Desarrollo Avanzado de Software

Laboratorio 2.- Trabajo con interfaces gráficas e idiomas

Contenido:

1 ORIENTACIÓN DE LA INTERFAZ, MANTENIMIENTO Y RECUPERACIÓN DE DATOS 2

2 APLICACIONES MULTI-IDIOMA 3

Objetivos: Comprender el ciclo de vida de las actividades y tenerlo en cuenta a la hora de desarrollar aplicaciones. Aprender a manejarse con el desarrollo de interfaces gráficas para Android y ser capaces de mantener datos, aunque las actividades se destruyan. Aprender a desarrollar aplicaciones multi-idioma.

Desarrollo Avanzado de Software

1 Orientación de la interfaz, mantenimiento y recuperación de datos

Al cambiar la orientación del dispositivo, la interfaz gráfica se adaptará al cambio de orientación automáticamente, pero no siempre será lo que deseemos. A veces querremos cambiar la forma de la interfaz gráfica para adaptarnos mejor a la nueva orientación.

Cuando queramos que una actividad tenga dos interfaces gráficas distintas en función de la orientación, habrá que crear, en la carpeta *res*, una carpeta que se llame *layout-land* y almacenar en ella las versiones horizontales de las interfaces gráficas verticales que se almacenan en la carpeta *layout*.

El fichero XML debe tener el mismo nombre en ambas carpetas (el nombre que usemos en el método `setContentView()`) y los elementos de la interfaz también pueden (si nos interesa) tener los mismos identificadores. De este modo, desde el código Java podremos acceder a un elemento concreto independientemente de que la interfaz gráfica que esté visualizando el usuario sea la horizontal o la vertical.

> 💡 Además, dependiendo del tipo de elemento (como por ejemplo los *EditText*), si tiene el mismo identificador en la versión horizontal y en la vertical, el propio sistema operativo se encargará de mantener la información de dicho elemento.

Cread una aplicación con dos interfaces gráficas distintas en función de la orientación. Lo único que necesitáis que tengan en común ambas interfaces es al menos una etiqueta de texto (*TextView*) que usaremos en el siguiente punto. Y por supuesto, todos aquellos elementos a los que se acceda desde el código, ya que, si se intenta acceder a un elemento que en dicha interfaz no es visible, obtendremos un *Null Pointer* .

- Usando el *TextView* común a ambas orientaciones, implementad un contador que vaya mostrando en dicha etiqueta el número de cambios de orientación sufridos por la aplicación.

> 💡 Tened en cuenta que, cada vez que cambiéis la orientación, la actividad se reinicia, por lo que habrá que almacenar en algún sitio (que no se destruya junto con la actividad) la información del contador de cambio de orientaciones.

- Añadid otro *TextView* donde vayáis concatenando un texto que os indique los estados por los que pasa la actividad (`onCreate`, `onStart`, `onPause`, etc.).

2 Aplicaciones multi-idioma

Como se ha visto en clase, todos los textos que se muestren en una aplicación deberían estar almacenados en el fichero ***strings.xml*** de la carpeta ***res/values***. De ese modo, hacer que esa misma aplicación soporte múltiples idiomas es muy sencillo. Sólo hay que copiar el fichero ***strings.xml*** en una carpeta llamada ***values***, seguida del sufijo del idioma correspondiente según la ISO 639-1 (Pinchad aquí para ver el listado).

Por ejemplo, los textos en castellano deberían estar en la carpeta ***values-es***, los textos en euskera en la carpeta ***values-eu*** y en la carpeta ***values-fr*** se almacenaría el fichero strings.xml con las traducciones en francés.

Automáticamente, la aplicación usará los textos almacenados en la carpeta que se corresponda con el idioma en el que se encuentre el sistema operativo, es lo que se llama localización. Sólo en el caso de que no se encuentre una carpeta específica para el idioma del sistema operativo, se usa la carpeta ***values*** (sin sufijo), que contendrá los textos a utilizar por defecto.

- Incluid en vuestra aplicación varios textos (en etiquetas, botones, etc.) y generad los ficheros ***strings.xml*** con los textos de vuestra aplicación correspondientes a varios idiomas y modificando el idioma del teléfono comprobad que la aplicación es multi-idioma.

Android Studio facilita el trabajo de generación de las carpetas y los distintos ficheros strings.xml gracias a su editor. Para usarlo, abrid el fichero strings.xml y pulsad la opción "Open editor" (ver Fig. 1)

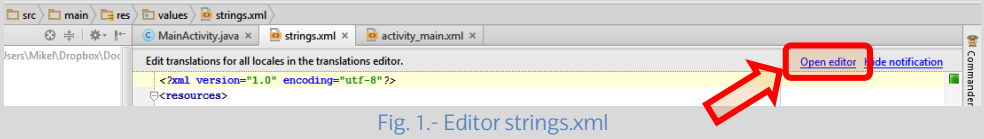

Fig. 1.- Editor strings.xml

En el editor que se abre, se pueden añadir fácilmente nuevas entradas en el fichero strings.xml (pulsando el símbolo +) y nuevas localizaciones para otros idiomas pulsando el símbolo del mapamundi (ver Fig. 2).

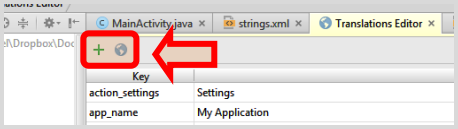

Fig. 2.- Añadir entrada o traducción

Al añadir una localización el editor no se actualiza correctamente, hay que cerrarlo y volver a abrirlo para que aparezca la nueva localización.

Sin embargo, no es cómodo tener que modificar el idioma del teléfono para tener que ejecutar una aplicación en otro idioma. Por ejemplo, hasta Android 5.0 el euskera no estaba oficialmente soportado ¿Y si queremos ejecutar nuestra aplicación en euskera aunque el teléfono no lo soporte? A partir de Android 13 se puede cambiar el idioma de una app desde la configuración del sistema, pero ¿y si queremos que sea el usuario desde la propia aplicación el que decida en qué idioma quiere ejecutarla?

Para esas ocasiones se puede forzar la localización desde dentro de la propia aplicación. Para ello haremos uso de la clase *Locale* del paquete *java.util*, definiendo una nueva localización mediante el código de dos letras del idioma y creando una nueva configuración para la aplicación, donde la localización se

corresponda con la que acabamos de crear. La clase *Configuration* está dentro del paquete *android.content.res*.

```
Locale nuevaloc = new Locale("eu");
Locale.setDefault(nuevaloc);

Configuration configuration =
      getBaseContext().getResources().getConfiguration();
configuration.setLocale(nuevaloc);
configuration.setLayoutDirection(nuevaloc);
```

Por último, habrá que actualizar la configuración de todos los recursos de la aplicación mediante el método `updateConfiguration` que recibe como parámetro la nueva configuración y las `DisplayMetrics` de la aplicación que contienen información del tamaño de la pantalla, la densidad en píxeles, etc.

```
Context context =
getBaseContext().createConfigurationContext(configuration);
getBaseContext().getResources().updateConfiguration(configu
ration, context.getResources().getDisplayMetrics());
```

Pero para que la interfaz gráfica se actualice por completo (todos los elementos, incluidos los del menú), hay que recargar de nuevo la actividad. Para ello finalizaremos la instancia en curso (método `finish()`) y lanzaremos una nueva (método `startActivity()`).

```
finish();
startActivity(getIntent());
```

- Cread la versión en euskera y en castellano de todos los textos de vuestra aplicación. Añadid a vuestra aplicación un botón que fuerce la localización de la aplicación al euskera y otro que la fuerce al castellano. Cuando esté listo, añadid un tercer botón que la traduzca a inglés.

Al destruirse una actividad, la configuración se resetea a la localización que el dispositivo tenga por defecto por lo que habremos perdido la localización elegida por el usuario.

- Modificad y verificad que la aplicación mantiene la localización seleccionada, aunque se cambie la orientación del dispositivo.

> 💡 La localización debería estar definida antes de que se ejecute el método `setContentView` que es el que, al cargar los elementos gráficos, realmente se encarga de buscar los valores concretos en el fichero ***strings.xml*** de la carpeta correspondiente.

- Añadid una segunda actividad a vuestra aplicación con algún elemento de texto y un botón para pasar de la primera a la segunda actividad. ¿Qué ocurre con la localización en la segunda actividad?