Vinculacíon de vistas

Contenido

1 Vinculacíon de vistas

La *vinculación de vista* es una función que te permite escribir más fácilmente código que interactúa con las vistas. Una vez que la vinculación de vista está habilitada en un módulo, genera una *clase de vinculación* para cada archivo de diseño XML presente en ese módulo. Una instancia de una clase de vinculación contiene referencias directas a todas las vistas que tienen un ID en el diseño correspondiente.
En la mayoría de los casos, la vinculación de vistas reemplaza a findViewById.
Instrucciones de configuración
Nota: La vinculación de vistas está disponible en Android Studio 3.6 Canary 11 y versiones posteriores.
La vinculación de vista se habilita módulo por módulo. Para habilitar la vinculación de vista en un módulo, agrega el elemento viewBinding a su archivo build.gradle, como se muestra en el siguiente ejemplo:

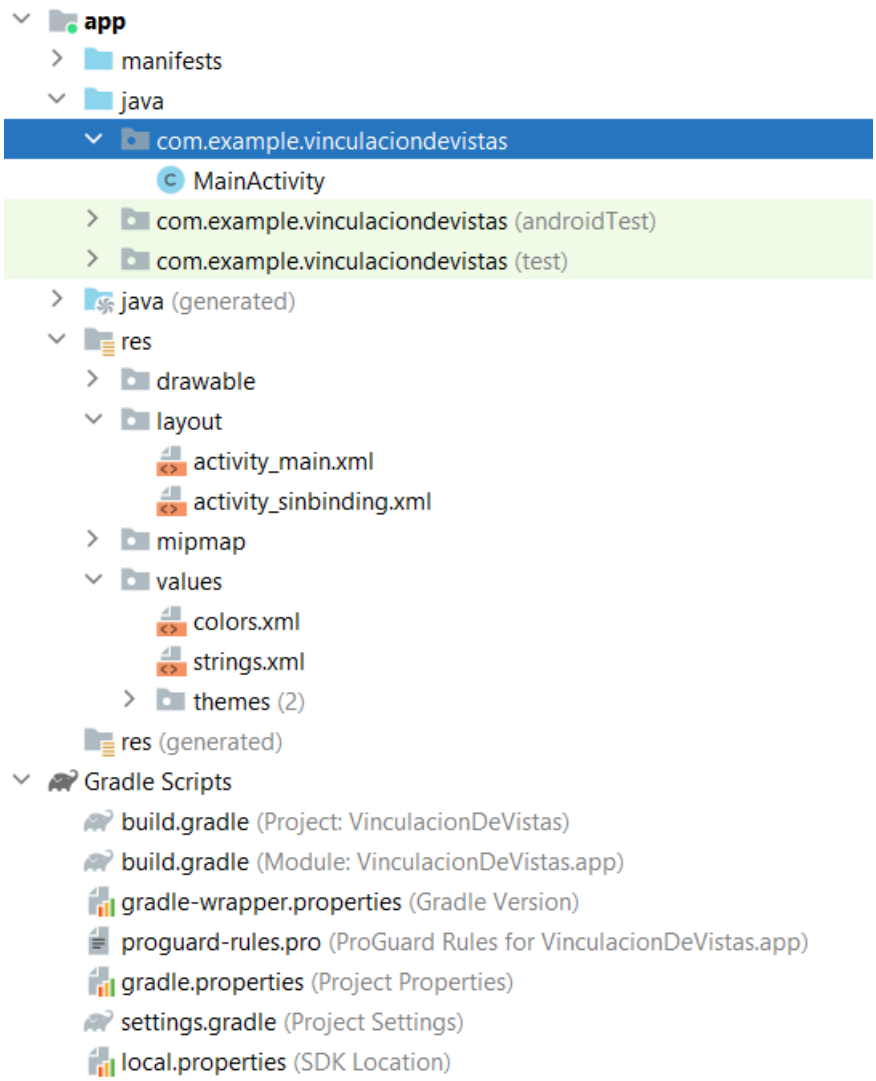
```
android {
        ...
        viewBinding {
            enabled = true
        }
    }
```

Si deseas que se ignore un archivo de diseño mientras se generan clases de vinculación, agrega el atributo tools:viewBindingIgnore="true" a la vista raíz de ese archivo:

```
<LinearLayout
            ...
            tools:viewBindingIgnore="true" >
        ...
    </ConstraintLayout>
```

2 Proyecto VinculacionDeVistas.

Lo vamos a ver sobre el ejemplo de proyecto VinculacionDeVistas que encontrareis en un zip VinculacionDeVistas.zip con la siguiente estructura.



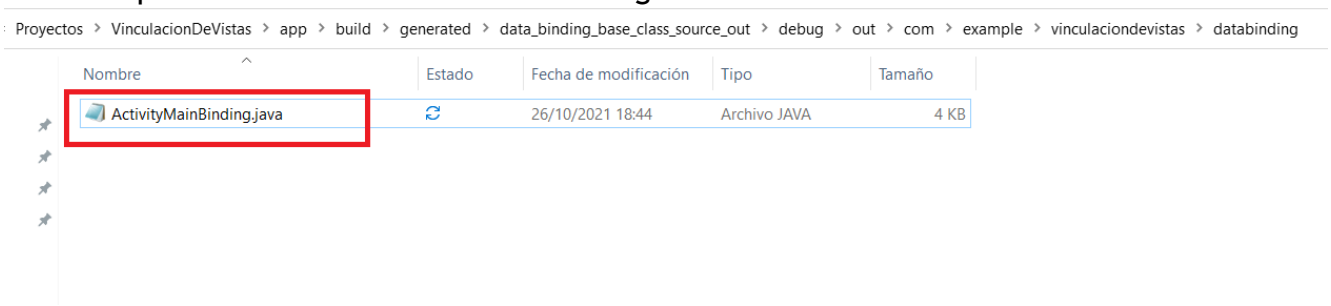
2.1 *Uso*

Si se habilita la vinculación de vista para un módulo, se genera una clase de vinculación para cada archivo de diseño XML que contiene el módulo. Cada clase de vinculación contiene referencias a la vista raíz y a todas las vistas que tienen un ID. El nombre de la clase de vinculación se genera convirtiendo el nombre del archivo XML según la convención de mayúsculas y minúsculas, y agregando la palabra "Binding" al final. En este caso `ActivityMainBinding.java`.

Encontrareis la clase en el directorio de archivos generados en la carpeta databinding dentro de vuestro proyecto, y vuestro paquete. En este caso:

VinculacionDeVistas\app\build\generated\data_binding_base_class_source_out\debug\out\c

om\example\vinculaciondevistas\databinding




```
private ActivityMainBinding binding;
```

Por ejemplo, en un archivo de diseño llamado activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guidelineStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintGuide_percent="0.04"
        android:orientation="vertical"/>

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guidelineEnd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintGuide_percent="0.96"
        android:orientation="vertical"/>
    <TextView
        android:id="@+id/textNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/Nombre"
        android:layout_marginTop="34dp"
        app:layout_constraintStart_toStartOf="@id/guidelineStart"
        app:layout_constraintTop_toTopOf="parent"
        />

    <EditText
        android:id="@+id/editTextNombre"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Introduce el nombre"
        app:layout_constraintStart_toStartOf="@id/guidelineStart"
        app:layout_constraintEnd_toEndOf="@id/guidelineEnd"
        app:layout_constraintTop_toBottomOf="@id/textNombre"
        android:autofillHints="name" />
    <TextView
```

```
        android:id="@+id/textApellidos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/Apellidos"
        app:layout_constraintTop_toBottomOf="@id/editTextNombre"
        app:layout_constraintStart_toStartOf="@id/guidelineStart"
        />
    <EditText
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="@id/guidelineEnd"
        app:layout_constraintStart_toStartOf="@id/guidelineStart"
        android:hint="Introduce el apellido"
        app:layout_constraintTop_toBottomOf="@id/textApellidos"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

La clase de vinculación generada se llama ResultProfileBinding. Esta clase tiene dos campos: un TextView llamado name y un Button llamado button. El campo ImageView del diseño no tiene ningún ID, por lo que no se hace referencia a él en la clase de vinculación.
Cada clase de vinculación también incluye un método getRoot(), que proporciona una referencia directa para la vista raíz del archivo de diseño correspondiente. En este ejemplo, el método getRoot() de la clase ResultProfileBinding muestra la vista raíz LinearLayout.
Las siguientes secciones demuestran el uso de las clases de vinculación generadas en actividades y fragmentos.

2.1.1 Cómo usar la vinculación de vista en actividades

A fin de configurar una instancia de la clase de vinculación para usarla con una actividad, realiza los siguientes pasos en el método onCreate() de la actividad:

1. Llama al método inflate() estático incluido en la clase de vinculación generada. Esto crea una instancia de la clase de vinculación para la actividad que se usará.
2. Para obtener una referencia a la vista raíz, llama al método getRoot()
3. Pasa la vista raíz a setContentView() para que sea la vista activa en la pantalla.

```
    private ActivityMainBinding binding;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
}
```

Ahora puedes usar la instancia de la clase de vinculación para hacer referencia a cualquiera de las vistas:

2.1.2 Cómo usar la vinculación de vista en fragmentos

Aunque no esta en el proyecto de ejemplo, lo dejamos. A fin de configurar una instancia de la clase de vinculación para su uso con un fragmento, realiza los siguientes pasos en el método onCreateView() del fragmento:

1. Llama al método inflate() estático incluido en la clase de vinculación generada. Esto crea una instancia de la clase de vinculación para que la use el fragmento.
2. Para obtener una referencia a la vista raíz, llama al método getRoot()
3. Muestra la vista raíz del método onCreateView() para convertirla en la vista activa de la pantalla.

Nota: El método **inflate()** requiere que transfieras un inflador de diseño. Si el diseño ya está inflado, puedes llamar al método estático **bind()** de la clase de vinculación.

```
private ResultProfileBinding binding;

@Override
public View onCreateView (LayoutInflater inflater,
                          ViewGroup container,
                          Bundle savedInstanceState) {
   binding = ResultProfileBinding.inflate(inflater, container, false);
   View view = binding.getRoot();
   return view;
}

@Override
public void onDestroyView() {
   super.onDestroyView();
   binding = null;
}
```

Nota: Los fragmentos sobreviven a sus vistas. Asegúrate de borrar las referencias a la instancia de clase de vinculación que se encuentran en el método **onDestroyView()** del fragmento.

2.1.3 Diferencias de findViewById

La vinculación de vistas tiene ventajas importantes frente al uso de findViewById:

- **Seguridad nula:** Debido a que la vinculación de vista crea referencias directas a las vistas, no hay riesgo de una excepción de puntero nulo debido a un ID de vista no válido. Además, cuando una vista solo está presente en algunas configuraciones de un diseño, el campo que contiene su referencia en la clase de vinculación se marca con @Nullable.
- **Seguridad de tipos:** Los campos de cada clase de vinculación tienen tipos que coinciden con las vistas a las que hacen referencia en el archivo XML. Esto significa que no hay riesgo de una excepción de transmisión de clase.

Estas diferencias significan que las incompatibilidades entre tu diseño y tu código harán que falle la compilación durante el momento de compilación en lugar de hacerlo en el tiempo de ejecución.

Comparación con la vinculación de datos

La vinculación de vistas y **la vinculación de datos** generan clases de vinculación que puedes usar para hacer referencia a vistas directamente. Sin embargo, la vinculación de vistas está diseñada para procesar casos de uso más simples y proporciona los siguientes beneficios por sobre la vinculación de datos:

- **Compilación más rápida:** la vinculación de vistas no requiere procesamiento de anotaciones, por lo que los tiempos de compilación son más rápidos.
- **Facilidad de uso:** la vinculación de vistas no requiere archivos de diseño XML

etiquetados especialmente, por lo que es más rápido adoptarlos en tus apps. Una vez que habilites la vinculación de vista en un módulo, se aplicará automáticamente a todos los diseños de ese módulo.

En cambio, la vinculación de vista tiene las siguientes limitaciones en comparación con la vinculación de datos:

- La vinculación de vistas no admite variables ni expresiones de diseño, por lo que no se puede usar para declarar contenido de IU dinámico directamente desde archivos de diseño XML.
- La vinculación de vista no admite la vinculación de datos bidireccional.

Debido a estas consideraciones, en algunos casos es mejor usar la vinculación de vistas y la vinculación de datos en un proyecto. Puedes usar la vinculación de datos en diseños que requieren funciones avanzadas y usar la vinculación de vista en diseños que no lo requieren.

2.2 *La clase Autogenerada*

En la clase autogenerada se crea una propiedad por cada widget o componente que habéis añadido en vuestra ventana.

Se infla el layout asociado en el método estático public `inflate` que llamaremos desde nuestra actividad.

```
  @NonNull
  public static ActivityMainBinding inflate(@NonNull LayoutInflater inflater,
      @Nullable ViewGroup parent, boolean attachToParent) {
    View root = inflater.inflate(R.layout.activity_main, parent, false);
    if (attachToParent) {
      parent.addView(root);
    }
    return bind(root);
  }
```

Se busca cada widget con findChildViewById, como podéis ver:

```
EditText editTextNombre = ViewBindings.findChildViewById(rootView, id);
```

```
  @NonNull
  public static ActivityMainBinding bind(@NonNull View rootView) {
    // The body of this method is generated in a way you would not otherwise
write.
    // This is done to optimize the compiled bytecode for size and performance.
    int id;
    missingId: {
      id = R.id.editTextNombre;
      EditText editTextNombre = ViewBindings.findChildViewById(rootView, id);
      if (editTextNombre == null) {
```

```
        break missingId;
      }

      id = R.id.guidelineEnd;
      Guideline guidelineEnd = ViewBindings.findChildViewById(rootView, id);
      if (guidelineEnd == null) {
        break missingId;
      }

      id = R.id.guidelineStart;
      Guideline guidelineStart = ViewBindings.findChildViewById(rootView, id);
      if (guidelineStart == null) {
        break missingId;
      }
```

```
// Generated by view binder compiler. Do not edit!
package com.example.vinculaciondevistas.databinding;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.constraintlayout.widget.Guideline;
import androidx.viewbinding.ViewBinding;
import androidx.viewbinding.ViewBindings;
import com.example.vinculaciondevistas.R;
import java.lang.NullPointerException;
import java.lang.Override;
import java.lang.String;

public final class ActivityMainBinding implements ViewBinding {
  @NonNull
  private final ConstraintLayout rootView;

  @NonNull
  public final EditText editTextNombre;

  @NonNull
  public final Guideline guidelineEnd;

  @NonNull
  public final Guideline guidelineStart;

  @NonNull
  public final TextView textApellidos;

  @NonNull
  public final TextView textNombre;

  private ActivityMainBinding(@NonNull ConstraintLayout rootView, @NonNull
EditText editTextNombre,
      @NonNull Guideline guidelineEnd, @NonNull Guideline guidelineStart,
      @NonNull TextView textApellidos, @NonNull TextView textNombre) {
    this.rootView = rootView;
```

```
    this.editTextNombre = editTextNombre;
    this.guidelineEnd = guidelineEnd;
    this.guidelineStart = guidelineStart;
    this.textApellidos = textApellidos;
    this.textNombre = textNombre;
  }

  @Override
  @NonNull
  public ConstraintLayout getRoot() {
    return rootView;
  }

  @NonNull
  public static ActivityMainBinding inflate(@NonNull LayoutInflater inflater) {
    return inflate(inflater, null, false);
  }

  @NonNull
  public static ActivityMainBinding inflate(@NonNull LayoutInflater inflater,
      @Nullable ViewGroup parent, boolean attachToParent) {
    View root = inflater.inflate(R.layout.activity_main, parent, false);
    if (attachToParent) {
      parent.addView(root);
    }
    return bind(root);
  }

  @NonNull
  public static ActivityMainBinding bind(@NonNull View rootView) {
    // The body of this method is generated in a way you would not otherwise
write.
    // This is done to optimize the compiled bytecode for size and performance.
    int id;
    missingId: {
      id = R.id.editTextNombre;
      EditText editTextNombre = ViewBindings.findChildViewById(rootView, id);
      if (editTextNombre == null) {
        break missingId;
      }

      id = R.id.guidelineEnd;
      Guideline guidelineEnd = ViewBindings.findChildViewById(rootView, id);
      if (guidelineEnd == null) {
        break missingId;
      }

      id = R.id.guidelineStart;
      Guideline guidelineStart = ViewBindings.findChildViewById(rootView, id);
      if (guidelineStart == null) {
        break missingId;
      }

      id = R.id.textApellidos;
      TextView textApellidos = ViewBindings.findChildViewById(rootView, id);
      if (textApellidos == null) {
        break missingId;
      }

      id = R.id.textNombre;
      TextView textNombre = ViewBindings.findChildViewById(rootView, id);
      if (textNombre == null) {
```

```
        break missingId;
      }

      return new ActivityMainBinding((ConstraintLayout) rootView, editTextNombre,
guidelineEnd,
          guidelineStart, textApellidos, textNombre);
    }
    String missingId = rootView.getResources().getResourceName(id);
    throw new NullPointerException("Missing required view with ID:
".concat(missingId));
  }
}
```

2.3 *La clase ActivityMain*

En la clase ActivityMain hacemos uso de ActivityMainBinding. Recogemos el binding del método binding = ActivityMainBinding.*inflate*(getLayoutInflater()); Recordar que el layout ya esta inflado en la clase ActivityMAin. El método `binding.getRoot()` nos devuelve el la vista del Layout para cargarla en el setContentView.

```
package com.example.vinculaciondevistas;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import com.example.vinculaciondevistas.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
    }
}
```

2.4 *Layout sin binding.*

En el proyecto hay un segundo layout llamado activitysinbinding.xml, que se le ha desactivado el binding añadiendo la propiedad `tools:viewBindingIgnore="true"`. Comprobar como para este layout no se ha generado la clase de binding.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    tools:viewBindingIgnore="true" >

    <TextView
        android:id="@+id/textNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/SinBinding"
        android:layout_marginTop="34dp"
        android:textSize="40sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Nota Importante: Si hay binding cuidado por que hay que usarlo. Mi recomendación es que para los proyectos de clase no lo useis.