UAX Universidad Alfonso X el Sabio

Unidad Didáctica 1: Primeros Pasos y Experiencia de 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 los conceptos fundamentales de la programación dirigida por eventos. A lo largo de esta unidad, aprenderás cómo los eventos, tales como clics de botones o entradas de datos, pueden ser manejados eficazmente para crear aplicaciones interactivas. Nos enfocaremos en el uso de Android Studio para el desarrollo de aplicaciones móviles, específicamente en la creación de interfaces de usuario interactivas y en la gestión de datos de usuarios.

Ten en cuenta:

El propósito de esta unidad es proporcionar una base sólida en la programación dirigida por eventos, lo cual es esencial para desarrollar aplicaciones modernas que respondan de manera dinámica a las acciones del usuario. La programación dirigida por eventos es un paradigma que se aplica ampliamente en el desarrollo de software, especialmente en el desarrollo de aplicaciones móviles y de escritorio.

Temas que se tratarán en esta unidad:

- Introducción a Android Studio y creación de la primera aplicación (Hello World).
- Diseño de interfaces de usuario interactivas utilizando el editor de diseño de Android Studio.
- Manejo de vistas de desplazamiento y texto en Android.
- Creación y configuración de actividades e intents en Android.
- Manejo del ciclo de vida de las actividades y estado de la aplicación.
- Implementación de eventos y manejo de recursos en una aplicación Android.
- Depuración y pruebas de aplicaciones en Android Studio.
- Uso de bibliotecas de soporte para mejorar la compatibilidad y funcionalidad de las aplicaciones.

Al finalizar esta unidad, deberías ser capaz de crear aplicaciones básicas que respondan a los eventos del usuario y gestionen datos de manera eficiente, utilizando las mejores prácticas en el desarrollo de aplicaciones Android.

2. Objetivos

2.1 Objetivo general

- Proporcionar a los estudiantes una comprensión fundamental de la programación dirigida por eventos y su aplicación en el desarrollo de aplicaciones Android, utilizando Android Studio.

2.2 Objetivos específicos

En esta unidad se establecen tres objetivos específicos:

- Comprender los conceptos básicos de la programación dirigida por eventos: Los estudiantes aprenderán cómo se manejan los eventos en las aplicaciones Android y cómo utilizar Android Studio para desarrollar aplicaciones interactivas.
- Desarrollar habilidades prácticas en el diseño de interfaces de usuario: A través de ejercicios y proyectos, los estudiantes crearán interfaces de usuario intuitivas y funcionales que respondan a eventos del usuario.
- Aplicar técnicas de depuración y pruebas en el desarrollo de aplicaciones: Los estudiantes aprenderán a identificar y corregir errores en sus aplicaciones, utilizando las herramientas de depuración y pruebas proporcionadas por Android Studio.

3. Tu Primera Aplicación Android

- 1.1: Android Studio y Hello World
- 1.2 (parte A): Tu primera IU interactiva
- 1.2 (parte B): El editor de diseño
- 1.3: Vistas de desplazamiento y texto

3.1. Android Studio y Hello World

PRERREQUISITOS

Antes de comenzar con la creación de tu aplicación, es necesario contar con los siguientes conocimientos:

- **Lenguaje de programación Java**: Conocimientos sólidos en Java, incluyendo conceptos como sintaxis, estructuras de control, y manejo de excepciones.
- **Programación orientada a objetos (OOP)**: Entender los principios de OOP, como clases, objetos, herencia, y polimorfismo.
- **XML (Extensible Markup Language)**: Familiaridad con XML para definir propiedades y atributos en los archivos de diseño.
- **Uso de un IDE (Entorno de Desarrollo Integrado)**: Experiencia en el uso de IDEs para el desarrollo y depuración de aplicaciones.

ANDROID STUDIO

¿Qué es Android Studio?

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Proporciona una serie de herramientas y características que facilitan el desarrollo, prueba y depuración de aplicaciones. Entre sus características se incluyen:

- **Plantillas de proyectos y actividades**: Para iniciar rápidamente nuevos proyectos y actividades.
- **Editor de diseño**: Herramientas visuales para diseñar interfaces de usuario.
- **Herramientas de prueba**: Incluyen emuladores y herramientas de depuración.
- **Compilación basada en Gradle**: Sistema de construcción flexible y eficiente.
- **Consola de registros y depurador**: Para monitorear y depurar aplicaciones.

- **Emuladores**: Para probar aplicaciones en diferentes versiones de Android y dispositivos.

Interfaz de Android Studio

La interfaz de Android Studio está compuesta por varios componentes clave:

- **Barra de herramientas**: Acceso rápido a funciones importantes como la ejecución y depuración de aplicaciones.
- **Barra de navegación**: Para navegar entre los diferentes componentes de tu proyecto.
- **Panel de proyecto**: Muestra la estructura del proyecto y permite acceder a los archivos.
- **Editor**: Donde se escribe y edita el código.
- **Pestañas para otros paneles**: Acceso a herramientas adicionales como la consola de Gradle, el panel de Logcat, etc.

Instalación de Android Studio

Para instalar Android Studio en tu máquina, sigue estos pasos:

1. **Descarga Android Studio**: Visita developer.android.com/studio y descarga el instalador adecuado para tu sistema operativo (Mac, Windows o Linux).
2. **Instalación**: Sigue las instrucciones del asistente de instalación para instalar Android Studio.
3. **Configuración inicial**: Una vez instalado, abre Android Studio y sigue las instrucciones para configurarlo por primera vez, incluyendo la instalación del SDK de Android y otros componentes necesarios.

Creación de tu Primera Aplicación Android

Paso 1: Iniciar Android Studio

1. **Abrir Android Studio**: Haz doble clic en el icono de Android Studio para abrir el IDE.
2. **Crear un nuevo proyecto**: En la pantalla de bienvenida, selecciona "Iniciar un nuevo proyecto de Android Studio".

Paso 2: Configurar el Nuevo Proyecto

1. **Nombrar tu aplicación**: Introduce un nombre para tu aplicación. Este nombre aparecerá en el dispositivo del usuario.

2. **Seleccionar la plantilla de actividad**: Elige una plantilla para tu actividad principal. Para una primera aplicación, selecciona "Actividad Vacía" o "Actividad Básica". Estas plantillas proporcionan una configuración mínima necesaria para comenzar.

Paso 3: Nombrar tu Actividad Principal

Es importante seguir buenas prácticas de nomenclatura:

- **Actividad principal**: Nombra tu actividad principal MainActivity.
- **Archivo de diseño**: Nombra el archivo de diseño asociado activity_main.
- **Compatibilidad**: Usa AppCompat para asegurar la compatibilidad con versiones anteriores de Android.
- **Conveniencia**: Generar el archivo de diseño automáticamente facilita el desarrollo.

ESTRUCTURA DEL PROYECTO

Un proyecto de Android se organiza en varias carpetas y archivos clave:

- **manifests**: Contiene el archivo AndroidManifest.xml, que describe los componentes de la aplicación y sus configuraciones.
- **java**: Carpeta que contiene el código fuente Java organizado en paquetes.
- **res**: Carpeta de recursos que incluye subcarpetas para diseños (layouts), cadenas de texto (strings), imágenes (drawables), dimensiones (dimens), y colores.
- **build.gradle**: Archivos de configuración de Gradle que controlan la compilación y las dependencias del proyecto.

Sistema de Compilación Gradle

Gradle es el sistema de construcción utilizado por Android Studio. Permite definir tareas de construcción y gestionar dependencias. En un proyecto típico, encontrarás tres archivos build.gradle:

- **Project-level build.gradle**: Configuración global del proyecto.
- **Module-level build.gradle**: Configuración específica de cada módulo.
- **Settings.gradle**: Define la estructura del proyecto y los módulos incluidos.

Para aprender más sobre Gradle, visita gradle.org.

Ejecución de tu Aplicación

Paso 1: Ejecutar la Aplicación

1. **Ejecutar**: Haz clic en el botón "Ejecutar" en la barra de herramientas.
2. **Seleccionar dispositivo**: Elige un dispositivo virtual (emulador) o un dispositivo físico conectado.
3. **Aceptar**: Confirma la selección para iniciar la ejecución de la aplicación.

Paso 2: Crear un Dispositivo Virtual

Para probar tu aplicación en diferentes versiones de Android y dispositivos:

1. **Abrir AVD Manager**: Ve a Herramientas > Android > AVD Manager.
2. **Crear un nuevo dispositivo virtual**: Sigue las instrucciones para seleccionar el hardware y la versión de Android deseada.

Paso 3: Configurar el Dispositivo Virtual

1. **Elegir hardware**: Selecciona el tipo de dispositivo (teléfono, tablet, etc.).
2. **Seleccionar versión de Android**: Elige la versión de Android que deseas emular.
3. **Finalizar configuración**: Completa el proceso y lanza el emulador.

Paso 4: Ejecutar en un Dispositivo Físico

Para ejecutar tu aplicación en un dispositivo físico:

1. **Activar opciones de desarrollador**:
    - Ve a Configuración > Acerca del teléfono.
    - Toca "Número de compilación" siete veces para activar las opciones de desarrollador.
    - Activa la depuración USB en Configuración > Opciones de desarrollador > Depuración USB.
2. **Conectar el dispositivo**: Usa un cable USB para conectar el dispositivo a tu computadora.
3. **Configuración adicional para Windows/Linux**: Instala los controladores USB necesarios para tu dispositivo.

Depuración y Registro

Añadir Registros a tu Aplicación

El panel de Logcat en Android Studio es una herramienta crucial para la depuración:

- **Visualizar información**: El panel de Logcat muestra registros e información relevante mientras la aplicación se ejecuta.
- **Añadir declaraciones de registro**: Incluye declaraciones de registro (Log.d, Log.e, etc.) en tu código para monitorear el comportamiento de la aplicación.

Ejemplo de Declaración de Registro en Java

```
import android.util.Log;

// Usa el nombre de la clase como etiqueta
private static final String TAG = MainActivity.class.getSimpleName();

// Mostrar mensaje en el logcat del monitor de Android
Log.d(TAG, "Creando la URI...");
```

Uso del Panel Logcat

- **Abrir Logcat**: Haz clic en la pestaña Logcat en la parte inferior de Android Studio.
- **Filtrar registros**: Usa el menú de nivel de registro para seleccionar el tipo de mensajes que deseas ver (debug, error, etc.).
- **Buscar usando etiquetas**: Filtra los mensajes por etiqueta para encontrar información relevante rápidamente.

3.2. Layouts y Recursos para la UI en Android

VISTAS

¿Qué es una vista?

Una vista es el bloque básico de construcción de la interfaz de usuario en Android. Todas las interfaces de usuario en una aplicación Android están compuestas de vistas. Las subclases de la clase View incluyen:

- **TextView**: Para mostrar texto.
- **EditText**: Para editar texto.
- **Button**: Para botones.
- **ScrollView** y **RecyclerView**: Para vistas desplazables.
- **ImageView**: Para mostrar imágenes.
- **ConstraintLayout** y **LinearLayout**: Para agrupar vistas.

Ejemplos de subclases de vistas

- **Button**
- **EditText**
- **Slider**
- **CheckBox**
- **RadioButton**
- **Switch**

Atributos de las vistas

Las vistas pueden tener varios atributos, como:

- **Color**
- **Dimensiones**
- **Posicionamiento**
- **Interactividad** (por ejemplo, respuesta a clics)
- **Visibilidad**

CREACIÓN DE VISTAS Y LAYOUTS

Editor de Diseño en Android Studio

El editor de diseño en Android Studio proporciona una representación visual del archivo XML de diseño, facilitando la creación y modificación de la UI.

Componentes del Editor de Diseño

- **Archivo de diseño XML**
- **Pestañas de Diseño y Texto**
- **Panel de Paleta**
- **Árbol de Componentes**
- **Paneles de Diseño y Esquema**
- **Pestaña de Atributos**

Definición de vistas en XML

Ejemplo de una vista `TextView` en XML:

```
<TextView
    android:id="@+id/show_count"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:background="@color/myBackgroundColor"
    android:text="@string/count_initial_value"
    android:textColor="@color/colorPrimary"
    android:textSize="@dimen/count_text_size"
    android:textStyle="bold" />
```

Creación de vistas en código Java

Ejemplo en una actividad:

```
TextView myText = new TextView(this);
myText.setText("Display this text!");
```

Contexto en Android

El contexto es una interfaz para acceder a la información global sobre el entorno de la aplicación. Se obtiene usando `getApplicationContext()`. Una actividad puede actuar como su propio contexto:

```
TextView myText = new TextView(this);
```

Grupos de Vistas y Jerarquía de Vistas

ViewGroup y jerarquía de vistas

Un `ViewGroup` es un contenedor que puede contener otras vistas. Ejemplos de `ViewGroup` incluyen `ConstraintLayout`, `ScrollView` y `RecyclerView`.

Layouts Comunes

- **LinearLayout**: Organiza las vistas en una fila horizontal o vertical.
- **ConstraintLayout**: Conecta las vistas con restricciones.
- **GridLayout**: Organiza las vistas en una cuadrícula.
- **TableLayout**: Organiza las vistas en filas y columnas.
- **FrameLayout**: Muestra una vista de una pila de vistas.

Ejemplo de LinearLayout en XML

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button ... />
```

```
    <TextView ... />
    <Button ... />
</LinearLayout>
```

Ejemplo de LinearLayout en código Java

```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);
TextView myText = new TextView(this);
myText.setText("Display this text!");
linearL.addView(myText);
setContentView(linearL);
```

Mejores Prácticas para la Jerarquía de Vistas

- Usa el menor número posible de vistas simples.
- Mantén la jerarquía plana, limitando el anidamiento de vistas y grupos de vistas.

EL EDITOR DE DISEÑO Y CONSTRAINTLAYOUT

¿Qué es ConstraintLayout?

`ConstraintLayout` es el layout predeterminado para nuevos proyectos en Android Studio. Ofrece flexibilidad para el diseño de la interfaz, permitiendo definir restricciones que determinan la posición y alineación de los elementos de la UI.

Barra de Herramientas del Editor de Diseño

- **Select Design Surface**: Paneles de Diseño y Esquema.
- **Orientation in Editor**: Retrato y Paisaje.
- **Device in Editor**: Elegir dispositivo para vista previa.
- **API Version in Editor**: Elegir API para vista previa.
- **Theme in Editor**: Elegir tema para vista previa.
- **Locale in Editor**: Elegir idioma/localización para vista previa.

Barra de Herramientas de constraintlayout

- **Show Constraints**: Mostrar restricciones.
- **Autoconnect**: Habilitar o deshabilitar autoconexión.
- **Clear All Constraints**: Borrar todas las restricciones en el diseño.
- **Infer Constraints**: Crear restricciones por inferencia.
- **Default Margins**: Establecer márgenes predeterminados.
- **Pack**: Agrupar o expandir elementos seleccionados.
- **Align**: Alinear elementos seleccionados.
- **Guidelines**: Añadir guías verticales u horizontales.

- **Zoom controls**: Acercar o alejar.

USO DE AUTOCONNECT

Habilita Autoconnect en la barra de herramientas y arrastra un elemento a cualquier parte del diseño. Autoconnect generará restricciones contra el layout padre.

MANEJO DE EVENTOS

Eventos en Android

Un evento es algo que ocurre en la UI (como un clic, toque o arrastre) o en el dispositivo (como detectar una actividad).

Manejadores de Eventos

Los métodos que responden a eventos específicos se llaman manejadores de eventos. Estos métodos se activan en respuesta a un evento específico.

Adjuntar Manejadores en XML e Implementar en Java

Ejemplo en XML:

```
<Button
    android:onClick="showToast"
    ... />
```

Ejemplo en Java:

```
public void showToast(View view) {
    String msg = "Hello Toast!";
    Toast toast = Toast.makeText(this, msg, Toast.LENGTH_SHORT);
    toast.show();
}
```

Alternativa: Establecer el Manejador de Clic en Java

```
final Button button = findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, Toast.LENGTH_SHORT);
        toast.show();
    }
```

```
});
```

RECURSOS Y MEDICIONES

Recursos

Separa los datos estáticos del código en tus layouts. Los recursos pueden incluir cadenas de texto, dimensiones, imágenes, menús, colores y estilos. Esto es útil para la localización.

Ubicación de los Recursos en tu Proyecto

Los recursos y archivos de recursos se almacenan en la carpeta `res`.

Referenciar Recursos en el Código

- **Layout**:

```
java
Copiar código
setContentView(R.layout.activity_main);
```

- **Vista**:

```
java
Copiar código
rv = findViewById(R.id.recyclerview);
```

- **Cadena de Texto**:
  - En Java: `R.string.title`
  - En XML: `android:text="@string/title"`

Mediciones

- **Density-independent Pixels (dp)**: Para vistas.
- **Scale-independent Pixels (sp)**: Para texto.
- No uses unidades dependientes del dispositivo o la densidad: Pixels reales (px), Medición real (in, mm), Puntos tipográficos (pt).

Notas sobre Mediciones

- Los píxeles independientes de la densidad (dp) son independientes de la resolución de la pantalla. Por ejemplo, 10px se verá mucho más pequeño en una pantalla de alta

resolución, pero Android escalará 10dp para que se vea bien en diferentes resoluciones.
- `sp` hace lo mismo para el tamaño del texto.

3.3. Vistas de Texto y Desplazamiento en Android

TEXTVIEW

¿Qué es un TextView?

`TextView` es una subclase de `View` utilizada para mostrar texto de una o varias líneas. `EditText` es una subclase de `TextView` que permite la edición de texto. Los `TextView` pueden ser controlados mediante atributos de diseño.

Configuración del Texto

El texto puede establecerse de dos formas:

- **Estáticamente**: Desde un recurso de cadena en XML.
- **Dinámicamente**: Desde código Java y cualquier otra fuente.

Formateo de Texto en Recursos de Cadena

- Usa etiquetas HTML `<b>` y `<i>` para negritas y cursivas.
- Todas las demás etiquetas HTML son ignoradas.
- Los recursos de cadena deben estar en una sola línea continua para cada párrafo.
- `\n` inicia una nueva línea o párrafo.
- Escapa apóstrofes y comillas con una barra invertida (`\" \`).
- Escapa cualquier carácter no ASCII con una barra invertida (`\`).

Creación de un TextView en XML

Ejemplo:

```
<TextView android:id="@+id/textview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/my_story"/>
```

Atributos Comunes de TextView

- `android:text`: Texto a mostrar.

- `android:textColor`: Color del texto.
- `android:textAppearance`: Estilo o tema predefinido.
- `android:textSize`: Tamaño del texto en `sp`.
- `android:textStyle`: Estilo del texto (normal, negrita, cursiva, negrita/cursiva).
- `android:typeface`: Tipo de letra (normal, sans-serif o monospace).
- `android:lineSpacingExtra`: Espacio extra entre líneas en `sp`.

Formateo de Enlaces Web Activos

Ejemplo en XML:

```
<string name="article_text">... www.rockument.com ...</string>

<TextView
   android:id="@+id/article"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:autoLink="web"
   android:text="@string/article_text"/>
```

Valores para `autoLink`: "web", "email", "phone", "map", "all".

Creación de TextView en Código Java

Ejemplo:

```
TextView myTextView = new TextView(this);
myTextView.setWidth(LayoutParams.MATCH_PARENT);
myTextView.setHeight(LayoutParams.WRAP_CONTENT);
myTextView.setMinLines(3);
myTextView.setText(R.string.my_story);
myTextView.append(userComment);
```

SCROLLVIEW

¿Qué es un ScrollView?

`ScrollView` es una subclase de `FrameLayout` que permite el desplazamiento de su contenido. Es útil para manejar grandes cantidades de texto, como noticias o artículos.

Uso de ScrollView

Para desplazar un `TextView`, insértalo dentro de un `ScrollView`. Solo un elemento `View` (generalmente `TextView`) está permitido dentro de un `ScrollView`. Para desplazar múltiples elementos, usa un `ViewGroup` (como `LinearLayout`) dentro del `ScrollView`.

Ejemplo de ScrollView con TextView

```
<ScrollView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_below="@id/article_subheading">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    .../>

</ScrollView>
```

Ejemplo de ScrollView con un Grupo de Vistas

```
<ScrollView>
   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="vertical">

       <TextView
           android:id="@+id/article_subheading"
           .../>

       <TextView
           android:id="@+id/article"
           ... />

   </LinearLayout>
</ScrollView>
```

Ejemplo de ScrollView con Imagen y Botón

```
<ScrollView>
    <LinearLayout>
        <ImageView .../>
        <Button .../>
        <TextView .../>
    </LinearLayout>
</ScrollView>
```

4. Actividades e Intents en Android

2.1: Intents y actividades

2.2: Estado y ciclo de vida de la actividad

2.3: Intents implícitos

4.1. Intents y actividades

ACTIVIDADES

¿Qué es una Actividad?

Una `Activity` es un componente de aplicación que representa una ventana o una jerarquía de vistas en Android. Aunque generalmente llena toda la pantalla, una `Activity` puede ser incrustada en otra o aparecer como una ventana flotante. En la mayoría de los casos, una `Activity` está representada por una clase Java individual.

Funciones de una Actividad

- Representa una tarea específica, como ordenar comestibles, enviar un correo electrónico o obtener direcciones.
- Maneja interacciones del usuario como clics de botones, entrada de texto o verificación de inicio de sesión.
- Puede iniciar otras actividades dentro de la misma aplicación o en aplicaciones diferentes.
- Tiene un ciclo de vida que incluye los estados de creación, inicio, ejecución, pausa, reanudación, detención y destrucción.

Ejemplos de Actividades

- Ordenar comestibles
- Enviar correos electrónicos
- Obtener direcciones

Aplicaciones y Actividades

Las actividades se enlazan entre sí para formar una aplicación. La primera actividad que ve el usuario suele llamarse "actividad principal" (main activity). Las actividades pueden organizarse en relaciones de padre-hijo en el manifiesto de Android para facilitar la navegación.

Diseños y Actividades

Una Activity generalmente tiene un diseño de UI que se define en uno o más archivos XML. La actividad "infla" el diseño como parte de su creación.

IMPLEMENTACIÓN DE ACTIVIDADES

Pasos para Implementar Nuevas Actividades

1. **Definir el diseño en XML**

   Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="¡Vamos a comprar comida!" />
</RelativeLayout>
```

2. **Definir la clase Java de la Actividad**

   Ejemplo:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
   }
}
```

3. **Conectar la actividad con el diseño**

   Ejemplo:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
}
```

4. **Declarar la actividad en el manifiesto de Android**

   Ejemplo:

```
<activity android:name=".MainActivity">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER"
/>
   </intent-filter>
</activity>
```

INTENTS

¿Qué es un Intent?

Un `Intent` es una descripción de una operación a realizar. Es un objeto utilizado para solicitar una acción de otro componente de la aplicación a través del sistema Android.

Funciones de los Intents

- **Iniciar una Actividad**: Un clic en un botón puede iniciar una nueva actividad para la entrada de texto.
- **Iniciar un Servicio**: Por ejemplo, iniciar la descarga de un archivo en segundo plano.
- **Entregar un Broadcast**: El sistema informa a todas las aplicaciones de que el teléfono está cargando.

Intents Explícitos e Implícitos

- **Intent Explícito**: Inicia una actividad específica. Ejemplo: La actividad principal inicia la actividad `ViewShoppingCart`.
- **Intent Implícito**: Solicita al sistema que encuentre una actividad que pueda manejar la solicitud. Ejemplo: Al hacer clic en "Compartir", se abre un selector con una lista de aplicaciones.

INICIO DE ACTIVIDADES

Iniciar una Actividad con un Intent Explícito

Ejemplo:

```
Intent intent = new Intent(this, ActivityName.class);
startActivity(intent);
```

Iniciar una Actividad con un Intent Implícito

Ejemplo:

```
Intent intent = new Intent(action, uri);
startActivity(intent);
```

Ejemplos de Intents Implícitos

- **Mostrar una página web**:

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW, uri);
startActivity(it);
```

- **Marcar un número de teléfono**:

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

Enviar y Recibir Datos

Tipos de Datos con Intents

- **Data**: Una pieza de información cuya ubicación puede ser representada por un URI.
- **Extras**: Una o más piezas de información como una colección de pares clave-valor en un `Bundle`.

Enviar y Recuperar Datos

1. **En la primera actividad (envío)**:
   - Crear el objeto `Intent`.
   - Colocar los datos o extras en el `Intent`.
   - Iniciar la nueva actividad con `startActivity()`.
2. **En la segunda actividad (recepción)**:
   - Obtener el objeto `Intent` con el que se inició la actividad.
   - Recuperar los datos o extras del objeto `Intent`.

Ejemplo de Envío de Datos con Extras

Ejemplo:

```
public static final String EXTRA_MESSAGE_KEY =
"com.example.android.twoactivities.extra.MESSAGE";

Intent intent = new Intent(this, SecondActivity.class);
String message = "¡Hola, Actividad!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

Ejemplo de Recuperación de Datos

Ejemplo:

```
int level = intent.getIntExtra("level", 0);
Bundle bundle = intent.getExtras();
```

Retornar Datos a la Actividad Iniciadora

Usa `startActivityForResult()` para iniciar la segunda actividad y retornar datos:

1. **En la primera actividad**:

```
public static final int CHOOSE_FOOD_REQUEST = 1;
Intent intent = new Intent(this, ChooseFoodItemsActivity.class);
startActivityForResult(intent, CHOOSE_FOOD_REQUEST);
```

2. **En la segunda actividad**:

```
Intent replyIntent = new Intent();
replyIntent.putExtra(EXTRA_REPLY, reply);
setResult(RESULT_OK, replyIntent);
finish();
```

3. **Implementar `onActivityResult()` en la primera actividad**:

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == CHOOSE_FOOD_REQUEST) {
        if (resultCode == RESULT_OK) {
            String reply =
data.getStringExtra(SecondActivity.EXTRA_REPLY);
            // Procesar los datos retornados
        }
    }
}
```

NAVEGACIÓN

Pila de Actividades

Cuando se inicia una nueva actividad, la actividad anterior se detiene y se empuja en la pila de actividades. Al finalizar la actividad actual o al presionar el botón de retroceso, la actividad se elimina de la pila y se reanuda la actividad anterior.

Formas de Navegación

- **Navegación Temporal o Atrás**: Proporcionada por el botón de retroceso del dispositivo y controlada por la pila de retroceso del sistema Android.
- **Navegación Ancestral o Arriba**: Proporcionada por el botón de "arriba" en la barra de acción de la aplicación y controlada definiendo relaciones de padre-hijo entre actividades en el manifiesto de Android.

4.2. Ciclo de Vida y Estado de una Actividad en Android

CICLO DE VIDA DE UNA ACTIVIDAD

¿Qué es el Ciclo de Vida de una Actividad?

El ciclo de vida de una actividad es el conjunto de estados en los que puede encontrarse una actividad desde su creación hasta su destrucción. Formalmente, es un gráfico dirigido de todos los estados por los que puede pasar una actividad y las callbacks asociadas con la transición de un estado a otro.

Estados de la Actividad y Visibilidad de la Aplicación

- **Created (no visible aún)**
- **Started (visible)**
- **Resume (visible)**
- **Paused (parcialmente invisible)**
- **Stopped (oculta)**
- **Destroyed (eliminada de la memoria)**

Los cambios de estado se desencadenan por acciones del usuario, cambios de configuración como la rotación del dispositivo o acciones del sistema.

CALLBACKS DEL CICLO DE VIDA DE LA ACTIVIDAD

Callbacks y Cuándo se Llaman

1. `onCreate(Bundle savedInstanceState)`: Inicialización estática.
2. `onStart()`: Cuando la actividad (pantalla) se está volviendo visible.
3. `onRestart()`: Llamado si la actividad fue detenida (llama a `onStart()`).
4. `onResume()`: Comienza a interactuar con el usuario.
5. `onPause()`: Está a punto de reanudar la actividad anterior.
6. `onStop()`: Ya no es visible pero aún existe y toda la información del estado se conserva.
7. `onDestroy()`: Llamada final antes de que el sistema Android destruya la actividad.

Implementación y Sobrescritura de Callbacks

La única callback obligatoria es `onCreate()`. Es necesario sobrescribir las demás callbacks para cambiar el comportamiento predeterminado.

IMPLEMENTACIÓN DE CALLBACKS

`onCreate(Bundle savedInstanceState)`

Llamada cuando la actividad se crea por primera vez, por ejemplo, cuando el usuario toca el ícono de la aplicación. Realiza toda la configuración estática: crear vistas, vincular datos a listas, etc. Solo se llama una vez durante la vida de la actividad. Toma un `Bundle` con el estado previamente guardado de la actividad, si existía.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // La actividad está siendo creada.
}
onStart()
```

Llamada cuando la actividad se está volviendo visible para el usuario. Puede llamarse más de una vez durante el ciclo de vida. Seguido por `onResume()` si la actividad pasa al primer plano o `onStop()` si se oculta.

```
@Override
protected void onStart() {
    super.onStart();
    // La actividad está a punto de ser visible.
}
onRestart()
```

Llamada después de que la actividad ha sido detenida, inmediatamente antes de volver a iniciarse. Estado transitorio, siempre seguido por `onStart()`.

```
@Override
protected void onRestart() {
    super.onRestart();
    // La actividad está entre detenida e iniciada.
}
onResume()
```

Llamada cuando la actividad comenzará a interactuar con el usuario. La actividad ha pasado a la parte superior de la pila de actividades. Comienza a aceptar la entrada del usuario. Estado de ejecución, siempre seguido por `onPause()`.

```
@Override
protected void onResume() {
    super.onResume();
    // La actividad se ha vuelto visible.
    // Ahora está "resumida".
}
onPause()
```

Llamada cuando el sistema está a punto de reanudar una actividad anterior. La actividad es parcialmente visible pero el usuario está saliendo de la actividad. Generalmente se usa para comprometer cambios no guardados en datos persistentes, detener animaciones y cualquier cosa que consuma recursos. Las implementaciones deben ser rápidas porque la siguiente actividad no se reanuda hasta que este método regrese. Seguido por `onResume()` si la actividad vuelve al frente o `onStop()` si se vuelve invisible para el usuario.

```
@Override
protected void onPause() {
    super.onPause();
    // Otra actividad está tomando el foco
    // esta actividad está a punto de ser "pausada".
}
onStop()
```

Llamada cuando la actividad ya no es visible para el usuario. Una nueva actividad se está iniciando, una existente se trae al frente de esta o esta se está destruyendo. Operaciones que eran demasiado pesadas para `onPause()`. Seguido por `onRestart()` si la actividad vuelve a interactuar con el usuario o `onDestroy()` si la actividad desaparece.

```
@Override
protected void onStop() {
    super.onStop();
    // La actividad ya no es visible.
    // Ahora está "detenida".
}
onDestroy()
```

Llamada final antes de que la actividad sea destruida. El usuario navega de regreso a la actividad anterior o cambia la configuración. La actividad está finalizando o el sistema la está destruyendo para liberar espacio. Llama al método `isFinishing()` para verificar. El sistema puede destruir la actividad sin llamar a este método, por lo que se debe usar `onPause()` o `onStop()` para guardar datos o estado.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    // La actividad está a punto de ser destruida.
}
```

Estado de la Instancia de la Actividad

Cambios de Configuración

Los cambios de configuración invalidan el diseño actual u otros recursos en tu actividad cuando el usuario:

- Rota el dispositivo.
- Elige un idioma diferente del sistema, por lo que cambia la localización.
- Entra en modo de ventana múltiple (desde Android 7).

Qué Sucede en un Cambio de Configuración

En un cambio de configuración, Android:

1. Apaga la actividad llamando a:

   ```
   onPause()
   onStop()
   onDestroy()
   ```

2. Reinicia la actividad llamando a:

   ```
   onCreate()
   onStart()
   onResume()
   ```

Guardar y Restaurar el Estado de la Actividad

¿Qué Guarda el Sistema?

El sistema solo guarda:

- El estado de las vistas con ID único (`android:id`), como el texto ingresado en `EditText`.
- El intent que inició la actividad y los datos en sus extras.

Eres responsable de guardar otros datos de la actividad y el progreso del usuario.

Guardar el Estado de la Instancia

Implementa `onSaveInstanceState()` en tu actividad. Llamado por el tiempo de ejecución de Android cuando existe la posibilidad de que la actividad sea destruida. Guarda datos solo para esta instancia de la actividad durante la sesión actual.

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // Agregar información para guardar el contador de HelloToast
    // en el bundle outState.
    outState.putString("count", String.valueOf(mShowCount.getText()));
}
```

Restaurar el Estado de la Instancia

Dos formas de recuperar el `Bundle` guardado:

1. En `onCreate(Bundle mySavedState)`: Preferido para asegurar que la interfaz de usuario, incluido cualquier estado guardado, esté de nuevo en funcionamiento lo antes posible.

   ```
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      mShowCount = findViewById(R.id.show_count);

      if (savedInstanceState != null) {
          String count = savedInstanceState.getString("count");
          if (mShowCount != null)
              mShowCount.setText(count);
      }
   }
   ```

2. Implementar la callback `onRestoreInstanceState(Bundle mySavedState)` (llamada después de `onStart()`).

   ```
   @Override
   public void onRestoreInstanceState(Bundle mySavedState) {
      super.onRestoreInstanceState(mySavedState);

      if (mySavedState != null) {
          String count = mySavedState.getString("count");
   ```

```
            if (count != null)
                mShowCount.setText(count);
        }
    }
```

Estado de la Instancia y Reinicio de la Aplicación

Cuando detienes y reinicias una nueva sesión de la aplicación, los estados de instancia de la actividad se pierden y tus actividades volverán a su apariencia predeterminada. Si necesitas guardar datos del usuario entre sesiones de la aplicación, usa preferencias compartidas o una base de datos.

4.3. Intents Implícitos en Android

¿Qué es un Intent?

Un `Intent` es una descripción de una operación a realizar. Es un objeto de mensajería utilizado para solicitar una acción de otro componente de la aplicación a través del sistema Android.

- **App Component**: Componente de la aplicación.
- **Originator**: Componente que inicia el intent.
- **Intent**: Descripción de la acción.
- **Action**: Acción a realizar.
- **Android System**: Sistema que maneja el intent.

Funciones de un Intent

Un `Intent` se puede utilizar para:

- **Iniciar una Actividad**
- **Iniciar un Servicio**
- **Entregar un Broadcast**

Intents Explícitos vs Implícitos

- **Intent Explícito**: Inicia una actividad de una clase específica.
- **Intent Implícito**: Solicita al sistema que encuentre una clase de actividad con un manejador registrado que pueda manejar esta solicitud.

DESCRIPCIÓN GENERAL DE LOS INTENTS IMPLÍCITOS

¿Qué haces con un Intent Implícito?

Inicias una actividad en otra aplicación describiendo una acción que deseas realizar, como "compartir un artículo", "ver un mapa" o "tomar una foto". Especificas una acción y opcionalmente proporcionas datos con los que realizar la acción. No especificas la clase de actividad objetivo, solo la acción deseada.

¿Qué hace el sistema con un Intent Implícito?

El tiempo de ejecución de Android coincide con la solicitud de intent implícito con los manejadores de intent registrados. Si hay múltiples coincidencias, se abrirá un selector de aplicaciones para permitir que el usuario decida.

¿Cómo funciona un Intent Implícito?

El tiempo de ejecución de Android mantiene una lista de aplicaciones registradas. Las aplicaciones deben registrarse a través del archivo `AndroidManifest.xml`. El tiempo de ejecución recibe la solicitud y busca coincidencias utilizando filtros de intent. Si hay más de una coincidencia, muestra una lista de posibles coincidencias y permite que el usuario elija una. El tiempo de ejecución de Android inicia la actividad solicitada.

ENVÍO DE UN INTENT IMPLÍCITO

Envío de un Intent Implícito

1. **Crear un Intent para una acción**

   ```
   Intent intent = new Intent(Intent.ACTION_CALL_BUTTON);
   ```

   El usuario ha presionado el botón de llamada, inicia la actividad que puede realizar una llamada (no se pasan datos ni se devuelven).

2. **Iniciar la Actividad**

   ```
   if (intent.resolveActivity(getPackageManager()) != null) {
       startActivity(intent);
   }
   ```

Evitar Excepciones y Bloqueos

Antes de iniciar una actividad implícita, utiliza el administrador de paquetes para verificar que haya un paquete con una actividad que coincida con los criterios dados.

```
Intent myIntent = new Intent(Intent.ACTION_CALL_BUTTON);
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
```

```
}
```

Envío de un Intent Implícito con URI de Datos

1. **Crear un Intent para una acción**

```
Intent intent = new Intent(Intent.ACTION_DIAL);
```

2. **Proporcionar datos como un URI**

```
intent.setData(Uri.parse("tel:8005551234"));
```

3. **Iniciar la Actividad**

```
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

Ejemplos de URI

- **Teléfono**: `Uri.parse("tel:8005551234")`
- **Geolocalización**: `Uri.parse("geo:0,0?q=brooklyn+bridge,brooklyn,ny")`
- **Web**: `Uri.parse("http://www.android.com")`

Ejemplos de Intents Implícitos

- **Mostrar una página web**:

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW, uri);
startActivity(it);
```

- **Marcar un número de teléfono**:

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL, uri);
startActivity(it);
```

Envío de un Intent Implícito con Extras

1. **Crear un Intent para una acción**

```
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
```

2. **Poner Extras**

```
String query = edittext.getText().toString();
intent.putExtra(SearchManager.QUERY, query);
```

3. **Iniciar la Actividad**

```
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

Categoría

Información adicional sobre el tipo de componente para manejar el intent.

- **CATEGORY_OPENABLE**: Permite solo URIs de archivos que se pueden abrir.
- **CATEGORY_BROWSABLE**: Solo una actividad que pueda iniciar un navegador web para mostrar datos referenciados por el URI.

Envío de un Intent Implícito con Tipo y Categoría

1. **Crear un Intent para una acción**

```
Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
```

2. **Establecer tipo MIME y categoría para información adicional**

```
intent.setType("application/pdf");
intent.addCategory(Intent.CATEGORY_OPENABLE);
```

3. **Iniciar la Actividad**

```
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivityForResult(intent,
ACTIVITY_REQUEST_CREATE_FILE);
}
```

4. **Procesar el URI de contenido devuelto en `onActivityResult()`**

RECEPCIÓN DE UN INTENT IMPLÍCITO

Registrar tu Aplicación para Recibir un Intent

Declara uno o más filtros de intent para la actividad en `AndroidManifest.xml`. El filtro anuncia la capacidad de la actividad para aceptar un intent implícito y establece condiciones en el intent que la actividad acepta.

Filtro de Intent en `AndroidManifest.xml`

Ejemplo:

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Filtros de Intent: Acción y Categoría

- **Acción**: Coincide con una o más constantes de acción.
  - `android.intent.action.VIEW`: Coincide con cualquier intent con `ACTION_VIEW`.
  - `android.intent.action.SEND`: Coincide con cualquier intent con `ACTION_SEND`.
- **Categoría**: Información adicional (lista de categorías).
  - `android.intent.category.BROWSABLE`: Puede ser iniciado por un navegador web.
  - `android.intent.category.LAUNCHER`: Muestra la actividad como icono de inicio.

Filtros de Intent: Datos

- **Datos**: Filtrar en URIs de datos y tipos MIME.
  - `android:scheme="https"`: Requiere que los URIs sean del protocolo HTTPS.
  - `android:host="developer.android.com"`: Solo acepta un intent de hosts especificados.
  - `android:mimeType="text/plain"`: Limita los tipos de documentos aceptables.

Un Filtro puede tener Múltiples Acciones y Datos

Ejemplo:

```
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="image/*"/>
    <data android:mimeType="video/*"/>
</intent-filter>
```

5. Prueba, depuración y uso de bibliotecas de compatibilidad

5.1. El Depurador de Android Studio

TODOS LOS CÓDIGOS TIENEN ERRORES

Errores

Los errores en el código pueden manifestarse de varias formas:

- **Resultados incorrectos o inesperados**: Valores erróneos que no corresponden con los esperados.
- **Bloqueos y excepciones**: Fallos en tiempo de ejecución que pueden congelar la aplicación o provocar cierres inesperados.
- **Fugas de memoria**: Recursos que no se liberan adecuadamente, causando un uso excesivo de memoria.

Causas

- **Error humano**: Errores en el diseño o implementación del código que necesitan ser corregidos.
- **Fallas de software**: Problemas en las bibliotecas utilizadas, que pueden requerir soluciones alternativas.
- **Limitaciones o fallas de hardware**: Adaptación del software para funcionar con el hardware disponible.

Origen del Término "Bug"

El término "bug" tiene una historia interesante en el contexto de la informática, que se remonta a los primeros días de la computación.

DEPURACIÓN

Objetivo de la Depuración

La depuración es el proceso de encontrar y corregir errores en el código. El objetivo es corregir comportamientos inesperados y no deseados para asegurar que el software funcione correctamente.

- **Pruebas unitarias**: Ayudan a identificar errores específicos y a prevenir regresiones.
- **Pruebas de usuario**: Ayudan a identificar errores de interacción y usabilidad.

Herramientas de Depuración en Android Studio

Android Studio ofrece herramientas integradas que ayudan a:

- Identificar problemas en el código.
- Encontrar el origen de estos problemas dentro del código fuente para corregirlos eficazmente.

REGISTRO EN ANDROID STUDIO

Añadir Mensajes de Registro a tu Código

Para añadir mensajes de registro en tu código, sigue estos pasos:

1. **Importa la clase de registro**:

```
import android.util.Log;
```

2. **Usa una variable de clase con el nombre de la clase como etiqueta**:

```
private static final String TAG =
MainActivity.class.getSimpleName();
```

3. **Muestra un mensaje en el panel Logcat de Android Studio**:

```
Log.d(TAG, "Hello World");
```

Abrir el Panel de Logcat

Para visualizar los mensajes de registro, abre el panel de Logcat en Android Studio.

Inspeccionar Mensajes de Registro

Ejemplo de mensaje de registro:

```
Log.d("MainActivity", "Hello World");
```

Salida en Logcat:

```
09-12 14:28:07.971 4304 /com.example.android.helloworld
D/MainActivity: Hello World
```

Elegir el Nivel de Registro Visible

Elige el nivel de registro para mostrar los logs con ese nivel o superior.

Niveles de Registro

Verbose: Muestra todos los mensajes de registro detallados y del sistema.

Debug: Muestra registros de depuración, valores de variables y notas de depuración.

Info: Muestra información de estado, como la conexión a la base de datos.

Warning: Muestra comportamientos inesperados y problemas no fatales.

Error: Muestra condiciones de error graves, excepciones y bloqueos.

DEPURACIÓN CON ANDROID STUDIO

Funcionalidades del Depurador

El depurador de Android Studio permite:

- **Ejecutar en modo de depuración**: Con el depurador adjunto.
- **Establecer y configurar puntos de interrupción**: Para detener la ejecución en líneas específicas del código.
- **Inspeccionar pilas de ejecución**: Ver los frames de ejecución y valores de variables.
- **Cambiar valores de variables**: Durante la ejecución.
- **Recorrer el código línea por línea**: Utilizando comandos de paso.
- **Pausar y reanudar**: Un programa en ejecución.

Ejecutar en Modo de Depuración

1. Abre el menú: `Run > Debug 'tu aplicación'`.
2. El panel del depurador se abrirá automáticamente, mostrando el estado de la depuración.

Establecer Puntos de Interrupción

Para establecer un punto de interrupción, haz clic en el margen izquierdo junto a la línea de código donde deseas detener la ejecución.

Editar Propiedades del Punto de Interrupción

Puedes hacer clic derecho en un punto de interrupción existente para editar sus propiedades, incluyendo la condición bajo la cual se activará.

Hacer que los Puntos de Interrupción sean Condicionales

Puedes establecer condiciones para los puntos de interrupción utilizando cualquier expresión Java que devuelva un booleano. Esto se puede hacer en el cuadro de diálogo de propiedades del punto de interrupción o haciendo clic derecho en un punto de interrupción existente.

Inspección y Edición de Variables

Inspeccionar Marcos

El marco superior es donde se detiene la ejecución en tu código. Puedes inspeccionar los valores de las variables y el estado actual del programa en este punto.

Inspeccionar y Editar Variables

Para cambiar el valor de una variable durante la ejecución, haz clic derecho en la variable y selecciona la opción correspondiente del menú contextual.

Comandos Básicos de Recorrido

Comandos de Recorrido Básicos

- **Step Over (F8)**: Pasa a la siguiente línea en el archivo actual.
- **Step Into (F7)**: Entra en la siguiente línea ejecutada, incluyendo métodos.
- **Force Step Into (Shift + F7)**: Entra en un método en una clase que normalmente no se haría, como una clase estándar del JDK.
- **Step Out (Shift + F8)**: Pasa a la primera línea ejecutada después de regresar del método actual.
- **Run to Cursor (Alt + F9)**: Ejecuta hasta la línea donde está el cursor en el archivo.

Recorrer el Código

- **Mostrar punto de ejecución**: Indica la línea actual de ejecución.
- **Step Over**: Avanza una línea en el mismo método.
- **Step Into**: Entra en un método llamado en la línea actual.
- **Force Step Into**: Fuerza la entrada en métodos normalmente ignorados.
- **Step Out**: Sale del método actual y avanza a la siguiente línea después de la llamada al método.
- **Drop Frame**: Retrocede la ejecución al inicio del método actual.
- **Run to Cursor**: Ejecuta hasta la línea donde está el cursor.

Reanudar y Pausar

Comandos

- **Resume**: Reanuda la ejecución del programa.
- **Pause**: Pausa la ejecución del programa.

Puedes acceder a estos comandos desde el menú: `Run > Pause Program…` y `Run > Resume Program….`

Silenciar Todos los Puntos de Interrupción

Puedes silenciar todos los puntos de interrupción desde el menú del depurador, permitiendo que la ejecución continúe sin interrupciones.

5.2. Pruebas de Aplicaciones en Android

Por Qué Vale la Pena Hacer Pruebas

¿Por Qué Deberías Probar tu Aplicación?

Realizar pruebas en tu aplicación te permite:

- **Encontrar y corregir problemas temprano**: Detectar errores en las primeras etapas del desarrollo es menos costoso y requiere menos esfuerzo.
- **Reducir costos**: El costo de corregir errores aumenta exponencialmente con el tiempo, desde la especificación hasta el lanzamiento.
- **Mejorar la calidad del código**: Las pruebas ayudan a asegurar que el código funciona como se espera y facilita el mantenimiento a largo plazo.

Costos de Corregir Errores

El costo de corregir errores varía según el momento en que se descubren:

- **Especificación**: $1
- **Diseño**: $10
- **Código**: $100
- **QA**: $1000
- **Lanzamiento**: $10000

¡Es más barato atrapar errores temprano!

Tipos de Pruebas

Niveles de Pruebas

- **Componentes**: Pruebas de las unidades más pequeñas de código.
- **Integración**: Pruebas de la interacción entre componentes.
- **Protocolo**: Pruebas de la comunicación entre sistemas.
- **Sistema**: Pruebas del sistema completo.

Tipos de Pruebas

- **Instalación**
- **Compatibilidad**
- **Regresión**
- **Aceptación**
- **Rendimiento**
- **Escalabilidad**
- **Usabilidad**
- **Seguridad**
- **Interfaz de usuario e interacción**

Las herramientas de pruebas automáticas de UI se cubren en otro capítulo.

Desarrollo Guiado por Pruebas (TDD)

¿Qué es TDD?

El Desarrollo Guiado por Pruebas (TDD) es una metodología de desarrollo en la que se definen casos de prueba antes de escribir el código funcional. El proceso es el siguiente:

1. **Definir un caso de prueba para un requisito**
2. **Escribir pruebas que verifiquen todas las condiciones del caso de prueba**
3. **Escribir código para pasar las pruebas**
4. **Iterar y refactorizar el código hasta que pase las pruebas**
5. **Repetir hasta que todos los requisitos tengan casos de prueba, todas las pruebas pasen y toda la funcionalidad esté implementada**

PRUEBAS EN TU PROYECTO

Conjuntos de Fuentes en Android Studio

Android Studio crea tres conjuntos de fuentes para tu proyecto:

- **main**: Código y recursos principales.
- **test**: Pruebas unitarias locales.
- **androidTest**: Pruebas instrumentadas.

PRUEBAS UNITARIAS LOCALES

¿Qué son las Pruebas Unitarias?

Las pruebas unitarias son las partes más pequeñas del programa que se pueden probar de forma aislada. El objetivo es demostrar que los componentes individuales son correctos. En Java, esto incluye probar métodos con diferentes entradas y verificar los resultados.

Pruebas Unitarias Locales en JUnit

Las pruebas unitarias locales se compilan y ejecutan completamente en tu máquina local utilizando la Máquina Virtual de Java (JVM). Se utilizan para probar partes de tu aplicación (como la lógica interna) sin necesidad de acceso al marco de Android o a dispositivos/emuladores. Si puedes crear objetos simulados que se comporten como los equivalentes del marco, puedes realizar pruebas unitarias eficientemente.

PRUEBAS UNITARIAS LOCALES EN TU PROYECTO

Organización de las Pruebas

Las pruebas se encuentran en el mismo paquete que la clase de aplicación asociada. Solo se importa `org.junit`, sin clases de Android. La ruta del proyecto para las clases de prueba es: `.../module-name/src/test/java/.`

Importaciones para JUnit

```
// Anotaciones
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;

// Ejecutor básico de pruebas JUnit4
import org.junit.runners.JUnit4;

// Método assertThat
```

```
import static org.junit.Assert.assertThat;
```

Clase de Pruebas

Ejemplo de clase de pruebas:

```
@RunWith(JUnit4.class) // Especifica el ejecutor de pruebas
public class CalculatorTest { // Nombre de lo que estás probando
    // Métodos de prueba y configuraciones
}
```

Ejemplo de Prueba

```
/**
 * Prueba de suma simple.
 * Cada prueba está identificada por una anotación @Test.
 */
@Test
public void addTwoNumbers() {
    double resultAdd = mCalculator.add(1d, 1d);
    assertThat(resultAdd, is(equalTo(2d)));
}
```

Anotación @Test

La anotación `@Test` indica a JUnit que este método es un método de prueba. Proporciona información al ejecutor de pruebas y ya no es necesario prefijar los métodos de prueba con "test".

MÉTODO SETUP()

Configurar el Entorno de Pruebas

```
/**
 * Configurar el entorno para las pruebas
 */
@Before
public void setUp() {
    mCalculator = new Calculator();
}
```

El método `setUp()` inicializa las variables y objetos que se usan en múltiples pruebas.

Método tearDown()

Liberar Recursos Externos

```
/**
```

```
 * Liberar recursos externos
 */
@After
public void tearDown() {
    // Código para liberar recursos
}
```

El método `tearDown()` se utiliza para liberar recursos después de las pruebas.

EJECUTAR PRUEBAS EN ANDROID STUDIO

Iniciar una Ejecución de Prueba

Para iniciar una ejecución de prueba:

1. Haz clic derecho en la clase de prueba y selecciona `Run 'nombre_app' test`.
2. Haz clic derecho en el paquete de prueba y selecciona `Run tests in 'package'`.

Resultados de las Pruebas

- **Pass**: Las pruebas pasan con éxito.
- **Fail**: Las pruebas fallan, mostrando los detalles del fallo.

Pruebas de Resultados de Punto Flotante

Ten cuidado con las pruebas de punto flotante, ya que la aritmética de punto flotante no es precisa en binario.

Solución para las Pruebas de Punto Flotante

Asegúrate de que los valores de punto flotante sean iguales dentro de un margen de error aceptable, como 0.0005.

5.3. La Biblioteca de Soporte de Android

¿QUÉ SON LAS BIBLIOTECAS DE SOPORTE DE ANDROID?

Definición

Las bibliotecas de soporte de Android son un conjunto de más de 25 bibliotecas incluidas en el SDK de Android que proporcionan características adicionales que no están integradas en el

marco de Android. Estas bibliotecas facilitan la compatibilidad hacia atrás y la inclusión de nuevas características en aplicaciones que se ejecutan en versiones anteriores de Android.

Características de las Bibliotecas de Soporte

Funciones

Las bibliotecas de soporte proporcionan una serie de características importantes:

- **Versiones compatibles hacia atrás de componentes**: Permiten utilizar componentes modernos en versiones antiguas de Android.
- **Elementos adicionales de diseño y UI**: Como RecyclerView.
- **Compatibilidad con diferentes factores de forma**: Incluyendo TV y wearables.
- **Componentes de Material Design y otros elementos UI**: Disponibles para versiones antiguas de Android.

Compatibilidad hacia Atrás

Es recomendable utilizar siempre la versión de la biblioteca de soporte de un componente si está disponible. Esto evita la necesidad de crear diferentes versiones de la aplicación para diferentes versiones de Android, ya que el sistema selecciona la mejor versión del componente disponible.

Versiones de las Bibliotecas de Soporte

Las bibliotecas de soporte están disponibles para Android 2.3 (API nivel 9) y versiones superiores. Se recomienda incluir las bibliotecas v4 support y v7 appcompat para aprovechar al máximo las características disponibles.

Bibliotecas Seleccionadas

Bibliotecas de Soporte v4

Estas bibliotecas contienen el conjunto más grande de APIs y cubren una amplia gama de funcionalidades:

- **compat**: Envolturas de compatibilidad.
- **core-utils**: Clases de utilidades, como AsyncTaskLoader.
- **core-ui**: Diversos componentes de UI.

- **media-compat**: Versiones retrocompatibles del marco de medios.
- **fragment**: Componente de UI para manejar fragmentos.

Bibliotecas de Soporte v7

Estas bibliotecas ofrecen compatibilidad hacia atrás y componentes específicos para diferentes usos:

- **appcompat**: Envolturas de compatibilidad.
- **cardview**: Nuevo componente de UI basado en Material Design.
- **gridlayout**: Diseño en celda rectangular (matriz).
- **mediarouter**: Rutas para transmitir audio y video.
- **palette**: Extracción de colores de una imagen.
- **recyclerview**: Vista de desplazamiento eficiente.
- **preference**: Modificación de configuraciones de UI.

Biblioteca v7 appcompat

La biblioteca appcompat incluye ActionBar y acciones de compartir. Se recomienda incluir siempre esta biblioteca y hacer que AppCompatActivity sea la clase padre de tus actividades.

CONFIGURACIÓN Y USO DE LAS BIBLIOTECAS DE SOPORTE

Inclusión de Bibliotecas de Soporte

Las bibliotecas de soporte son parte del SDK de Android y están disponibles a través del SDK Manager. En Android Studio, se almacenan localmente en el Android Support Repository. Para utilizarlas en tu proyecto, debes incluirlas en el archivo `build.gradle` del módulo correspondiente.

Inicio del SDK Manager en Android Studio

Para iniciar el SDK Manager:

1. Ve a `Tools > Android > SDK Manager`.
2. En la pestaña `SDK Tools`, encuentra `Android Support Repository`.
3. Si no está instalado o hay una actualización disponible, marca la casilla y haz clic en `Apply`.
4. Confirma los componentes en el cuadro de diálogo y haz clic en `OK` para instalar.
5. Cuando finalice, haz clic en `Finish` y verifica que ahora esté instalado.

Encontrar el Identificador de Dependencia

Para encontrar el identificador de dependencia:

1. Abre la página de características de la Biblioteca de Soporte.
2. Encuentra la característica que deseas, por ejemplo, la biblioteca recomendada v7 appcompat.
3. Copia el identificador de dependencia del script de construcción para la biblioteca:

```
com.android.support:appcompat-v7:27.1.1
```

Añadir la Dependencia en build.gradle

1. Abre `build.gradle (Module: app)`.
2. En la sección de dependencias, añade la dependencia para la biblioteca de soporte si no está ya incluida en la plantilla:

```
dependencies {
    implementation 'com.android.support:appcompat-v7:27.1.1'
}
```

3. Actualiza el número de versión si se te solicita.
4. Sincroniza el proyecto cuando se te solicite haciendo clic en `Sync Now`.

Ejemplo de build.gradle Actualizado

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-
layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation
'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

6. Interacción del usuario

6.1 Botones e Imágenes Clicables en Android

Interacción del Usuario

Expectativas del Usuario

Los usuarios esperan interactuar con las aplicaciones de diversas formas:

- **Toques o clics**: Realizar acciones tocando botones o elementos de la interfaz.
- **Escritura**: Ingresar datos mediante el teclado.
- **Gestos**: Utilizar movimientos como deslizamientos y pellizcos.
- **Voz**: Comandos por voz para realizar acciones.

Diseño de Interacción del Usuario

Es crucial que el diseño de la interacción del usuario sea obvio, fácil y consistente. Considera los siguientes principios:

- **Simplicidad**: Minimiza el número de pasos necesarios para realizar una acción.
- **Accesibilidad**: Utiliza elementos de la interfaz de usuario que sean fáciles de acceder, entender y usar.
- **Buenas prácticas de Android**: Sigue las mejores prácticas de diseño de Android para cumplir con las expectativas de los usuarios.

Botones

¿Qué es un Botón?

Un `Button` es una vista que responde a toques o clics. Generalmente, el texto o los elementos visuales indican la acción que se realizará al ser tocado. Los botones pueden tener diferentes estados como:

- **Normal**
- **Enfocado**
- **Deshabilitado**
- **Presionado**
- **Activado/Desactivado**

Activos de Imagen para Botones

Para añadir imágenes a tus botones:

1. Haz clic derecho en `app/res/drawable`.
2. Selecciona `New > Image Asset`.
3. Escoge `Action Bar and Tab Items` del menú desplegable.
4. Haz clic en la imagen de `Clipart` (el logotipo de Android) para seleccionar una imagen.

También puedes experimentar seleccionando `New > Vector Asset` para añadir gráficos vectoriales.

Responder a Toques en Botones

En tu Código

Utiliza el `OnClickListener` para manejar eventos de clic en tu código:

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Haz algo en respuesta al clic del botón
    }
});
```

En XML

Usa el atributo `android:onClick` en el diseño XML para definir el método que se llamará al hacer clic:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

En este caso, cuando el usuario haga clic en el botón, el sistema Android llamará al método `sendMessage()` de la actividad.

IMÁGENES CLICABLES

¿Qué es una ImageView?

Una `ImageView` es un componente de UI que muestra una imagen. También puede configurarse para responder a toques, convirtiéndola en una imagen clicable.

Responder a Toques en ImageView

En tu Código

Utiliza el `OnClickListener` para manejar eventos de clic en tu código:

```
ImageView imageView = findViewById(R.id.imageView);
imageView.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Haz algo en respuesta al clic de la imagen
    }
});
```

En XML

Usa el atributo `android:onClick` en el diseño XML para definir el método que se llamará al hacer clic:

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/donut_circle"
    android:onClick="orderDonut" />
```

Al igual que con los botones, el sistema Android llamará al método `orderDonut()` de la actividad cuando se haga clic en la imagen.

BOTÓN DE ACCIÓN FLOTANTE (FAB)

¿Qué es un FAB?

El `Floating Action Button` (FAB) es un botón circular elevado que flota sobre el diseño. Representa la acción principal o "promovida" en una pantalla y se limita a uno por pantalla.

Usos Comunes del FAB

Ejemplo: Botón de "Agregar contacto" en una aplicación de contactos.

Implementación del FAB

En el Diseño XML

Inicia con la plantilla de `Basic Activity` y añade el FAB en el diseño:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:src="@drawable/ic_fab_chat_button_white"
```

```
... />
```

Tamaño del FAB

- **Tamaño por defecto**: 56 x 56 dp
- **Tamaño mini**: 30 x 40 dp (establecido con `app:fabSize="mini"`)
- **Tamaño normal**: 56 x 56 dp (establecido con `app:fabSize="normal"`)

GESTOS COMUNES

Tipos de Gestos

Los gestos táctiles incluyen:

- **Toque prolongado**
- **Doble toque**
- **Fling**
- **Arrastrar**
- **Desplazar**
- **Pellizcar**

Es importante no depender únicamente de gestos para el comportamiento básico de la aplicación.

Detección de Gestos

Para manejar gestos, puedes usar clases y métodos como:

- **GestureDetectorCompat**: Para gestos comunes.
- **MotionEvent**: Para eventos de movimiento.

Detección de Gestos

Para detectar todos los tipos de gestos, realiza los siguientes pasos esenciales:

1. **Recopilar datos sobre eventos táctiles**: Captura la interacción inicial del usuario y sigue la posición de los dedos.
2. **Interpretar los datos**: Determina si los datos cumplen con los criterios para los gestos soportados por tu aplicación.

Durante toda la interacción, un objeto de la clase `MotionEvent` es entregado a `onTouchEvent()`, proporcionando detalles de cada interacción. Tu aplicación puede usar los datos proporcionados por `MotionEvent` para determinar si un gesto ocurrió.

6.2 Controles de Entrada en Android

Visión General de los Controles de Entrada

Aceptación de Entrada del Usuario

Los controles de entrada son componentes esenciales en cualquier aplicación, ya que permiten a los usuarios interactuar y proporcionar datos. Los principales controles de entrada incluyen:

- **Texto y números libres**: Usando EditText con el teclado.
- **Provisión de opciones**: Con CheckBox, RadioButton, y Spinner.
- **Alternar encendido/apagado**: Con Toggle Switch.
- **Elección de valor dentro de un rango**: Con SeekBar.

Ejemplos de Controles de Entrada

- **EditText**: Para ingresar texto usando el teclado.
- **SeekBar**: Para deslizar hacia la izquierda o derecha hasta una configuración.
- **CheckBox**: Para elegir más de una opción.
- **RadioButton**: Para hacer una sola elección dentro de un grupo.
- **Switch**: Para alternar entre encendido y apagado.
- **Spinner**: Para elegir un solo elemento de una lista desplegable.

Foco de la Vista

¿Qué es el Foco?

El foco determina cuál vista recibirá la entrada del usuario. Solo una vista puede tener el foco en un momento dado, y esto ayuda a dejar claro cuál vista está lista para aceptar la entrada.

Asignación del Foco

El foco se puede asignar de varias maneras:

- **Toque del usuario en una vista**: La vista bajo el toque del usuario obtiene el foco.
- **Guía de la aplicación**: La aplicación puede guiar al usuario de un control de texto a otro utilizando la tecla de retorno, tabulador o las teclas de flecha.
- **Método requestFocus()**: Llama a `requestFocus()` en cualquier vista que sea "focusable" para asignarle el foco.

Vista Clicable versus Vista Enfocable

- **Clicable**: Una vista que puede responder a clics o toques.

- **Enfocable**: Una vista que puede obtener el foco para aceptar entrada.

Los controles de entrada como los teclados envían la entrada a la vista que tiene el foco.

Determinación del Foco

La vista que obtiene el foco es generalmente la más cercana al toque del usuario, con prioridad de izquierda a derecha y de arriba hacia abajo. El foco puede cambiar cuando el usuario interactúa con un control direccional.

Guía del Foco

Para una navegación fluida y predecible, es importante indicar visualmente cuál vista tiene el foco y cuáles pueden tenerlo. La disposición lógica de los controles de entrada en el diseño, de izquierda a derecha y de arriba hacia abajo, facilita la navegación del usuario.

Métodos para Manejar el Foco

- **setFocusable()**: Establece si una vista puede tener el foco.
- **requestFocus()**: Da el foco a una vista específica.
- **setOnFocusChangeListener()**: Establece un oyente para cuando una vista gana o pierde el foco.
- **onFocusChanged()**: Llamado cuando el foco en una vista cambia.

Encontrar la Vista con Foco

- **Activity.getCurrentFocus()**: Obtiene la vista actualmente enfocada en una actividad.
- **ViewGroup.getFocusedChild()**: Obtiene el hijo enfocado en un grupo de vistas.

TEXTO LIBRE Y NÚMEROS

Uso de EditText

`EditText` es el control de entrada principal para texto y números. Puede configurarse para aceptar varios tipos de entrada y personalizarse con diferentes atributos de entrada (`inputType`).

Configuración del Tipo de Entrada

El tipo de entrada se configura en el panel de atributos del editor de diseño o directamente en XML:

```
<EditText
    android:id="@+id/name_field"
    android:inputType="textPersonName"
    ... />
```

Ejemplos de Configuración de EditText

- **Texto de mensaje**:

```
<EditText
    android:id="@+id/message"
    android:inputType="textShortMessage"
    ... />
```

- **Número de teléfono**:

```
<EditText
    android:id="@+id/phone_number"
    android:inputType="phone"
    ... />
```

Obtener Texto de EditText

Para obtener el texto ingresado en un `EditText`:

```
EditText simpleEditText = findViewById(R.id.edit_simple);
String strValue = simpleEditText.getText().toString();
```

Tipos Comunes de Entrada

- **textCapCharacters**: Todo en mayúsculas.
- **textCapSentences**: Cada oración comienza con mayúscula.
- **textPassword**: Contraseña oculta.
- **number**: Solo números.
- **textEmailAddress**: Teclado con `@` convenientemente ubicado.
- **phone**: Teclado numérico de teléfono.
- **datetime**: Teclado numérico con barra y dos puntos para fecha y hora.

PROVISIÓN DE OPCIONES

Elementos de UI para Proveer Opciones

- **CheckBox**: Permite al usuario seleccionar cualquier número de opciones. Cada CheckBox es una vista independiente que puede tener un controlador de clics.
- **RadioButton**: Deben agruparse en un RadioGroup para que el usuario pueda seleccionar solo una opción. Comúnmente se usan junto con un botón de envío para el grupo.
- **ToggleButton y Switch**: Permiten alternar entre encendido y apagado.
- **Spinner**: Permite al usuario elegir un solo elemento de una lista desplegable.

Ejemplo de CheckBox

```
<CheckBox
    android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option" />
```

Ejemplo de RadioButton en un RadioGroup

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Choice 1" />
    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Choice 2" />
</RadioGroup>
```

Ejemplo de ToggleButton

```
<ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="ON"
    android:textOff="OFF" />
```

Ejemplo de Spinner

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

6.3 Menús y Selectores en Android

Visión General de Menús y Selectores

Tipos de Menús

Android ofrece diferentes tipos de menús que puedes utilizar en tu aplicación para proporcionar opciones y acciones adicionales a los usuarios:

- **Barra de aplicaciones con menú de opciones**
- **Menú contextual**
- **Menú emergente**

Además, hay diálogos y selectores que permiten al usuario seleccionar fechas, horas y proporcionar alertas.

Barra de Aplicaciones con Menú de Opciones

¿Qué es la Barra de Aplicaciones?

La barra de aplicaciones es una barra ubicada en la parte superior de cada pantalla, común en todos los dispositivos Android. Incluye:

- Icono de navegación para abrir el cajón de navegación.
- Título de la actividad actual.
- Iconos para elementos del menú de opciones.
- Botón de desbordamiento de acciones para ver más opciones.

¿Qué es el Menú de Opciones?

El menú de opciones proporciona accesos directos a configuraciones y acciones importantes. Aparece en la esquina derecha de la barra de aplicaciones y se puede acceder tocando el botón de desbordamiento de acciones (tres puntos).

Pasos para Implementar el Menú de Opciones

1. **Crear un recurso de menú XML** (`menu_main.xml`):

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/option_settings"
          android:title="Settings" />
    <item android:id="@+id/option_favorites"
```

```
            android:title="Favorites" />
</menu>
```

2. **Sobrescribir `onCreateOptionsMenu()` en la actividad para inflar el menú**:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

3. **Manejar clics en los elementos del menú**:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.option_settings:
            showSettings();
            return true;
        case R.id.option_favorites:
            showFavorites();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Añadir Iconos a los Elementos del Menú

1. Haz clic derecho en `drawable`.
2. Selecciona `New > Image Asset`.
3. Elige `Action Bar and Tab Icons`.
4. Edita el nombre del icono y selecciona la imagen de clipart.
5. Haz clic en `Next` y luego en `Finish`.

Ejemplo de item con icono:

```
<item
    android:id="@+id/action_favorites"
    android:icon="@drawable/ic_favorite"
    android:orderInCategory="30"
    android:title="@string/action_favorites"
    app:showAsAction="ifRoom" />
```

MENÚS CONTEXTUALES

¿Qué son los Menús Contextuales?

Los menús contextuales permiten a los usuarios realizar acciones en una vista seleccionada. Pueden ser desplegados en cualquier vista y son comúnmente usados en elementos de `RecyclerView`, `GridView` u otras colecciones de vistas.

Tipos de Menús Contextuales

Menú contextual flotante: Se activa con una pulsación larga en una vista.

Barra de acción contextual: Aparece temporalmente en lugar de la barra de aplicaciones, permitiendo acciones en múltiples elementos seleccionados.

Pasos para Implementar un Menú Contextual Flotante

1. **Crear un recurso de menú XML** (`menu_context.xml`):

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/context_edit"
          android:title="Edit"
          android:orderInCategory="10" />
    <item android:id="@+id/context_share"
          android:title="Share"
          android:orderInCategory="20" />
</menu>
```

2. **Registrar la vista para el menú contextual** en `onCreate()` de la actividad:

```
TextView articleText = findViewById(R.id.article);
registerForContextMenu(articleText);
```

3. **Implementar `onCreateContextMenu()` para inflar el menú**:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_context, menu);
}
```

4. **Implementar `onContextItemSelected()` para manejar los clics en el menú**:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
```

```
        switch (item.getItemId()) {
            case R.id.context_edit:
                editNote();
                return true;
            case R.id.context_share:
                shareNote();
                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }
```

MENÚS EMERGENTES

¿Qué es un Menú Emergente?

Un menú emergente es una lista vertical de elementos anclada a una vista. Se utiliza típicamente para acciones que no afectan directamente el contenido de la vista, como opciones de respuesta o reenvío en una aplicación de correo.

Pasos para Implementar un Menú Emergente

1. **Crear un recurso de menú XML** (`menu_popup.xml`):

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/option_forward"
          android:title="Forward" />
</menu>
```

2. **Añadir un `ImageButton` para el icono del menú emergente en el diseño de la actividad**:

```
<ImageButton
    android:id="@+id/button_popup"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_action_popup" />
```

3. **Asignar `onClickListener` al botón en `onCreate()`**:

```
ImageButton button = findViewById(R.id.button_popup);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showPopup(v);
    }
});
```

4. **Implementar el método `showPopup()` para inflar y mostrar el menú emergente**:

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.getMenuInflater().inflate(R.menu.menu_popup,
popup.getMenu());
    popup.setOnMenuItemClickListener(new
PopupMenu.OnMenuItemClickListener() {
        @Override
        public boolean onMenuItemClick(MenuItem item) {
            switch (item.getItemId()) {
                case R.id.option_forward:
                    // Implementa la acción para el botón de
reenvío
                    return true;
                default:
                    return false;
            }
        }
    });
    popup.show();
}
```

DIÁLOGOS Y SELECTORES

Diálogos

Los diálogos aparecen encima de la actividad actual, interrumpiendo el flujo normal y requiriendo una acción del usuario para ser descartados. Un tipo común de diálogo es el `AlertDialog`.

AlertDialog

Un `AlertDialog` puede mostrar:

- **Título** (opcional)
- **Área de contenido**: Mensaje, lista o diseño personalizado
- **Botones de acción**: Hasta tres botones (positivo, negativo y neutral)

Ejemplo de creación de un `AlertDialog`:

```
public void showAlert(View view) {
    AlertDialog.Builder alertDialog = new AlertDialog.Builder(this);
    alertDialog.setTitle("Connect to Provider");
    alertDialog.setMessage(R.string.alert_message);
    alertDialog.setPositiveButton("OK", new
DialogInterface.OnClickListener() {
        @Override
```

```
        public void onClick(DialogInterface dialog, int which) {
            // Acción para el botón OK
        }
    });
    alertDialog.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Acción para el botón Cancel
        }
    });
    alertDialog.show();
}
```

Selectores

Los selectores permiten a los usuarios elegir una fecha u hora.

DatePickerDialog

Para crear un `DatePickerDialog`, añade un fragmento que extienda `DialogFragment` e implemente `DatePickerDialog.OnDateSetListener`.

TimePickerDialog

Para crear un `TimePickerDialog`, añade un fragmento que extienda `DialogFragment` e implemente `TimePickerDialog.OnTimeSetListener`.

6.4 Navegación del Usuario en Android

NAVEGACIÓN HACIA ATRÁS

¿Qué es la Navegación hacia Atrás?

La navegación hacia atrás permite a los usuarios regresar a las pantallas anteriores en orden inverso al que las visitaron. Es provista por el botón de retroceso del dispositivo y es controlada por la pila de retroceso del sistema Android.

Gestión del Comportamiento del Botón de Retroceso

El sistema Android maneja la pila de retroceso y el comportamiento del botón de retroceso. En general, no se recomienda cambiar este comportamiento, excepto en casos específicos donde se deba satisfacer la expectativa del usuario, como en un navegador embebido.

Sobrescribir onBackPressed()

Para personalizar el comportamiento del botón de retroceso, puedes sobrescribir el método `onBackPressed()` en tu actividad:

```
java
Copiar código
@Override
public void onBackPressed() {
    // Añade el manejador de la tecla de retroceso aquí.
    return;
}
```

NAVEGACIÓN JERÁRQUICA

¿Qué es la Navegación Jerárquica?

La navegación jerárquica permite a los usuarios navegar dentro de una jerarquía de pantallas organizadas en niveles. La pantalla principal o de inicio actúa como un punto de partida, y los usuarios pueden descender a pantallas secundarias o volver a la pantalla principal.

Patrones de Navegación Jerárquica

- **Pantalla principal**: Habilita la navegación hacia pantallas secundarias.
- **Colección de hermanos**: Pantallas que permiten la navegación a una colección de pantallas secundarias.
- **Sección de hermanos**: Pantallas con contenido, como historias o detalles.

Tipos de Navegación Jerárquica

1. **Navegación descendente**: Desde una pantalla principal hacia una pantalla secundaria.
2. **Navegación ancestral**: Desde una pantalla secundaria o hermana hacia la pantalla principal.
3. **Navegación lateral**: Entre pantallas hermanas.

Ejemplo de Navegación Jerárquica

Un ejemplo de jerarquía en una aplicación de noticias:

- **Pantalla principal**: Lista de categorías de noticias (Titulares, Tecnología, Cocina).
- **Pantalla de colección**: Lista de titulares dentro de una categoría.
- **Pantalla de sección**: Detalles de una noticia específica.

NAVEGACIÓN DESCENDENTE

¿Qué es la Navegación Descendente?

La navegación descendente permite a los usuarios navegar desde una pantalla principal hacia una pantalla secundaria. Este tipo de navegación es común en aplicaciones que presentan una jerarquía clara de contenido.

Ejemplo de Navegación Descendente

En una aplicación de noticias, navegar desde la pantalla principal de categorías de noticias hasta una lista de titulares, y luego a los detalles de una noticia específica.

Controles para la Navegación Descendente

- **Navigation Drawer**: Un menú deslizante desde el borde de la pantalla.
- **Botones e imágenes clicables**: En la pantalla principal.
- **List items**: En pantallas de colección.

Implementación del Navigation Drawer

Layouts para el Navigation Drawer

1. **Crear el `DrawerLayout` como el `ViewGroup` raíz** de la actividad.
2. **Añadir un `NavigationView`** para el contenido del drawer.
3. **Incluir un `AppBarLayout`** que contenga un botón de icono de navegación.
4. **Añadir un layout de contenido para la actividad** que muestre el navigation drawer.
5. **Crear un layout para el encabezado del navigation drawer**.

Pasos para Implementar el Navigation Drawer

1. **Poblar el menú del navigation drawer** con títulos de elementos e íconos.
2. **Configurar el navigation drawer y los listeners de los elementos** en el código de la actividad.
3. **Manejar las selecciones de los elementos del menú**.

NAVEGACIÓN ANCESTRAL

¿Qué es la Navegación Ancestral?

La navegación ancestral permite a los usuarios regresar de una pantalla secundaria o de detalle a la pantalla principal o de inicio. Es provista por el botón de "Up" en la barra de aplicaciones.

Declarar la Actividad Padre en AndroidManifest

Para implementar la navegación ancestral, debes declarar la actividad padre en el archivo `AndroidManifest.xml`:

```
<activity android:name=".OrderActivity"
  android:label="@string/title_activity_order"
  android:parentActivityName="com.example.android.MainActivity">
  <meta-data
     android:name="android.support.PARENT_ACTIVITY"
     android:value=".MainActivity"/>
</activity>
```

NAVEGACIÓN LATERAL

¿Qué es la Navegación Lateral?

La navegación lateral permite a los usuarios moverse entre pantallas hermanas, como pestañas dentro de una misma sección. Es común en aplicaciones con múltiples secciones que el usuario puede cambiar rápidamente sin regresar a la pantalla principal.

Beneficios de Usar Pestañas y Deslizamientos

- **Acceso rápido a contenido relacionado** sin necesidad de navegación adicional.
- **Interfaz persistente** que permite a los usuarios cambiar entre secciones relacionadas sin regresar a la pantalla principal.

Mejores Prácticas con Pestañas

- **Disposición horizontal**: Colocar las pestañas a lo largo de la parte superior de la pantalla.
- **Persistencia**: Mantener las pestañas visibles en todas las pantallas relacionadas.
- **Sin historial**: El cambio entre pestañas no debe tratarse como navegación en el historial.

Pasos para Implementar Pestañas

1. **Definir el layout de pestañas usando `TabLayout`.**
2. **Implementar un `Fragment` y su layout para cada pestaña.**
3. **Implementar un `PagerAdapter`** desde `FragmentPagerAdapter` o `FragmentStatePagerAdapter`.
4. **Crear una instancia del layout de pestañas.**
5. **Utilizar `PagerAdapter` para gestionar las pantallas** (cada pantalla es un `Fragment`).
6. **Establecer un listener para determinar qué pestaña fue seleccionada.**

Ejemplo de Implementación de Pestañas

1. **Añadir `TabLayout` debajo de la `Toolbar`:**

```
<android.support.design.widget.TabLayout
    android:id="@+id/tab_layout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/toolbar"
    android:background="?attr/colorPrimary"
    android:minHeight="?attr/actionBarSize"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>
```

2. **Añadir `ViewPager` debajo de `TabLayout`:**

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:layout_below="@id/tab_layout"/>
```

3. **Crear la configuración de `TabLayout` y `ViewPager` en `onCreate()`:**

```
TabLayout tabLayout = findViewById(R.id.tab_layout);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);

final ViewPager viewPager = findViewById(R.id.pager);
final PagerAdapter adapter = new
PagerAdapter(getSupportFragmentManager(),
tabLayout.getTabCount());
viewPager.setAdapter(adapter);

viewPager.addOnPageChangeListener(new
TabLayout.TabLayoutOnPageChangeListener(tabLayout));
```

```
tabLayout.addOnTabSelectedListener(new
TabLayout.OnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        viewPager.setCurrentItem(tab.getPosition());
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {}

    @Override
    public void onTabReselected(TabLayout.Tab tab) {}
});
```

6.5 RecyclerView en Android

¿Qué es un RecyclerView?

Definición

`RecyclerView` es un contenedor desplazable para grandes conjuntos de datos. Es eficiente y reutiliza un número limitado de elementos de vista, actualizando rápidamente los datos cambiantes.

Ventajas

- **Eficiencia**: Reutiliza elementos de vista que ya no son visibles para el usuario.
- **Actualización rápida**: Maneja cambios en los datos de forma eficiente.

Componentes de RecyclerView

Descripción General

Los componentes principales de RecyclerView incluyen:

- **Datos**: El conjunto de datos que se mostrará en la lista.
- **RecyclerView**: El contenedor desplazable para los elementos de la lista.
- **Layout para un ítem de datos**: Archivo XML que define el diseño de un ítem de datos.
- **Layout Manager**: Gestiona la organización de los componentes de la UI en una vista (`RecyclerView.LayoutManager`).
- **Adapter**: Conecta los datos con `RecyclerView` (`RecyclerView.Adapter`).
- **ViewHolder**: Contiene la información de la vista para mostrar un ítem (`RecyclerView.ViewHolder`).

IMPLEMENTACIÓN DE RECYCLERVIEW

Pasos para Implementar un RecyclerView

1. **Añadir la dependencia de RecyclerView en `build.gradle`:**

```
dependencies {
    ...
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    ...
}
```

2. **Añadir RecyclerView al diseño XML de la actividad:**

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v7.widget.RecyclerView>
```

3. **Crear el diseño XML para un ítem de lista:**

```
<LinearLayout ...>
    <TextView
        android:id="@+id/word"
        style="@style/word_title" />
</LinearLayout>
```

4. **Implementar el Adapter:**

```
public class WordListAdapter extends
RecyclerView.Adapter<WordListAdapter.WordViewHolder> {
    private final LinkedList<String> mWordList;
    private LayoutInflater mInflater;

    public WordListAdapter(Context context, LinkedList<String>
wordList) {
        mInflater = LayoutInflater.from(context);
        this.mWordList = wordList;
    }

    @Override
    public WordViewHolder onCreateViewHolder(ViewGroup parent,
int viewType) {
        View mItemView =
mInflater.inflate(R.layout.wordlist_item, parent, false);
        return new WordViewHolder(mItemView, this);
    }

    @Override
```

```
    public void onBindViewHolder(WordViewHolder holder, int
position) {
        String mCurrent = mWordList.get(position);
        holder.wordItemView.setText(mCurrent);
    }

    @Override
    public int getItemCount() {
        return mWordList.size();
    }

    class WordViewHolder extends RecyclerView.ViewHolder {
        public final TextView wordItemView;
        final WordListAdapter mAdapter;

        public WordViewHolder(View itemView, WordListAdapter
adapter) {
            super(itemView);
            wordItemView = itemView.findViewById(R.id.word);
            this.mAdapter = adapter;
        }
    }
}
```

5. **Crear el RecyclerView en `onCreate()` de la actividad**:

```
RecyclerView mRecyclerView = findViewById(R.id.recyclerview);
WordListAdapter mAdapter = new WordListAdapter(this, mWordList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

LAYOUT MANAGER

¿Qué es un Layout Manager?

Cada `ViewGroup` tiene un layout manager que se utiliza para posicionar elementos de vista dentro de un `RecyclerView`. Reutiliza elementos de vista que ya no son visibles para el usuario.

Tipos de Layout Managers

- **LinearLayoutManager**: Dispone los elementos en una lista vertical u horizontal.
- **GridLayoutManager**: Dispone los elementos en una cuadrícula.
- **StaggeredGridLayoutManager**: Dispone los elementos en una cuadrícula de forma escalonada.

ADAPTER

¿Qué es un Adapter?

El `Adapter` actúa como intermediario entre los datos y las vistas. Gestiona la creación, actualización, adición y eliminación de elementos de vista a medida que cambian los datos subyacentes.

Métodos Requeridos del Adapter

- **onCreateViewHolder()**: Crea nuevas vistas.
- **onBindViewHolder()**: Vincula los datos a una vista.
- **getItemCount()**: Devuelve el número de elementos en el conjunto de datos.

Ejemplo de Adapter

```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
    View mItemView = mInflater.inflate(R.layout.wordlist_item, parent,
false);
    return new WordViewHolder(mItemView, this);
}

@Override
public void onBindViewHolder(WordViewHolder holder, int position) {
    String mCurrent = mWordList.get(position);
    holder.wordItemView.setText(mCurrent);
}

@Override
public int getItemCount() {
    return mWordList.size();
}
```

VIEWHOLDER

¿Qué es un ViewHolder?

El `ViewHolder` contiene la información de la vista para mostrar un ítem. Es utilizado por el `Adapter` para preparar una vista con datos para un elemento de lista.

Ejemplo de ViewHolder

```
class WordViewHolder extends RecyclerView.ViewHolder {
    public final TextView wordItemView;
    final WordListAdapter mAdapter;

    public WordViewHolder(View itemView, WordListAdapter adapter) {
```

```
            super(itemView);
            wordItemView = itemView.findViewById(R.id.word);
            this.mAdapter = adapter;
            itemView.setOnClickListener(this);
        }
}
```

APRENDE MÁS

Para obtener más información sobre la implementación y uso de `RecyclerView` en Android, consulta los siguientes recursos:

- **RecyclerView Class**
- **RecyclerView.Adapter Class**
- **RecyclerView.ViewHolder Class**
- **RecyclerView.LayoutManager Class**

7. Experiencia del usuario atractiva

7.1 Dibujos, Estilos y Temas en Android

DIBUJOS (DRAWABLES)

¿Qué es un Drawable?

Un `Drawable` es una clase genérica de Android utilizada para representar cualquier tipo de gráfico. Todos los `Drawables` se almacenan en la carpeta `res/drawable` del proyecto.

Tipos de Clases Drawable

Existen varios tipos de `Drawables` en Android, incluyendo:

- **Bitmap File**
- **Nine-Patch File**
- **Layer List Drawable**
- **Shape Drawable**
- **State List Drawable**
- **Level List Drawable**
- **Transition Drawable**
- **Vector Drawable**
- **Custom Drawables**

Formatos de Imagen Soportados

Los Drawables pueden ser imágenes en formatos como:

- PNG (.png)
- JPG (.jpg)
- GIF (.gif)
- BMP sin comprimir (.bmp)
- WebP (4.0 y superior)

Estas imágenes crean un tipo de datos `BitmapDrawable` y se colocan directamente en `res/drawable`.

Referenciar drawables

Puedes referenciar `Drawables` en XML:

```
<ImageView
    android:layout_height="wrap_content"
```

```
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

O en código Java:

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

NINE-PATCH FILES

¿Qué son los Nine-Patch Files?

Los archivos de Nine-Patch (.9.png) son archivos PNG con regiones estirables, que solo se expanden, no se reducen. Estos son útiles para los fondos de elementos de la UI que deben cambiar de tamaño, como los botones.

Crear Archivos Nine-Patch

Para crear un archivo Nine-Patch:

1. Coloca un archivo PNG pequeño en `res/drawable`.
2. Haz clic derecho y selecciona `Create 9-Patch file`.
3. Abre el editor de 9-Patch para especificar las regiones estirables.

Editar Archivos Nine-Patch

Utiliza el editor de 9-Patch para marcar las regiones estirables para el ancho y la altura. Puedes hacer clic para marcar los píxeles y convertirlos en negros, o usar `Shift-clic` (ctrl-clic en Mac) para desmarcarlos.

LISTAS DE CAPAS (LAYER LIST)

¿Qué es una Layer List?

Puedes crear imágenes en capas de manera similar a las herramientas de dibujo como GIMP. En Android, cada capa está representada por un `Drawable` y se organizan y gestionan en un archivo XML.

Crear una Layer List

Ejemplo de una Layer List en XML:

```
<layer-list>
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
```

```
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>
```

DIBUJOS DE FORMA Y GRADIENTDRAWABLE

¿Qué es un Shape Drawable?

Un `Shape Drawable` permite definir una forma y sus propiedades en XML, como rectángulo, óvalo, anillo o línea. Se puede estilizar con atributos como `<corners>`, `<gradient>`, `<padding>`, `<size>`, `<solid>` y `<stroke>`.

Crear un GradientDrawable

Ejemplo de un GradientDrawable:

```
<shape android:shape="rectangle">
    <gradient
        android:startColor="@color/white"
        android:endColor="@color/blue"
        android:angle="45"/>
    <corners android:radius="8dp" />
</shape>
```

En Java:

```
Resources res = getResources();
Drawable shape = res.getDrawable(R.drawable.gradient_box);
TextView tv = findViewById(R.id.textview);
tv.setBackground(shape);
```

DIBUJOS DE TRANSICIÓN

¿Qué es un Transition Drawable?

Un `Transition Drawable` es un `Drawable` que puede realizar una transición suave entre dos otros `Drawables`. Está representado por `TransitionDrawable` en código Java.

Crear Transition Drawables

Ejemplo en XML:

```
<transition>
```

```
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

En Java:

```
ImageButton button = findViewById(R.id.button);
TransitionDrawable drawable = (TransitionDrawable)
button.getDrawable();
drawable.startTransition(500);
```

DIBUJOS VECTORIALES

¿Qué es un Vector Drawable?

Los `Vector Drawables` se escalan suavemente para todos los tamaños de pantalla y son compatibles con Android API Nivel 21 y superiores. Utiliza Vector Asset Studio para crear estos `Drawables`.

Crear Vector Drawables

Ejemplo de un Vector Drawable:

```
<vector
    android:height="256dp"
    android:width="256dp"
    android:viewportWidth="32"
    android:viewportHeight="32">
    <path android:fillColor="@color/red"
          android:pathData="M20.59.5c-1.9550-3.831.268-4.53c-0.67-
1.732-2.547-3-4.5-3..." />
</vector>
```

ESTILOS

¿Qué es un Estilo?

Un estilo es una colección de atributos que definen la apariencia visual de una vista. Ayuda a reducir la duplicación, hacer el código más compacto y gestionar la apariencia visual de muchos componentes con un solo estilo.

Definir Estilos en styles.xml

Ejemplo de definir un estilo en styles.xml:

```
<resources>
    <style name="CodeFont">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
```

```
</resources>
```

Uso de Estilos para Reducir Desorden

Ejemplo de uso de estilos:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

Herencia de Estilos

Ejemplo de herencia de estilos:

```
<resources>
    <style name="RedCode" parent="@style/CodeFont">
        <item name="android:textColor">#FF0000</item>
    </style>
</resources>
```

TEMAS

¿Qué es un Tema?

Un tema es un estilo aplicado a una actividad completa o incluso a toda la aplicación. Los temas se aplican en el archivo AndroidManifest.xml.

Personalizar el Tema de la Aplicación

Ejemplo de personalización de un tema en AndroidManifest.xml:

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

Ejemplo de definición de un tema:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

Recursos de Estilos y Temas

Recursos Disponibles

Android ofrece una colección de estilos y temas integrados:

- **Android Styles**
- **Android Themes**
- **Styles and Themes Guide**
- **DayNight Theme Guide**

7.2 Material Design en Android

El Metáfora del Material (Material Metaphor)

¿Qué es Material Design?

Material Design es un conjunto de pautas de diseño que proporciona un lenguaje visual coherente para los productos digitales. Combina principios clásicos del buen diseño con la innovación y las posibilidades de la tecnología y la ciencia.

Metáfora del Material

Material Design se basa en una metáfora de materiales tridimensionales que contienen luz, material y sombras. Las superficies y bordes proporcionan pistas visuales que están arraigadas en la realidad. Los fundamentos de la luz, la superficie y el movimiento transmiten cómo los objetos se mueven, interactúan y existen en el espacio y en relación entre sí.

Implementación en tu Aplicación

Los elementos en tu aplicación Android deben comportarse de manera similar a los materiales del mundo real, arrojando sombras, ocupando espacio e interactuando entre sí.

Imágenes

Uso de Imágenes

Las imágenes en Material Design deben ser relevantes, informativas y agradables. Las mejores prácticas para el uso de imágenes incluyen:

- **Uso junto con texto**: Las imágenes deben complementar y mejorar el contenido textual.

- **Imágenes originales**: Utiliza imágenes originales para proporcionar un punto de enfoque y construir una narrativa.
- **Comunicación y diferenciación**: Las imágenes ayudan a comunicar y diferenciar tu aplicación.

TIPOGRAFÍA

Tipografía Roboto

Roboto es la tipografía estándar en Android y tiene seis pesos:

- Thin
- Light
- Regular
- Medium
- Bold
- Black

Estilos y Escala de Fuente

Utilizar demasiados tamaños de fuente puede ser confuso y desordenado. Es mejor utilizar un conjunto limitado de tamaños que funcionen bien juntos. Los estilos de texto en Material Design incluyen:

- Display 4, Display 3, Display 2, Display 1
- Headline, Title, Subheading
- Body 2, Body 1, Caption, Button

Configuración de la Apariencia del Texto

Puedes configurar la apariencia del texto en XML:

```
android:textAppearance="@style/TextAppearance.AppCompat.Display3"
```

Fuentes como Recursos

Puedes agrupar fuentes como recursos en el paquete de tu aplicación (APK) y crear una carpeta de fuentes dentro de `res`. Para acceder a un recurso de fuente:

```
@font/myfont
R.font.myfont
```

Fuentes Descargables

Descargar fuentes de un proveedor de aplicaciones reduce el tamaño del APK, aumenta la tasa de éxito de la instalación de la aplicación y mejora la salud general del sistema, ahorrando datos móviles, memoria del teléfono y espacio en disco.

COLOR

Uso del Color

Material Design recomienda usar un color primario junto con algunas tonalidades y un color de acento para crear una experiencia de usuario audaz. La paleta de colores de Material Design incluye tonos atrevidos, entornos atenuados, sombras profundas y reflejos brillantes.

Paleta de Colores

Android Studio crea una paleta de colores para ti. La definición del tema de la aplicación en `styles.xml` puede incluir:

- **colorPrimary**: Branding de la AppBar.
- **colorPrimaryDark**: Contraste de la barra de estado.
- **colorAccent**: Llama la atención del usuario en interruptores, FAB, etc.

Los colores se definen en `colors.xml`.

Contraste de Color y Accesibilidad

El contraste es importante para la separación visual, la legibilidad y la accesibilidad. No todas las personas ven los colores de la misma manera. El tema maneja el texto por defecto:

- **Theme.AppCompat.Light**: El texto será casi negro
- **Theme.AppCompat.Light.DarkActionBar**: El texto será casi blanco

MOVIMIENTO

Movimiento en Material Design

El movimiento en Material Design describe relaciones espaciales, funcionalidad e intención. Es receptivo, natural, consciente e intencional. El material debe responder rápidamente a la entrada del usuario, ser consciente de su entorno y moverse de manera natural e intencional para guiar el enfoque al lugar correcto en el momento correcto.

Movimiento en tu Aplicación

Para mantener la continuidad y resaltar elementos o acciones en tu aplicación:

- **Mantén la continuidad**: Transiciones naturales entre acciones o estados.
- **Resalta elementos**: Usa el movimiento para dibujar el enfoque.
- **Retroalimentación sensible**: Proporciona interacción receptiva.

DISEÑO

Diseño en Material Design

El diseño debe usar píxeles independientes de densidad para vistas (`dp`) y píxeles escalables para texto (`sp`). Los elementos deben alinearse a una cuadrícula con un espaciado consistente.

Planificación del Diseño

- **Planifica tu diseño**: Usa plantillas para patrones de diseño comunes.
- **Alineación de la cuadrícula**: Asegúrate de que todos los elementos se alineen correctamente.
- **Espaciado y tamaño**: Mantén un espaciado y tamaños consistentes.

COMPONENTES

Componentes en Material Design

Material Design proporciona directrices sobre el uso e implementación de componentes en Android, como:

- **Navegación inferior**
- **Botones**
- **Tarjetas**
- **Chips**
- **Tablas de datos**
- **Diálogos**
- **Divisores**
- **Deslizadores**
- **Snackbar**
- **Toast**
- **Steppers**
- **Subheaders**
- **Campos de texto**
- **Barras de herramientas**

Consistencia de Componentes

La consistencia ayuda a la intuición del usuario. Componentes como pestañas, FAB, snackbar y navigation drawer deben ser usados de manera coherente para facilitar la navegación y la interacción.

7.3 Recursos para Diseños Adaptativos en Android

DISEÑOS ADAPTATIVOS Y RECURSOS

¿Qué son los Diseños Adaptativos?

Los diseños adaptativos son layouts que se ven bien en diferentes tamaños de pantalla, orientaciones y dispositivos. Estos diseños se ajustan automáticamente a:

- **Tamaño de la pantalla**
- **Orientación del dispositivo**
- **Configuración regional (locale)**
- **Versión de Android instalada**

Implementación de Diseños Adaptativos

Los diseños adaptativos proporcionan recursos alternativos y utilizan layouts flexibles como `GridLayout`. Esto permite que la interfaz de usuario se ajuste dinámicamente según las diferentes configuraciones del dispositivo.

Estructura de Carpetas de Recursos

Ejemplo de estructura de carpetas en un proyecto pequeño:

```
MyProject/
    src/
    res/
        drawable/
            graphic.png
        layout/
            activity_main.xml
            list_iteminfo.xml
        mipmap/
            ic_launcher_icon.png
        values/
            strings.xml
```

Coloca todos los recursos de tu proyecto en la carpeta `res`.

Directorios Comunes de Recursos

- **drawable/**: Imágenes y gráficos.
- **layout/**: Archivos XML de diseño.
- **menu/**: Archivos XML de menús.
- **values/**: Archivos XML de valores simples como cadenas o colores.
- **xml/**: Archivos XML arbitrarios.
- **raw/**: Archivos arbitrarios en su forma cruda.
- **mipmap/**: Dibujos para diferentes densidades de iconos de lanzador.

RECURSOS ALTERNATIVOS

¿Qué son los Recursos Alternativos?

Diferentes configuraciones de dispositivo pueden requerir diferentes recursos, tales como:

- Cadenas localizadas
- Resoluciones de imagen
- Dimensiones de diseño

Android carga automáticamente los recursos apropiados según la configuración del dispositivo.

Creación de Carpetas de Recursos Alternativos

Usa carpetas alternativas para proporcionar recursos específicos para diferentes configuraciones de dispositivo. Los nombres de estas carpetas tienen el formato `nombre-recurso-calificador-config`.

Nombres de Carpetas de Recursos Alternativos

Ejemplos de nombres de carpetas de recursos alternativos:

- **drawable-hdpi**: Dibujos para pantallas de alta densidad.
- **layout-land**: Diseños para orientación horizontal.
- **layout-v7**: Diseños para versiones específicas de la plataforma.
- **values-fr**: Archivos de valores para la configuración regional francesa.

Orientación de la Pantalla

Proporciona diseños alternativos según la orientación de la pantalla:

- **res/layout-port**: Para diseños específicos en orientación vertical.
- **res/layout-land**: Para diseños específicos en orientación horizontal.

Evita las dimensiones codificadas para reducir la necesidad de diseños especializados.

Ejemplo de Layout Adaptativo Simple

Usando `GridLayout` para ajustar la cantidad de columnas según la orientación:

En `values/integer.xml`:

```
<integer name="grid_column_count">1</integer>
```

En `values/integer.xml-land`:

```
<integer name="grid_column_count">2</integer>
```

RECURSOS PREDETERMINADOS

Importancia de Proveer Recursos Predeterminados

Siempre proporciona recursos predeterminados para asegurar que tu aplicación funcione correctamente en cualquier configuración de dispositivo. Los recursos predeterminados se colocan en carpetas sin calificadores específicos:

- **res/layout**
- **res/values**
- **res/drawable**

Android recurre a los recursos predeterminados cuando no encuentra recursos específicos que coincidan con la configuración del dispositivo.

LOCALIZACIÓN

Proveer Recursos Localizados

Proporciona cadenas y otros recursos para locales específicos para aumentar la audiencia potencial de tu aplicación. La localización se basa en la configuración del dispositivo.

Ejemplo de archivo de cadenas localizado:

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

8. Cómo probar tu IU

8.1 Pruebas de Interfaz de Usuario (UI) en Android

Visión General de las Pruebas de UI

¿Qué es una Prueba de UI?

Las pruebas de UI verifican que la interfaz de usuario de la aplicación funcione correctamente. Estas pruebas realizan todas las acciones de UI del usuario con elementos `View`, tales como:

- Tocar una `View` y entrar datos o hacer una elección.
- Examinar los valores de las propiedades de cada `View`.
- Proporcionar entrada a todos los elementos `View`.
- Probar valores inválidos.
- Verificar la salida devuelta.
- Comprobar si los valores son correctos o esperados.
- Confirmar la presentación correcta de la UI.

Problemas con las Pruebas Manuales

Las pruebas manuales pueden ser:

- **Consumen mucho tiempo y son tediosas**.
- **Propensas a errores**.
- **La UI puede cambiar y necesitar pruebas frecuentes**.
- **A medida que la aplicación se vuelve más compleja, las secuencias posibles de acciones pueden crecer exponencialmente**.

Beneficios de las Pruebas Automáticas

- **Libera tiempo y recursos** para otros trabajos.
- **Más rápidas que las pruebas manuales**.
- **Repetibles**.
- **Ejecutan pruebas para diferentes estados y configuraciones del dispositivo**.
- **Verifican que la UI se comporte como se espera**.
- **Comprueban que la aplicación devuelva la salida de UI correcta en respuesta a las interacciones del usuario**.

CONFIGURACIÓN DEL ENTORNO DE PRUEBAS Y ESPRESSO

Instalación de la Biblioteca de Soporte de Android

1. En Android Studio, elige `Tools > Android > SDK Manager`.
2. Haz clic en `SDK Tools` y busca `Android Support Repository`.
3. Si es necesario, actualiza o instala la biblioteca.

Añadir Dependencias en build.gradle

```
dependencies {
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation
'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

Configuración de defaultConfig en build.gradle

```
android {
    defaultConfig {
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
}
```

Preparación del Dispositivo

- Activa la **Depuración USB**.
- Desactiva todas las animaciones en `Developer Options > Drawing`:
  - Escala de animación de ventana
  - Escala de animación de transición
  - Escala de duración del animador

Creación de Pruebas

Guarda las pruebas en `module-name/src/androidTests/java/`. En Android Studio, navega a `app > java > module-name (androidTest)` y crea las pruebas como clases JUnit.

CREACIÓN DE PRUEBAS CON ESPRESSO

Anotaciones y Definiciones de Clases de Pruebas

```
@RunWith(AndroidJUnit4.class) // Anotación requerida para pruebas
@LargeTest // Basado en los recursos que la prueba usa y el tiempo de
ejecución
public class ChangeTextBehaviorTest {
    // Clase de prueba
```

```
}
```

Definición del Contexto de la Prueba

```
@Rule
public ActivityTestRule<MainActivity> mActivityRule =
    new ActivityTestRule<>(MainActivity.class);
```

Métodos @Before y @After

```
@Before
public void initValidString() {
    mStringToBetyped = "Espresso";
}

@After
public void tearDown() {
    // Liberar recursos después de la prueba
}
```

Estructura del Método de Prueba @Test

```
@Test
public void changeText_sameActivity() {
    // 1. Encontrar una View
    // 2. Realizar una acción
    // 3. Verificar que la acción fue realizada
}
```

Uso de Hamcrest Matchers

Hamcrest es un framework para crear matchers y assertions personalizados. Simplifica las pruebas al permitir definir reglas de coincidencia de manera declarativa y precisa.

EJEMPLOS DE PRUEBAS CON ESPRESSO

Ejemplo Básico de Prueba

```
@Test
public void changeText_sameActivity() {
    // 1. Encontrar la vista por Id
    onView(withId(R.id.editTextUserInput))
        // 2. Realizar acción: escribir texto y cerrar el teclado
        .perform(typeText(mStringToBetyped), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());
    // 3. Verificar que el texto fue cambiado
    onView(withId(R.id.textToBeChanged))
        .check(matches(withText(mStringToBetyped)));
}
```

Encontrar Vistas con onView

```
onView(withId(R.id.editTextUserInput)) // Buscar una vista con el Id
especificado
```

```
onView(withText("Sample Text")) // Buscar una vista con texto
específico
onView(allOf(withId(R.id.word), withText("Clicked! Word 15"),
isDisplayed())) // Buscar una vista que coincida con múltiples
condiciones
```

Realizar Acciones

```
onView(withId(R.id.editTextUserInput))
    .perform(typeText(mStringToBetyped), closeSoftKeyboard());
onView(withId(R.id.changeTextBt)).perform(click());
```

Verificar Resultados

```
onView(withId(R.id.textToBeChanged))
    .check(matches(withText(mStringToBetyped)));
```

Ejemplo de Resultado de Prueba Fallida

```
onView(withId(R.id.text_message))
    .check(matches(withText("This is a failing test.")));

// Resultado esperado: "This is a failing test."
// Resultado obtenido: "AppCompatTextView{id=2131427417 res-
name=text_message ..."
```

GRABACIÓN DE PRUEBAS

Grabación de una Prueba con Espresso

1. En Android Studio, selecciona `Run > Record Espresso Test`.
2. Haz clic en `Restart app`, selecciona el objetivo y haz clic en `OK`.
3. Interactúa con la aplicación para realizar las acciones que deseas probar.
4. Haz clic en `Add Assertion` y selecciona un elemento de la UI.
5. Elige `text is` e ingresa el texto que esperas ver.
6. Haz clic en `Save Assertion` y luego en `Complete Recording`.

9. Conclusiones

A lo largo de esta primera unidad didáctica, hemos explorado los conceptos fundamentales de la programación dirigida por eventos, particularmente en el contexto del desarrollo de aplicaciones móviles con Android Studio. La comprensión de este paradigma es esencial para crear aplicaciones interactivas y dinámicas que respondan adecuadamente a las acciones del usuario. Desde la introducción a Android Studio y la creación de la primera aplicación “Hello World”, hasta el manejo de eventos, el ciclo de vida de las actividades y el uso de vistas, hemos cubierto las herramientas y técnicas clave que permiten a los desarrolladores implementar interfaces de usuario eficientes y amigables.

Un aspecto crucial es la capacidad de manejar eventos, como los clics y entradas del usuario, lo que constituye la base de las aplicaciones interactivas. La correcta gestión del ciclo de vida de una actividad asegura que la aplicación sea robusta y capaz de manejar transiciones, cambios de estado y eventos inesperados, como la rotación de la pantalla o la interrupción de una llamada telefónica.

Otro punto destacado es la importancia de las herramientas de depuración y prueba proporcionadas por Android Studio, que facilitan la identificación y corrección de errores durante el desarrollo. Además, la implementación de bibliotecas de soporte asegura que las aplicaciones sean compatibles con versiones anteriores de Android, ampliando su alcance a un público más amplio.

En conclusión, los conocimientos adquiridos en esta unidad sientan las bases para abordar proyectos más complejos en el desarrollo de aplicaciones móviles. El manejo efectivo de eventos y la implementación de mejores prácticas en la creación de interfaces y la gestión de actividades resultan esenciales para el éxito de las aplicaciones modernas. A medida que avanzamos en las siguientes unidades, estos fundamentos permitirán una evolución hacia el desarrollo de aplicaciones más sofisticadas y con mayor impacto en el entorno digital.

10. Bibliografía

Para la elaboración de este tema se han consultado las siguientes fuentes y recursos:

- Android Developers. (2023). *Android Studio User Guide*. Google. Disponible en: https://developer.android.com/studio
- Deitel, P., & Deitel, H. (2017). *Android 6 for Programmers: An App-Driven Approach*. Prentice Hall.
- Phillips, B., Stewart, C., Hardy, K., & Marsicano, B. (2019). *Android Programming: The Big Nerd Ranch Guide*. 4ª edición. Big Nerd Ranch.
- Meier, R. (2015). *Professional Android: Building Apps with Android Studio*. John Wiley & Sons.
- Burnette, E. (2018). *Hello, Android: Introducing Google's Mobile Development Platform*. 4ª edición. Pragmatic Bookshelf.
- Shields, R., & Tofferi, D. (2022). *Learning Android Development*. Packt Publishing.

Este conjunto de materiales proporcionó un apoyo esencial para entender los conceptos y prácticas de programación en Android, así como las técnicas de desarrollo de interfaces interactivas y la gestión eficiente de eventos dentro de las aplicaciones móviles.

UAX Universidad Alfonso X el Sabio

G R A C I A S

UAX.COM