Unidad Didáctica 6:
CÓMO AGREGAR FUNCIONES GEOGRÁFICAS A TUS APPS

Programación Dirigida por Eventos

UAX Universidad Alfonso X el Sabio

ÍNDICE

Contenido

1. Introducción

¡Bienvenido a la asignatura Programación Dirigida por Eventos!

En esta unidad didáctica, exploraremos "Cómo agregar funciones geográficas a tus apps". Aprenderás a integrar Google Maps y otros servicios de localización en tus aplicaciones Android, permitiendo a los usuarios interactuar con el entorno geográfico a través de sus dispositivos. Este conocimiento es fundamental para desarrollar aplicaciones modernas que requieren servicios de localización y mapas.

Ten en cuenta:

La integración de funciones geográficas no solo enriquece la experiencia del usuario, sino que también proporciona funcionalidades esenciales para aplicaciones de navegación, entrega, servicios basados en la ubicación y muchas más. Es crucial entender cómo gestionar los permisos de localización y optimizar el uso de la batería mientras se utilizan servicios de ubicación.

Temas que se tratarán en esta unidad:

1. **Servicios de Localización en Android:** Comprender los diferentes servicios de localización disponibles y cómo utilizarlos en una aplicación Android.
2. **Configuración de Google Play Services:** Instalación y configuración de Google Play Services para servicios de localización.
3. **Permisos de Localización:** Manejo de permisos de localización en tiempo de ejecución y en el manifiesto.
4. **FusedLocationProviderClient:** Uso de este cliente para obtener la ubicación del dispositivo de manera eficiente.
5. **Geocodificación y Geocodificación Inversa:** Conversión de direcciones en coordenadas de latitud/longitud y viceversa.
6. **Creación de LocationRequest:** Configuración de las solicitudes de actualización de ubicación.
7. **Trabajar con la Configuración del Usuario:** Verificación y solicitud de cambios en la configuración del dispositivo para servicios de localización.
8. **Solicitar Actualizaciones de Ubicación:** Implementación de solicitudes regulares de actualización de ubicación.
9. **API de Places en Android:** Uso de la API de Places para acceder a datos de lugares y negocios.
10. **Uso de Mapas en tus Aplicaciones Android:** Integración de Google Maps en aplicaciones Android.
11. **Configuración y Personalización de Mapas:** Configuración inicial y personalización de mapas en aplicaciones.
12. **Agregar Interactividad a los Mapas:** Implementación de eventos y controles de UI en mapas.
13. **Uso de Superposiciones y Formas:** Añadir superposiciones y dibujar formas en mapas.
14. **Google Street View:** Integración de vistas panorámicas de 360 grados.

15. **Vistas Personalizadas en Android:** Creación y uso de vistas personalizadas en aplicaciones Android.
16. **Animaciones:** Implementación de animaciones para mejorar la interactividad y experiencia del usuario.
17. **Reproducción de Medios:** Uso de VideoView y ExoPlayer para la reproducción de medios en aplicaciones.
18. **Componentes de Arquitectura:** Uso de Room, ViewModel y Repository para gestionar datos en aplicaciones Android.

Al finalizar esta unidad, deberías ser capaz de integrar funciones geográficas en tus aplicaciones Android, ofreciendo una experiencia rica y dinámica a los usuarios.

2. Objetivos

Objetivo general:

- Proporcionar a los estudiantes las habilidades necesarias para agregar y gestionar funciones geográficas en aplicaciones Android, utilizando servicios de localización y Google Maps.

Objetivos específicos:

En esta unidad se establecen tres objetivos específicos:

1. **Comprender y utilizar servicios de localización en Android:** Aprender a configurar y utilizar servicios de localización, gestionar permisos y optimizar el uso de la batería.
2. **Integrar y personalizar Google Maps en aplicaciones:** Desarrollar la capacidad de integrar Google Maps, añadir interactividad y personalizar mapas en aplicaciones Android.
3. **Implementar API de Places y técnicas de geocodificación:** Aplicar técnicas para obtener y manejar datos de lugares, realizar geocodificación y geocodificación inversa en aplicaciones Android.

3. Ubicación

3.1. Servicios de Localización en Android

Visión General

Introducción

Los teléfonos móviles se destacan por su capacidad de movilidad, permitiendo a los usuarios moverse y desplazarse. Tu aplicación puede detectar y utilizar la ubicación del dispositivo para personalizar la experiencia del usuario.

Uso de la Ubicación en tu Aplicación

Para utilizar la ubicación en tu aplicación, sigue estos pasos generales:

1. Verificar si se ha concedido el permiso de ubicación.
2. Solicitar el permiso si es necesario.
3. Solicitar la ubicación más reciente.
4. Solicitar actualizaciones de ubicación.

Cada uno de estos pasos se detalla a continuación.

Configuración de Google Play Services

Instalación del Repositorio de Google

Los servicios de localización son proporcionados por Google Play Services. Para configurar estos servicios, instala el Google Repository en Android Studio:

1. Selecciona **Tools > Android > SDK Manager**.
2. Ve a la pestaña **SDK Tools**.
3. Expande **Support Repository**.
4. Selecciona **Google Repository** y haz clic en **OK**.

Añadir Google Play a tu Proyecto

Agrega la dependencia en build.gradle (Module: app):

```
dependencies {
   implementation 'com.google.android.gms:play-services-location:xx.x.x'
}
```

Reemplaza xx.x.x con el número de versión sugerido por Android Studio.

Permisos de Localización

Permisos y Configuración del Usuario

Desde Marshmallow (API 23) en adelante, los usuarios pueden conceder o denegar el acceso a su ubicación para cada aplicación. Además, pueden cambiar el permiso en cualquier momento. Las aplicaciones deben solicitar permisos explícitos para acceder a la ubicación del dispositivo.

Permisos en el Manifiesto

Para solicitar acceso a la ubicación en el manifiesto, agrega:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Solicitar Permisos en Tiempo de Ejecución

Los permisos de ubicación deben verificarse y solicitarse en tiempo de ejecución, ya que los usuarios pueden revocarlos en cualquier momento.

Ejemplo de Verificación y Solicitud de Permisos

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_LOCATION_PERMISSION);
} else {
   Log.d(TAG, "getLocation: permissions granted");
}
```

Manejar la Respuesta del Usuario

Sobrescribe onRequestPermissionsResult() para verificar si el usuario ha concedido el permiso.

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
   if (requestCode == REQUEST_LOCATION_PERMISSION) {
      if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
         // Permiso concedido
      } else {
```

```
            // Permiso denegado
        }
    }
}
```

Obtener la Ubicación del Dispositivo

Uso de FusedLocationProviderClient

FusedLocationProviderClient combina GPS, Wi-Fi y la red celular para equilibrar resultados rápidos y precisos con un mínimo consumo de batería. Este cliente devuelve un objeto Location con la latitud y longitud.

Configurar FusedLocationProviderClient

```
FusedLocationProviderClient flpClient = LocationServices.getFusedLocationProviderClient(context);
```

Solicitar la Última Ubicación Conocida

Para obtener la última ubicación conocida:

```
flpClient.getLastLocation().addOnSuccessListener(new OnSuccessListener<Location>() {
    @Override
    public void onSuccess(Location location) {
        if (location != null) {
            double lat = location.getLatitude();
            double lng = location.getLongitude();
            // Usar la latitud y longitud
        } else {
            // Manejar "no location"
        }
    }
});
```

Manejar Fallos en la Solicitud de Ubicación

```
flpClient.getLastLocation().addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.e(TAG, "onFailure: ", e);
    }
});
```

Geocodificación y Geocodificación Inversa

Uso de la Clase Geocoder

Geocoder se utiliza para convertir una dirección en coordenadas de latitud/longitud y viceversa.

Geocodificación Inversa

Convierte coordenadas de latitud/longitud en una dirección legible.

```
Geocoder geocoder = new Geocoder(context, Locale.getDefault());
List<Address> addresses = geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), 1);
if (addresses != null && !addresses.isEmpty()) {
   Address address = addresses.get(0);
   String addressString = address.getAddressLine(0);
   // Usar la dirección
}
```

Geocodificación de Direcciones

Convierte una dirección en coordenadas de latitud/longitud.

```
List<Address> addresses = geocoder.getFromLocationName("1600 Amphitheatre Parkway, Mountain View, CA", 1);
if (addresses != null && !addresses.isEmpty()) {
   Address address = addresses.get(0);
   double latitude = address.getLatitude();
   double longitude = address.getLongitude();
   // Usar latitud y longitud
}
```

Creación de un Objeto LocationRequest

Configurar LocationRequest

LocationRequest se utiliza para establecer parámetros para las solicitudes de actualización de ubicación.

Parámetros de LocationRequest

- **setInterval()**: Frecuencia de las actualizaciones.
- **setFastestInterval()**: Límite de la tasa de actualización para evitar sobrecarga de datos.

- **setPriority()**: Establece la prioridad de la solicitud y las fuentes.

Ejemplo de Creación de LocationRequest

```
private LocationRequest getLocationRequest() {
   LocationRequest locationRequest = new LocationRequest();
   locationRequest.setInterval(10000);
   locationRequest.setFastestInterval(5000);
   locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
   return locationRequest;
}
```

Trabajar con la Configuración del Usuario

Configuración del Usuario para Servicios de Localización

Los usuarios pueden controlar el equilibrio entre precisión y consumo de energía. Tu aplicación puede detectar la configuración del dispositivo y solicitar al usuario que la cambie si es necesario.

Verificar la Configuración del Dispositivo

1. Crear un objeto LocationSettingsRequest.
2. Crear un objeto SettingsClient.
3. Usar checkLocationSettings() para verificar si la configuración del dispositivo coincide con LocationRequest.

Ejemplo de Verificación de Configuración

```
LocationSettingsRequest settingsRequest = new LocationSettingsRequest.Builder().addLocationRequest(mLocationRequest).build();
SettingsClient client = LocationServices.getSettingsClient(this);
Task<LocationSettingsResponse> task = client.checkLocationSettings(settingsRequest);
task.addOnFailureListener(this, new OnFailureListener() {
   @Override
   public void onFailure(@NonNull Exception e) {
      int statusCode = ((ApiException) e).getStatusCode();
      if (statusCode == LocationSettingsStatusCodes.RESOLUTION_REQUIRED) {
         // Mostrar diálogo al usuario
         try {
            ResolvableApiException resolvable = (ResolvableApiException) e;
            resolvable.startResolutionForResult(MainActivity.this, REQUEST_CHECK_SETTINGS);
         } catch (IntentSender.SendIntentException sendEx) {
            // Manejar error
         }
```

```
        }
    }
});
```

Manejar la Decisión del Usuario

Sobrescribe onActivityResult() en la actividad para asegurarte de que el requestCode coincida con la constante en startResolutionForResult().

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CHECK_SETTINGS) {
        if (resultCode == RESULT_CANCELED) {
            // Detener el seguimiento de ubicación
        } else if (resultCode == RESULT_OK) {
            // Iniciar el seguimiento de ubicación
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

Solicitar Actualizaciones de Ubicación

Uso de LocationRequest con FusedLocationProviderClient

Para recibir actualizaciones regulares de ubicación:

1. Crear un objeto LocationRequest.
2. Sobrescribir LocationCallback.onLocationResult().
3. Usar requestLocationUpdates() en FusedLocationProviderClient para iniciar las actualizaciones regulares.

Ejemplo de Solicitud de Actualizaciones de Ubicación

```
mLocationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        for (Location location : locationResult.getLocations()) {
            // Actualizar UI con los datos de ubicación
        }
    }
};

FusedLocationProviderClient flpClient = LocationServices.getFusedLocationProviderClient(this);
flpClient.requestLocationUpdates(getLocationRequest(), mLocationCallback, null);
```

4. Lugares

4.1. API de Places en Android

Visión General y Configuración de la API

¿Qué es un "lugar" en Google Maps?

Un **lugar** es un espacio físico que tiene un nombre y está en un mapa. Incluye negocios, puntos de interés y ubicaciones geográficas.

Un **objeto Place** representa un lugar y puede incluir:

- Nombre
- Dirección
- Ubicación geográfica
- ID del lugar
- Número de teléfono
- Tipo de lugar
- URL del sitio web

API de Places

La API de Places proporciona acceso a los mismos datos de lugares utilizados por Google Maps.

Acceso a la API de Places

La API de Places incluye varios servicios:

- **PlacePicker**: Permite a los usuarios buscar y seleccionar un lugar en un mapa.
- **PlaceDetectionApi**: Proporciona acceso al lugar actual del dispositivo.
- **GeoDataApi**: Accede a la base de datos de Google de información local de lugares y negocios, incluyendo imágenes.
- **PlaceAutocomplete**: Permite a los usuarios buscar un lugar específico con sugerencias de autocompletado.

Requisitos para usar la API de Places

1. **Configurar Google Play Services**
2. **Agregar la dependencia de Google Play Services al proyecto**
3. **Registrar tu aplicación y obtener la clave de la API**

Configuración de Google Play Services

Para instalar Google Repository en Android Studio:

1. Selecciona **Tools > Android > SDK Manager**.
2. Ve a la pestaña **SDK Tools**.
3. Expande **Support Repository**.
4. Selecciona **Google Repository** y haz clic en **OK**.

Añadir Google Play Services al Proyecto

Agrega la dependencia en build.gradle (Module: app):

```
dependencies {
   implementation 'com.google.android.gms:play-services-location:xx.x.x'
}
```

Reemplaza xx.x.x con el número de versión sugerido por Android Studio.

Registrar tu Aplicación y Obtener la Clave de la API

1. Registra tu aplicación en Google API Console.
2. Obtén la información para el certificado de depuración y el certificado de liberación.
3. Habilita Google Places API y solicita la clave de la API.
4. Añade la clave de la API a AndroidManifest.xml:

```
<application>
 <meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY"/>
</application>
```

Configuración de Permisos

Solicitar Permisos en Tiempo de Ejecución

Los usuarios pueden revocar los permisos en cualquier momento, por lo que es importante verificar y solicitar permisos cada vez que tu aplicación utilice la ubicación.

Permiso de Ubicación

Tu aplicación debe solicitar el permiso ACCESS_FINE_LOCATION.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Verificación y Solicitud de Permisos

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_LOCATION_PERMISSION);
} else {
   Log.d(TAG, "getLocation: permissions granted");
}
```

Manejar la Respuesta del Usuario

Sobrescribe onRequestPermissionsResult() para verificar si el usuario ha concedido el permiso.

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
   if (requestCode == REQUEST_LOCATION_PERMISSION) {
      if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
         // Permiso concedido
      } else {
         // Permiso denegado
      }
   }
}
```

Uso de los Detalles de un Lugar

Uso de Place ID

Un **Place ID** es un identificador único de un lugar. Para obtener el ID de un lugar, llama a Place.getId(). Usa el Place ID para obtener un objeto Place.

Ejemplo de Obtener el ID de un Lugar

```
mGeoDataClient.getPlaceById(placeId).addOnCompleteListener(new OnCompleteListener<PlaceBufferResponse>() {
   @Override
   public void onComplete(@NonNull Task<PlaceBufferResponse> task) {
      if (task.isSuccessful()) {
```

```
            PlaceBufferResponse places = task.getResult();
            Place myPlace = places.get(0);
            Log.i(TAG, "Place found: " + myPlace.getName());
            places.release();
        } else {
            Log.e(TAG, "Place not found.");
        }
    }
});
```

Obtener Datos de un Objeto Place

Puedes obtener diversos datos de un objeto Place:

- getName(): Nombre del lugar.
- getAddress(): Dirección en formato legible.
- getId(): ID del lugar.
- getPhoneNumber(): Número de teléfono.
- getWebsiteUri(): URL del sitio web.
- getLatLng(): Latitud y longitud del lugar.
- getViewport(): LatLngBounds para mostrar el lugar en un mapa.
- getLocale(): Configuración regional del lugar.
- getPlaceTypes(): Lista de tipos de lugar.
- getPriceLevel(): Nivel de precio (0 a 4).
- getRating(): Calificación agregada de usuarios (1.0 a 5.0).

Uso de la Interfaz de Usuario de PlacePicker

PlacePicker

PlacePicker muestra un mapa interactivo y una lista de lugares cercanos. Los usuarios pueden seleccionar o buscar un lugar, y la aplicación recupera los detalles del lugar seleccionado.

Ejemplo de Uso de PlacePicker

```
PlacePicker.IntentBuilder builder = new PlacePicker.IntentBuilder();
try {
    startActivityForResult(builder.build(MainActivity.this), REQUEST_PICK_PLACE);
} catch (GooglePlayServicesRepairableException | GooglePlayServicesNotAvailableException e)
{
    e.printStackTrace();
}
```

Obtener el Lugar Seleccionado por el Usuario

En onActivityResult(), llama a PlacePicker.getPlace() para obtener el lugar seleccionado por el usuario.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   if (resultCode == RESULT_OK) {
      Place place = PlacePicker.getPlace(this, data);
      setAndroidType(place);
      mLocationTextView.setText(getString(R.string.address_text,                place.getName(),
place.getAddress(), System.currentTimeMillis()));
   }
   super.onActivityResult(requestCode, resultCode, data);
}
```

Obtención de Lugares Cercanos

Uso de PlaceDetectionClient

Para obtener lugares en la ubicación del dispositivo, usa PlaceDetectionClient.getCurrentPlace(), que devuelve una lista de objetos PlaceLikelihood.

Ejemplo de Obtener Lugares Cercanos

```
Task<PlaceLikelihoodBufferResponse>                placeResult                =
mPlaceDetectionClient.getCurrentPlace(null);
placeResult.addOnCompleteListener(new
OnCompleteListener<PlaceLikelihoodBufferResponse>() {
   @Override
   public void onComplete(@NonNull Task<PlaceLikelihoodBufferResponse> task) {
      if (task.isSuccessful()) {
         PlaceLikelihoodBufferResponse likelyPlaces = task.getResult();
         float maxLikelihood = 0;
         Place currentPlace = null;
         for (PlaceLikelihood placeLikelihood : likelyPlaces) {
            if (maxLikelihood < placeLikelihood.getLikelihood()) {
               maxLikelihood = placeLikelihood.getLikelihood();
               currentPlace = placeLikelihood.getPlace();
            }
         }
         // Actualizar la UI
         likelyPlaces.release();
      } else {
```

```
            Log.e(TAG, "Error getting places");
        }
    }
});
```

Filtrar la Lista de Lugares

Usa PlaceFilter para limitar los resultados incluidos en PlaceLikelihoodBuffer.

```
PlaceFilter placeFilter = new PlaceFilter(true, Arrays.asList("place_id1", "place_id2"));
Task<PlaceLikelihoodBufferResponse> placeResult = mPlaceDetectionClient.getCurrentPlace(placeFilter);
```

Uso del Servicio de Autocompletado de Lugares

PlaceAutocomplete

El autocompletado de lugares proporciona sugerencias basadas en las consultas de búsqueda del usuario. Puedes utilizar el widget de UI de autocompletado o crear una interfaz de usuario personalizada.

Ejemplo de Uso del Widget de Autocompletado

Para usar el widget de autocompletado, elige entre fragmento o actividad:

```
PlaceAutocompleteFragment autocompleteFragment = (PlaceAutocompleteFragment)
      getFragmentManager().findFragmentById(R.id.place_autocomplete_fragment);

autocompleteFragment.setOnPlaceSelectedListener(new PlaceSelectionListener() {
    @Override
    public void onPlaceSelected(Place place) {
        // Manejar el lugar seleccionado
    }

    @Override
    public void onError(Status status) {
        // Manejar el error
    }
});
```

Obtener Predicciones de Lugares en una Interfaz de Usuario Personalizada

Llama a GeoDataClient.getAutocompletePredictions() para obtener una lista de nombres de lugares y/o direcciones predichos.

```
Task<AutocompletePredictionBufferResponse> results =
mGeoDataClient.getAutocompletePredictions(query, bounds, filter);
results.addOnSuccessListener(new
OnSuccessListener<AutocompletePredictionBufferResponse>() {
   @Override
   public void onSuccess(AutocompletePredictionBufferResponse response) {
      // Manejar las predicciones de autocompletado
   }
});
```

5. Mapas

5.1. Uso de Mapas en tus Aplicaciones Android

Configuración de la API

Introducción a la API de Maps

Google Maps API te permite incluir mapas de Google en tu aplicación. Puedes personalizar los mapas con controles, estilos, marcadores y más.

Configuración del Proyecto y Obtención de Claves de API

Registro de la Aplicación y Obtención de la Clave de la API

1. **Registrar tu aplicación en Google API Console**.
2. **Obtener el certificado de depuración y el certificado de liberación**.
3. **Solicitar la clave de la API**.

Incluir Google Maps en tu Aplicación

1. **Obtener las claves de API para usar Google Maps**.
2. **Incluir Google Maps en tu aplicación**.
3. **Cambiar la apariencia y el comportamiento del mapa**.

Uso de la Plantilla de Actividad de Google Maps en Android Studio

La plantilla de Google Maps en Android Studio crea el código base necesario y proporciona un enlace para obtener la clave de la API.

xml

Copiar código
<string name="google_maps_key"
templateMergeStrategy="preserve">YOUR_KEY_HERE</string>

Mostrar el Mapa

Añadir el Mapa al Diseño

Para agregar un mapa a tu diseño, usa:

- **MapView**
- **MapFragment**

Es más sencillo usar MapFragment ya que no requiere que reenvíes todos los métodos del ciclo de vida.

Ejemplo de Uso de MapFragment

```
<fragment
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"/>
```

onMapReadyCallback

La actividad que muestra el mapa debe implementar la interfaz OnMapReadyCallback. Cuando el mapa está listo, se pasa a onMapReady().

Ejemplo de Implementación

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_maps);
  SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
  mapFragment.getMapAsync(this);
}

@Override
public void onMapReady(GoogleMap googleMap) {
  LatLng sydney = new LatLng(-34, 151);
```

```
    googleMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
```

Configurar el Estado Inicial del Mapa

Configuración Inicial del Mapa

Puedes configurar el estado inicial del mapa en código o como atributos XML para MapView o MapFragment.

Atributos XML Personalizados

```
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    map:cameraBearing="112.5"
    map:cameraTargetLat="-33.8"
    map:cameraTargetLng="151"
    map:cameraTilt="30"
    map:cameraZoom="13"
    map:mapType="normal"
    map:uiCompass="false"
    map:uiRotateGestures="true"
    map:uiTiltGestures="true"
    map:uiZoomGestures="true"/>
```

Configuración de Opciones del Mapa en Código

```
GoogleMapOptions options = new GoogleMapOptions()
    .mapType(GoogleMap.MAP_TYPE_SATELLITE)
    .compassEnabled(false)
    .rotateGesturesEnabled(false)
    .tiltGesturesEnabled(false);
```

Tipos de Mapa

Tipos de Mapa Disponibles

- **Normal**: Mapa de carreteras.
- **Satellite**: Foto satelital.

- **Hybrid**: Satélite y carreteras.
- **Terrain**: Datos topográficos.
- **None**: Sin mosaicos.

Ejemplo de Configuración del Tipo de Mapa

```
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

Estilos de Mapa y Modo Lite

Estilos de Mapa

Puedes cambiar el estilo de tus mapas usando el asistente de estilo de mapas (mapstyle.withgoogle.com). El asistente genera un archivo JSON que puedes usar para aplicar estilos a tu mapa.

Ejemplo de Uso del Estilo de Mapa

```
try {
   boolean success = googleMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this,
R.raw.map_style));
   if (!success) {
      Log.e(TAG, "Style parsing failed.");
   }
} catch (Resources.NotFoundException e) {
   Log.e(TAG, "Can't find style file. Error: ", e);
}
```

Modo Lite

El modo lite muestra una imagen de mapa con interactividad limitada. Es útil para listas de mapas o mapas pequeños sin interacción.

Ejemplo de Implementación del Modo Lite

```
<fragment
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:map="http://schemas.android.com/apk/res-auto"
   android:name="com.google.android.gms.maps.SupportMapFragment"
   android:id="@+id/map"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   map:liteMode="true"/>
```

Agregar Interactividad a los Mapas

Escuchar Eventos del Mapa

El objeto GoogleMap tiene muchos listeners para eventos como clics en el mapa, cambios en la cámara, clics en marcadores y más.

Ejemplo de Configuración de un Listener

```
map.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
   @Override
   public void onMapClick(LatLng latLng) {
      // Manejar el clic en el mapa
   }
});
```

Mover la Cámara y la Vista

Posición de la Cámara

La posición de la cámara especifica la ubicación, orientación y perspectiva del mapa.

- **Target**: Centro del mapa (Latitud/Longitud).
- **Bearing**: Dirección vertical medida en grados en el sentido de las agujas del reloj desde el norte.
- **Tilt**: Ángulo de visión.
- **Zoom**: Escala del mapa.

Niveles de Zoom

- **1**: Mundo.
- **5**: Continente.
- **10**: Ciudad.
- **15**: Calles.
- **20**: Edificios.

Pasos para Mover la Cámara

```
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(home, zoom));
```

Marcadores

Añadir Marcadores

Puedes agregar marcadores para señalar ubicaciones específicas en el mapa.

Ejemplo de Añadir un Marcador

```
LatLng sydney = new LatLng(-34, 151);
mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
```

Personalización de Marcadores

Puedes cambiar el color, la imagen o el punto de anclaje del marcador.

Negocios y Puntos de Interés (POIs)

Puntos de Interés

Los POIs incluyen atracciones, negocios, edificios gubernamentales, instalaciones médicas, parques, lugares de culto, escuelas y complejos deportivos.

Uso de OnPoiClickListener

Puedes responder a los clics en los POIs y mostrar una ventana de información.

Ejemplo de Uso de OnPoiClickListener

```
@Override
public void onPoiClick(PointOfInterest poi) {
   Toast.makeText(getApplicationContext(), "Clicked: " + poi.name,
Toast.LENGTH_SHORT).show();
}
```

Controles de UI

Controles de UI en el Mapa

Puedes habilitar o deshabilitar los controles de UI del mapa utilizando UiSettings.

Ejemplo de Configuración de Controles

```
UiSettings uiSettings = mMap.getUiSettings();
uiSettings.setZoomControlsEnabled(true);
uiSettings.setCompassEnabled(true);
uiSettings.setMapToolbarEnabled(true);
```

Gestos

Puedes habilitar o deshabilitar gestos como zoom, desplazamiento, inclinación y rotación.

Ejemplo de Configuración de Gestos

```
uiSettings.setZoomGesturesEnabled(true);
uiSettings.setScrollGesturesEnabled(true);
uiSettings.setTiltGesturesEnabled(true);
uiSettings.setRotateGesturesEnabled(true);
```

Superposiciones y Formas

Uso de Superposiciones de Terreno y Mosaico

- **Superposición de Terreno**: Imagen fija en un punto específico del mapa.
- **Superposición de Mosaico**: Conjunto de imágenes superpuestas a los mosaicos del mapa base.

Ejemplo de Añadir una Superposición de Terreno

```
GroundOverlayOptions homeOverlay = new GroundOverlayOptions()
   .image(BitmapDescriptorFactory.fromResource(R.drawable.android))
   .position(home, 100);
mMap.addGroundOverlay(homeOverlay);
```

Dibujar Formas en el Mapa

- **Polilíneas**: Serie de segmentos de línea conectados.
- **Polígonos**: Forma cerrada.
- **Círculos**: Proyección geográficamente precisa de un círculo.

Ejemplo de Dibujar una Polilínea

```
PolylineOptions polylineOptions = new PolylineOptions()
   .add(new LatLng(-35.016, 143.321))
```

```
   .add(new LatLng(-34.747, 145.592));
mMap.addPolyline(polylineOptions);
```

Street View

Google Street View

Street View proporciona una vista panorámica de 360 grados. Usa StreetViewPanoramaFragment para fragmentos y StreetViewPanoramaView para vistas.

Ejemplo de Configuración de Street View

```
StreetViewPanoramaFragment streetViewPanoramaFragment =
   (StreetViewPanoramaFragment)
getFragmentManager().findFragmentById(R.id.streetviewpanorama);
streetViewPanoramaFragment.getStreetViewPanoramaAsync(this);
```

6. Vistas personalizadas

6.1 Vistas Personalizadas en Android

Visión General de las Vistas Personalizadas

La Clase View y sus Subclases

La clase View es el bloque básico de la UI en Android. Puedes utilizar subclases de View (como EditText, Button, etc.) para habilitar la interacción del usuario y mostrar información. Puedes extender View o cualquier subclase de View para personalizar la apariencia y el comportamiento, incluyendo la interacción del usuario.

Personalización de Subclases de View

- **Heredar y personalizar el aspecto y comportamiento de la subclase**: Por ejemplo, una EditText extendida con un botón de reinicio (X) para borrar el texto.
- **Dibujar cualquier forma o tamaño**: Crear tus propios elementos interactivos y comportamientos.

Creación de una Clase de Vista Personalizada

Pasos para Crear Vistas Personalizadas

1. **Crear una clase que extienda View o una subclase de View**.
2. **Sobrescribir métodos de View**:
    - Extender View: Dibujar la vista completa sobrescribiendo onDraw().
    - Extender una subclase de View: Sobrescribir comportamiento o apariencia.
3. **Usar la clase de vista personalizada en un diseño**.

Crear una Clase de Vista Personalizada

Ejemplo de Creación de Clase Personalizada

```
public class CustomViewExample extends AppCompatEditText {
   // Constructores requeridos
   public CustomViewExample(Context context) {
      super(context);
      init();
   }

   public CustomViewExample(Context context, AttributeSet attrs) {
      super(context, attrs);
```

```
    init();
  }

  public CustomViewExample(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);
    init();
  }

  // Método de inicialización
  private void init() {
    // Definir variable miembro para el drawable
    mClearButtonImage = ResourcesCompat.getDrawable(getResources(),
R.drawable.ic_clear_opaque_24dp, null);
    // Configurar acciones para el botón de limpieza
  }
}
```

Dibujo de la Vista Personalizada

Extender una Subclase para Usar su Apariencia

No es necesario escribir código para dibujar o redibujar si extiendes una subclase de View. La subclase extendida hereda la apariencia y el comportamiento de la subclase original.

Ejemplo: Extender EditText

Sobrescribir métodos de la subclase EditText para personalizar la apariencia (agregar un botón X):

```
@Override
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  // Dibujar el botón X
}
```

Métodos de Dibujo de Canvas y Paint

- **drawText()**: Dibujar texto.
- **setTypeface()**: Establecer la fuente del texto.
- **setColor()**: Establecer el color del texto.
- **drawRect(), drawOval(), drawArc()**: Dibujar formas.
- **setStyle()**: Establecer el relleno y contorno de las formas.
- **drawBitmap()**: Dibujar bitmaps.

Ejemplo de Métodos de Dibujo

```
@Override
protected void onDraw(Canvas canvas) {
   super.onDraw(canvas);
   canvas.drawCircle(mWidth / 2, mHeight / 2, mRadius, mDialPaint);
   canvas.drawText(Integer.toString(i), x, y, mTextPaint);
}
```

Cálculo del Tamaño

Calcular el tamaño en onSizeChanged(), que se llama cuando la vista personalizada aparece por primera vez y cuando cambia de tamaño:

```
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
   mWidth = w;
   mHeight = h;
   mRadius = (float) (Math.min(mWidth, mHeight) / 2 * 0.8);
}
```

Redibujar después de la Interacción del Usuario

Para forzar un redibujado después de la interacción del usuario, usa invalidate() para llamar a onDraw() nuevamente:

```
setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {
      invalidate();
   }
});
```

Uso de la Vista Personalizada en un Diseño

Añadir la Vista Personalizada a la UI

Agrega la vista personalizada al diseño XML de la actividad. Controla la apariencia con los atributos heredados de la subclase:

```
<com.example.customfancontroller.DialView
   android:id="@+id/dialView"
   android:layout_width="@dimen/dial_width"
   android:layout_height="@dimen/dial_height"
   android:layout_marginTop="@dimen/standard_margin"
   android:layout_marginRight="@dimen/standard_margin"/>
```

Uso de Atributos de Subclase de View

Ejemplo de atributos XML para una clase extendida de EditText:

```
<com.example.android.customedittext.EditTextWithClear
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:inputType="textCapSentences"
   android:textAppearance="@style/Base.TextAppearance.AppCompat.Display1"
   android:hint="@string/last_name"/>
```

Uso de Atributos Personalizados

Creación de Atributos Personalizados

Define un atributo y su tipo en XML, establece el valor del atributo y recupéralo en el constructor de la vista.

Declarar la Vista Personalizada como Styleable

Define el nombre y tipo en <declare-styleable> en res/values/attrs.xml:

```
<resources>
   <declare-styleable name="DialView">
      <attr name="fanOnColor" format="reference|color" />
      <attr name="fanOffColor" format="reference|color" />
   </declare-styleable>
</resources>
```

Añadir Atributos Personalizados al Diseño XML

Especifica valores para los atributos personalizados en el diseño XML:

```
<com.example.customfancontroller.DialView
   android:id="@+id/dialView"
   app:fanOffColor="@color/yellow1"
   app:fanOnColor="@color/cyan1"/>
```

Obtener Atributos Personalizados en el Constructor de la Vista

Los atributos del bundle de recursos se pasan al constructor de la vista. Usa obtainStyledAttributes() para obtener un TypedArray de valores de atributos:

```
public DialView(Context context, AttributeSet attrs) {
   super(context, attrs);
```

```
    TypedArray typedArray = context.obtainStyledAttributes(attrs, R.styleable.DialView, 0, 0);
    mFanOnColor = typedArray.getColor(R.styleable.DialView_fanOnColor, defaultFanOnColor);
    mFanOffColor = typedArray.getColor(R.styleable.DialView_fanOffColor, defaultFanOffColor);
    typedArray.recycle();
}
```

Uso de Atributos Personalizados en la Vista

Usa los valores de atributos recuperados en la vista personalizada:

```
if (fanSelection >= 1) {
    mDialPaint.setColor(mFanOnColor);
} else {
    mDialPaint.setColor(mFanOffColor);
}
```

Métodos Set y Get para Atributos Personalizados

Para permitir un comportamiento dinámico, crea métodos set y get para cada atributo personalizado:

```
public int getFanOnColor() {
    return mFanOnColor;
}

public void setFanOnColor(int color) {
    mFanOnColor = color;
}
```

Redibujar después de Cambios en Propiedades

Usa invalidate() después de cualquier cambio en propiedades para redibujar la vista. Si el cambio afecta el tamaño o la forma, usa requestLayout():

```
int newFanOnColor = getColorFromUser(...);
setFanOnColor(newFanOnColor);
invalidate(); // Forzar redibujado
```

7. Canvas

7.1 Uso de la Clase Canvas en Android

Introducción a Canvas

¿Por qué usar un Canvas?

El uso de un Canvas permite realizar dibujos más complejos de lo que es posible con vistas predefinidas. Es útil cuando tu aplicación necesita redibujar regularmente, por ejemplo, en animaciones o gráficos interactivos.

La Clase Canvas

La clase Canvas actúa como una superficie de dibujo lógica para gráficos 2D. Se utiliza en el método onDraw() que se ejecuta en el hilo de la UI. El recorte define las porciones visibles al usuario. Es importante monitorear el rendimiento de las operaciones de dibujo utilizando la herramienta de perfilado de renderizado de GPU.

Clases Requeridas para Dibujar

Para dibujar en Android, necesitas:

- **View**: Muestra el Bitmap.
- **Bitmap**: Superficie física de dibujo.
- **Canvas**: Proporciona la API para dibujar en el Bitmap.
- **Paint**: Estiliza lo que dibujas.

Dos Maneras de Dibujar en Canvas

1. **ImageView**: Dibujar en el Canvas en el manejador onClick. El dibujo puede cambiar cuando el usuario toca el ImageView.
2. **Custom View**: Dibujar en el Canvas en el método onDraw(). Permite una interacción más compleja con el usuario.

Dibujo Sencillo en Respuesta a la Acción del Usuario

Uso de un Canvas en un ImageView

Puedes usar un ImageView para dibujar en un Canvas definiendo el manejador onClick del ImageView para dibujar en el Canvas. Necesitas:

- **ImageView**

- **Bitmap**
- **Canvas**
- **Paint**

Ejemplo de Creación de ImageView en XML

```
<ImageView
   android:id="@+id/myimageview"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:onClick="drawSomething"/>
```

Crear un Objeto Paint

El objeto Paint almacena cómo dibujar: color, estilo, grosor de línea, tamaño de texto. Este objeto puede persistir fuera del manejador onClick.

```
private Paint mPaint = new Paint();
private Paint mPaintText = new Paint(Paint.UNDERLINE_TEXT_FLAG);

mPaint.setColor(Color.RED);
mPaintText.setColor(Color.BLUE);
mPaintText.setTextSize(70);
```

Manejador onClick en el ImageView

Define el manejador onClick para dibujar en el Canvas:

```
mBitmap = Bitmap.createBitmap(vWidth, vHeight, Bitmap.Config.ARGB_8888);
mImageView.setImageBitmap(mBitmap);
mCanvas = new Canvas(mBitmap);
mCanvas.drawColor(mColorBackground);
mCanvas.drawText(getString(R.string.my_string), 100, 100, mPaintText);
view.invalidate();
```

Uso de Canvas en una Vista Personalizada

Uso de una Vista Personalizada

Las vistas personalizadas permiten que el dibujo cambie en respuesta a interacciones de usuario más complejas. Puedes subclasificar una de las clases de View, sobrescribir onDraw() y onSizeChanged() para dibujar, y onTouchEvent() para manejar los toques del usuario.

Ejemplo de Uso

Sobrescribir onDraw()

```
@Override
protected void onDraw(Canvas canvas) {
   super.onDraw(canvas);
   canvas.drawBitmap(mBitmap, 0, 0, mPaint);
   canvas.drawPath(mPath, mPaint);
}
```

Sobrescribir onTouchEvent()

```
switch (event.getAction()) {
   case MotionEvent.ACTION_DOWN:
      touchStart(x, y);
      break;
   case MotionEvent.ACTION_MOVE:
      touchMove(x, y);
      invalidate();
      break;
   case MotionEvent.ACTION_UP:
      touchUp();
      invalidate();
      break;
}
```

Operaciones con Canvas

¿Qué se Puede Hacer en un Canvas?

- **Rellenar el Canvas con Color**: mCanvas.drawColor(mColorBackground);
- **Dibujar Formas Primitivas**: Rectángulos, óvalos, arcos.
- **Dibujar Formas Complejas**: Usar la clase Path.
- **Dibujar Texto**: Usar drawText() y estilizar con setTypeface(), setColor().
- **Aplicar Transformaciones**: Traslación, rotación, sesgado (skew).

Ejemplo de Dibujar Texto con Transformaciones

```
mPaint.setTextSize(120);
canvas.translate(100, 1800);
canvas.skew(0.2f, 0.3f);
canvas.drawText("Transformed", 400, 60, mPaint);
```

Recortes (Clipping)

¿Qué es el Recorte?

El recorte (clipping) es una manera de definir regiones de una imagen, lienzo o bitmap que se dibujan o no en la pantalla. Ayuda a reducir el redibujado excesivo y mejora el rendimiento.

Ejemplo de Uso de clipRect()

```
canvas.clipRect(x, y, right, bottom);
```

Ejemplo de Uso de clipPath()

```
mPath.addCircle(radius, x, y, Path.Direction.CCW);
canvas.clipPath(mPath, Region.Op.DIFFERENCE);
```

Guardar y Restaurar el Estado del Canvas

Guardar y Restaurar

El contexto de una actividad mantiene una pila de estados de dibujo, que incluye las transformaciones y regiones de recorte actualmente aplicadas. No puedes eliminar las regiones de recorte, y deshacer una transformación invirtiéndola es propenso a errores.

Ejemplo de Uso de save() y restore()

```
canvas.save();
mPaint.setTextSize(120);
canvas.translate(100, 1800);
canvas.skew(0.2f, 0.3f);
canvas.drawText("Skewing", 400, 60, mPaint);
canvas.restore();

canvas.save();
mPaint.setColor(Color.CYAN);
canvas.translate(600, 1800);
canvas.drawText("Save/Restore", 400, 60, mPaint);
canvas.restore();
```

Resumen

Resumen de Canvas

Para dibujar en Android necesitas una View, un Canvas, un Paint y un Bitmap. El Bitmap es la superficie física de dibujo, el Canvas proporciona la API para dibujar en el Bitmap, el Paint estiliza lo que dibujas y la View muestra el Bitmap.

- **Clipping**: Define regiones visibles.
- **Transformaciones**: Aplica traslación, rotación, y sesgado.
- **Guardar y Restaurar**: Maneja estados de dibujo de manera eficiente.

7.2 La Clase SurfaceView en Android

Visión General de SurfaceView

¿Qué es SurfaceView?

SurfaceView es una clase en Android que permite dibujar en una superficie separada del hilo principal de la UI. Esto es especialmente útil para realizar operaciones de dibujo intensivas que podrían afectar el rendimiento de la UI si se ejecutan en el hilo principal.

Características Clave de SurfaceView

- **Superficie Separada**: SurfaceView proporciona una superficie de dibujo separada que se renderiza detrás de la superficie de la aplicación.
- **Dibujo Fuera del Hilo de UI**: Permite dibujar en la superficie desde un hilo separado, reduciendo el impacto en el rendimiento del hilo de UI.
- **Combinación de Superficies**: El sistema Android combina y renderiza todas las superficies.

Trabajando con SurfaceView

Pasos Generales para Usar SurfaceView

1. **Crear una Clase de Vista Personalizada**: Extiende SurfaceView e implementa Runnable.
2. **Obtener SurfaceHolder**: Controla y monitorea la superficie a través del SurfaceHolder.
3. **Verificar la Disponibilidad de la Superficie**: Implementa los métodos surfaceCreated() y surfaceDestroyed() para manejar la disponibilidad de la superficie.
4. **Implementar el Método run() del Hilo**: Realiza las operaciones de dibujo en este método.
5. **Añadir Métodos de Pausa y Reanudación**: Maneja el ciclo de vida del hilo de dibujo.

Creación de una Vista Personalizada

Ejemplo de Creación de una Clase que Extiende SurfaceView

```
public class GameView extends SurfaceView implements Runnable {
   private SurfaceHolder mSurfaceHolder;
   private boolean mRunning;
   private Thread mGameThread;

   public GameView(Context context) {
      super(context);
      init();
   }

   public GameView(Context context, AttributeSet attrs) {
      super(context, attrs);
      init();
   }

   public GameView(Context context, AttributeSet attrs, int defStyleAttr) {
      super(context, attrs, defStyleAttr);
      init();
   }

   private void init() {
      mSurfaceHolder = getHolder();
   }

   @Override
   public void run() {
      while (mRunning) {
         if (mSurfaceHolder.getSurface().isValid()) {
            Canvas canvas = mSurfaceHolder.lockCanvas();
            // Realizar operaciones de dibujo aquí
            mSurfaceHolder.unlockCanvasAndPost(canvas);
         }
      }
   }

   public void pause() {
      mRunning = false;
      try {
         mGameThread.join();
      } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }

    public void resume() {
        mRunning = true;
        mGameThread = new Thread(this);
        mGameThread.start();
    }
}
```

Dibujo en SurfaceView

Bloqueo y Desbloqueo del Canvas

Para dibujar en un SurfaceView, debes bloquear el Canvas, realizar las operaciones de dibujo y luego desbloquearlo y publicar el contenido.

Ejemplo de Bloqueo y Desbloqueo del Canvas

```
if (mSurfaceHolder.getSurface().isValid()) {
    Canvas canvas = mSurfaceHolder.lockCanvas();
    // Realizar operaciones de dibujo aquí
    mSurfaceHolder.unlockCanvasAndPost(canvas);
}
```

Operaciones de Dibujo

Puedes realizar varias operaciones de dibujo en el Canvas, como dibujar formas, texto e imágenes. Utiliza el objeto Paint para estilizar lo que dibujas.

Ejemplo de Dibujo en el Canvas

```
@Override
public void run() {
    while (mRunning) {
        if (mSurfaceHolder.getSurface().isValid()) {
            Canvas canvas = mSurfaceHolder.lockCanvas();
            canvas.drawColor(Color.BLACK); // Limpiar el Canvas con color negro
            Paint paint = new Paint();
            paint.setColor(Color.WHITE);
            paint.setTextSize(50);
            canvas.drawText("Dibujando en SurfaceView", 100, 100, paint);
            mSurfaceHolder.unlockCanvasAndPost(canvas);
```

```
    }
  }
}
```

Manejo de la Interacción del Usuario

Manejo de Eventos de Toque

Para manejar la interacción del usuario, como toques en la pantalla, sobrescribe el método onTouchEvent().

Ejemplo de Manejo de Eventos de Toque

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
      // Manejar el toque
      break;
    case MotionEvent.ACTION_MOVE:
      // Manejar el movimiento
      break;
    case MotionEvent.ACTION_UP:
      // Manejar el levantamiento del dedo
      break;
  }
  return true;
}
```

Ejemplo Completo de Dibujo y Manejo de Interacción

Combina el dibujo y la interacción del usuario para crear una experiencia interactiva.

```
public class GameView extends SurfaceView implements Runnable {
  private SurfaceHolder mSurfaceHolder;
  private boolean mRunning;
  private Thread mGameThread;
  private Paint mPaint;

  public GameView(Context context) {
    super(context);
    init();
  }

  private void init() {
```

```
    mSurfaceHolder = getHolder();
    mPaint = new Paint();
    mPaint.setColor(Color.WHITE);
    mPaint.setTextSize(50);
}

@Override
public void run() {
    while (mRunning) {
        if (mSurfaceHolder.getSurface().isValid()) {
            Canvas canvas = mSurfaceHolder.lockCanvas();
            canvas.drawColor(Color.BLACK); // Limpiar el Canvas con color negro
            canvas.drawText("Dibujando en SurfaceView", 100, 100, mPaint);
            mSurfaceHolder.unlockCanvasAndPost(canvas);
        }
    }
}

public void pause() {
    mRunning = false;
    try {
        mGameThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

public void resume() {
    mRunning = true;
    mGameThread = new Thread(this);
    mGameThread.start();
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            // Manejar el toque
            break;
        case MotionEvent.ACTION_MOVE:
            // Manejar el movimiento
            break;
        case MotionEvent.ACTION_UP:
            // Manejar el levantamiento del dedo
            break;
    }
```

```
        return true;
    }
}
```

Ejemplo de Código

Implementación Completa de una Clase SurfaceView

```
public class CustomSurfaceView extends SurfaceView implements SurfaceHolder.Callback, Runnable {
    private SurfaceHolder mSurfaceHolder;
    private Thread mThread;
    private boolean mRunning;
    private Paint mPaint;

    public CustomSurfaceView(Context context) {
        super(context);
        init();
    }

    private void init() {
        mSurfaceHolder = getHolder();
        mSurfaceHolder.addCallback(this);
        mPaint = new Paint();
        mPaint.setColor(Color.RED);
        mPaint.setStyle(Paint.Style.FILL);
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        mRunning = true;
        mThread = new Thread(this);
        mThread.start();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        mRunning = false;
        try {
            mThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }
    }

    @Override
    public void run() {
        while (mRunning) {
            if (mSurfaceHolder.getSurface().isValid()) {
                Canvas canvas = mSurfaceHolder.lockCanvas();
                canvas.drawColor(Color.BLACK);
                canvas.drawCircle(getWidth() / 2, getHeight() / 2, 100, mPaint);
                mSurfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            mPaint.setColor(mPaint.getColor() == Color.RED ? Color.BLUE : Color.RED);
            return true;
        }
        return super.onTouchEvent(event);
    }
}
```

8. Animaciones

8.1 Animaciones en Android

¿Qué es la Animación?

Definición de Animación

La animación es una técnica para crear la ilusión de un objeto en movimiento mostrando una serie de imágenes discretas que cambian con el tiempo.

Ejemplos de Animación

- **Flipbook**: Un libro con una imagen diferente en cada página que, al pasar rápidamente, parece moverse.
- **Claymation**: Un tipo de animación en stop-motion.
- **Animaciones de UI**: Como deslizar un elemento de una lista.
- **Juegos Móviles**: Con animaciones de personajes, entornos y UI.

Frames y Tasa de Frames

- **Frame**: Una imagen en una secuencia animada.
- **Tasa de Frames (Frame Rate)**: La velocidad a la que se muestran los frames.
- **Tasa de Refresco**: La frecuencia con la que el sistema Android redibuja la pantalla.

Relación entre Frame Rate y Tasa de Refresco

- Si la tasa de frames es más lenta que la tasa de refresco, la animación puede entrecortarse.
- Si es más rápida, la app desperdicia recursos.
- Idealmente, la tasa de frames debe coincidir con la tasa de refresco de la pantalla para animaciones suaves.

Gestión de la Tasa de Frames

Afortunadamente, el sistema Android gestiona automáticamente la tasa de frames en la mayoría de las situaciones, por lo que no es necesario gestionarla manualmente.

Tipos de Animación en Android

1. **Animación de Vistas**

2. **Animación de Propiedades**
3. **Animación de Drawable**
4. **Animación Basada en Física**

Animaciones de Vistas

Sistema de Animación de Vistas

El sistema de animación de vistas es un sistema más antiguo limitado a objetos de vista (View). Es relativamente fácil de configurar y ofrece capacidades suficientes para muchas necesidades de aplicaciones.

Especificaciones de la Animación de Vistas

- Modifica solo el lugar donde se dibuja la vista, no la vista en sí.
- Si animas un botón para que se mueva por la pantalla, la ubicación donde se puede tocar el botón no cambia.
- Requiere menos tiempo y código que otras animaciones como la animación de propiedades.

Animaciones de Propiedades

Sistema de Animación de Propiedades

Disponible desde Android 3.0 (API nivel 11), permite animar propiedades de cualquier objeto. Es extensible para animar propiedades de tipos personalizados y es mejor usarlo en lugar de la animación de vistas.

Características de la Animación de Propiedades

- Puedes animar casi cualquier cosa como el color, tamaño y posición.
- Cambia el valor de una propiedad durante un período de tiempo especificado.
- Asigna animadores a propiedades para animarlas.
- Define aspectos de la animación como duración o interpolación.

Ejemplo: Animar un círculo para que crezca aumentando su radio con el tiempo

```
ObjectAnimator circleAnimator = ObjectAnimator.ofFloat(circleView, "radius", 0f, 100f);
circleAnimator.setDuration(1000); // Duración de 1 segundo
```

```
circleAnimator.start();
```

Propiedad a Animar

Puedes animar cualquier cosa cuyo valor pueda establecerse en un método "setter". La propiedad no tiene que ser un atributo existente, pero debe tener un método "setter".

Ejemplo de Animación de una Propiedad Existente

Para animar el tamaño del texto de una vista de texto:

```
TextView textView = findViewById(R.id.text_view);
ObjectAnimator textSizeAnimator = ObjectAnimator.ofFloat(textView, "textSize", 12f, 24f);
textSizeAnimator.setDuration(500); // Duración de 500 ms
textSizeAnimator.start();
```

Definición de las Características de la Animación

El sistema de animación de propiedades te permite definir características como:

- **Duración**: Cuánto tiempo dura una animación (por defecto 300 milisegundos).
- **Interpolación Temporal**: Especifica cómo se calculan los valores de la propiedad en función del tiempo transcurrido.
- **Repetición**: Especifica si una animación se repite al final de la duración, cuántas veces y cómo (reversa, ida y vuelta).

Duración

Define cuánto tiempo debe durar una animación. La duración predeterminada es de 300 milisegundos, pero puede ajustarse según las necesidades de la animación.

Interpolación Temporal

Especifica cómo se calculan los valores para la propiedad animada a lo largo del tiempo de la animación. Los valores pueden cambiar de manera lineal o acelerada/desacelerada.

Ejemplo de Interpolación

```
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "translationX", 0f, 100f);
animator.setInterpolator(new AccelerateDecelerateInterpolator());
animator.setDuration(500);
animator.start();
```

Repetición

Permite especificar cuántas veces se repetirá la animación y el comportamiento de repetición, como reproducir en reversa o alternar entre ida y vuelta.

Interpoladores

¿Qué es un Interpolador Temporal?

Define cómo cambian los valores en una animación con el tiempo. Puedes usar interpoladores lineales (cambio uniforme) o no lineales (aceleración/desaceleración).

Interpoladores Predefinidos

- **LinearInterpolator**: Cambio constante.
- **AccelerateDecelerateInterpolator**: Cambio lento al inicio y al final, acelerando en el medio.
- **AnticipateInterpolator**: El cambio comienza hacia atrás y luego avanza rápidamente, como una banda elástica.

Ejemplo de Uso de un Interpolador

```
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "translationX", 0f, 100f);
animator.setInterpolator(new AccelerateDecelerateInterpolator());
animator.setDuration(500);
animator.start();
```

Creación de un Interpolador Personalizado

Puedes crear tus propios interpoladores implementando la interfaz TimeInterpolator.

Ejemplo de Interpolador Personalizado

```
public class CustomInterpolator implements TimeInterpolator {
   @Override
   public float getInterpolation(float input) {
      return input * input; // Ejemplo simple de interpolación cuadrática
   }
}

// Uso del interpolador personalizado
```

```
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "translationX", 0f, 100f);
animator.setInterpolator(new CustomInterpolator());
animator.setDuration(500);
animator.start();
```

ObjectAnimator

Creación de Instancias de ObjectAnimator

ObjectAnimator tiene métodos de fábrica para crear instancias que animan de diferentes maneras:

- ofArgb(): Anima entre valores de color.
- ofFloat(): Anima entre valores flotantes.
- ofInt(): Anima entre valores enteros.

Ejemplo de Creación de un ObjectAnimator

```
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "alpha", 0f, 1f);
animator.setDuration(300);
animator.start();
```

Configuración de ObjectAnimator

1. **Crear instancia**: Usa un método de fábrica para crear el animador.
2. **Establecer interpolador**: Opcional, para definir cómo cambiarán los valores con el tiempo.
3. **Establecer duración**: Tiempo que durará la animación.
4. **Iniciar la animación**: Llama al método start().

Ejemplo Completo

```
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotation", 0f, 360f);
animator.setInterpolator(new AccelerateDecelerateInterpolator());
animator.setDuration(1000);
animator.start();
```

AnimatorSet

Uso de AnimatorSet

AnimatorSet permite agrupar animaciones en conjuntos lógicos que se ejecutan juntos, secuencialmente o después de ciertos retrasos.

Ejemplo de AnimatorSet

```
ObjectAnimator scaleX = ObjectAnimator.ofFloat(view, "scaleX", 1f, 2f);
ObjectAnimator scaleY = ObjectAnimator.ofFloat(view, "scaleY", 1f, 2f);
ObjectAnimator rotation = ObjectAnimator.ofFloat(view, "rotation", 0f, 360f);

AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(scaleX).with(scaleY).before(rotation);
animatorSet.setDuration(1000);
animatorSet.start();
```

Animaciones en XML

Definición de Animaciones en XML

Guarda las animaciones en el directorio res/animator/. Usa etiquetas como <objectAnimator> y <set> para definir animaciones.

Ejemplo de XML

```
<set android:ordering="sequentially">
   <objectAnimator android:propertyName="x" android:duration="500" android:valueTo="400"
android:valueType="intType"/>
   <objectAnimator android:propertyName="y" android:duration="500" android:valueTo="300"
android:valueType="intType"/>
   <objectAnimator android:propertyName="alpha" android:duration="500"
android:valueTo="1f"/>
</set>
```

Ejecución de Animaciones Definidas en XML

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(context,
R.anim.property_animator);
set.setTarget(view);
set.start();
```

Animaciones de Drawable

Animación de Drawable

Las animaciones de drawable muestran recursos Drawable uno tras otro, como una película.

Creación de Animaciones de Drawable desde XML

Guarda el archivo en res/drawable/ con el elemento <animation-list> como nodo raíz.

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
android:oneshot="true">
   <item android:drawable="@drawable/frame1" android:duration="200"/>
   <item android:drawable="@drawable/frame2" android:duration="200"/>
   <item android:drawable="@drawable/frame3" android:duration="200"/>
</animation-list>
```

Ejemplo de Uso de AnimationDrawable

```
ImageView imageView = findViewById(R.id.image_view);
imageView.setBackgroundResource(R.drawable.rocket_thrust);
AnimationDrawable rocketAnimation = (AnimationDrawable) imageView.getBackground();
rocketAnimation.start();
```

Ejemplo Completo de Animación de Drawable

```
public class MainActivity extends AppCompatActivity {
   private AnimationDrawable rocketAnimation;

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

      ImageView rocketImage = findViewById(R.id.rocket_image);
      rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
      rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
   }

   @Override
   public boolean onTouchEvent(MotionEvent event) {
      if (event.getAction() == MotionEvent.ACTION_DOWN) {
         rocketAnimation.start();
         return true;
      }
      return super.onTouchEvent(event);
   }
}
```

Animaciones Basadas en Física

¿Qué es una Animación Basada en Física?

Las animaciones basadas en física se apoyan en las leyes de la física para lograr un alto grado de realismo en la animación. Utilizan los fundamentos de la física para construir animaciones.

Beneficios de Animaciones Basadas en Física

- **Aspecto Natural**: Imitan movimientos en tiempo real.
- **Corrección de Curso**: Mantienen el impulso cuando el objetivo cambia.
- **Reducción de Disrupciones Visuales**: Son más fluidas y aparentan ser más receptivas.

Ejemplo de Spring Animation

```
final SpringAnimation anim = new SpringAnimation(view, DynamicAnimation.TRANSLATION_Y,
0);
anim.setStartVelocity(10000);
anim.getSpring().setStiffness(SpringForce.STIFFNESS_LOW);
anim.start();
```

Ejemplo de Fling Animation

```
FlingAnimation fling = new FlingAnimation(view, DynamicAnimation.ROTATION_X);
fling.setStartVelocity(150).setMinValue(0).setMaxValue(1000).setFriction(0.1f).start();
```

Uso de la API de Animaciones Basadas en Física

La biblioteca android.support.animation fue introducida en la versión 25.3.0 de la biblioteca de soporte e incluye clases para animaciones basadas en física.

Ejemplo de Configuración

```
dependencies {
   implementation "com.android.support:support-dynamic-animation:25.3.0"
}
```

9. Cómo reproducir video

9.1 Reproducción de Medios con VideoView en Android

Formatos y Fuentes de Medios

Ubicación de los Medios Reproducibles

Los medios que pueden ser reproducidos en una aplicación Android pueden estar:

- **Incrustados en la aplicación**: Medios incluidos como recursos.
- **En el almacenamiento externo del dispositivo**: Como una tarjeta SD.
- **Transmitidos por internet**: Streaming desde un servidor remoto.

Formatos de Medios

Diferentes reproductores de medios soportan diferentes formatos. Existen dos tipos principales de formatos:

1. **Formato de muestra**: Codificación del medio (codec).
2. **Formato de contenedor**: Medio + metadatos. La extensión indica el contenedor, por ejemplo, .mp3.

Ejemplos de Formatos de Audio

Formato de Muestra	Formato de Contenedor
MP3	.mp3
AAC	.aac
FLAC	.flac
PCM/WAVE	.wav

Ejemplos de Formatos de Video

Formato de Muestra	Formato de Contenedor
H.263, H.264 AVC	.mp4, .3gp
MPEG-4	.3gp
VP8, VP9	.webm, .mkv

Otras Consideraciones

- **Adaptive Streaming**: DASH, Smooth Streaming, HLS.
- **Gestión de Derechos Digitales (DRM)**: Widevine o PlayReady.

Para más información sobre formatos de medios soportados:

- **Supported Media Formats (Android platform)**
- **Supported formats (ExoPlayer)**

Reproductores y Controles de Medios

Reproductores de Medios

Un reproductor de medios toma una fuente de medios como entrada, la decodifica y la renderiza como audio o video. Puede o no tener una vista asociada para mostrar video.

Opciones de Reproductores de Medios

- **MediaPlayer**: Sencillo, limitado, parte de la plataforma Android.
- **VideoView**: Envoltura para MediaPlayer para mostrar video.
- **ExoPlayer**: Mejor opción para la mayoría de aplicaciones multimedia.
- **YouTube Android Player API**: Para medios alojados en YouTube.
- **Reproductor personalizado**: Solo para casos de uso muy complejos.

Controles de Medios

También llamados controles de transporte. Incluyen:

- **Reproducir/pausar**
- **Avance rápido/retroceso**
- **Saltar adelante/atrás**
- **Barra de progreso o barra de búsqueda**

Opciones de Controles de Medios

- **MediaController**: Sencillo pero limitado.
- **PlayerControlView (ExoPlayer)**: Muy personalizable.
- **Crear tus propios controles en el diseño de la aplicación**.

Reproducción de Medios con Intents

Uso de Intents Implícitos

Para reproducir medios con intents implícitos, se utiliza Intent.ACTION_VIEW y la URI del medio a reproducir.

Ejemplo de Intent para Reproducir Medios

```
Intent mediaIntent = new Intent();
mediaIntent.setAction(Intent.ACTION_VIEW);
mediaIntent.setData(Uri.parse("https://www.youtube.com/watch?v=LBBqTd6uOd4"));

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

Ventajas y Desventajas de los Intents para Reproducir Medios

Ventajas:

- Fácil de usar para reproducir la mayoría de los medios.
- No necesitas implementar reproductores o controles.
- La forma más sencilla de reproducir videos de YouTube.

Desventajas:

- Requiere que haya una aplicación disponible en el dispositivo para manejar el intent.
- El usuario sale de tu aplicación para reproducir los medios.

Reproducción de Medios con VideoView

Clase VideoView

VideoView es la forma más sencilla de incrustar video en el diseño de una aplicación. Combina MediaPlayer para la reproducción y SurfaceView para la visualización.

Ejemplo de Diseño con VideoView

```
<VideoView
   android:id="@+id/videoview"
```

```
android:layout_width="0dp"
android:layout_height="0dp"
android:layout_margin="8dp"
app:layout_constraintDimensionRatio="4:3"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
```

Configuración de VideoView y Controles de Medios

En onCreate():

1. Obtener el objeto VideoView.
2. Crear un nuevo MediaController.
3. Conectar el MediaController al VideoView con setMediaController().
4. Conectar el VideoView al controlador con setMediaPlayer().

Ejemplo de Configuración en onCreate()

```
mVideoView = findViewById(R.id.videoview);
MediaController controller = new MediaController(this);
controller.setMediaPlayer(mVideoView);
mVideoView.setMediaController(controller);
```

Establecer la Fuente de Medios

Usa setVideoURI() con la URI del video a reproducir.

Ejemplo de Establecimiento de la Fuente de Medios

```
Uri videoUri = Uri.parse("android.resource://" + getPackageName() + "/raw/" + "videofile");
mVideoView.setVideoURI(videoUri);
```

Medios en Almacenamiento Externo

```
String fullPath = Environment.getExternalStorageDirectory() + "/" + "videofile.mp4";
File file = new File(fullPath);
Uri videoUri = Uri.fromFile(file);
mVideoView.setVideoURI(videoUri);
```

Medios Transmitidos desde Internet

```
String mediaName = "http://myserver.com/videofile.mp4";
Uri videoUri = Uri.parse(mediaName);
```

```
mVideoView.setVideoURI(videoUri);
```

Permisos Necesarios

- **Almacenamiento Externo**: <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
- **Internet**: <uses-permission android:name="android.permission.INTERNET" />

Control de la Reproducción

Métodos de VideoView para controlar la reproducción:

- start(): Iniciar la reproducción.
- pause(): Pausar la reproducción.
- stopPlayback(): Detener la reproducción y liberar los recursos.
- seekTo(int msec): Mover la posición de reproducción actual a la posición en milisegundos.
- getCurrentPosition(): Obtener la posición de reproducción actual en milisegundos.

Ejemplo de Control de la Reproducción

```
mVideoView.start();
mVideoView.pause();
mVideoView.stopPlayback();
mVideoView.seekTo(1000); // Mover a 1 segundo
int position = mVideoView.getCurrentPosition();
```

Ciclo de Vida de la Aplicación y VideoView

- Crear y conectar el controlador de medios en onCreate().
- Llamar a setVideoURI() en onStart().
- Llamar a stopPlayback() en onStop().
- Llamar a pause() en onPause() solo para SDK < 24.

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
   mVideoView.pause();
}
```

Preservar la Posición de Reproducción

VideoView no preserva la posición del reproductor a través de cambios en el ciclo de vida. Debes guardar la posición de reproducción en el estado de la instancia y restaurarla en onCreate().

Ejemplo de Preservación de la Posición de Reproducción

```
@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);
   outState.putInt("current_position", mVideoView.getCurrentPosition());
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   mCurrentPosition = savedInstanceState.getInt("current_position");
   mVideoView.seekTo(mCurrentPosition);
}
```

Listeners de Eventos de VideoView

Listeners de MediaPlayer para manejar diferentes eventos:

- **onPrepared()**: El medio está listo para reproducirse.
- **onCompletion()**: El medio ha terminado de reproducirse.
- **onInfo()**: Información o advertencia está disponible.
- **onError()**: Ha ocurrido un error.

Ejemplo de Listener onPrepared()

```
mVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
   @Override
   public void onPrepared(MediaPlayer mediaPlayer) {
      mBufferingTextView.setVisibility(View.INVISIBLE);

      if (mCurrentPosition > 0) {
         mVideoView.seekTo(mCurrentPosition);
      }

      mVideoView.start();
   }
});
```

Ejemplo de Listener onCompletion()

```
mVideoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
   @Override
   public void onCompletion(MediaPlayer mediaPlayer) {
```

```
        Toast.makeText(MainActivity.this, "Playback completed", Toast.LENGTH_SHORT).show();
        mVideoView.seekTo(1);
    }
});
```

Reproducción de Medios con ExoPlayer

ExoPlayer

ExoPlayer es un reproductor de medios de código abierto que ofrece más funcionalidades y flexibilidad que MediaPlayer.

Características de ExoPlayer

- Soporte para más tecnologías que MediaPlayer.
- Fácil de personalizar y extender.
- Recomendado para la mayoría de las aplicaciones multimedia.

Recursos de ExoPlayer

- **Guía del desarrollador de ExoPlayer**
- **Proyecto de ExoPlayer en GitHub**
- **Blog del desarrollador de ExoPlayer**

Arquitectura de Aplicaciones Multimedia en Android

Beneficios de la Arquitectura de Aplicaciones Multimedia

- Implementa mejores prácticas para aplicaciones multimedia complejas.
- Integra tus aplicaciones con la plataforma Android, incluyendo Android Auto y Android Wear.
- Habilita el uso de controles de hardware, como botones de auriculares.
- Mantiene el estado del reproductor entre múltiples aplicaciones.
- Proporciona una API de navegador para habilitar el descubrimiento de medios.
- Proporciona un servicio para la reproducción de música en segundo plano.

Recursos Adicionales

- **Visión general de aplicaciones multimedia**

10. Componentes de Arquitectura

10.1 Componentes de Arquitectura en Android

Componentes de Arquitectura

¿Qué son los Componentes de Arquitectura?

Los Componentes de Arquitectura son un conjunto de bibliotecas de Android diseñadas para ayudarte a estructurar tu aplicación de una manera robusta, comprobable y mantenible.

Características Clave

- **Buenas prácticas de arquitectura**: Facilitan la implementación de arquitecturas recomendadas.
- **Menos código repetitivo**: Reducen la cantidad de código boilerplate.
- **Testabilidad**: Separación clara de responsabilidades facilita las pruebas.
- **Mantenimiento**: Fewer dependencies hacen que el mantenimiento sea más fácil.

Ventajas

- **Robustez**: Implementaciones confiables y sostenibles.
- **Facilidad de pruebas**: Mejor estructura facilita pruebas unitarias y de integración.
- **Mantenibilidad**: Componentes desacoplados facilitan la actualización y el mantenimiento.

Overview

Estructura General

La arquitectura recomendada para aplicaciones Android incluye los siguientes componentes:

1. **UI Controller (Activity/Fragment)**: Muestra los datos y maneja los eventos de la UI.
2. **ViewModel**: Mantiene y maneja los datos necesarios para la UI.
3. **Repository**: Proporciona una API limpia para acceder a los datos.
4. **Room Database**: Base de datos que gestiona los datos locales.
5. **DAO (Data Access Object)**: Define las operaciones de acceso a la base de datos.
6. **Entity**: Define el esquema de la base de datos.
7. **LiveData**: Mantiene los datos actualizados y notifica a la UI sobre cambios.
8. **Lifecycle**: Maneja el ciclo de vida de los componentes.

Room

Overview de Room

Room es una biblioteca de mapeo de objetos SQL robusta que genera código de SQLite para Android. Proporciona una API simple para interactuar con la base de datos.

Componentes de Room

1. **Entity**: Define el esquema de la base de datos.
2. **DAO**: Define las operaciones de lectura/escritura en la base de datos.
3. **RoomDatabase**: Clase abstracta que extiende RoomDatabase y sirve como contenedor de la base de datos.

Creación de una Base de Datos Room

Definir una Entidad

```
@Entity(tableName = "person")
public class Person {
   @PrimaryKey(autoGenerate = true)
   private int uid;

   @ColumnInfo(name = "first_name")
   private String firstName;

   @ColumnInfo(name = "last_name")
   private String lastName;

   // Getters y setters
}
```

Definir un DAO

```
@Dao
public interface PersonDao {
   @Insert
   void insert(Person person);

   @Update
   void update(Person... persons);

   @Delete
```

```
    void delete(Person person);

    @Query("SELECT * FROM person")
    List<Person> getAll();
}
```

Definir la Base de Datos

```
@Database(entities = {Person.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract PersonDao personDao();
}
```

Crear la Instancia de la Base de Datos

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
    AppDatabase.class, "database-name").build();
```

Notas sobre Room

- **Verificaciones en tiempo de compilación**: Verifica las declaraciones SQL en tiempo de compilación.
- **Operaciones en hilos de fondo**: No ejecutar operaciones de base de datos en el hilo principal.
- **Singleton**: Usualmente, la base de datos Room debe ser una instancia singleton.

ViewModel

¿Qué es un ViewModel?

ViewModel es un objeto que proporciona datos a los componentes de la UI y sobrevive a los cambios de configuración, como la rotación del dispositivo.

Características Clave

- **Persistencia**: Los datos sobreviven a cambios de configuración.
- **Interfaz limpia**: Proporciona una interfaz clara entre la UI y los datos.

Ejemplo de ViewModel

```
public class PersonViewModel extends ViewModel {
    private MutableLiveData<List<Person>> persons;

    public LiveData<List<Person>> getPersons() {
```

```
        if (persons == null) {
            persons = new MutableLiveData<List<Person>>();
            loadPersons();
        }
        return persons;
    }

    private void loadPersons() {
        // Hacer la carga de datos de forma asíncrona
    }
}
```

Uso de ViewModel con Repositorio

```
public class PersonViewModel extends AndroidViewModel {
    private PersonRepository repository;
    private LiveData<List<Person>> allPersons;

    public PersonViewModel(Application application) {
        super(application);
        repository = new PersonRepository(application);
        allPersons = repository.getAllPersons();
    }

    public LiveData<List<Person>> getAllPersons() {
        return allPersons;
    }

    public void insert(Person person) {
        repository.insert(person);
    }
}
```

Repository

¿Qué es un Repository?

El Repository proporciona una API limpia para acceder a los datos. Maneja la lógica de datos y decide si debe obtener datos de una red o de una base de datos local.

Ejemplo de Repository

```
public class PersonRepository {
    private PersonDao personDao;
    private LiveData<List<Person>> allPersons;
```

```
    public PersonRepository(Application application) {
        AppDatabase db = AppDatabase.getDatabase(application);
        personDao = db.personDao();
        allPersons = personDao.getAll();
    }

    public LiveData<List<Person>> getAllPersons() {
        return allPersons;
    }

    public void insert(Person person) {
        new insertAsyncTask(personDao).execute(person);
    }

    private static class insertAsyncTask extends AsyncTask<Person, Void, Void> {
        private PersonDao asyncTaskDao;

        insertAsyncTask(PersonDao dao) {
            asyncTaskDao = dao;
        }

        @Override
        protected Void doInBackground(final Person... params) {
            asyncTaskDao.insert(params[0]);
            return null;
        }
    }
}
```

LiveData

¿Qué es LiveData?

LiveData es una clase de contenedor de datos que es consciente del ciclo de vida. Mantiene un valor y permite que este valor sea observado.

Características Clave

- **Observable**: Notifica a los observadores cuando los datos cambian.
- **Consciente del ciclo de vida**: Se actualiza automáticamente cuando la actividad o fragmento está en un estado activo.

Ejemplo de Uso de LiveData

Definir LiveData en DAO

```
@Query("SELECT * FROM person")
LiveData<List<Person>> getAllPersons();
```

Observar LiveData en la UI

```
personViewModel.getAllPersons().observe(this, new Observer<List<Person>>() {
   @Override
   public void onChanged(@Nullable final List<Person> persons) {
      // Actualizar la UI
   }
});
```

Lifecycle

Componentes Conscientes del Ciclo de Vida

Los componentes conscientes del ciclo de vida realizan acciones en respuesta a un cambio en el estado del ciclo de vida de otro componente.

Ejemplo de LifecycleObserver

```
public class MyObserver implements LifecycleObserver {
   @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
   public void onResume() {
      // Acción cuando el componente está en estado RESUME
   }

   @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
   public void onPause() {
      // Acción cuando el componente está en estado PAUSE
   }
}
```

Uso de LifecycleObserver

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

```
        getLifecycle().addObserver(new MyObserver());
    }
}
```

Beneficios de Lifecycle

- **Gestión Automática**: Los componentes gestionan automáticamente su comportamiento basado en el estado del ciclo de vida.
- **Responsabilidad Descentralizada**: En lugar de que una actividad o fragmento maneje todas las responsabilidades del ciclo de vida, los componentes individuales manejan sus propias responsabilidades.

11. Conclusiones

En esta sexta unidad didáctica, hemos explorado cómo agregar funciones geográficas a las aplicaciones Android, centrándonos en la integración de servicios de localización y la API de Google Maps. El uso de estas tecnologías es crucial en el desarrollo de aplicaciones que dependen de la ubicación del usuario, como aquellas relacionadas con la navegación, el transporte o los servicios de entrega.

Uno de los aspectos fundamentales abordados fue la configuración de **Google Play Services** y el manejo de los **permisos de localización**, lo que asegura que las aplicaciones puedan acceder a los servicios de ubicación de manera eficiente y con el consentimiento explícito del usuario. También aprendimos a utilizar el **FusedLocationProviderClient**, una herramienta que facilita la obtención de la ubicación del dispositivo de forma más precisa y eficiente en términos de consumo de batería.

Además, exploramos técnicas avanzadas como la **geocodificación** y la **geocodificación inversa**, que permiten convertir coordenadas en direcciones legibles y viceversa, añadiendo funcionalidad y precisión a las aplicaciones. Por otro lado, la implementación de la **API de Places** en Android ofrece la capacidad de acceder a información sobre negocios y lugares cercanos, mejorando significativamente la experiencia del usuario.

En resumen, la correcta implementación de funciones geográficas permite a los desarrolladores crear aplicaciones que no solo interactúan con el entorno físico del usuario, sino que también brindan una experiencia de usuario más rica, dinámica y personalizada. Estas habilidades serán esenciales para el desarrollo de aplicaciones móviles modernas.

12. Bibliografía

Android Developers. (2023). Location and Maps in Android. Google. Disponible en: https://developer.android.com/training/location

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

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

Nagpal, S. (2018). Mastering Android Location Services. Packt Publishing.

Firebase. (2023). Firebase Realtime Database and Location Services. Google. Disponible en: https://firebase.google.com/docs/database

Estas referencias ofrecen una base sólida para comprender e implementar funciones geográficas en aplicaciones Android, así como para dominar las herramientas y API relacionadas con la ubicación.

UAX Universidad Alfonso X el Sabio

G R A C I A S

UAX.COM