Unidad Didáctica 4:
CÓMO EXPANDIR LA EXPERIENCIA DEL USUARIO

Programación Dirigida por Eventos

UAX Universidad Alfonso X el Sabio

ÍNDICE

Contenido

1. Introducción

¡Bienvenido a la asignatura Programación Dirigida por Eventos!

En esta unidad didáctica, exploraremos cómo expandir la experiencia del usuario en aplicaciones Android. Aprenderás a utilizar fragmentos y widgets para crear interfaces de usuario más dinámicas y adaptables. Los fragmentos permiten modularizar la interfaz de usuario y reutilizar componentes, mientras que los widgets ofrecen funcionalidades adicionales y actualizaciones en tiempo real directamente en la pantalla de inicio del dispositivo.

Ten en cuenta:

La expansión de la experiencia del usuario es crucial para desarrollar aplicaciones atractivas y funcionales. A través de esta unidad, adquirirás habilidades para diseñar y gestionar componentes de interfaz de usuario que mejoren la interactividad y la personalización de tus aplicaciones.

Temas que se tratarán en esta unidad:

1. **Fragmentos en Android:** Comprender qué son los fragmentos y cómo se utilizan para modularizar la interfaz de usuario.
2. **Creación y uso de fragmentos:** Pasos para crear y agregar fragmentos a una actividad, tanto estática como dinámicamente.
3. **Ciclo de vida de los fragmentos:** Aprender el ciclo de vida de un fragmento y cómo interactúa con el ciclo de vida de la actividad.
4. **Comunicación entre fragmentos y actividades:** Métodos para enviar datos y comunicarse entre fragmentos y actividades.
5. **Widgets de aplicación:** Qué son los widgets, cómo se crean y cómo se integran en la pantalla de inicio.
6. **Actualización de widgets:** Implementar actualizaciones en tiempo real para widgets.
7. **Configuración y personalización de widgets:** Permitir al usuario configurar widgets según sus preferencias.
8. **Interacción entre widgets y aplicaciones:** Manejar eventos y acciones desde widgets hacia la aplicación.

Al finalizar esta unidad, deberías ser capaz de implementar fragmentos y widgets en tus aplicaciones Android, mejorando significativamente la experiencia del usuario y la funcionalidad de la aplicación.

2. Objetivos

Objetivo general:

- Proporcionar a los estudiantes las habilidades necesarias para expandir la experiencia del usuario en aplicaciones Android mediante el uso de fragmentos y widgets.

Objetivos específicos:

En esta unidad se establecen tres objetivos específicos:

1. **Comprender y utilizar fragmentos en Android:** Aprender a crear, gestionar y comunicar fragmentos dentro de una actividad para modularizar y mejorar la interfaz de usuario.
2. **Desarrollar e integrar widgets de aplicación:** Adquirir la capacidad de crear widgets funcionales y personalizables que interactúen con la aplicación y proporcionen información en tiempo real.
3. **Mejorar la interactividad y personalización de la UI:** Implementar técnicas para actualizar y personalizar widgets y fragmentos, ofreciendo una experiencia de usuario rica y dinámica.

3. Fragmentos

3.1. Fragments en Android

Comprender la Clase Fragment

Definición

Un Fragment es un componente reutilizable de la interfaz de usuario con su propio ciclo de vida. Es una sección de la interfaz de usuario que se puede reutilizar en múltiples actividades y tiene su propio comportamiento y estados del ciclo de vida, similares a una Actividad. Cada instancia de Fragment está ligada exclusivamente a la Actividad que la contiene.

Características

- **Portion de la UI**: Contiene una parte de la interfaz de usuario y su comportamiento.
- **Ciclo de vida propio**: Tiene sus propios estados del ciclo de vida.
- **Reutilizable**: Puede ser compartido entre múltiples actividades.
- **Diseño y comportamiento definidos**: El código del Fragment define su diseño y comportamiento.
- **Representa secciones de UI**: Útil para diferentes tamaños de pantalla y layouts.

Crear un Fragment

Pasos para Crear un Fragment

1. **Subclase de Fragment**: Crear una subclase de Fragment.
2. **Crear un layout para el Fragment**: Diseñar un archivo XML para el layout.
3. **Añadir el Fragment a una Actividad**: Puede ser estáticamente en el layout de la actividad o dinámicamente usando transacciones de Fragment.

Ejemplo de Creación

Clase Fragment

java
Copiar código

```
public class SimpleFragment extends Fragment {
    public SimpleFragment() {
        // Constructor vacío requerido
    }

    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        // Inflar el layout del Fragment y devolverlo como la vista raíz
        return inflater.inflate(R.layout.fragment_simple, container, false);
    }
}
```

Crear un Nuevo Fragment en Android Studio

1. Expandir app > java en el proyecto y seleccionar el nombre del paquete.
2. Elegir File > New > Fragment > Fragment (Blank).
3. Marcar la opción para crear el layout XML.
4. Opciones adicionales: incluir métodos de fábrica (newInstance()) y callbacks de interfaz.

Usar un Layout para un Fragment

Crear un Layout XML para el Fragment

Android Studio puede crear automáticamente el archivo XML del layout al crear un nuevo Fragment. El método onCreateView() del Fragment crea y devuelve la vista del layout.

Inflar el Layout del Fragment

El método onCreateView() infla el layout del Fragment y lo devuelve como la vista raíz.

Ejemplo de onCreateView()

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    // Inflar el layout del Fragment y devolverlo como vista raíz
    return inflater.inflate(R.layout.fragment_simple, container, false);
}
```

Detalles de Inflar el Layout

- **ID del recurso del layout**: R.layout.fragment_simple.
- **ViewGroup**: El contenedor padre del layout inflado.
- **Booleano**: Si el layout debe adjuntarse al padre (normalmente false).

Añadir un Fragment a una Actividad

Métodos para Añadir un Fragment

1. **Estáticamente**: En el layout de la actividad, visible durante todo el ciclo de vida de la actividad.
2. **Dinámicamente**: Añadir y remover mientras la actividad está en ejecución usando transacciones de Fragment.

Añadir un Fragment Estáticamente

Declara el Fragment dentro del layout de la actividad (activity_main.xml) usando la etiqueta <fragment>. Especifica las propiedades del layout para el Fragment como si fuera una vista.

Ejemplo Estático

```
<fragment
   android:name="com.example.appname.SimpleFragment"
   android:id="@+id/simple_fragment"
   android:layout_weight="2"
   android:layout_width="0dp"
   android:layout_height="match_parent" />
```

Añadir un Fragment Dinámicamente

1. **Especificar ViewGroup**: Define el contenedor para el Fragment en el layout.
2. **Instanciar el Fragment**: Crear una instancia del Fragment.
3. **Instanciar FragmentManager**: Obtener una instancia de FragmentManager.
4. **Usar transacciones de Fragment**: Realizar operaciones de transacción para añadir, remover o reemplazar Fragments.

Ejemplo Dinámico

1. **Especificar ViewGroup**

```
<FrameLayout
   android:id="@+id/fragment_container"
   android:name="SimpleFragment"
   tools:layout="@layout/fragment_simple"
   ... />
```

2. **Instanciar el Fragment**

```
public static SimpleFragment newInstance() {
   return new SimpleFragment();
```

```
}

SimpleFragment fragment = SimpleFragment.newInstance();
```

3. **Instanciar FragmentManager**

```
FragmentManager fragmentManager = getSupportFragmentManager();
```

4. **Usar transacciones de Fragment**

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.addToBackStack(null);
fragmentTransaction.commit();
```

Operaciones de Transacción de Fragment

- **Añadir un Fragment**: add().
- **Remover un Fragment**: remove().
- **Reemplazar un Fragment**: replace().
- **Mostrar/Ocultar un Fragment**: show() y hide().
- **Añadir a la pila de retroceso**: addToBackStack(null).

Ejemplo de Remoción de Fragment

```
SimpleFragment fragment = (SimpleFragment) fragmentManager.findFragmentById(R.id.fragment_container);
if (fragment != null) {
   FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
   fragmentTransaction.remove(fragment).commit();
}
```

3.2. Ciclo de Vida y Comunicaciones de Fragmentos en Android

Comprender el Ciclo de Vida de un Fragment

Definición

El ciclo de vida de un Fragment es similar al ciclo de vida de una Actividad. Los callbacks del ciclo de vida definen cómo se comporta el Fragment en cada estado.

Estados del Ciclo de Vida de un Fragment

Un Fragment puede estar en los siguientes estados:

- **Activo (o reanudado)**
- **Pausado**
- **Detenido**

Cómo la Actividad Afecta el Ciclo de Vida del Fragment

Estados de la Actividad y Callbacks del Fragment

- **Creado**
  - Callbacks: onAttach(), onCreate(), onCreateView(), onActivityCreated()
  - Estado del Fragment: El Fragment es añadido y su layout es inflado
- **Iniciado**
  - Callback: onStart()
  - Estado del Fragment: El Fragment está activo y visible
- **Reanudado**
  - Callback: onResume()
  - Estado del Fragment: El Fragment está activo y listo para la interacción del usuario
- **Pausado**
  - Callback: onPause()
  - Estado del Fragment: El Fragment está pausado porque la Actividad está pausada
- **Detenido**
  - Callback: onStop()
  - Estado del Fragment: El Fragment está detenido y ya no es visible
- **Destruido**
  - Callbacks: onDestroyView(), onDestroy(), onDetach()
  - Estado del Fragment: El Fragment es destruido

Uso de Callbacks del Ciclo de Vida de un Fragment

Callbacks para Activar el Fragment

- **onCreate()**: Inicializa los componentes y variables del Fragment.
- **onCreateView()**: Infla el layout XML del Fragment.

Ejemplo de onCreate()

@Override

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Inicializa componentes y variables
}
```

Ejemplo de onCreateView()

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
   // Infla el layout del Fragment y lo devuelve como la vista raíz
   return inflater.inflate(R.layout.fragment_simple, container, false);
}
```

Más Callbacks del Ciclo de Vida del Fragment

- **onAttach()**: Llamado cuando el Fragment es primero adjuntado a la Actividad.
- **onActivityCreated()**: Llamado cuando el método onCreate() de la Actividad ha retornado.
- **onDestroyView()**: Llamado cuando la vista creada por onCreateView() es separada del Fragment.

Métodos del Fragment y Contexto de la Actividad

Uso del Contexto de la Actividad

Cuando el Fragment está activo o reanudado, puedes usar getActivity() para obtener la Actividad que inició el Fragment. Esto es útil para encontrar vistas en el layout de la Actividad.

Ejemplo de Uso del Contexto

```
View listView = getActivity().findViewById(R.id.list);
```

Llamar Métodos en el Fragment

Puedes obtener un Fragment llamando a findFragmentById() en FragmentManager y luego llamar métodos en el Fragment.

Ejemplo de Llamada de Método

```
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fragment);
mData = fragment.getSomeData();
```

Uso de la Pila de Retroceso

Añadir un Fragment a la pila de retroceso permite que la Actividad anfitriona mantenga la pila de retroceso incluso después de que el Fragment sea removido. El usuario puede navegar de vuelta al Fragment usando el botón de retroceso.

Ejemplo de Añadir a la Pila de Retroceso

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.addToBackStack(null);
fragmentTransaction.commit();
```

Comunicación entre Fragment y Actividad

Enviar Datos al Fragment

Puedes enviar datos al Fragment a través del método de fábrica newInstance(), creando un Bundle y usando setArguments(Bundle) para suministrar los argumentos de construcción del Fragment.

Ejemplo de Método de Fábrica

```
public static SimpleFragment newInstance(int choice) {
   SimpleFragment fragment = new SimpleFragment();
   Bundle arguments = new Bundle();
   arguments.putInt(CHOICE, choice);
   fragment.setArguments(arguments);
   return fragment;
}
```

Usar Datos de la Actividad en el Fragment

Antes de dibujar la vista del Fragment, obtén los argumentos del Bundle usando getArguments(). Puedes hacerlo en onCreate() o onCreateView().

Ejemplo de Uso de Argumentos

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   if (getArguments().containsKey(CHOICE)) {
      mRadioButtonChoice = getArguments().getInt(CHOICE);
```

```
    }
}
```

Recuperar Datos del Fragment

Define una interfaz en el Fragment con métodos de callback, y luego, en la Actividad, implementa la interfaz para recuperar datos del Fragment.

Ejemplo de Definición de Interfaz

```
public interface OnFragmentInteractionListener {
    void onRadioButtonChoice(int choice);
}
```

Ejemplo de Implementación de la Interfaz en la Actividad

```
public class MainActivity extends AppCompatActivity implements
SimpleFragment.OnFragmentInteractionListener {
    @Override
    public void onRadioButtonChoice(int choice) {
        mRadioButtonChoice = choice;
    }
}
```

4. Widgets de apps

4.1. Widgets de Aplicación en Android

Visión General de los Widgets de Aplicación

¿Qué son los Widgets de Aplicación?

Los widgets de aplicación son vistas de aplicaciones en miniatura que aparecen en la pantalla de inicio y se actualizan con nuevos datos incluso si las aplicaciones no están en ejecución.

Instalación de Widgets

1. **Instalación del Widget**:
   - Cuando el usuario instala la aplicación, los widgets asociados aparecen en el selector de widgets.
   - El usuario mantiene presionada la pantalla de inicio y toca "Widgets".
   - El selector de widgets muestra una lista de widgets disponibles.
   - El usuario selecciona un widget para colocarlo en la pantalla de inicio.
2. **Funciones de los Widgets**:
   - Mostrar información.
   - Realizar funciones simples como mostrar la hora, resumir eventos del calendario o controlar la reproducción de música.
   - Proporcionar una lista o colección desplazable.
   - Abrir la aplicación asociada cuando se toca.

Componentes del Widget de Aplicación

Archivos y Clases Necesarios

1. **Archivo provider-info XML**: Define los metadatos del widget.
2. **Archivo layout XML**: Define la interfaz de usuario del widget.
3. **Clase AppWidgetProvider**: Código Java para el widget.
4. **Actividad de Configuración (opcional)**: Permite al usuario configurar el widget.

Ejemplo de Archivo provider-info XML

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
  android:initialLayout="@layout/new_app_widget"
  android:minHeight="180dp"
  android:minWidth="110dp"
  android:previewImage="@drawable/example_appwidget_preview"
  android:resizeMode="horizontal|vertical"
```

```
  android:updatePeriodMillis="1800000"
  android:widgetCategory="home_screen">
</appwidget-provider>
```

Clase AppWidgetProvider

Los widgets de aplicación son receptores de difusión que extienden AppWidgetProvider, la cual a su vez extiende BroadcastReceiver. Los objetos AppWidgetProvider reciben solo difusiones relevantes para el widget, incluyendo intents para actualizar, habilitar, deshabilitar y eliminar.

Agregar un Widget a una Aplicación

Pasos en Android Studio

1. Selecciona File > New > Widget > App Widget.
2. El nombre de la clase debe incluir la palabra "Widget".
3. Configurar la colocación y las opciones de pantalla de configuración (si es necesario).

Archivos Relacionados con el Widget

- Clase provider: NewAppWidget.java que extiende AppWidgetProvider.
- Archivo de layout: res/layouts/new_app_widget.xml.
- Archivo provider-info: res/xml/new_app_widget_info.xml.
- Actividad de configuración (opcional): NewAppWidgetConfigurationActivity.java.
- El manifiesto de Android se actualiza para incluir las clases provider y de configuración.

Actualizar el archivo provider-info

Atributos Importantes

- **minHeight** y **minWidth**: Tamaño mínimo del widget.
- **initialLayout**: Layout inicial del widget.
- **updatePeriodMillis**: Intervalo de actualización.
- **previewImage**: Imagen de vista previa.
- **resizeMode**: Modos de redimensionamiento.
- **widgetCategory**: Categoría del widget (pantalla de inicio).
- **configure**: Clase de la actividad de configuración.

Ejemplo de Actualización

```
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
```

```
  android:minHeight="40dp"
  android:minWidth="40dp"
  android:initialLayout="@layout/new_app_widget"
  android:updatePeriodMillis="86400000"
  android:previewImage="@drawable/new_appwidget_preview"
  android:resizeMode="horizontal|vertical"
  android:widgetCategory="home_screen"
  android:configure="com.example.android.widgettest.MyAppWidgetConfigureActivity">
</appwidget-provider>
```

Definir el Layout del Widget

Diseño del Layout

El layout del widget debe ser pequeño y mostrar información limitada. Sigue las guías de diseño de widgets y aplicaciones de Android.

Ejemplo de Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="@dimen/widget_margin">
  <ImageView
    android:id="@+id/widget_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <TextView
    android:id="@+id/widget_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

Margenes del Widget

Los widgets necesitan espacio adicional alrededor de los bordes. Android Studio crea recursos para versiones anteriores y nuevas.

Crear la Clase App Widget Provider

Implementación de la Clase

Extiende AppWidgetProvider y anula onUpdate() para construir el layout y gestionar las actualizaciones.

Ejemplo de Clase

```
public class NewAppWidget extends AppWidgetProvider {
   @Override
   public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
      for (int appWidgetId : appWidgetIds) {
         // Actualizar el widget
         RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.new_app_widget);
         views.setTextViewText(R.id.appwidget_text, String.valueOf(appWidgetId));
         appWidgetManager.updateAppWidget(appWidgetId, views);
      }
   }
}
```

Métodos Adicionales

- **onDeleted()**: Llamado cuando las instancias del widget son eliminadas.
- **onEnabled()**: Llamado cuando el widget es instanciado.
- **onDisabled()**: Llamado cuando la última instancia del widget es eliminada.

Declarar el Provider en el Manifiesto

```
<receiver android:name="NewAppWidgetProvider">
   <intent-filter>
      <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
   </intent-filter>
   <meta-data android:name="android.appwidget.provider" android:resource="@xml/example_appwidget_info" />
</receiver>
```

Actualizaciones y Acciones del Widget

Implementar Actualizaciones

Anula onUpdate() para recibir datos actualizados y configurar el widget.

Ejemplo de Implementación

```
@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
  for (int appWidgetId : appWidgetIds) {
    RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.new_app_widget);
    views.setTextViewText(R.id.appwidget_id, String.valueOf(appWidgetId));
    appWidgetManager.updateAppWidget(appWidgetId, views);
  }
}
```

Proveer Acciones al Widget

Adjunta acciones al widget con intents pendientes. Usa setOnClickPendingIntent() para conectar un PendingIntent a una o más vistas en el widget.

Ejemplo de Acción del Widget

```
@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
  for (int appWidgetId : appWidgetIds) {
    Intent intent = new Intent(context, MainActivity.class);
    PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
    RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.new_app_widget);
    views.setOnClickPendingIntent(R.id.appwidget_layout, pendingIntent);
    appWidgetManager.updateAppWidget(appWidgetId, views);
  }
}
```

Uso de una Actividad de Configuración

Configurar el Widget

Algunos widgets necesitan configuración por parte del usuario. La actividad de configuración se lanza cuando el usuario agrega el widget a la pantalla de inicio.

Añadir Actividad de Configuración

Selecciona la opción de pantalla de configuración al agregar el widget. Declara la actividad en el manifiesto con un filtro de intent que acepta ACTION_APPWIDGET_CONFIGURE.

```
<activity android:name=".ExampleAppWidgetConfigure">
   <intent-filter>
      <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
   </intent-filter>
</activity>
```

Implementar la Actividad de Configuración

Obtén el ID del widget desde los extras del intent, establece el resultado predeterminado y solicita la actualización del widget.

Ejemplo de Implementación

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

   Intent intent = getIntent();
   Bundle extras = intent.getExtras();
   if (extras != null) {
      mAppWidgetId = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID,
AppWidgetManager.INVALID_APPWIDGET_ID);
   }

   setResult(RESULT_CANCELED);

   // Configurar el widget y solicitar actualización
   AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(this);
   RemoteViews views = new RemoteViews(this.getPackageName(), R.layout.new_app_widget);
   appWidgetManager.updateAppWidget(mAppWidgetId, views);

   Intent resultValue = new Intent();
   resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
   setResult(RESULT_OK, resultValue);
   finish();
}
```

5. Sensores

5.1. Fundamentos de Sensores en Android

Categorías y Tipos de Sensores

Introducción a los Sensores

Los sensores en Android permiten medir condiciones ambientales, movimiento y orientación del dispositivo. Estos sensores se dividen en tres categorías principales:

- **Sensores de Movimiento**
- **Sensores Ambientales**
- **Sensores de Posición**

Sensores de Movimiento

Miden el movimiento del dispositivo. Los tipos incluyen:

- **Acelerómetros**
- **Sensores de gravedad**
- **Giroscopios**
- **Sensores de vector de rotación**

Sensores Ambientales

Miden las condiciones ambientales. Los tipos incluyen:

- **Barómetros**
- **Fotómetros (sensores de luz)**
- **Termómetros**

Sensores de Posición

Miden la posición física del dispositivo. Los tipos incluyen:

- **Magnetómetros (sensores de campo geomagnético)**
- **Sensores de proximidad**

Tipos de Sensores

Sensores Basados en Hardware

Son componentes físicos integrados en el dispositivo que derivan datos midiendo directamente propiedades específicas.

Ejemplos de Sensores Basados en Hardware

- **Sensor de luz**
- **Sensor de proximidad**
- **Magnetómetro**
- **Acelerómetro**

Sensores Basados en Software

Son sensores virtuales o compuestos que derivan datos de uno o más sensores de hardware.

Ejemplos de Sensores Basados en Software

- **Aceleración lineal**
- **Orientación**

Disponibilidad de Sensores

Variabilidad de Sensores

La disponibilidad de sensores varía de un dispositivo a otro y también puede variar entre versiones de Android.

- La mayoría de los dispositivos tienen acelerómetros y magnetómetros.
- Algunos dispositivos tienen barómetros o termómetros.
- Un dispositivo puede tener más de un sensor de un tipo dado.

Emulación de Sensores

Emulación en el Emulador de Android

Los controles de sensores virtuales permiten probar aplicaciones en el emulador. Se pueden simular diversos sensores desde el panel de sensores virtuales.

Pestaña de Acelerómetro

Permite probar aplicaciones para cambios en la posición y orientación del dispositivo.

Pestaña de Sensores Adicionales

Permite simular sensores de posición y ambientales como:

- **Temperatura ambiente**
- **Campo magnético**
- **Proximidad**
- **Luz**
- **Presión**
- **Humedad relativa**

Marco de Trabajo de Sensores en Android

Clases y Interfaces Importantes

SensorManager

Accede y escucha sensores, registra y desregistra los escuchadores de eventos de sensores, adquiere información de orientación y proporciona constantes para precisión, tasas de adquisición de datos y calibración.

Clases Clave

- **Sensor**: Determina las capacidades específicas de un sensor.
- **SensorEvent**: Información sobre un evento, incluidos los datos brutos del sensor.
- **SensorEventListener**: Recibe notificaciones sobre eventos del sensor.

Descubrimiento de Sensores y sus Capacidades

Identificación de Sensores

Crear una Instancia de SensorManager

SensorManager mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

Obtener la Lista de Sensores

Para obtener todos los sensores del dispositivo:

List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);

Para obtener sensores de un tipo específico:

java
Copiar código
List<Sensor> gravitySensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);

Métodos para Identificar Características del Sensor

- getResolution()
- getMaximumRange()
- getPower()
- getVendor()
- getVersion()
- getMinDelay()

Ejemplo de Identificación de Magnetómetro

```
private SensorManager mSensorManager;

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
   // Éxito: hay un magnetómetro.
} else {
   // Error: no hay magnetómetro.
}
```

Manejo de Configuraciones de Sensores

Filtros de Google Play

Usa filtros para dirigir configuraciones de sensores específicas y excluir dispositivos que no las tienen.

Declaración en el Manifiesto

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
android:required="true" />
```

Detección de Sensores en Tiempo de Ejecución

Usa getDefaultSensor() para detectar sensores específicos y ajustar las características de la aplicación en consecuencia.

Monitoreo de Eventos de Sensores

Registro de Escuchadores de Eventos de Sensores

Registra los escuchadores en onStart() y desregístralos en onStop() para garantizar que los sensores usen energía solo cuando la aplicación está en primer plano.

Ejemplo de Registro en onStart()

```
@Override
protected void onStart() {
   super.onStart();
   mSensorManager.registerListener(this, mSensorLight, SensorManager.SENSOR_DELAY_UI);
}
```

Ejemplo de Desregistro en onStop()

```
@Override
protected void onStop() {
   super.onStop();
   mSensorManager.unregisterListener(this);
}
```

Implementar SensorEventListener

Implementa la interfaz SensorEventListener con callbacks onSensorChanged() y onAccuracyChanged().

Ejemplo de SensorEventListener

```
public class SensorActivity extends Activity implements SensorEventListener {
   // ...

   @Override
   public void onSensorChanged(SensorEvent sensorEvent) {
      // Hacer algo cuando los datos del sensor cambien.
   }
```

```
    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Hacer algo si la precisión del sensor cambia.
    }
}
```

5.2. Sensores de Movimiento y Posición en Android

Visión General de los Sensores de Movimiento y Posición

Sensores de Movimiento y Posición

Los sensores de movimiento y posición monitorean el movimiento del dispositivo o su posición en el espacio, respectivamente. Ambos devuelven matrices multidimensionales de valores de sensores para cada evento del sensor.

- **Ejemplo**: El acelerómetro devuelve datos de fuerza de aceleración para los tres ejes de coordenadas (x, y, z) relativos al dispositivo.

Sistemas de Coordenadas

Sistema de Coordenadas del Dispositivo

Algunos sensores usan un sistema de coordenadas relativo al dispositivo.

- **Ejemplo**: Acelerómetros

Sistema de Coordenadas de la Tierra

Otros sensores usan un sistema de coordenadas relativo a la superficie de la Tierra.

- **Ejemplo**: Magnetómetro

Coordenadas del Dispositivo

Definición de Coordenadas del Dispositivo

Relativas al dispositivo físico, independientemente de su posición en el mundo.

- **Eje x**: Horizontal y apunta a la derecha
- **Eje y**: Vertical y apunta hacia arriba
- **Eje z**: Apunta hacia afuera de la pantalla

- **Eje z negativo**: Apunta detrás de la pantalla

Orientación del Dispositivo

Relativas a la pantalla del dispositivo en su orientación predeterminada. Los ejes no se intercambian cuando cambia la orientación por rotación. La aplicación debe transformar los datos entrantes del sensor para que coincidan con la rotación.

Coordenadas de la Tierra

Definición de Coordenadas de la Tierra

- **Eje y**: Apunta al norte magnético a lo largo de la superficie de la Tierra
- **Eje x**: 90 grados del eje y, apuntando al este
- **Eje z**: Se extiende hacia el espacio
- **Eje z negativo**: Se extiende hacia el suelo

Determinación de la Orientación del Dispositivo

Orientación del Dispositivo

Posición del dispositivo relativa a las coordenadas de la Tierra (el eje y apunta al norte magnético). Se determina usando el acelerómetro y el sensor de campo geomagnético con métodos en SensorManager.

Componentes de la Orientación

- **Azimuth**: Ángulo entre la dirección de la brújula del dispositivo y el norte magnético
- **Pitch**: Ángulo entre el plano paralelo a la pantalla del dispositivo y el plano paralelo al suelo
- **Roll**: Ángulo entre el plano perpendicular a la pantalla del dispositivo y el plano perpendicular al suelo

Métodos de SensorManager

- getRotationMatrix(): Genera una matriz de rotación a partir del acelerómetro y el sensor de campo geomagnético.
- getOrientation(): Usa la matriz de rotación para calcular los ángulos de la orientación del dispositivo.

Ejemplo de Determinación de la Orientación

```
private SensorManager mSensorManager;

final float[] rotationMatrix = new float[9];
mSensorManager.getRotationMatrix(rotationMatrix, null, accelerometerReading, magnetometerReading);

final float[] orientationAngles = new float[3];
mSensorManager.getOrientation(rotationMatrix, orientationAngles);
```

Comprensión de la Rotación del Dispositivo

Transformación de Coordenadas para la Rotación

Si la aplicación dibuja vistas basadas en datos de sensores:

- El sistema de coordenadas de la pantalla o la actividad rota con el dispositivo.
- El sistema de coordenadas del sensor no rota.
- Necesitas transformar las coordenadas del sensor a las coordenadas de la actividad.

Manejo de la Rotación del Dispositivo y la Actividad

Consulta la orientación del dispositivo con getRotationMatrix() y vuelve a mapear la matriz de rotación desde los datos del sensor a las coordenadas de la actividad con remapCoordinateSystem().

Constantes Devueltas por getRotation()

- ROTATION_0: Predeterminado (retrato para teléfonos)
- ROTATION_90: Lateral (paisaje para teléfonos)
- ROTATION_180: Al revés (si el dispositivo lo permite)
- ROTATION_270: Lateral en la dirección opuesta

Muchos dispositivos devuelven ROTATION_90 o ROTATION_270 independientemente de la rotación en el sentido de las agujas del reloj o en sentido contrario.

Ejemplo de Manejo de la Rotación del Dispositivo

```
float[] rotationMatrix = new float[9];
boolean rotationOK = SensorManager.getRotationMatrix(rotationMatrix, null, mAccelerometerData, mMagnetometerData);

float[] rotationMatrixAdjusted = new float[9];
switch (mDisplay.getRotation()) {
```

```
    case Surface.ROTATION_0:
        rotationMatrixAdjusted = rotationMatrix.clone();
        break;
    case Surface.ROTATION_90:
        SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.AXIS_Y,
SensorManager.AXIS_MINUS_X, rotationMatrixAdjusted);
        break;
    case Surface.ROTATION_180:
        SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.AXIS_MINUS_X,
SensorManager.AXIS_MINUS_Y, rotationMatrixAdjusted);
        break;
    case Surface.ROTATION_270:
        SensorManager.remapCoordinateSystem(rotationMatrix, SensorManager.AXIS_MINUS_Y,
SensorManager.AXIS_X, rotationMatrixAdjusted);
        break;
}
```

Uso de Sensores de Movimiento

Monitoreo del Movimiento del Dispositivo

Los sensores de movimiento monitorean el movimiento del dispositivo, como inclinación, sacudida, rotación o balanceo.

- **Entrada directa del usuario**: Relativa al dispositivo/aplicación (por ejemplo, dirigir un coche en un juego).
- **Movimiento del dispositivo relativo a la Tierra**: El dispositivo está contigo mientras conduces.

Tipos de Sensores de Movimiento

- **Acelerómetro (TYPE_ACCELEROMETER)**: Mide la aceleración a lo largo de los tres ejes del dispositivo, incluida la gravedad.
- **Aceleración sin gravedad (TYPE_LINEAR_ACCELERATION)**: Mide la aceleración sin incluir la gravedad.
- **Gravedad (TYPE_GRAVITY)**: Mide la fuerza de gravedad sin aceleración.
- **Giroscopio (TYPE_GYROSCOPE)**: Mide la velocidad de rotación en radianes por segundo.

Ejemplo de Uso del Acelerómetro

```
private SensorManager mSensorManager;
private Sensor mSensor;
```

```
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

Uso de Sensores de Posición

Determinación de la Posición Física del Dispositivo en la Tierra

Sensor Geomagnético (Magnetómetro)

- **TYPE_MAGNETIC_FIELD**: Mide la fuerza de los campos magnéticos alrededor del dispositivo en cada uno de los tres ejes, incluido el campo magnético de la Tierra.
- **Unidades**: Microtesla (uT)
- **Uso**: Determinar la posición del dispositivo respecto al mundo externo (brújula).

Sensor de Vector de Rotación

- **TYPE_ROTATION_VECTOR**: Proporciona la orientación respecto a las coordenadas de la Tierra como un cuaternión unitario.
- **Sensor de software**: Integra datos del acelerómetro, magnetómetro y giroscopio (si está disponible).
- **Uso**: Método eficiente y preciso para determinar la orientación del dispositivo.

Contador de Pasos y Detector de Pasos

- **TYPE_STEP_COUNTER**: Mide los pasos del usuario desde el último reinicio.
- **Uso eficiente de la batería**: Usar JobScheduler para recuperar el valor actual a intervalos específicos.
- **TYPE_STEP_DETECTOR**: Sensor de hardware que desencadena un evento para cada paso.

6. Conclusiones

A lo largo de esta cuarta unidad didáctica, hemos explorado cómo mejorar y expandir la experiencia del usuario en aplicaciones Android mediante el uso de **fragmentos** y **widgets**. Estas herramientas permiten crear interfaces más flexibles, modulares y personalizables, aspectos fundamentales en el desarrollo de aplicaciones modernas.

El uso de **fragmentos** facilita la creación de aplicaciones más escalables al permitir la reutilización de componentes de la interfaz en diferentes actividades, mejorando tanto la modularidad como la gestión de los recursos. También abordamos el ciclo de vida de los fragmentos, que es crucial para asegurar su correcta integración con la actividad que los contiene, asegurando una experiencia de usuario fluida y sin interrupciones.

En cuanto a los **widgets**, aprendimos que son una excelente forma de proporcionar información y funcionalidad de manera directa en la pantalla de inicio del dispositivo, sin necesidad de que la aplicación esté abierta. La capacidad de actualizar los widgets en tiempo real y permitir la interacción directa mejora significativamente la usabilidad y personalización, ofreciendo a los usuarios experiencias más atractivas y prácticas.

En conclusión, la correcta implementación de fragmentos y widgets en las aplicaciones Android permite no solo una mejor gestión de recursos y modularidad, sino también una interacción más rica y adaptable para el usuario, lo que incrementa tanto la funcionalidad como el atractivo visual de las aplicaciones.

7. Bibliografía

Android Developers. (2023). *Fragments: Modularizing UI Components*. Google. Disponible en: https://developer.android.com/guide/fragments

Meier, R. (2015). *Professional Android: Building Apps with Android Studio*. John Wiley & Sons.

Phillips, B., Stewart, C., Hardy, K., & Marsicano, B. (2019). *Android Programming: The Big Nerd Ranch Guide*. Big Nerd Ranch.

Burnette, E. (2018). *Creating Android Widgets: Adding Home Screen Functionality*. Apress.

Firebase. (2023). *Firebase Real-Time Data Updates for Widgets*. Google. Disponible en: https://firebase.google.com/docs/database

Estas referencias proporcionan un marco teórico y práctico robusto para entender e implementar fragmentos y widgets en el desarrollo de aplicaciones Android, mejorando la interacción y personalización para los usuarios.

UAX Universidad Alfonso X el Sabio

G R A C I A S

UAX.COM