Flexibilidad de App Android

en idioma, def. pantallas, ver. plataforma.

Diferentes Lenguajes

Siempre es una buena práctica para extraer cadenas de la app de su código y mantenerlos en un archivo externo. En el directorio de recursos podemos definir:

```
MyProject/
    res/
        values/
            strings.xml
        values-es/
            strings.xml
        values-fr/
            strings.xml
```

Diferentes Lenguajes

En cada archivo hay que agregar los valores de cadena para cada entorno.

En tiempo de ejecución, el sistema Android utiliza el conjunto basado en la configuración regional establecida actualmente para el dispositivo del usuario.

Diferentes Lenguajes

Por ejemplo, las siguientes son algunas diferentes archivos de recursos de String en diferentes idiomas.

Inglés (por default)**, /values-es/strings.xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

Español, /values-es/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Aplicación</string>
    <string name="hello_world">Hola Mundo!</string>
</resources>
```

Frances, /values-fr/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Usar los Recursos String

En el código fuente, se puede hacer referencia a los string usando la sintaxis R.string.<string_name>. Hay varios metodos que aceptan usar los string de esta manera:

Por ejemplo:

```
// Obtener un recurso de string desde los Recursos de la  app
String hello = getResources().getString(R.string.hello_world);


// O suplantar un recurso string con un método que requiere un string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

Usar los Recursos String

En otros archivos XML, se puede hacer referencia a un string con la sintaxis @string/<string_name> cuando el atributo XML acepta ese valor.

Por ejemplo:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Soportar Diferentes Pantallas

Android categoriza pantallas de dispositivos mediante dos propiedades generales:

- Tamaño
- Densidad.

Como las aplicaciones pueden ser instaladas en dispositivos con pantallas que varían en tamaño y densidad, hay que incluir algunos recursos que optimizan el aspecto de su aplicación para diferentes tamaños de pantalla y densidades.

Hay 4 tamaños generalizadas:

- **pequeño,**
- **normal,**
- **grande,**

Y cuatro densidades generalizadas:

- **xlarge bajo (LDPI),**
- **media (MDPI),**
- **alta (IPAP),**
- **extra alta (xhdpi)**

Soportar Diferentes Pantallas

También hay que tener en cuenta que la orientación pantallas horizontal o vertical se debe considerar una variación en el tamaño de la pantalla, por lo que muchas de las aplicaciones deben revisar el diseño para optimizar la experiencia del usuario en cada orientación.

Crear Diseños Diferentes (Layouts)

Se debe crear un archivo de formato XML único para cada tamaño de pantalla que desea soportar.
Cada diseño debe ser guardado en el directorio de recursos apropiados, nombrada con un sufijo -<screen_size>. Por ejemplo, un diseño único para pantallas grandes se debe guardar bajo res/layout-large/.

Importante: Android escala automáticamente la disposición con el fin de encajar correctamente en la pantalla. Así que los diseños para diferentes tamaños de pantalla no tienen que preocuparse por el tamaño absoluto de los elementos de la interfaz de usuario, sino que se centran en la estructura de disposición (como el tamaño o la posición en relacion a los puntos de vista importantes con hermanos y visitas).

Crear Diseños Diferentes (Layouts)

Por ejemplo, este proyecto incluye un diseño predeterminado y un diseño alternativo para pantallas grandes:

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

Los nombres de los archivos deben ser exactamente el mismo, pero su contenido es diferente con el fin de proporcionar una interfaz de usuario optimizada para el tamaño de la pantalla correspondiente.

Crear Diseños Diferentes (Layouts)

Simplemente hay que hacer referencia al archivo layout de su aplicación:

```
@Override
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

El sistema carga el archivo de diseño del directorio layout adecuado en función de tamaño de la pantalla del dispositivo en el que se ejecuta la aplicación. Más información acerca de cómo Android selecciona el recurso apropiado está disponible en la guía de recursos que dan.

Crear Diseños Diferentes (Layouts)

Como otro ejemplo, aquí hay un proyecto con un diseño alternativo para la orientación horizontal:

```
MyProject/
    res/
        layout/
            main.xml
        layout-land/
            main.xml
```

Por defecto, el archivo layout/main.xml es usado para la orientación vertical.

Crear Diseños Diferentes (Layouts)

Si se quiere brindar un diseño especial para horizontal (landscape), incluyendo pantallas grandes, hay que usar los dos calificadores large y land:

```
MyProject/
    res/
        layout/                 # default (portrait)
            main.xml
        layout-land/            # landscape
            main.xml
        layout-large/           # large (portrait)
            main.xml
        layout-large-land/      # large landscape
            main.xml
```

Crear Diferentes Bitmaps

Siempre se debe proporcionar recursos de mapa de bits que estén debidamente a escala para cada uno de los cubos de densidad generalizadas:

- baja,
- media,
- alta densidad y
- extra-alta.

Esto le ayuda a conseguir buena calidad y el rendimiento gráfico en todas las densidades de pantalla.

Para generar estas imágenes, usted debe comenzar con su recurso en bruto en formato vectorial y generar las imágenes para cada densidad utilizando la escala siguiente del tamaño:

- xhdpi: 2,0
- hdpi: 1.5
- mdpi: 1.0 (línea de base)
- ldpi: 0.75

Crear Diferentes Bitmaps

Significa que si se genera una imagen de 200x200 para dispositivos xhdpi, se debe generar el mismo recurso en 150x150 para hdpi,
100x100 para mdpi y
75x 75 para dispositivos ldpi.

Colocar los archivos en el directorio apropiado de recursos drawable:

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Cada vez que se hace referencia a @drawable/awesomeimage, el sistema selecciona el mapa de bits apropiada basada en la densidad de la pantalla.

Nota: Los recursos de baja densidad (ldpi) no siempre son necesarios. Cuando usted proporciona recursos hdpi, el sistema les reduce proporcionalmente a la mitad para pantallas ldpi para encajar correctamente.

Para obtener más consejos y directrices sobre iconos para su aplicación, consulte la guía de diseño Iconografía. https://developer.android.com/design/style/iconography.html

Soportar Versiones Diferentes Android

En el AndroidManifest.xml describe los detalles de la app y identifica las versiones de Android soportadas. El atributo minSdkVersion y targetSdkVersion del elemento <uses-sdk identifica la min nivel de API compatible y la máx nivel de API. Eso define las versiones de Android para las que hay que diseñar y testear.

For example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

Chequear Versiones Android en Runtime

Android proporciona un código único para cada versión de API en la clase de las constantes de generación. Utilice estos códigos dentro de su aplicación para crear condiciones que aseguren el código que depende de los niveles más altos de la API se ejecuta sólo cuando esas APIs están disponibles en el sistema.

For example:

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Usar Temas y Estilos en la Aplicación

Android proporciona temas para que la experiencia del usuario tengan la apariencia del sistema operativo subyacente. Estos temas se pueden aplicar a su aplicación en el archivo AndroidManifest. Mediante el uso de estos estilos y temas la aplicación seguirá naturalmente la apariencia de la última versión de Android con cada nueva versión.

Para hacer que la aplicación luzca como dialog box:

```
<activity android:theme="@android:style/Theme.Dialog">
```

Para hacer que la activity luzca un transparent background:

```
<activity android:theme="@android:style/Theme.Translucent">
```

Usar Temas y Estilos en la Aplicación

Para hacer que la aplicación luzca con el estilo del usuario definido en /res/values/styles.xml:

```
<activity android:theme="@style/CustomTheme">
```

Para aplicar el tema a toda la app (todas las Activitys), agregar el atributo android:theme en el elemento <application> en el AndroidManifest

```
<application android:theme="@style/CustomTheme">
```

Para más información en la creación de estilos y temas, leer la guía Styles and Themes. https://developer.android.com/guide/topics/ui/themes.html