book 1 details
Books
Book 2 Title
AUTHOR 2
USERS ALSO VIEWED
Book 2 details
Movies
Movie 1 Title
GENRE
Movie 1 details

TAB 1
TAB 2
TAB 3
TAB 1
TAB 2
TAB 3


SATURDAY
December 1, 2012
Day
December 1
Week
Nov 25 – 30
Month
December
Agenda
December 1

Navegación en Action Bar

En muchos casos, el icono actúa como botón de “Home”, permitiendo volver al inicio de la aplicación.

```
actionBar.setHomeButtonEnabled(true);
```

Al pulsar sobre el icono, Android lanza una notificación como si un ítem del menú hubiese sido lanzado.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case (android.R.id.home) :
            Intent intent = new Intent(this, ActionBarActivity.class);
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Navegación en Action Bar

En otros casos, puede ser interesante navegar a un nivel superior en la estructura de la aplicación.

```
actionBar.setDisplayHomeAsUpEnabled(true);
```

Al pulsar sobre el icono, Android lanza una notificación como si un ítem del menú hubiese sido lanzado.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case (android.R.id.home) :
            //Implementar la llamada al nivel superior
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Pestañas de navegación

Además del icono de navegación, Android nos ofrece la posibilidad de navegar a través de pestañas o listas desplegables.

Estos métodos de navegación están pensados para trabajar directamente con fragmentos.

Especificamos el método de navegación al crear la actividad.

```
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

Pestañas de navegación

El siguiente paso es crear las pestañas, y especificar los fragmentos asociados a cada una de ellas

```
Tab tabOne = actionBar.newTab();

tabOne.setText(“First Tab”)
    .setIcon(R.drawable.ic_launcher)
    .setContentDescription(“Tab the First”)
    .setTabListener(
        new TabListener<MyFragment>
            (this, R.id.fragmentContainer, MyFragment.class));

actionBar.addTab(tabOne);
```

Es necesario implementar el escuchador para el cambio de pestañas

```
public static class TabListener<T extends Fragment> implements
ActionBar.TabListener {

    private Fragment fragment;
    private Activity activity;
    private Class<T> fragmentClass;
    private int fragmentContainer;

    public TabListener(Activity activity, int fragmentContainer,
                       Class<T> fragmentClass) {
      this.activity = activity;
      this.fragmentContainer = fragmentContainer;
      this.fragmentClass = fragmentClass;
    }

// Called when a new tab has been selected
public void onTabSelected(Tab tab, FragmentTransaction ft) {
  if (fragment == null) {
    String fragmentName = fragmentClass.getName();
    fragment = Fragment.instantiate(activity, fragmentName);
    ft.add(fragmentContainer, fragment, fragmentName);
  } else
    ft.attach(fragment);
}
```

```
// Called on the currently selected tab when a different tag is
// selected.
public void onTabUnselected(Tab tab, FragmentTransaction ft) {
  if (fragment != null)
    ft.detach(fragment);
}

// Called when the selected tab is selected.
public void onTabReselected(Tab tab, FragmentTransaction ft) {
  if (fragment != null)
    ft.attach(fragment);
}
```

Listas desplegables

Otra manera de añadir navegación o filtros a la Action Bar es a través de listas desplegables.

```
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);

ArrayList<CharSequence> al = new ArrayList<CharSequence>();
al.add(“Item 1”);
al.add(“Item 2”);

ArrayAdapter<CharSequence> dropDownAdapter =
    new ArrayAdapter<CharSequence>(this,
        android.R.layout.simple_list_item_1,
        al);
```

Listas desplegables

Para manejar la selección del usuario, implementamos un escuchador.

```
// Select the drop-down navigation mode.
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);

// Create a new Spinner Adapter that contains the values to
// be displayed in the drop down.
ArrayAdapter dropDownAdapter =
  ArrayAdapter.createFromResource(this,
                                  R.array.my_dropdown_values,
                                  android.R.layout.simple_list_item_1);

// Assign the callbacks to handle drop-down selections.
actionBar.setListNavigationCallbacks(dropDownAdapter,
  new OnNavigationListener() {
    public boolean onNavigationItemSelected(int itemPosition,
                                            long itemId) {
      // TODO Modify your UI based on the selection
      return true;
    }
 });
```

Acciones

La parte derecha de la barra de acción es utilizada para añadir acciones directas y un menú.

Las acciones, son en realidad, elementos de menú lo suficientemente importantes como para acceder a ellos directamente.

Action Bar

Menús y acciones

Menús

Los menús son una manera de disponer de acciones ocultas, sin sacrificar espacio en pantalla.

Android dispone de un menú que se muestra en tres etapas:

Acciones en el Action bar: accesos directos a acciones representados por un icono. Las acciones que no entran en el Action Bar, se muestran en el `Menu Overflow`.

Menu Overflow: contiene todos los elementos que no han sido marcados como acción, y las acciones que no caben en la barra de acción.

Submenu: menú que se muestra en una ventana flotante.

Crear un menú

Para añadir un menú en la actividad, sobreescribimos el método `onCreateOptionsMenu`.

Disponemos del método `add` del objeto `Menu` para añadir nuevos elementos. Para cada elemento debemos especificar:

Identificador de grupo: permite agrupar los elementos del menú.

Identificador único del nuevo elemento, utilizado posteriormente para identificar el elemento sobre el que se ha pulsado.

Orden en el que se muestra el elemento en el menú.

El **texto** a mostrar por el elemento en el menú.

Elementos del Action Bar

Para indicar que un elemento en concreto debe posicionarse en la Action Bar, utilizamos su método `setShowAsActionFlags`, pasando alguno de estos parámetros:

`SHOW_AS_ACTION`: se muestra siempre como una acción.

`SHOW_AS_IF_ROOM`: se muestra sólo si hay espacio disponible.

Por defecto, las acciones únicamente muestran el icono. Para mostrar el texto, añadir el parámetro `SHOW_AS_ACTION_WITH_TEXT`.

Menús desde fragmentos

Al igual que el resto de elementos de la interfaz, podemos incluir elementos del menú dentro de los propios fragmentos.

Registramos el fragmento como contribuyente a la generación del menú:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}
```

Implementamos el método `onCreateOptionsMenu` en el fragmento para incluir los elementos del menú.

Crear menús desde XML

En lugar de construir el menú directamente en código, es conveniente definirlo en un XML.

Esto nos permite crear alternativas por dispositivo, idiomas, configuraciones...

Cada elemento del menú es definido como un nuevo nodo, pudiendo especificar todas las opciones disponibles para un `MenuItem`

```
<menu xmlns:android=”http://schemas.android.com/apk/res/android”>
  <item
    android:id=”@+id/action_view_item”
    android:icon=”@drawable/action_view_icon”
    android:title=”@string/action_view_title”
    android:showAsAction=”ifRoom|collapseActionView”
    android:actionLayout=”@layout/my_action_view”>
  </item>
</menu>
```

Modificar menús

Tenemos la posibilidad de modificar los menús dinámicamente, justo en el momento anterior a que sean mostrados.

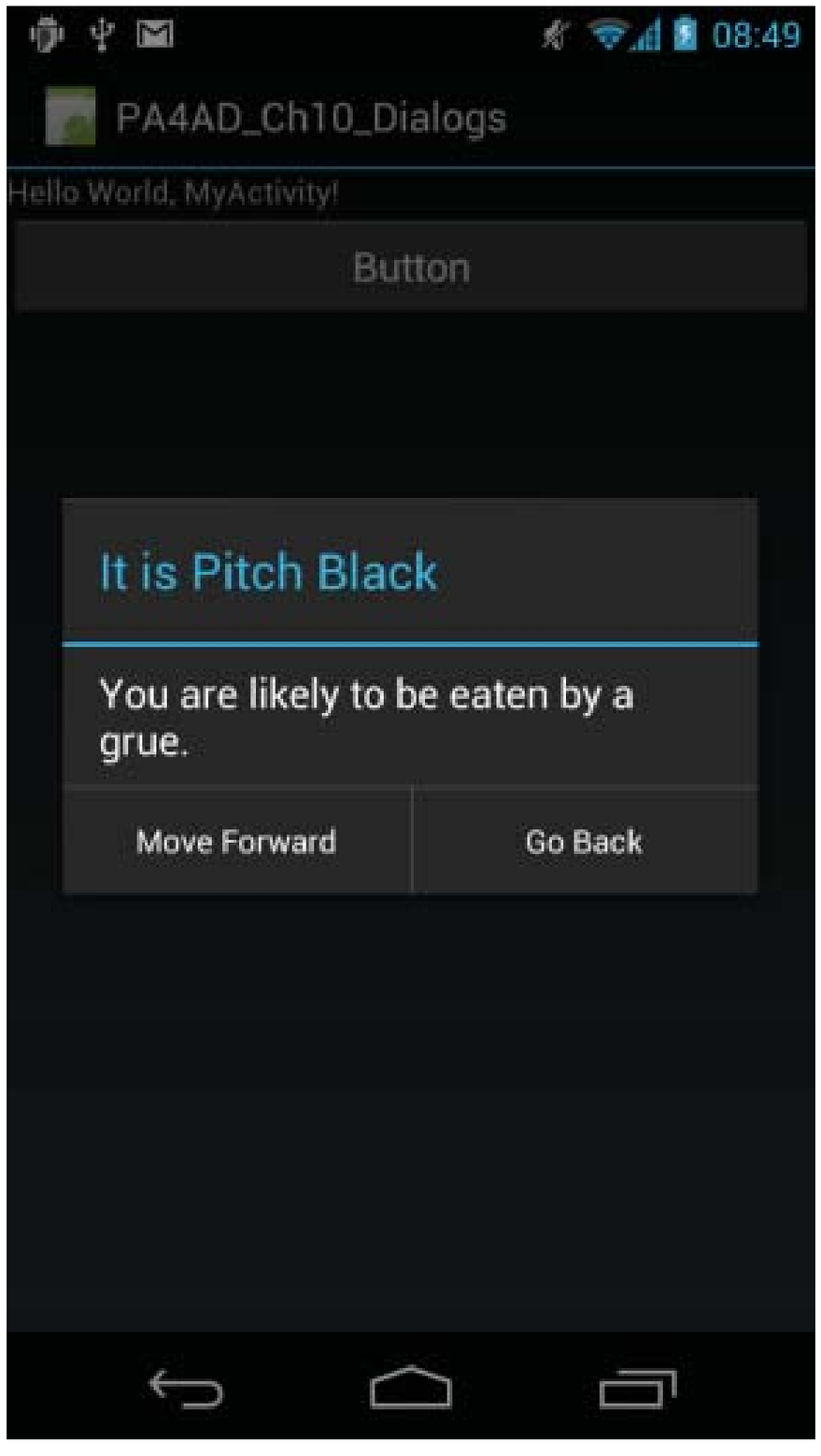
Podemos, por ejemplo, eliminar un elemento si sabemos que no va funcionar (no hay red...)

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    MenuItem menuItem = menu.findItem(MENU_ITEM);

    [ ... modify Menu Items ... ]

    return true;
}
```

Ejercicio

Cuadros de diálogo

Introducción



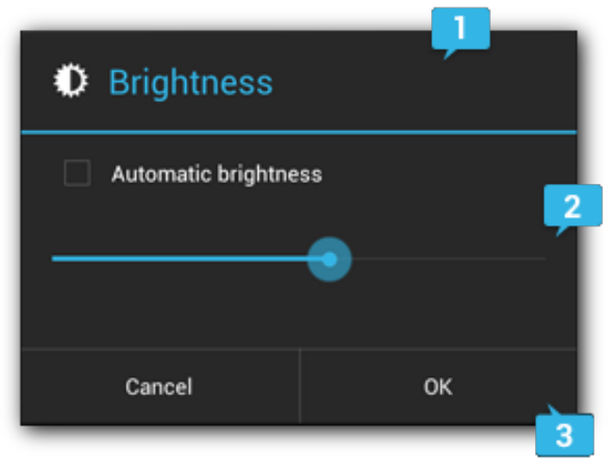
Al igual que en el software de escritorio, disponemos de cuadros de diálogo para interactuar con el usuario.

Generalmente son actividades o fragmentos que oscurecen la UI que las ha lanzado.

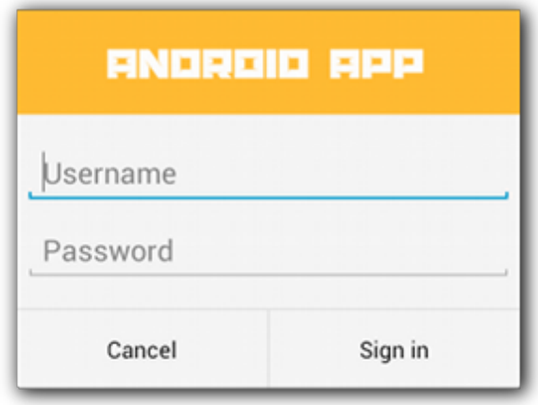
Pueden simplemente informar o esperar una respuesta.

Cuadros de diálogo

Tenemos tres opciones para implementar cuadros de diálogo en Android:



Extender la clase `Dialog`: Android incluye varios tipos de cuadros de diálogo, más allá de las clásicas alertas.



Actividades con estilo de cuadro de diálogo.

Toasts: pequeños mensajes de texto que pueden ser lanzados desde Servicios o Recibidores de Broadcast.

Mostrar un cuadro de diálogo

Para crear un simple cuadro de diálogo, instanciar la clase Dialog, definir sus características y mostrar el cuadro.

```
// Create the new Dialog.
Dialog dialog = new Dialog(MyActivity.this);

// Set the title.
dialog.setTitle(“Dialog Title”);
// Inflate the layout.
dialog.setContentView(R.layout.dialog_view);

// Update the Dialog’s contents.
TextView text = (TextView)dialog.findViewById(R.id.dialog_text_view);
text.setText(“This is the text in my dialog”);

// Display the Dialog.
dialog.show();
```

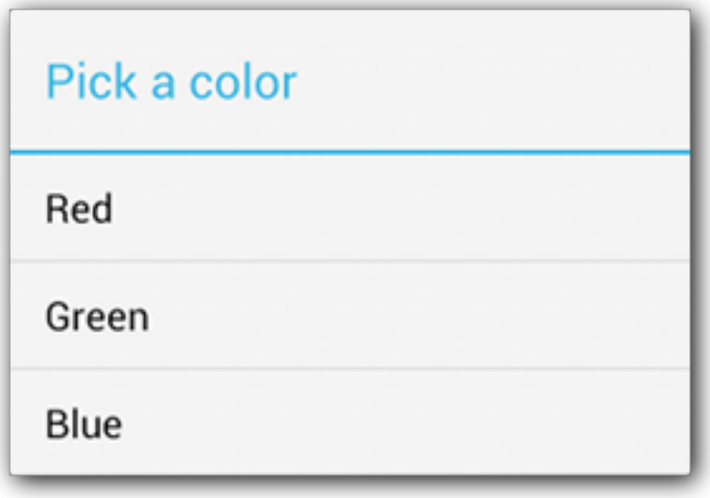
Cuadro de alerta

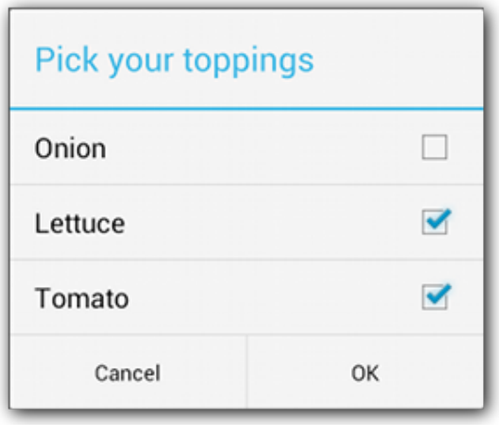
La clase `AlertDialog` es la implementación más común de cuadro de diálogo.

Pueden presentar un mensaje con los clásicos botones de Sí, No, OK o Cancelar.

Pueden mostrar una lista de valores con checkbox o radiobutton

O puden proveer de una entrada de texto.





```
Context context = MyActivity.this;
String title = “It is Pitch Black”;
String message = “You are likely to be eaten by a Grue.”;
String button1String = “Go Back”;
String button2String = “Move Forward”;

AlertDialog.Builder ad = new AlertDialog.Builder(context);
ad.setTitle(title);
ad.setMessage(message);

ad.setPositiveButton(
  button1String,
  new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int arg1) {
      eatenByGrue();
    }
  }
);

ad.setNegativeButton(
  button2String,
  new DialogInterface.OnClickListener(){
    public void onClick(DialogInterface dialog, int arg1) {
      // do nothing
    }
  }
);
```

Cuadros de diálogo especializados

Android dispone de cuadros de diálogo especializados para obtener un tipo de dato concreto del usuario.

`CharacterPickerDialog`: permite seleccionar caracteres especiales en función de la codificación actual.

`DatePickerDialog`: permite seleccionar una fecha.

`DatePickerDialog`: permite seleccionar una hora.

`ProgressDialog`: muestra una barra de progreso dentro de la caja. Bloquea la interfaz de usuario.

Dialog Fragments

Una mejor alternativa para mostrar diálogos es utilizar fragmentos.

Un `DialogFragment` encapsula la vista y el comportamiento de un diálogo, facilitando su reutilización.

```
public class MyDialogFragment extends DialogFragment {

  private static String CURRENT_TIME = "CURRENT_TIME";

  public static MyDialogFragment newInstance(String currentTime) {
    // Create a new Fragment instance with the specified
    // parameters.
    MyDialogFragment fragment = new MyDialogFragment();
    Bundle args = new Bundle();
    args.putString(CURRENT_TIME, currentTime);
    fragment.setArguments(args);

    return fragment;
  }

  @Override
  public Dialog onCreateDialog(Bundle savedInstanceState) {
    // Create the new Dialog using the AlertBuilder.
    AlertDialog.Builder timeDialog =
      new AlertDialog.Builder(getActivity());

    // Configure the Dialog UI.
    timeDialog.setTitle("The Current Time Is...");
    timeDialog.setMessage(getArguments().getString(CURRENT_TIME));

    // Return the configured Dialog.
    return timeDialog.create();
  }
}
```

Notificaciones

Notificaciones

Son utilizadas para notificar al usuario de eventos que se han producido.

Las notificaciones son gestionadas por el `NotificationManager`, que tiene la capacidad de:

Mostrar un icono en la barra de estado

Encender los LED

Hacer vibrar el teléfono

Emitir sonidos

Enviar Intents a través de Broadcast

Crear notificaciones

La manera más sencilla es utilizar `Notification.builder` para crear una nueva notificación.

```
Notification.Builder builder =
  new Notification.Builder(MyActivity.this);
builder.setSmallIcon(R.drawable.ic_launcher)
       .setTicker("Notification")
       .setWhen(System.currentTimeMillis())
       .setDefaults(Notification.DEFAULT_SOUND |
                    Notification.DEFAULT_VIBRATE)
       .setSound(
          RingtoneManager.getDefaultUri(
            RingtoneManager.TYPE_NOTIFICATION))
       .setVibrate(new long[] { 1000, 1000, 1000, 1000, 1000 })
       .setLights(Color.RED, 0, 1);

Notification notification = builder.getNotification();
```

Personalizar notificaciones

La apariencia al expander la notificación, puede ser altamente personalizada, utilizando una UI estándar o personalizada.

```
builder.setSmallIcon(R.drawable.ic_launcher)
    .setTicker("Notification")
    .setWhen(System.currentTimeMillis())
    .setContentTitle("Title")
    .setContentText("Subtitle")
    .setContentInfo("Info")
    .setLargeIcon(myIconBitmap)
    .setContentIntent(pendingIntent);
```


```
builder.setSmallIcon(R.drawable.ic_launcher)
    .setTicker("Notification")
    .setWhen(System.currentTimeMillis())
    .setContentTitle("Progress")
    .setProgress(100, 50, false)
    .setContent(myRemoteView);
```

Lanzar, actualizar y cancelar

Para lanzar una notificación, pedimos al `NotificationManager` que lo haga, indicando la notificación y un identificador.

```
String svc = Context.NOTIFICATION_SERVICE;

NotificationManager notificationManager
      = (NotificationManager)getSystemService(svc);

int NOTIFICATION_REF = 1;
Notification notification = builder.getNotification();

notificationManager.notify(NOTIFICATION_REF, notification);
```

Lanzar, actualizar y cancelar

Para actualizar una notificación que ya ha sido lanzada, volvemos a lanzar la nueva notificación con el mismo identificador.

```
builder.setSmallIcon(R.drawable.ic_launcher)
    .setTicker("Notification")
    .setWhen(System.currentTimeMillis())
    .setContentTitle("Progress")
    .setProgress(100, 75, false)
    .setContent(myRemoteView)
    .setOngoing(true)
    .setOnlyAlertOnce(true);

Notification notification = builder.getNotification();
notificationManager.notify(NOTIFICATION_REF, notification);
```

Lanzar, actualizar y cancelar

Al cancelar una notificación, ésta es eliminada totalmente de la barra de notificaciones.

Como norma general, cuando el usuario pulsa en la notificación o accede a la aplicación asociada, se debe eliminar la notificación.

```
builder.setSmallIcon(R.drawable.ic_launcher)
    .setTicker("Notification")
    .setWhen(System.currentTimeMillis())
    .setContentTitle("Progress")
    .setProgress(100, 75, false)
    .setContent(myRemoteView)
    .setContentIntent(pendingIntent)
    .setAutoCancel(true);

notificationManager.cancel(NOTIFICATION_REF);
```