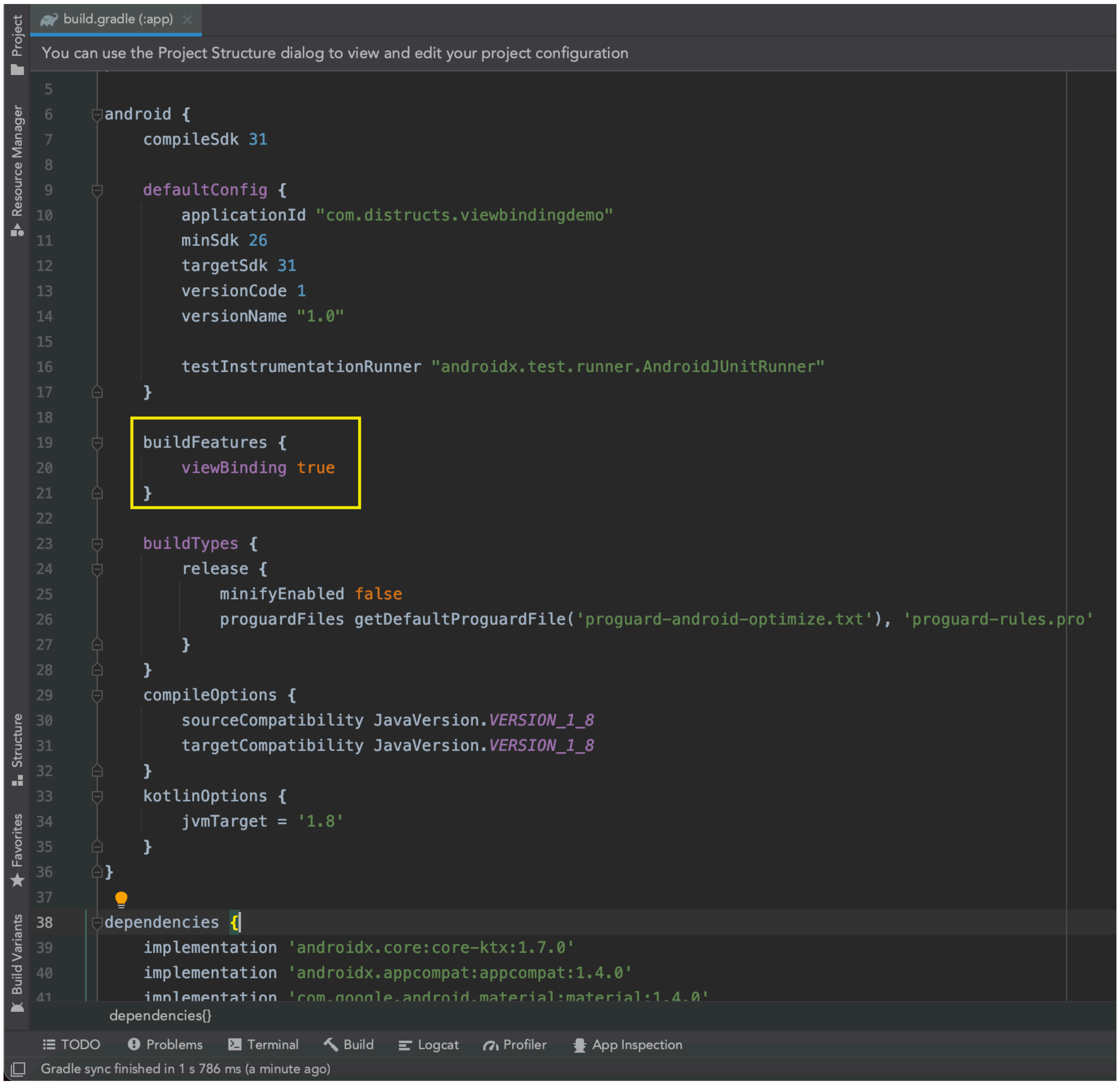
Setup instructions

Create a new project in Android Studio (API 21+, Kotlin 1.3+).
Set the view binding build option to true in the module-level build.gradle file and rebuild the project.



```
android {
    compileSdk 31

    defaultConfig {
        applicationId "com.distructs.viewbindingdemo"
        minSdk 26
        targetSdk 31
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildFeatures {
        viewBinding true
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'com.google.android.material:material:1.4.0'
```

Gradle sync finished in 1 s 786 ms (a minute ago)

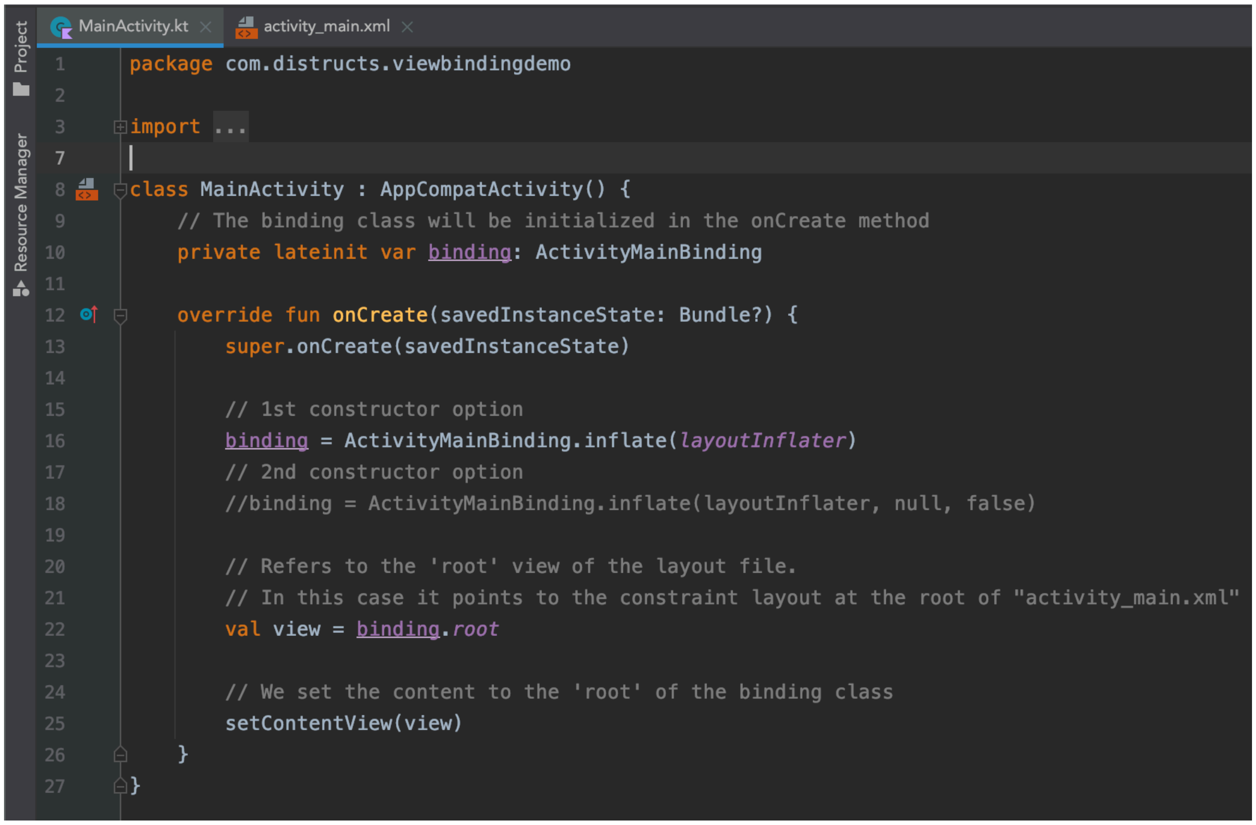
Usage

Once view binding is enabled for a module, a binding class is created for each XML layout file that the module contains
The name of the class is generated by converting the name of the XML file to Pascal case and adding the word “Binding” to the end. (e.g., A layout file named “result_profile.xml” would generate a binding class called “ResultProfileBinding”)

Using view binding in activities

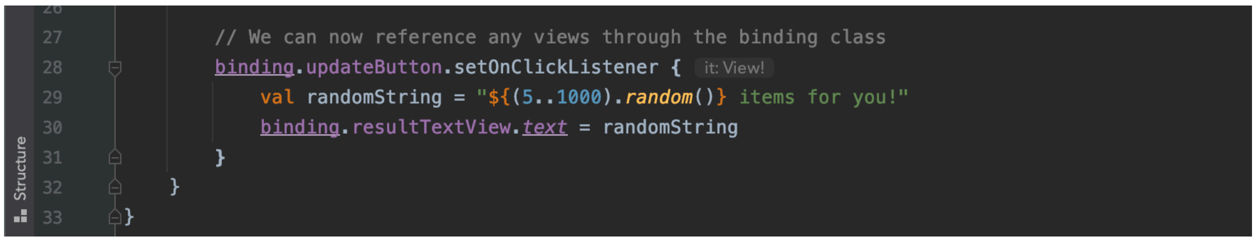
To set up an instance of the binding class for use in an activity, perform the following steps in the activity's onCreate() method:

1. Call the static inflate() method included in the generated binding class. This creates an instance of the binding class for the activity to use.
2. Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax.
3. Pass the root view to setContentView() to make it the active view on the screen.



```kotlin
package com.distructs.viewbindingdemo

import ...

class MainActivity : AppCompatActivity() {
    // The binding class will be initialized in the onCreate method
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // 1st constructor option
        binding = ActivityMainBinding.inflate(layoutInflater)
        // 2nd constructor option
        //binding = ActivityMainBinding.inflate(layoutInflater, null, false)

        // Refers to the 'root' view of the layout file.
        // In this case it points to the constraint layout at the root of "activity_main.xml"
        val view = binding.root

        // We set the content to the 'root' of the binding class
        setContentView(view)
    }
}
```

You can then use the binding instance to reference any views

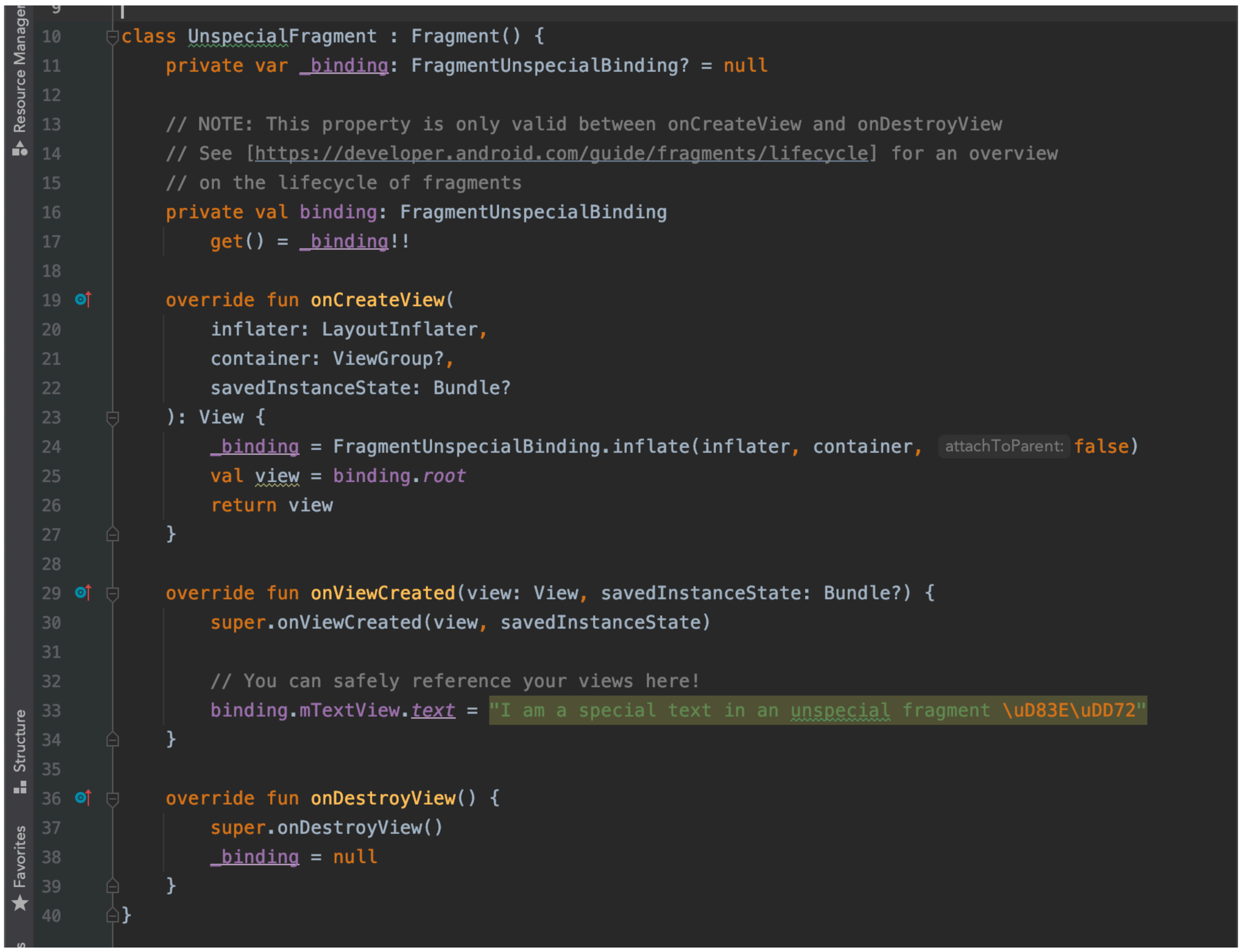


```kotlin
        // We can now reference any views through the binding class
        binding.updateButton.setOnClickListener { it: View!
            val randomString = "${(5..1000).random()} items for you!"
            binding.resultTextView.text = randomString
        }
    }
}
```

Using view binding in fragments

To set up an instance of the binding class for use with a fragment, perform the following steps in the fragments onCreateView() method:

1. Call the static inflate() method included in the generated binding class. This creates an instance of the binding class for the fragment to use.
2. Get a reference to the root view by either calling the getRoot() method or using Kotlin property syntax.
3. Return the root view from the onCreateView() method to make it the active view on the screen.
4. Fragments outlive their views so make sure you clean up any references to the binding class instance in the fragment's onDestroyView() method.



```kotlin
class UnspecialFragment : Fragment() {
    private var _binding: FragmentUnspecialBinding? = null

    // NOTE: This property is only valid between onCreateView and onDestroyView
    // See [https://developer.android.com/guide/fragments/lifecycle] for an overview
    // on the lifecycle of fragments
    private val binding: FragmentUnspecialBinding
        get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentUnspecialBinding.inflate(inflater, container,  attachToParent: false)
        val view = binding.root
        return view
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // You can safely reference your views here!
        binding.mTextView.text = "I am a special text in an unspecial fragment \uD83E\uDD72"
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```