Departamento de Informática | Universidad de Valladolid

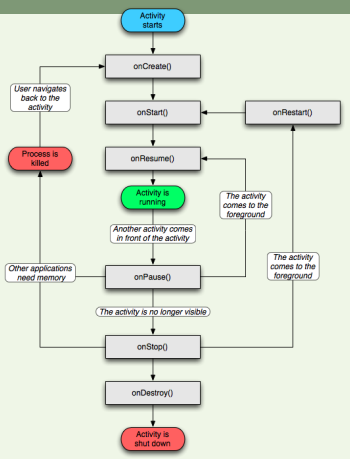
Sistemas Móviles

Hola, Mundo con pretensiones

Félix Prieto

Curso 2013/14

Ciclo de vida de las actividades



- Las actividades pueden pasar por varios estados
- El cambio de estado disparan la ejecución de ciertos métodos
- Podemos redefinir los métodos de cambio de estado
- Debemos asegurar que las actividades terminan «ade-cuadamente»

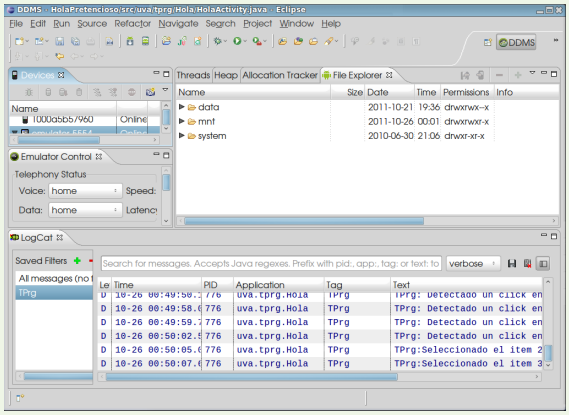
Añadir información al Log

- Disponemos de una clase `Log` que permite volcar información al log del sistema
- Varios métodos para diferentes niveles del log:
  - `Log.d` (Debug) `Log.e` (Error) `Log.i` (Info) `Log.v` (Verbose) `Log.w` (Warn)
- Habitualmente utilizamos

```
Log.d("Etiqueta","Mensaje");
```

- La información del Log aparece en la vista «LogCat» de la perspectiva DDMS
- La información puede ser filtrada para facilitar su lectura

La perspectiva DDMS



Acceder a los elementos de la interfaz

- Podemos obtener una referencia a cualquier elemento de la interfaz gráfica definida en XML mediante su etiqueta
- `findViewById()` proporciona una referencia a la vista identificada mediante la correspondiente etiqueta
  - En XML `android:id=@+id/etiqueta`
  - En Java `R.id.etiqueta`
- Es normal que deseemos una referencia a un tipo más específico de vista (`TextView`, por ejemplo)

```
texto = (TextView) findViewById(R.id.textView1)
```

- Podemos cambiar el aspecto de la vista una vez referenciada desde Java
- Podemos añadir elementos a una vista desde el código Java

Recursos de tipo `String`

- Podemos acceder a los recursos de tipo String desde Java
  - En XML `name="etiqueta"`
  - En Java `R.string.etiqueta`
- Podemos definir vectores de cadenas como recursos
  - Usamos «String Array» en el editor «listo»
  - Son útiles en vistas de tipo «botón de radio»
- Podemos definir cadenas «plurales», con varios valores en función de una cantidad
  - Asociado a uso de singular/plural en diferentes idiomas
  - Implementación parcial en el editor «listo», hay que retocar la definición en XML.
  - Los idiomas aplican reglas muy variadas respecto a los plurales

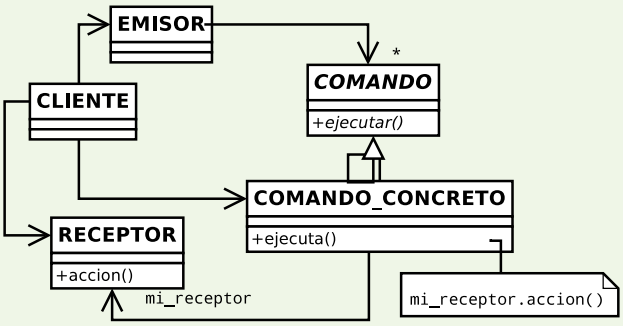
Un string «plural»

```
<!-- Desde java se puede invocar con:
     getQuantityString(
          R.plurals.principalContadorPulsaciones, pulsaciones,
          pulsaciones));
     donde pulsaciones es de tipo int
     Las cantidades utilizables son zero, one, two, few, many, other
     siempre en función del idioma -->
<plurals name="principalContadorPulsaciones">
  <item quantity="one">Has pulsado Boton %d vez</item>
  <item quantity="other">Has pulsado Boton %d veces</item>
</plurals>
```

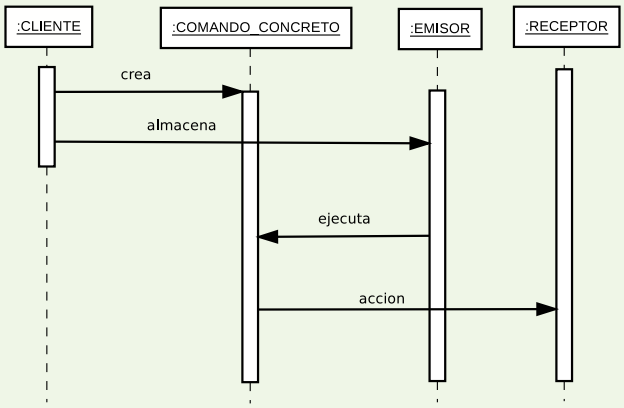

Gestión de eventos y patrón comando

- En el framework de Android, los eventos de la interfaz de usuario se gestionan mediante el patrón comando
- El emisor está dentro del framework
- Existe un comando abstracto para cada tipo de evento (`OnClick`, `OnLongClickListener`, `OnMenuItemClickListener`,...)
- El comando concreto es cualquier objeto que implemente la correspondiente interfaz
- El receptor suele estar entre las clases del dominio del problema
- Para cada evento existe un método específico que conecta el emisor con el comando concreto (por ejemplo `setOnClickListener(...)`)

Comando (Command) (GoF p.215)



Comando (Command) (GoF p.215)



Reaccionar a un «click»

- Fijar un «escuchador» para una vista: `setOnClickListener(...)`
- Hacer que el «escuchador» implemente la interfaz `OnClickListener`
- El «escuchador» suele ser el propio objeto `Activity` que contiene la vista
- Un solo objeto suele escuchar diferentes eventos «Click»
- El «click» puede ser generado en varios tipos de vista, no solo botones
- Al implementar el método `onClick()` recibimos como argumento la vista que disparó el evento
- Usamos esa información para producir diferentes comportamientos
- También podemos manejar el «Click» largo (`LongClick`)

Menú de actividad

- Podemos asociar un menú a cada actividad
- Definimos los elementos como recursos XML (directorio `res/menu`
  - *Item*: Una entrada normal
  - *Sub-Menu*: Una entrada con elementos internos
  - *Group*
- *No* hay que asignar el menú a la vista
- *Pero* hay que gestionar la creación/apertura del menú
- *Además* hay que gestionar la activación de los items

Ejemplo de menú

- Redefinimos `onCreateOptionsMenu(Menu menu)` para implementar la creación del «menú»
- Redefinimos `onPrepareOptionsMenu(Menu menu)` para implementar las pulsación de «menú»
- Utilizamos un objeto `Inflate` para construir el objeto `Menu`
- Utilizamos `invalidateOptionsMenu()` para obligar a la llamada al método `onCreateOptionsMenu(Menu menu)` en la siguiente activación del menú.

Creación de un menú

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Gestión de la creación de los menús
  Log.d(ASIMOV,"Inflando el menú");
  super.onCreateOptionsMenu(menu);
  MenuInflater inf = getMenuInflater();
  inf.inflate(R.menu.principal, menu);
  itemVariable = menu.findItem(R.id.item1);
  itemVariable.setEnabled(false);
  return true;
}

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
  Log.d(ASIMOV,"Preparando menú de opciones");
  return super.onPrepareOptionsMenu(menu);
}
```


Gestión de eventos del menú

- La gestión de los eventos es responsabilidad de la actividad
- Redefinimos `onOptionsItemSelected(...)`
  - `item`: El item que fue pulsado
- Podemos aplicar las mismas técnicas que en la gestión de un «click»
- Podemos escribir la gestión de menús en un ancestro común a varias actidades
- Podemos modificar el menú en tiempo de ejecución
  - `itemVariable = menu.findItem(R.id.item1);`

Gestión de un menú

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Gestión de la selección de opciones en un menú
  Log.d(ASIMOV,"Item seleccionado en el menú");
  switch (item.getItemId()) {
  case R.id.item1:
    Log.d(ASIMOV,"Procesando item1");
    break;
  case R.id.item2:
    Log.d(ASIMOV,"Procesando item2");
    break;
  case R.id.item3:
```

Gestión de un menú

```
      Log.d(ASIMOV,"Procesando item3");
      /* Activando la opción modificable */
      itemVariable.setEnabled(true);
      break;
    case R.id.item4:
      Log.d(ASIMOV,"Procesando item4");
      /* Desactivando la opción modificable */
      itemVariable.setEnabled(false);
      break;
    default:
      Log.d(ASIMOV,"Procesando item desconocido");
      break;
    }
    return true;
  }
```


Menú contextual

- Asociado a una vista en particular
- Se dispara con una pulsación larga
- La vista se registra mediante `registerForContextMenu(v);`
- Redefinimos `onCreateContextMenu(Menu menu)` para implementar la creación del «menú»
- Redefinimos `onPrepareContextMenu(Menu menu)` para implementar las pulsación de «menú»
- Utilizamos un objeto `Inflate` para construir el objeto `Menu`
- La gestión de los eventos es responsabilidad de la actividad
- Redefinimos `onContextItemSelected(...)`

Lanzar otra actividad

- Podemos lanzar otras actividades
- Al terminar nos devuelven el foco
- En la llamada utilizamos un objeto `Intent` explícito o implícito
- `new Intent(this,Aviso.class)` es explícito
- `startActivity(new Intent(this, Aviso.class));` lanza la actividad `Aviso`
- `startActivityForResult(...)` si debe devolver datos
- La actividad debe ser declarada como tal en `AndroidManifest.xml`

Preferencias

- Permiten modelar los «ajustes» que el usuario realiza sobre la aplicación
- Mantienen la persistencia entre ejecuciones
- El menú se puede construir en XML(`/res/xml`)
- Tipo de preferencias:
  - Preference
  - PreferenceScreen
  - CheckBoxPreference
  - EditTextPreference
  - RingTonePreference
  - ListPreference
  - PreferenceCategory

Uso de las preferencias

- Se gestionan desde una actividad que hereda de `PreferenceActivity`
- Las preferencias se cargan mediante `addPreferencesFromResource(...);`
- `PreferenceActivity` gestiona la persistencia
- Es preciso añadir una clave a cada preferencia
- Para recuperar la preferencia usamos `PreferenceManager .getDefaultSharedPreferences(context) .getBoolean(key,default));`

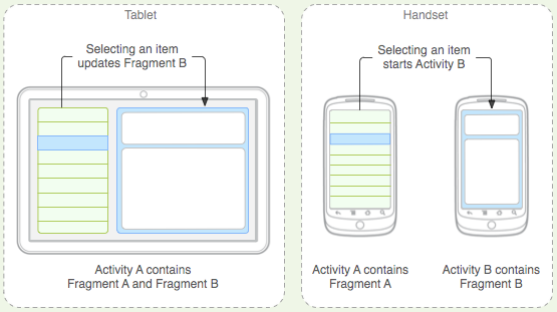
Persistencia en preferencias

- Las preferencias se guardan dentro del directorio `/data/data/<paquete>/shared_prefs`
- El fichero de preferencias se llama `<paquete>_preferences.xml`
- Se puede usar el mismo esquema para crear y acceder a otros ficheros en el mismo directorio
- Todos ellos son ficheros XML con la misma estructura clave-valor
- Usamos `getSharedPreferences(...)` para acceder a otros ficheros de preferencias
- Podemos usar un objeto de tipo `SharedPreferences.editor` para modificar y guardar las preferencias

Organizar las preferencias

- Podemos Organizar las preferencias de varios modos:
- Mediante pantallas diferentes
  - Utilizamos `<PreferenceScreen ...>`
- Mediante títulos, en una pantalla
  - Utilizamos `<PreferenceCategory ...>`

Fragmentos



Fragmentos

- Permiten aprovechar mejor la pantalla de las tabletas
  `http://developer.android.com/guide/components/fragments.html`
- Como consecuencia
  `addPreferencesFromResource(...)` de la clase
  `PreferenceActivity` está obsoleto
- Podemos seguir usándolo
- Podemos utilizar el sistema de fragmentos con una estructura tradicional de preferencias

Cargando un «Fragmento»

```
public class Preferencias extends PreferenceActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /* Obsoleto desde la versión 11 de la api */
    // addPreferencesFromResource(R.xml.preferencias);
    /* A partir de la versión 11 recomiendan utilizar fragmentos */
    getFragmentManager().beginTransaction()
     .replace(android.R.id.content, new MiFragmento()).commit();
  }
  /* Lo más fácil es definir una clase aquí mismo */
  public static class MiFragmento extends PreferenceFragment {
    @Override
    public void onCreate(final Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.preferencias);
    }
  }
  /* etc */
}
```