

# Introduction to Android Developing Web Applications

Çağatay Sönmez

05.12.2019

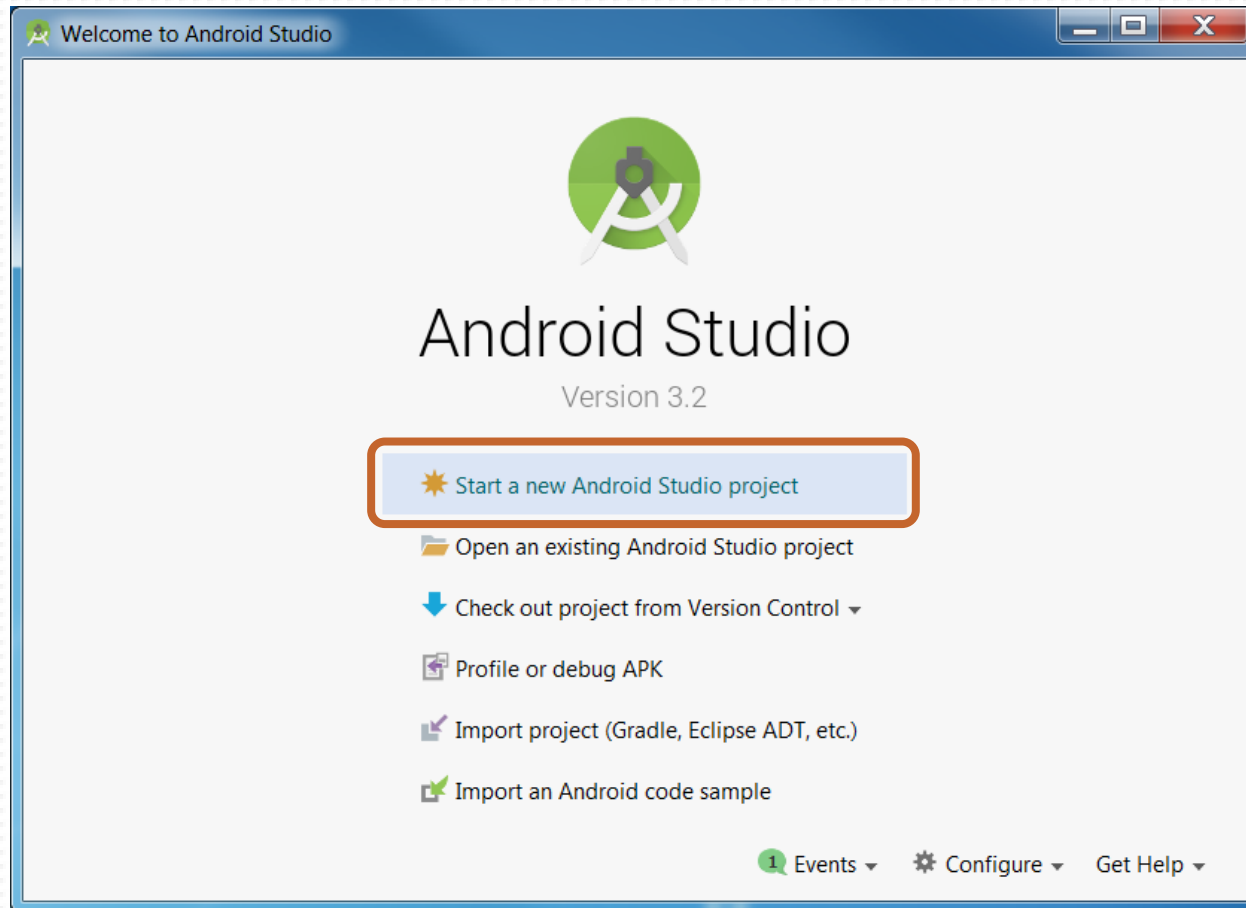
# Agenda

- Android WebView
- Adding WebView to Activity
- Configuring the WebView
- Using WebViewClient
- Using WebChromeClient
- Executing JavaScript Code via WebView
- Invoking Native Java Code on JavaScript Domain
- Remote Debugging WebViews

# Materials

- Application source code can be found on GitHub
  - <https://github.com/CagataySonmez/Android-for-Beginners/tree/master/5-IntroductionToAndroid-DevelopingWebApps>
- Android Studio version 3.5.2 is used on this training
- Android Studio can be downloaded from the official website
  - <https://developer.android.com/studio/>
- Free courses can be found on Google Developers Training website
  - <https://developers.google.com/training/android/>

# Creating Android TV App I



# Creating Android TV App II

Create New Project

Configure your project

Name  
AndroidWebApp

Package name  
com.arcelik.androidwebapp

Save location  
D:\Users\AR430805\AndroidStudioProjects\AndroidWebApp

Language  
Java

Minimum API level  
API 21: Android 5.0 (Lollipop)

*Help me choose*

☐ This project will support instant apps

☒ Use android.\* artifacts

Empty Activity

Creates a new empty activity

Previous Next Cancel Finish

# Building Web Apps in WebView

- Use WebView to deliver a web application as a part of a client application
- WebView does not include any features of a fully developed web browser, such as navigation controls or an address bar
- Using WebView is reasonable
  - if your app provides data to the user that always requires an Internet connection to retrieve data
  - If you already have a web page providing frequently updated information such as an end-user agreement or a user guide

# Adding WebView to Activity

The screenshot displays the Android Studio IDE with the following components:

- Palette:** Shows the 'Layouts' category selected, with 'ConstraintLayout' as the active layout. Other options include Guideline (Horizontal), Guideline (Vertical), LinearLayout (horizontal), LinearLayout (vertical), FrameLayout, TableLayout, TableRow, and Space.
- Component Tree:** Located at the bottom left, it shows a 'ConstraintLayout' containing a 'webView' widget, which is highlighted with a blue selection bar.
- Design View:** The central workspace shows a visual representation of the layout with two side-by-side rectangular areas, one grey and one teal, representing the WebView components.
- Attributes Panel:** On the right, it shows the properties for the selected 'webView' widget. The 'id' is set to 'webView'. Under the 'Layout' section, the 'Constraint Widget' is visualized as a diagram with four corner handles and dimension values (0, 50, 0, 50). Below this, the 'Constraints' list includes:
  - Start → StartOf parent (0dp)
  - End → EndOf parent (0dp)
  - Top → TopOf parent (0dp)
  - Bottom → BottomOf parent (0dp)The 'layout\_width' and 'layout\_height' are both set to '0dp'. The 'visibility' is set to 'visible'.

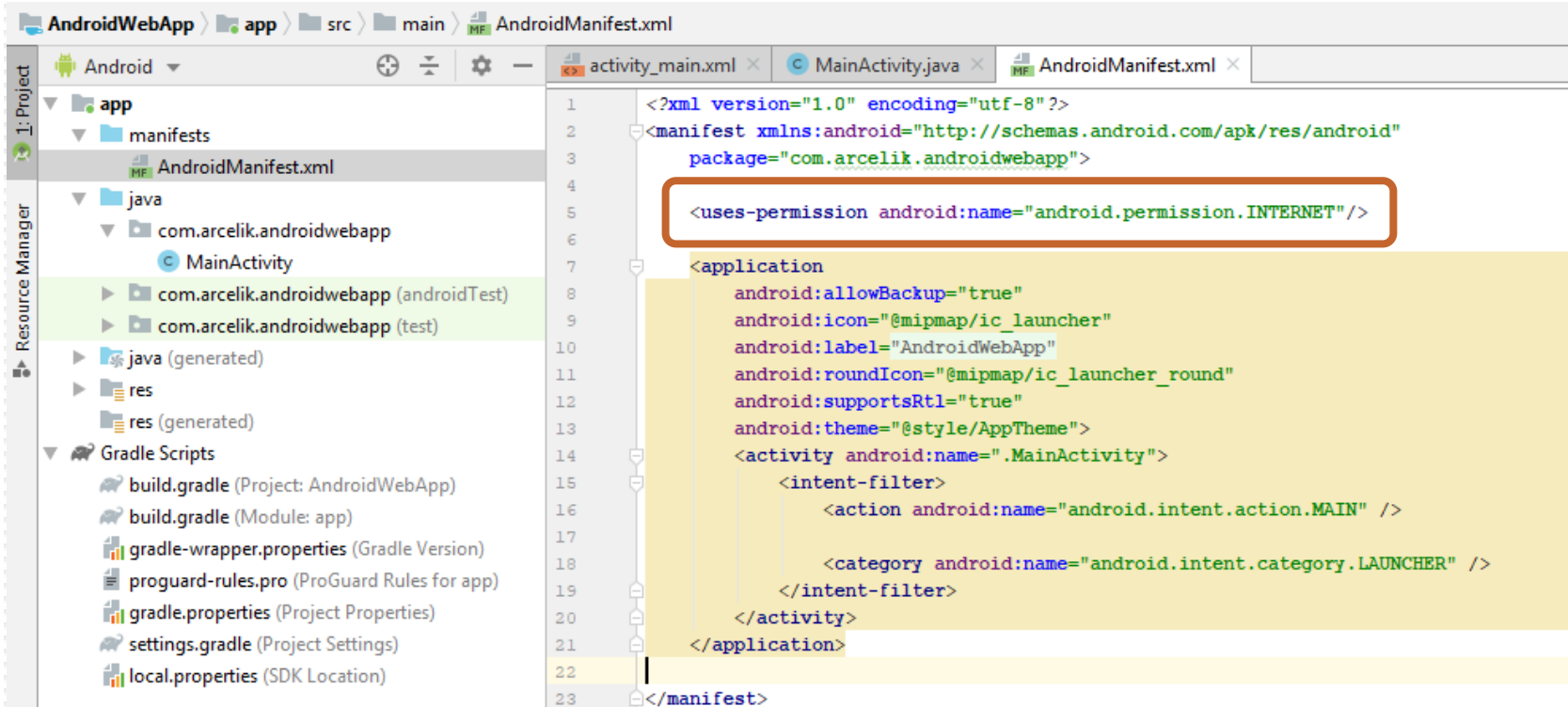
# Loading URL with WebView



```
activity_main.xml x MainActivity.java x jsInterface.java x webChromeHandler.java
10
11 public class MainActivity extends Activity {
12     private String TAG = "MainActivity";
13     private WebView myWebView;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         myWebView = (WebView) findViewById(R.id.webView);
21         myWebView.loadUrl("https://google.com/");
22     }
}
```



# Using INTERNET Permission

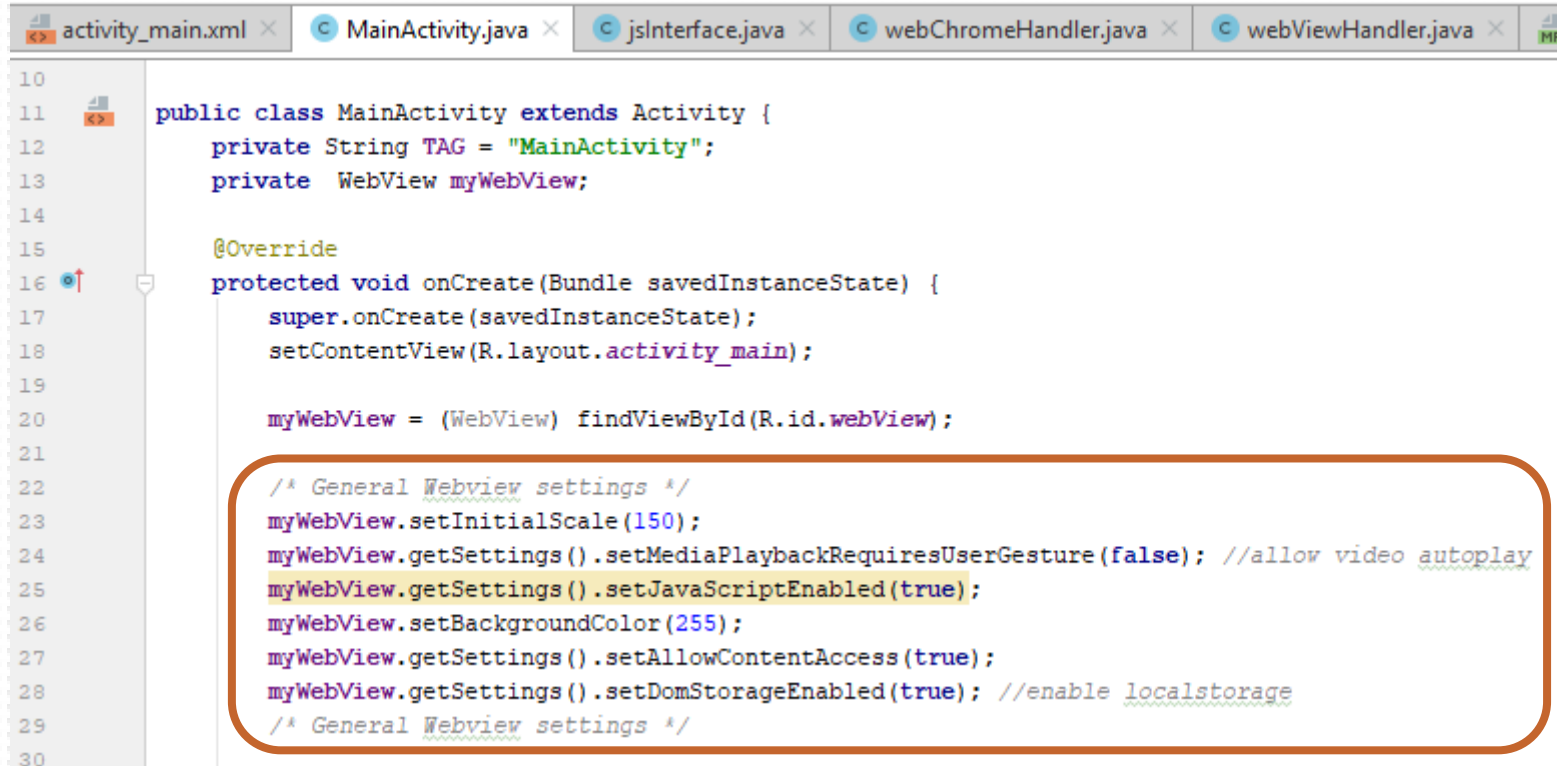


The screenshot displays the Android Studio interface for a project named "AndroidWebApp". The left sidebar shows the project structure, with the "app" module selected. The main editor window shows the "AndroidManifest.xml" file, which is open in the "src/main" directory. The file content is as follows:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3       package="com.arcelik.androidwebapp">
4
5     <uses-permission android:name="android.permission.INTERNET"/>
6
7     <application
8         android:allowBackup="true"
9         android:icon="@mipmap/ic_launcher"
10        android:label="AndroidWebApp"
11        android:roundIcon="@mipmap/ic_launcher_round"
12        android:supportsRtl="true"
13        android:theme="@style/AppTheme">
14        <activity android:name=".MainActivity">
15            <intent-filter>
16                <action android:name="android.intent.action.MAIN" />
17
18                <category android:name="android.intent.category.LAUNCHER" />
19            </intent-filter>
20        </activity>
21    </application>
22
23 </manifest>
```

The line `<uses-permission android:name="android.permission.INTERNET"/>` is highlighted with a red box, indicating the permission being added to the application.

# Configuring WebView I



```
activity_main.xml x MainActivity.java x jsInterface.java x webChromeHandler.java x webViewHandler.java x ME
10
11 public class MainActivity extends Activity {
12     private String TAG = "MainActivity";
13     private WebView myWebView;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         myWebView = (WebView) findViewById(R.id.webView);
21
22         /* General Webview settings */
23         myWebView.setInitialScale(150);
24         myWebView.getSettings().setMediaPlaybackRequiresUserGesture(false); //allow video autoplay
25         myWebView.getSettings().setJavaScriptEnabled(true);
26         myWebView.setBackgroundColor(255);
27         myWebView.getSettings().setAllowContentAccess(true);
28         myWebView.getSettings().setDomStorageEnabled(true); //enable localStorage
29         /* General Webview settings */
30
```

# Configuring WebView II

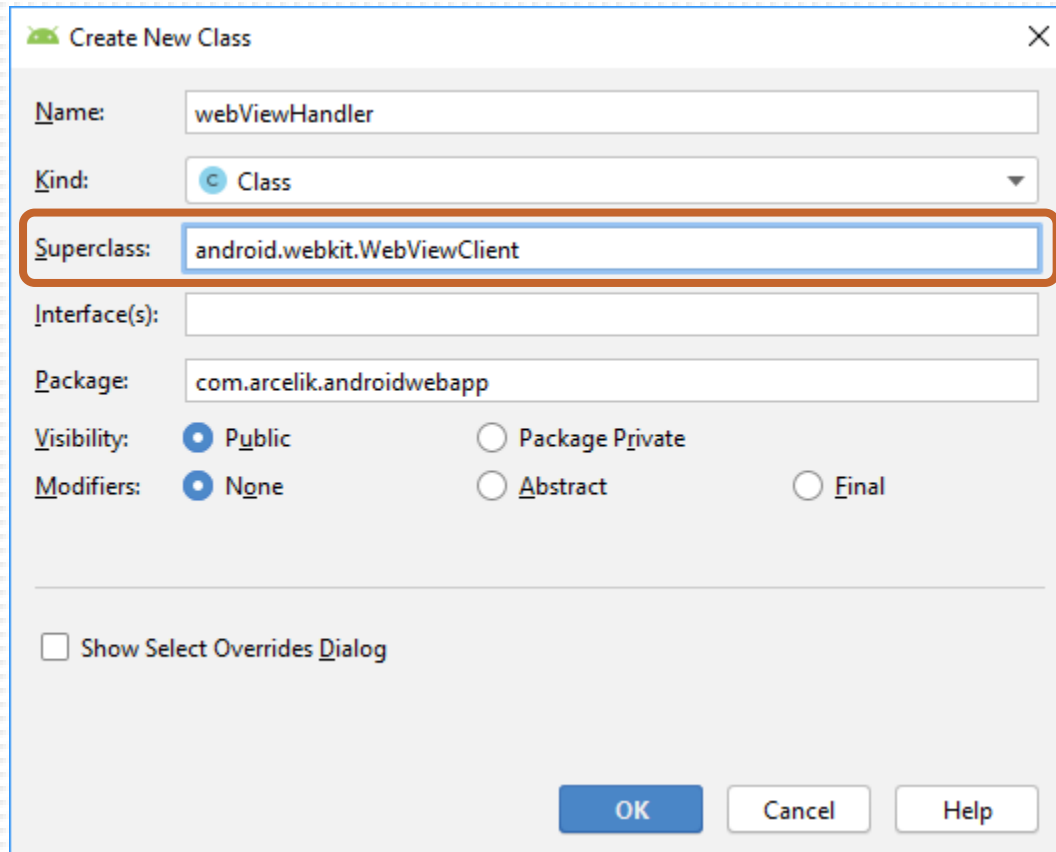
```
activity_main.xml x MainActivity.java x jsInterface.java x webChromeHandler.java x webViewHandler.java x
26 myWebView.setBackgroundColor(255);
27 myWebView.getSettings().setAllowContentAccess(true);
28 myWebView.getSettings().setDomStorageEnabled(true); //enable localStorage
29 /* General Webview settings */
30
31 //enable/disable remote inspection (disable for release app)
32 WebView.setWebContentsDebuggingEnabled(true);
33
34 //Access remote resources from local files
35 myWebView.getSettings().setAllowUniversalAccessFromFileURLs(true);
36
37 //define a custom user agent
38 myWebView.getSettings().setUserAgentString("Custom User Agent");
39
40 //allow http resources over https connection
41 myWebView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_ALWAYS_ALLOW);
42
```

# WebViewClient & WebChromeClient

- The WebView component basically handles the html rendering
- If you want to have more advanced controls, you should use WebViewClient and WebChromeClient
- They have too many functions that you can override; for example:
  - WebView allows taking control when a URL is about to be loaded in the current page
  - WebViewClient allows receiving the current progress of loading a page
- Most of the time WebViewClient and WebChromeClient are used without overriding their methods as shown below:

```
myWebView.setWebChromeClient(new WebChromeClient());  
myWebView.setWebViewClient(new WebViewClient());
```

# Creating WebViewClient I



Create New Class

Name:

Kind: ☒ Class

Superclass:

Interface(s):

Package:

Visibility: ☒ Public ☐ Package Private

Modifiers: ☒ None ☐ Abstract ☐ Final

☐ Show Select Overrides Dialog

OK Cancel Help

# Creating WebViewClient II

```
activity_main.xml × MainActivity.java × jsInterface.java × webChromeHandler.java × webViewHandler.java ×  
10 public class webViewHandler extends WebViewClient {  
11     private String TAG = "WebChromeClient";  
12     private Activity context;  
13  
14     public webViewHandler(Activity ct){  
15         this.context = ct;  
16     }  
17  
18     @Override  
19     public void onPageStarted(WebView view, String url, Bitmap favicon) {  
20         super.onPageStarted(view, url, favicon);  
21     }  
22  
23     @Override  
24     public boolean shouldOverrideUrlLoading(WebView view, String url) {  
25         return false;  
26     }  
27  
28     @Override  
29     public void onLoadResource(WebView view, String url) {  
30         super.onLoadResource(view, url);  
31     }  
32  
33     @Override  
34     public void onPageFinished(WebView view, String url) {  
35         super.onPageFinished(view, url);  
36     }  
37  
38     @Override  
39     public void onReceivedError(WebView view, int errorCode, String description, String failingUrl)
```

# Enabling WebViewClient

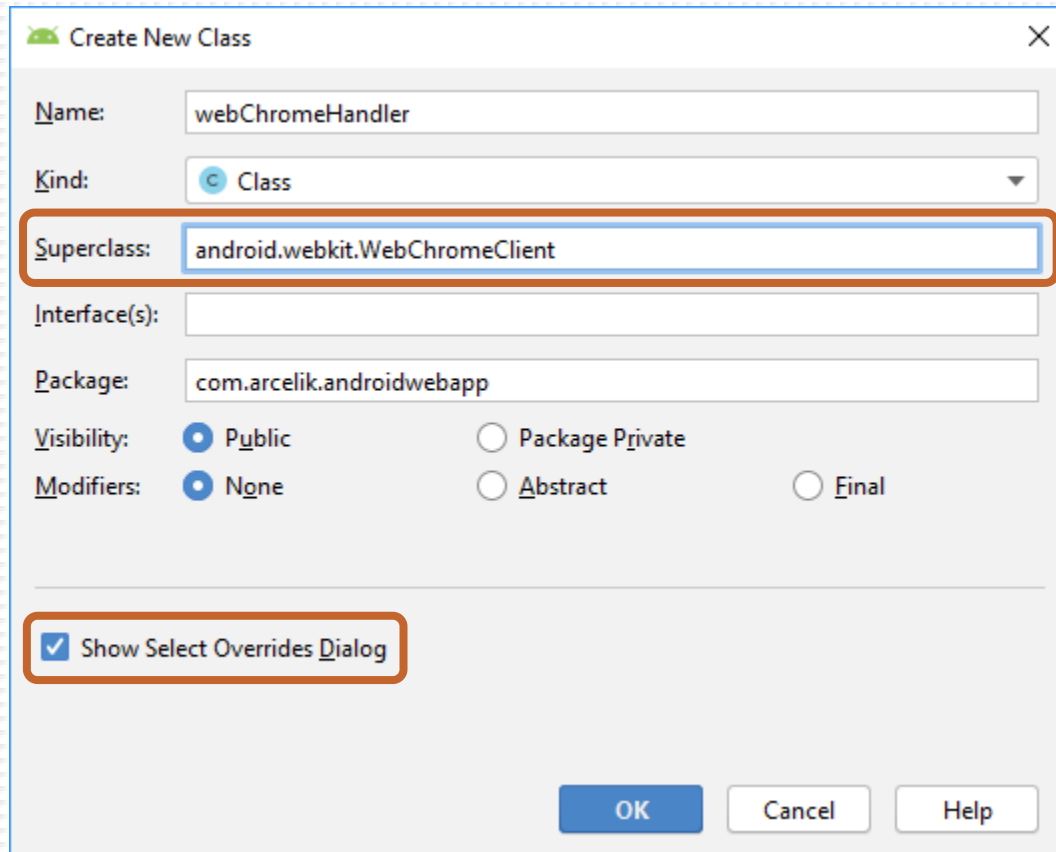


The screenshot shows an IDE with five tabs: activity\_main.xml, MainActivity.java, jsInterface.java, webChromeHandler.java, and webViewHandler.java. The MainActivity.java tab is active, displaying the following code:

```
30
31 //enable/disable remote inspection (disable for release app)
32 WebView.setWebContentsDebuggingEnabled(true);
33
34 //Access remote resources from local files
35 myWebView.getSettings().setAllowUniversalAccessFromFileURLs(true);
36
37 //define a custom user agent
38 myWebView.getSettings().setUserAgentString("Custom User Agent");
39
40 //allow http resources over https connection
41 myWebView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_ALWAYS_ALLOW);
42
43 //Enable WebViewClient
44 myWebView.setWebViewClient(new webViewHandler( ct: this));
45
46
47
```

The code on lines 43 and 44 is highlighted with an orange border. The code enables remote inspection, allows universal access from file URLs, sets a custom user agent, allows mixed content, and sets a custom WebViewClient.

# Creating WebChromeClient I



**Create New Class**

Name:

Kind: ☐ Class

Superclass:

Interface(s):

Package:

Visibility: ☒ Public ☐ Package Private

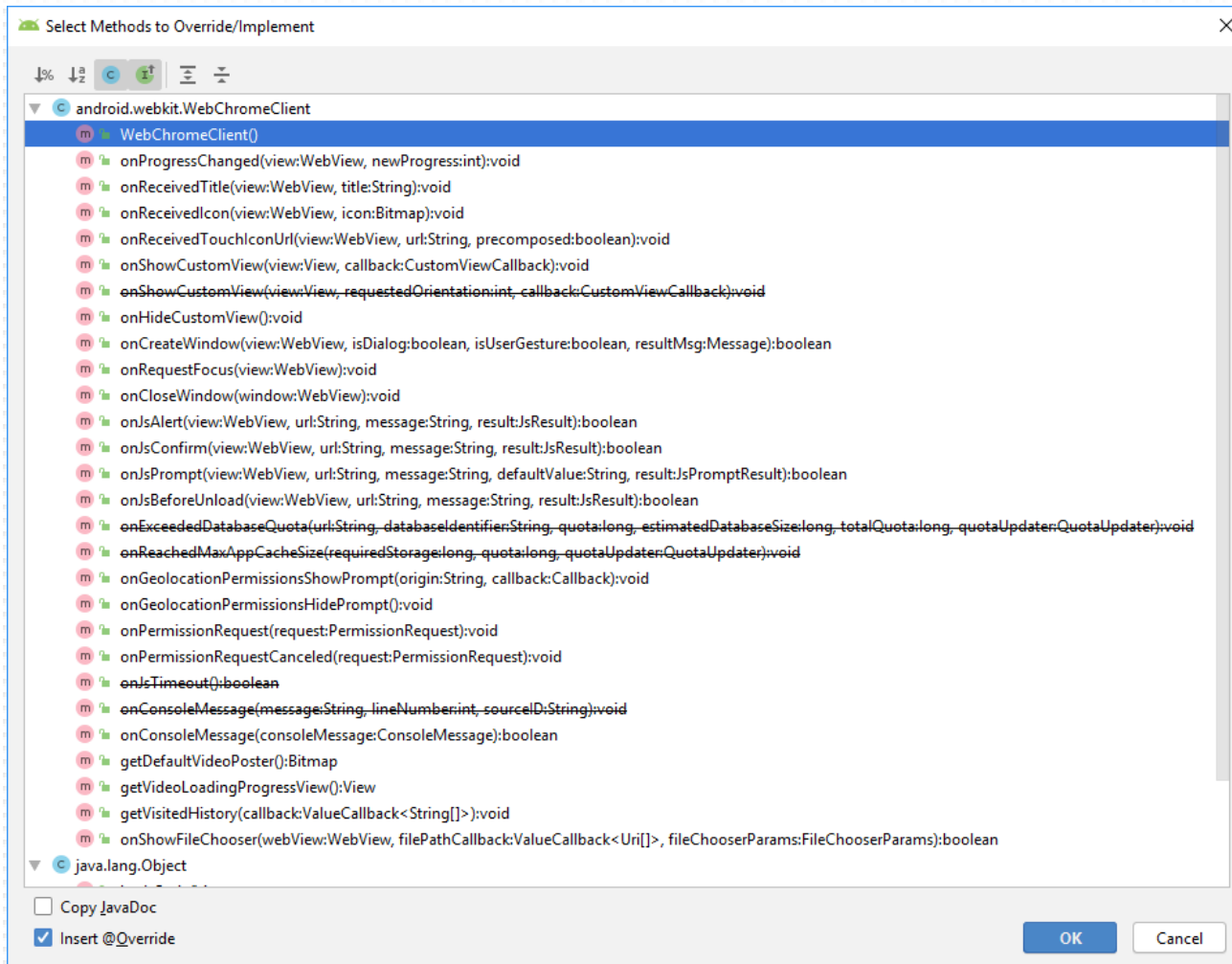
Modifiers: ☒ None ☐ Abstract ☐ Final

☒ Show Select Overrides Dialog

OK Cancel Help




# Creating WebChromeClient II



# Creating WebChromeClient III

```
activity_main.xml x MainActivity.java x jsInterface.java x webChromeHandler.java x
1 package com.arcelik.androidwebapp;
2
3 import android.app.Activity;
4 import android.util.Log;
5 import android.webkit.WebChromeClient;
6 import android.webkit.WebView;
7
8 public class webChromeHandler extends WebChromeClient {
9     private String TAG = "WebViewClient";
10    private Activity context;
11
12    public webChromeHandler(Activity ct){
13        this.context = ct;
14    }
15
16    @Override
17    public void onProgressChanged(WebView view, int progress)
18    {
19        // Return the app name after finish loading
20        if(progress == 100)
21            Log.d(TAG, msg: "Load finished!");
22        else
23            Log.d(TAG, msg: "progress: " + progress + "%");
24    }
25
26    @Override
27    public void onCloseWindow (WebView window){
28        super.onCloseWindow(window);
29        context.finish();
30    }
31 }
```

# Enabling WebChromeClient



The screenshot shows an IDE with five tabs: activity\_main.xml, MainActivity.java, jsInterface.java, webChromeHandler.java, and webViewHandler.java. The MainActivity.java tab is active, displaying Java code for enabling WebChromeClient. The code includes comments and method calls to configure WebView settings. The final two lines, which enable the WebChromeClient, are highlighted with a red rounded rectangle.

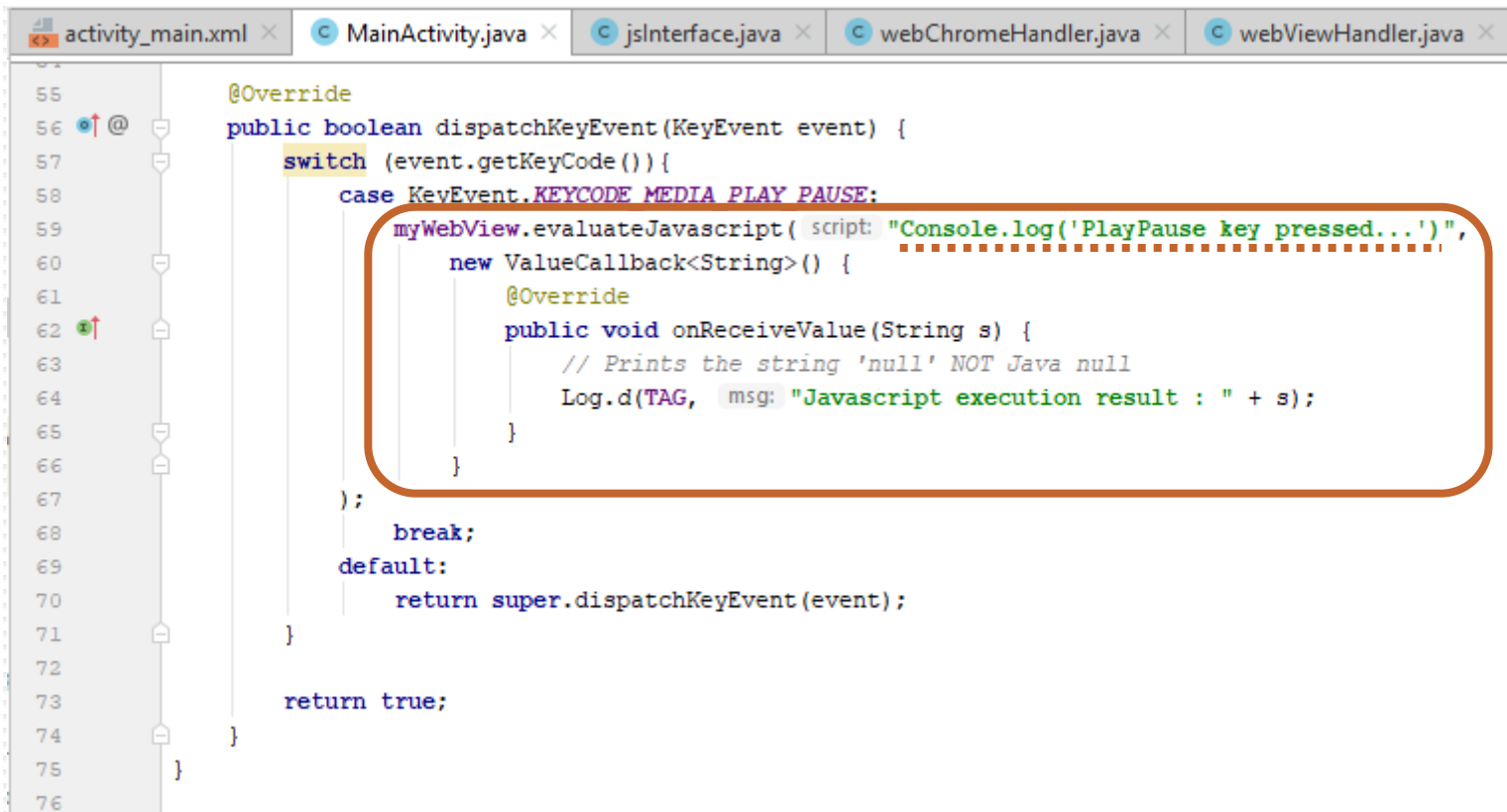
```
30
31 //enable/disable remote inspection (disable for release app)
32 WebView.setWebContentsDebuggingEnabled(true);
33
34 //Access remote resources from local files
35 myWebView.getSettings().setAllowUniversalAccessFromFileURLs(true);
36
37 //define a custom user agent
38 myWebView.getSettings().setUserAgentString("Custom User Agent");
39
40 //allow http resources over https connection
41 myWebView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_ALWAYS_ALLOW);
42
43 //Enable WebViewClient
44 myWebView.setWebViewClient(new webViewHandler( ct: this));
45
46 //Enable WebChromeClient
47 myWebView.setWebChromeClient(new webChromeHandler( ct: this));
48
49
50
```

# Execute JavaScript Code via WebView I

- You might need to execute JavaScript function on your Android application, for example you can;
  - Pass a value to the currently displayed page by invoking a function
  - Manipulate the currently displayed page via the JavaScript code
- **evaluateJavascript** function of WebView asynchronously evaluates JavaScript in the context of the currently displayed page
- This method must be called on the UI thread and the callback will be made on the UI thread.

```
public void evaluateJavascript (String script, ValueCallback<String> resultCallback)
```

# Execute JavaScript Code via WebView II



The screenshot shows the Android Studio IDE with the following tabs open: activity\_main.xml, MainActivity.java, jsInterface.java, webChromeHandler.java, and webViewHandler.java. The MainActivity.java file is selected, and the following code is visible:

```
55 @Override
56 public boolean dispatchKeyEvent(KeyEvent event) {
57     switch (event.getKeyCode()) {
58         case KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE:
59             myWebView.evaluateJavascript( script: "Console.log('PlayPause key pressed...')",
60                 new ValueCallback<String>() {
61                     @Override
62                     public void onReceiveValue(String s) {
63                         // Prints the string 'null' NOT Java null
64                         Log.d(TAG, msg: "Javascript execution result : " + s);
65                     }
66                 }
67             );
68             break;
69             default:
70                 return super.dispatchKeyEvent(event);
71     }
72 }
73 return true;
74 }
75 }
76 }
```

A red rounded rectangle highlights the code block from line 59 to 66, which demonstrates evaluating JavaScript code and handling the result via a ValueCallback.

# Invoke Java Function from JavaScript

- You might need to execute Java function in the context of the currently displayed page, for example you can;
  - build a communication channel between JavaScript and Java domains
  - handle operations which are not supported by HTML standards
- **addJavascriptInterface** function of WebView allows the Java object's methods to be accessed from JavaScript.
- The instance of your interface (object) is injected into all frames of the web page, including all the iframes, using the supplied name.

```
public void addJavascriptInterface (Object object, String name)
```

# Adding JavaScript Interface

```
activity_main.xml x MainActivity.java x jsInterface.java x webChromeHandler.java x
1 package com.arcelik.androidwebapp;
2
3 import android.app.Activity;
4 import android.webkit.JavascriptInterface;
5 import android.widget.Toast;
6
7 import java.util.Locale;
8
9 public class jsInterface {
10     private String TAG = "jsInterface";
11     private Activity context;
12     @ public jsInterface(Activity context) {
13         this.context = context;
14     }
15
16     @JavascriptInterface
17     public String getLanguage() {
18         return Locale.getDefault().getLanguage();
19     }
20
21     @JavascriptInterface
22     public String getCountry() {
23         return Locale.getDefault().getCountry();
24     }
25
26     @JavascriptInterface
27     public void showToast(String toast) {
28         Toast.makeText(context, toast, Toast.LENGTH_SHORT).show();
29     }
30 }
```

# Binding JavaScript Code to Java Code



```
activity_main.xml × MainActivity.java × jsInterface.java × webChromeHandler.java × webViewHandler.java ×
37 //define a custom user agent
38 myWebView.getSettings().setUserAgentString("Custom User Agent");
39
40 //allow http resources over https connection
41 myWebView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_ALWAYS_ALLOW);
42
43 //Enable WebViewClient
44 myWebView.setWebViewClient(new webViewHandler( ct: this));
45
46 //Enable WebChromeClient
47 myWebView.setWebChromeClient(new webChromeHandler( ct: this));
48
49 //Binding JavaScript Code to Java Code
50 myWebView.addJavascriptInterface(new jsInterface( context: this), name: "arSmartTV");
51
52 myWebView.loadUrl("https://html5test.com/");
53 }
54
```



# Invoking Native Java Function

```
<!DOCTYPE html>
<html>
<body>

<script>
    arSmartTV.showToast("Hello World!");
</script>

</body>
</html>
```



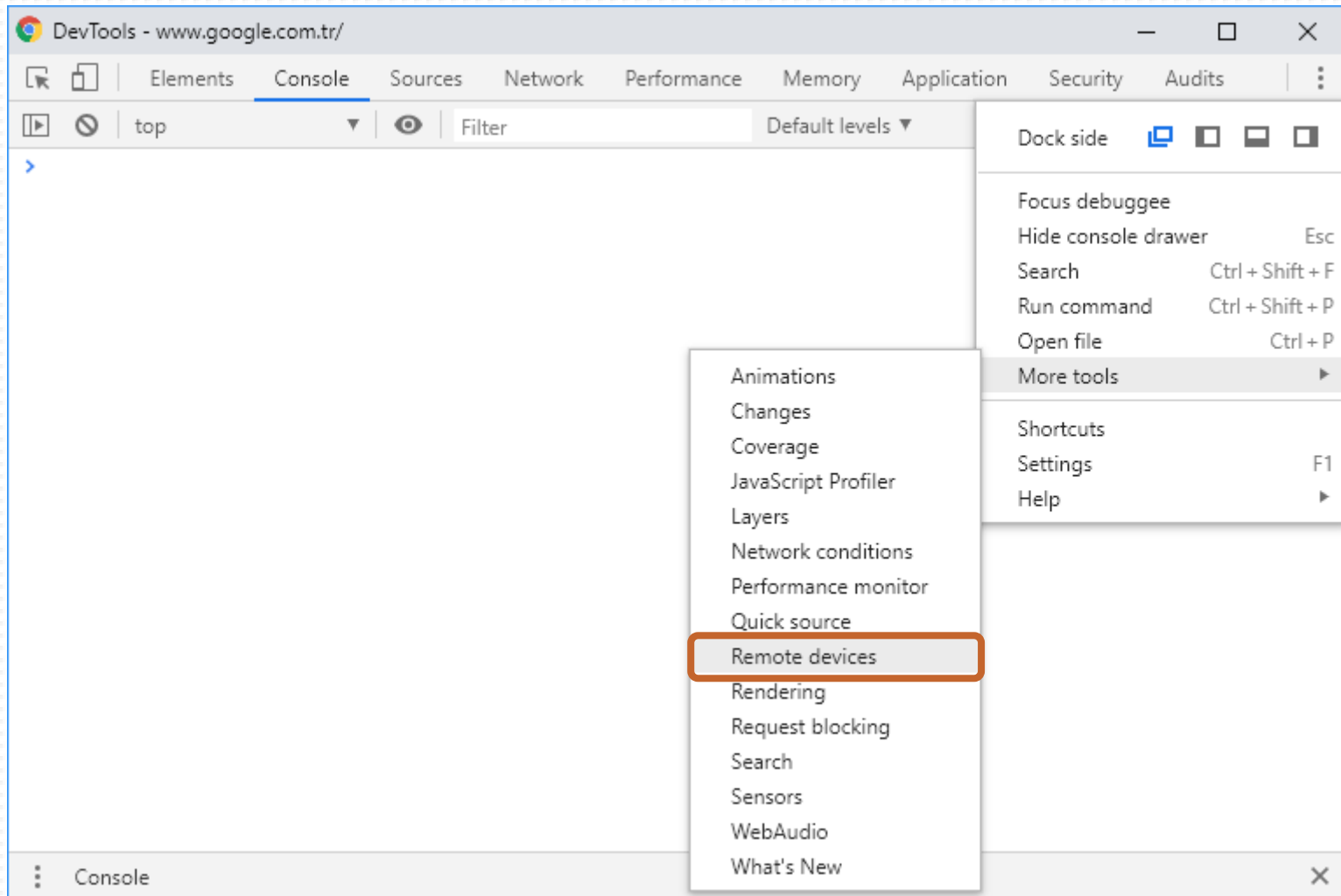
This will Show a toast message on the screen

# Remote Debugging WebViews

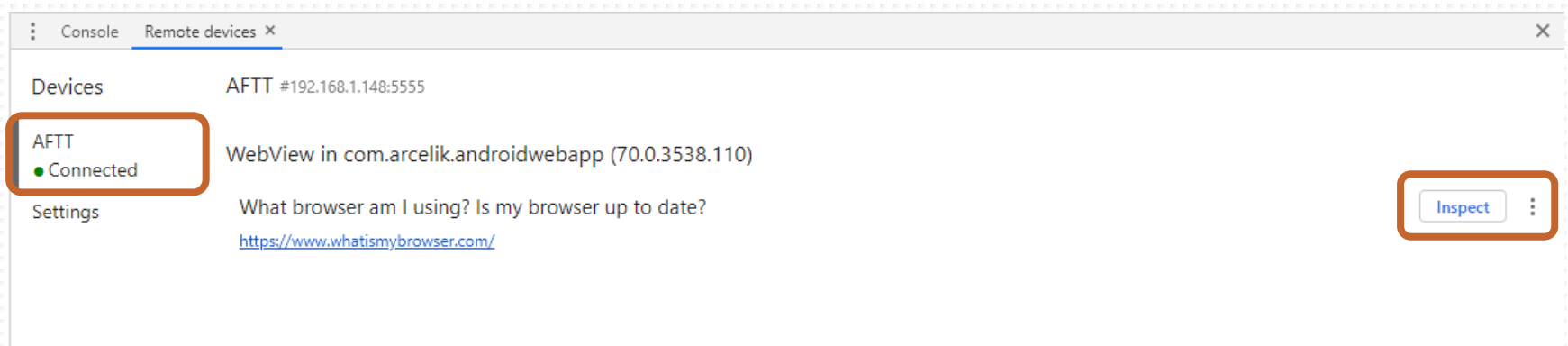
- You can use Chrome Developer Tools to debug WebViews in your native Android apps.
- To enable WebView debugging, call the following static method of the WebView class:
  - **setWebContentsDebuggingEnabled.**
- You can find a list of debug-enabled WebViews on your device via
  - The chrome://inspect page, or
  - DevTools options -> More Tools -> Remote Devices menu

```
WebView.setWebContentsDebuggingEnabled(true);
```

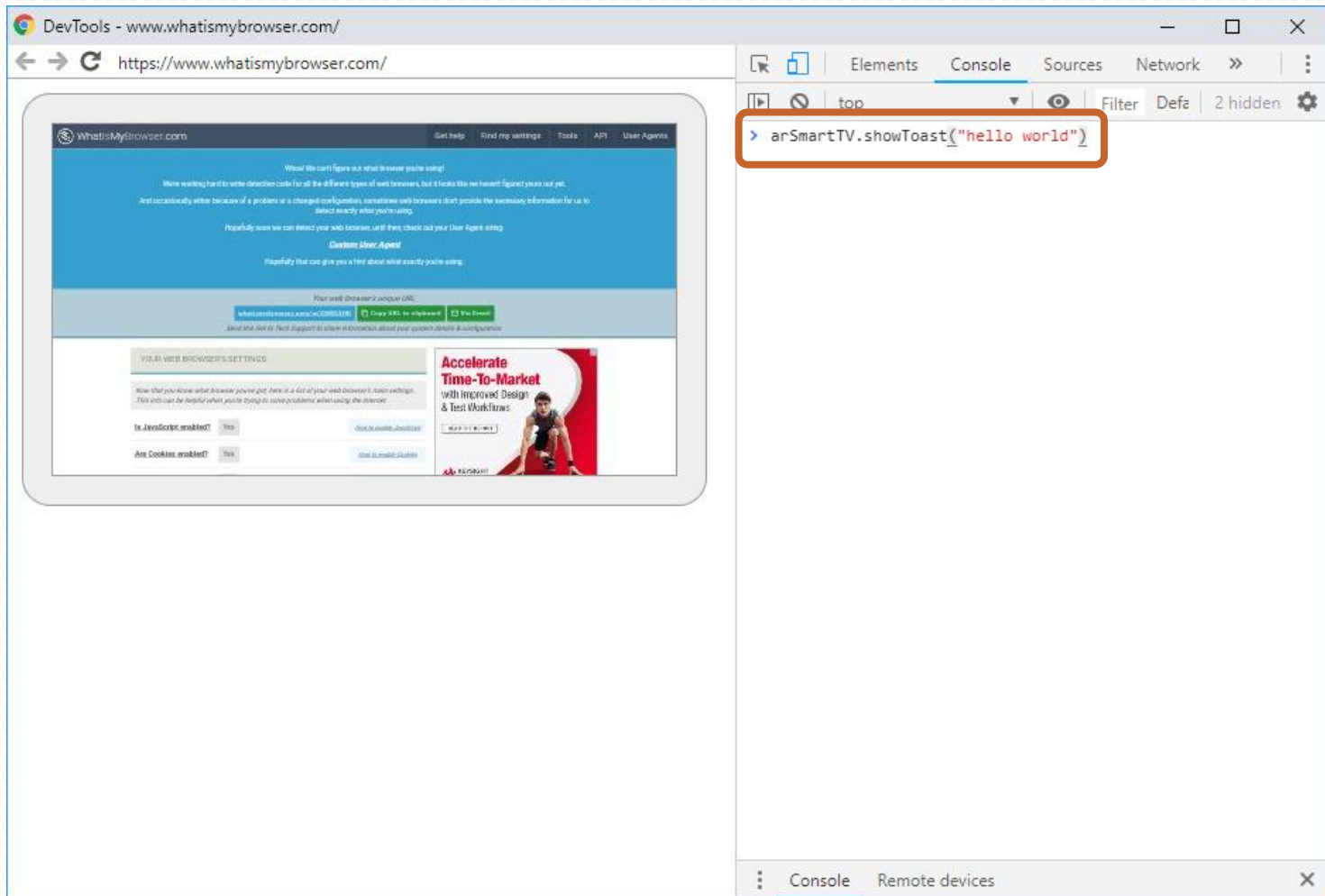
# Opening a WebView in DevTools



# Opening a WebView in DevTools II



# Opening a WebView in DevTools III



# QUESTIONS?