Departamento de Informática | Universidad de Valladolid

Tecnología de Programación

Hola, Mundo con pretensiones

Félix Prieto

Curso 2011/12

Actualización de Android

- Se ha producido una nueva actualización de Android:
  - Android SDK 14
  - Android Tools 8
  - Android Platforms Tools 8
  - Eclipse ADT 14
  - Android 4.0 (API 14)
- Podemos seguir usando las versiones anteriores
- Al intentar actualizar algo nos «ofrece» las nuevas versiones

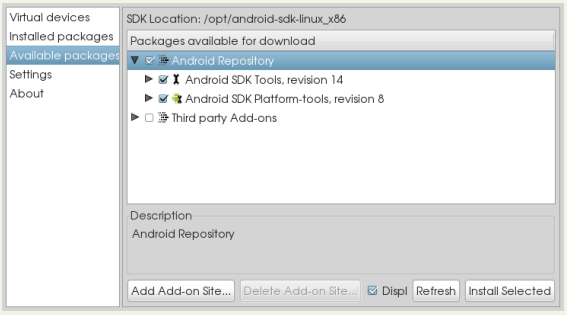
Principales novedades

- Android «Ice Cream Sandwich»
- Resuelto el problema con el «snapshot» de dispositivos virtuales
- Mejoras en la perspectiva DDMS
- Algunas modificaciones en los editores «listos»

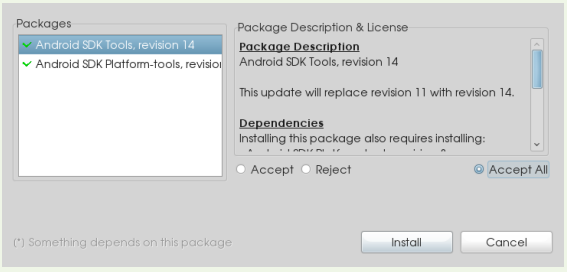
El proceso de actualización

- Arrancar android como root
- Actualizar Tools y Platform Tools
- Volver a arrancar Android como root
- Instalar la nueva documentación
- Si se desea: Instalar la nueva API
- Arrancar eclipse como root
- Help->Check For Updates
- Actualizar el ADT a la versión 14

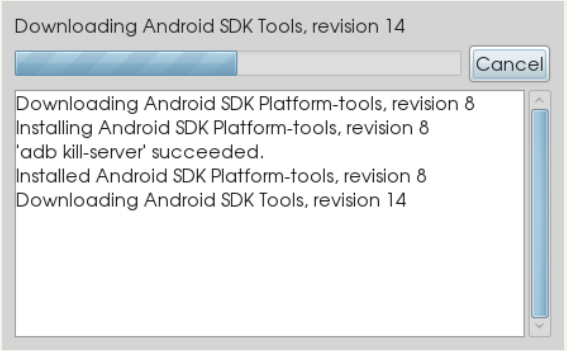
Inicio del proceso



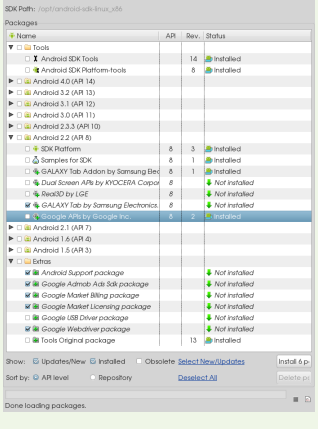
Aceptar las licencias



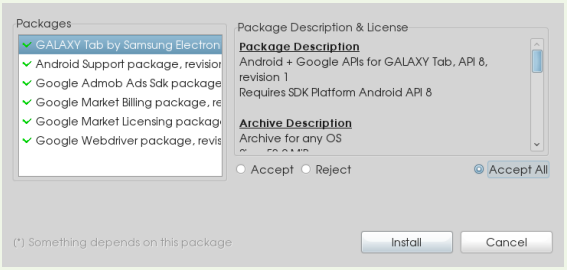
La descarga puede ser lenta



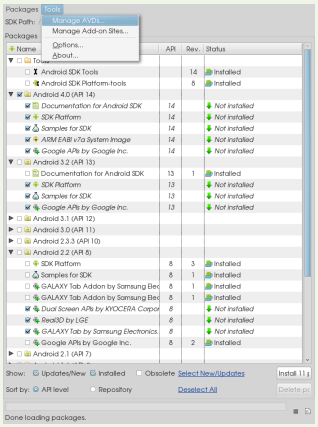
Selección de los elementos a instalar



Más licencias para aceptar



La gestión de dispositivos



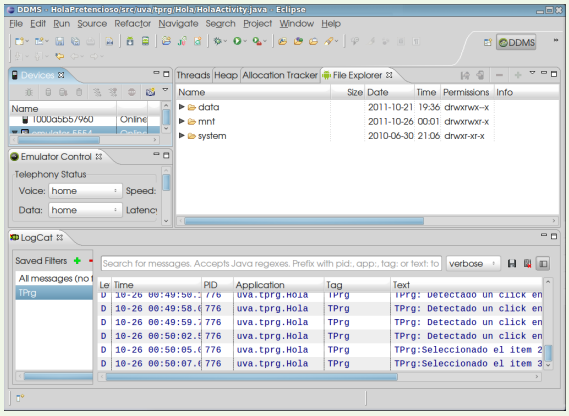
Instalación completa

- Se puede rehacer la instalación completa
- Eliminar también los directorios `.android` `.eclipse` y `.metadata` dentro del workspace de eclipse
- Los permisos del directorio y el nombre del directorio Android han cambiado
- Desde el usuario root hay que aplicar los comandos:
  - `chown -R root:root /opt/android-sdk-linux`
  - `chmod -R o+rx /opt/android-sdk-linux`
- El snapshot de los dispositivos virtuales funciona sin más que activarlo al editar la máquina
- No son precisos más argumentos al lanzar el dispositivo virtual
- Si se usa «snapshot» el dispositivo tarda en cerrarse

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



Métodos y atributos de clase

- `Log` no es un objeto, es una clase
- `d` es un método definido a nivel de clase
- El método puede ser enviado como mensaje a la clase o a cualquiera de sus instancias
- También podemos definir atributos de clase
- En ambos casos utilizamos el modificador `static` en la definición
- Las definiciones a nivel de clase se utilizan para definir constantes o comportamientos comunes a todas las instancias de la clase
- En ocasiones se utilizan de un modo similar a las funciones de una sola ejecución de Eiffel

Acceder a los elementos de la interfaz

- Podemos obtener una referencia a cualquier elemento de la interfaz gráfica definida en XML.
- `findViewById(R.id.textView1)` proporciona una referencia a la vista identificada mediante el entero `R.id.textView1`
- La etiqueta `textView1` puede ser modificada en el editor de «layouts»
- Es normal que deseemos una referencia a un tipo más específico de vista (`TextView`, por ejemplo)
- Lo conseguimos con un «casting»:
  `texto = (TextView) findViewById(R.id.textView1)`

Recursos de tipo `String`

- Podemos acceder a los recursos de tipo String desde Java
  - `getString(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
  - No implementado actualmente en el editor «listo», hay que definirlo directamente en XML.

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>
```

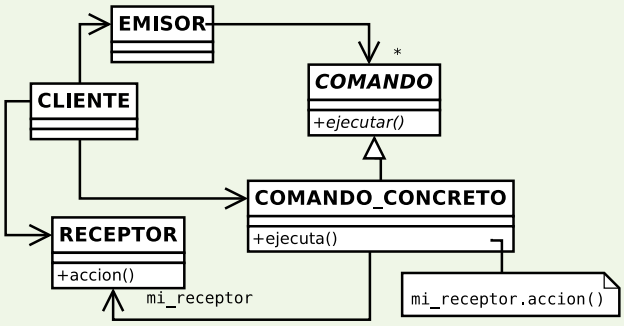

Interfaces en Java

- La noción de «interfaz» en Java equivale a una clase abstracta en Eiffel en que todos los métodos son completamente diferidos
- Una clase puede implementar tantas interfaces como deseemos
- Utilizamos `implements` en lugar de `extends`
- Nos comprometemos a implementar todos los métodos diferidos declarados en la interfaz
- Mediante interfaces también podemos utilizar polimorfismo y ligadura dinámica

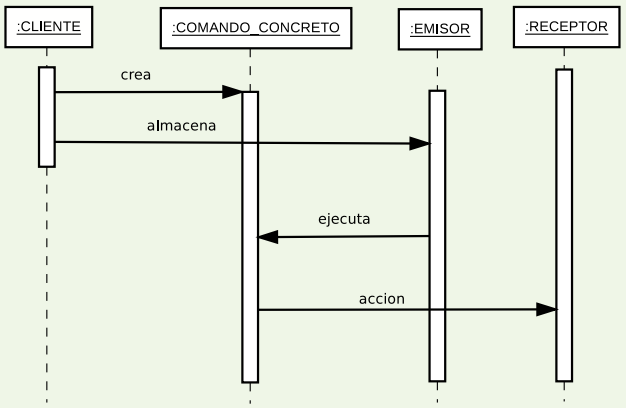
Gestión de eventos y patrón comando

- 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
  - *item*: Una entrada normal
  - *submenú*: Una entrada con elementos internos
  - *grupo*: Agrupa entradas desde el punto de vista del programador
- *No* hay que asignar el menú a la vista
- Hay que gestionar la creación/apertura del menú
- 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*

Creación de un menú

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Gestión de la creación de los menús
  Log.d(TAG, TAG + "␣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(TAG, TAG + "␣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(TAG, TAG + "Item␣seleccionado␣en␣el␣menú");
  switch (item.getItemId()) {
  case R.id.item1:
    Log.d(TAG, TAG + "␣Procesando␣item1");
    break;
  case R.id.item2:
    Log.d(TAG, TAG + "␣Procesando␣item2");
    break;
  case R.id.item3:
```


Gestión de un menú

```
    Log.d(TAG, TAG + "␣Procesando␣item3");
    /* Activando la opción modificable */
    itemVariable.setEnabled(true);
    break;
  case R.id.item4:
    Log.d(TAG, TAG + "␣Procesando␣item4");
    /* Desactivando la opción modificable */
    itemVariable.setEnabled(false);
    break;
  default:
    Log.d(TAG, TAG + "␣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));`
- Podemos encapsular esta llamada en nuestro gestor de preferencias

Persistencia en preferencias

- Las preferencias se guardan en un fichero con el nombre del paquete en el directorio `/data/data/shared_prefs`
- Se puede usar el mismo esquema para otros ficheros de preferencias
- Usamos `getSharedPreferences(...)` para acceder a otros ficheros de preferencias
- En este caso, usaremos un objeto de tipo `SharedPreferences.editor` para modificar y guardar las preferencias