MOBILE WEB APPS

Mobile Web App

- Applications that look like ‘native’ applications but make only use of web technologies Idea: use a single activity that shows a webview
- The webview reads .html pages from the *local* asset

- Apps are written with standard **web technology**

  - **Html/CSS** pages linked with each other to provide the same look and feel of a native applications, in particular they follow the **Single Page Application** pattern

  - **JavaScript** embedded in the pages (and any JS library, like jQuery, as well as techniques like AJAX)

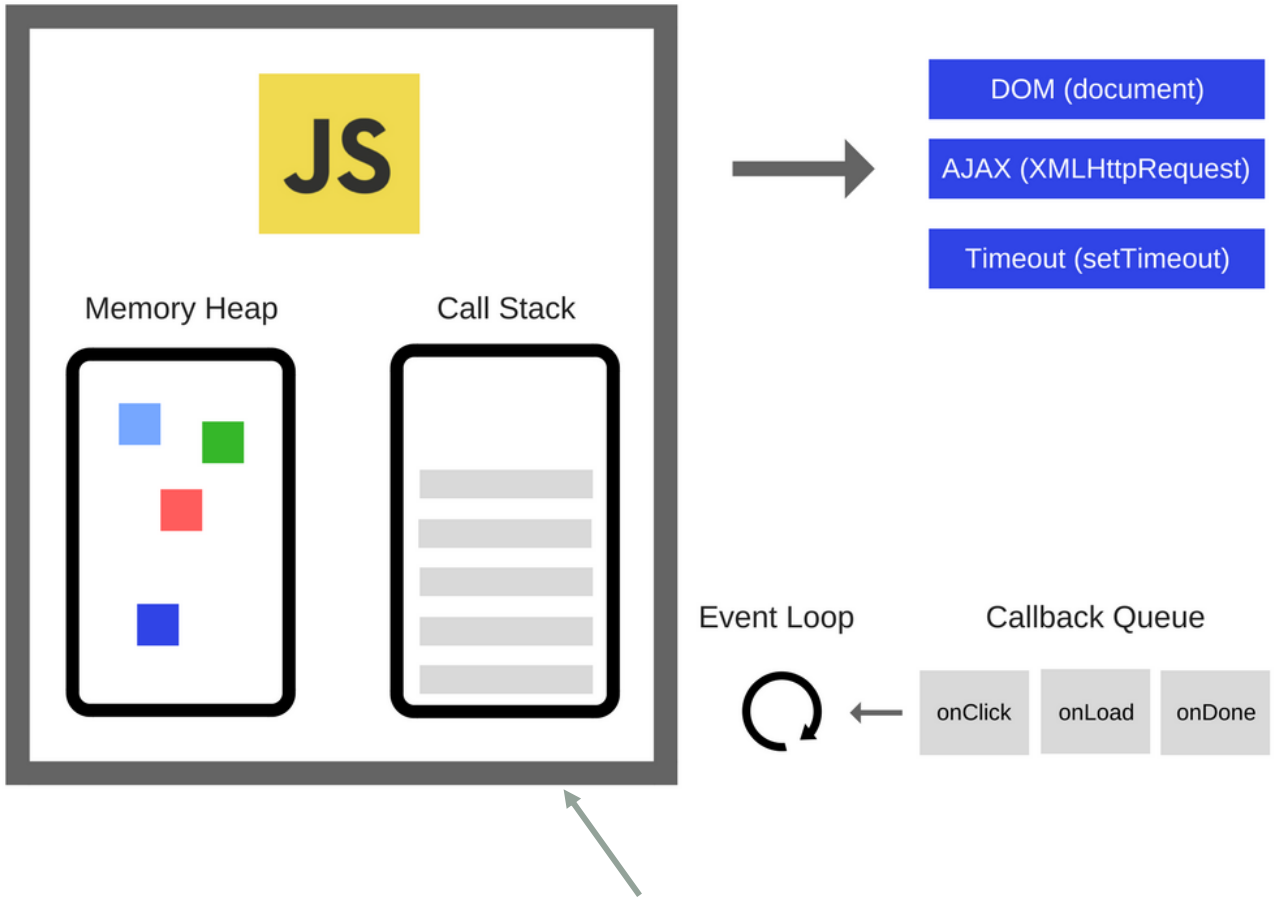
A web page is an active program

- **HTML5**
  - Content + Structure via a fixed set of **tags**

- **CSS3**:
  - How to display an element (colors, fonts, borders..)
  - How to layout elements wrt each other
  - Selector + declaration

- **JavaScript**:
  - How to react to an ‘event’
    - Changing the style of an element (e.g., onMouseOver)
    - Trigger some computation
    - Change the content of the page
- **Ajax**
  - Asynchronous interactions over HTTP(e.g. use a web-api)

Browser as a sandboxed VM



- Single thread
- Event-driven
- Web worker (HMTL5)

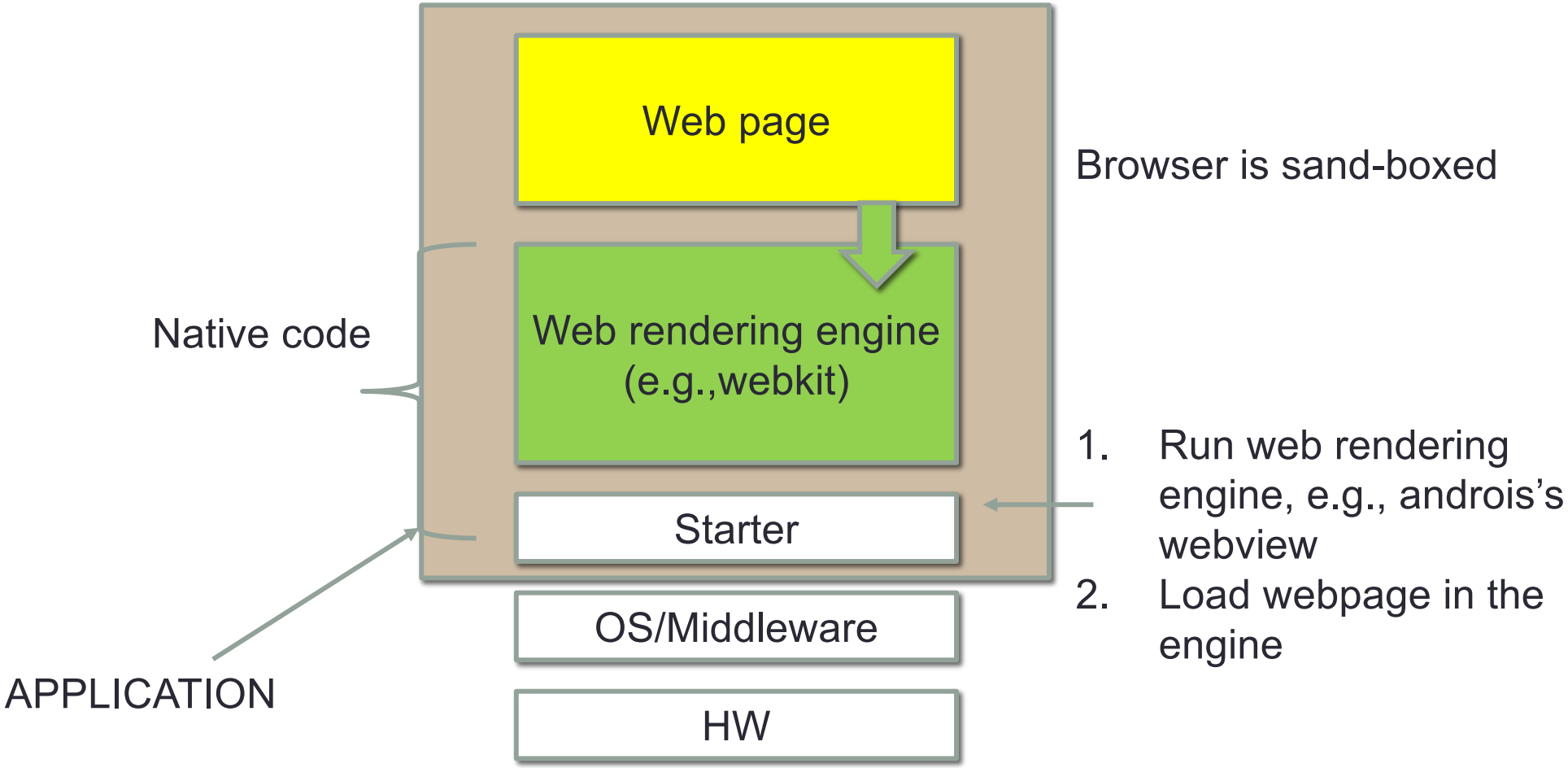
Sandbox (restrict device access)
Secure mechanism (JS is external, untrusted code…)

Multithreading

- **Web worker:**
  - Allows to run parallel JS code
  - Only one MAIN thread that access to DOM
- **Service worker**:
  - Much more flexible
- **WebAssemby**

Architecture of a Mobile Web App

- Idea: just avoid fetching page from the site
- Store the web page locally (of course no local web-server!)



Setting the viewport

- Most of the broswers support the viewport meta tag
- It allows to set the view port for the rendering process
- In particular, it allows to pass the real current viewport, so that redering is done on the real size
- A tag as simple as the following one solves the previous problem

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Some example of viewports

Apple Products

	Pixel Size	Viewport
iPhone		
iPhone X	1125 x 2436	375 x 812
iPhone 8 Plus	1080 x 1920	414 x 736
iPhone 8	750 x 1334	375 x 667
iPhone 7 Plus	1080 x 1920	414 x 736
iPhone 7	750 x 1334	375 x 667
iPhone 6 Plus/6S Plus	1080 x 1920	414 x 736
iPhone 6/6S	750 x 1334	375 x 667
iPHone 5	640 x 1136	320 x 568
iPod		
iPod Touch	640 x 1136	320 x 568
iPad		
iPad Pro	2048 x 2732	1024 x 1366
iPad Third & Fourth Generation	1536 x 2048	768 x 1024
iPad Air 1 & 2	1536 x 2048	768 x 1024
iPad Mini	768 x 1024	768 x 1024
iPad Mini 2 & 3	1536 x 2048	768 x 1024

320--1280

Android Devices

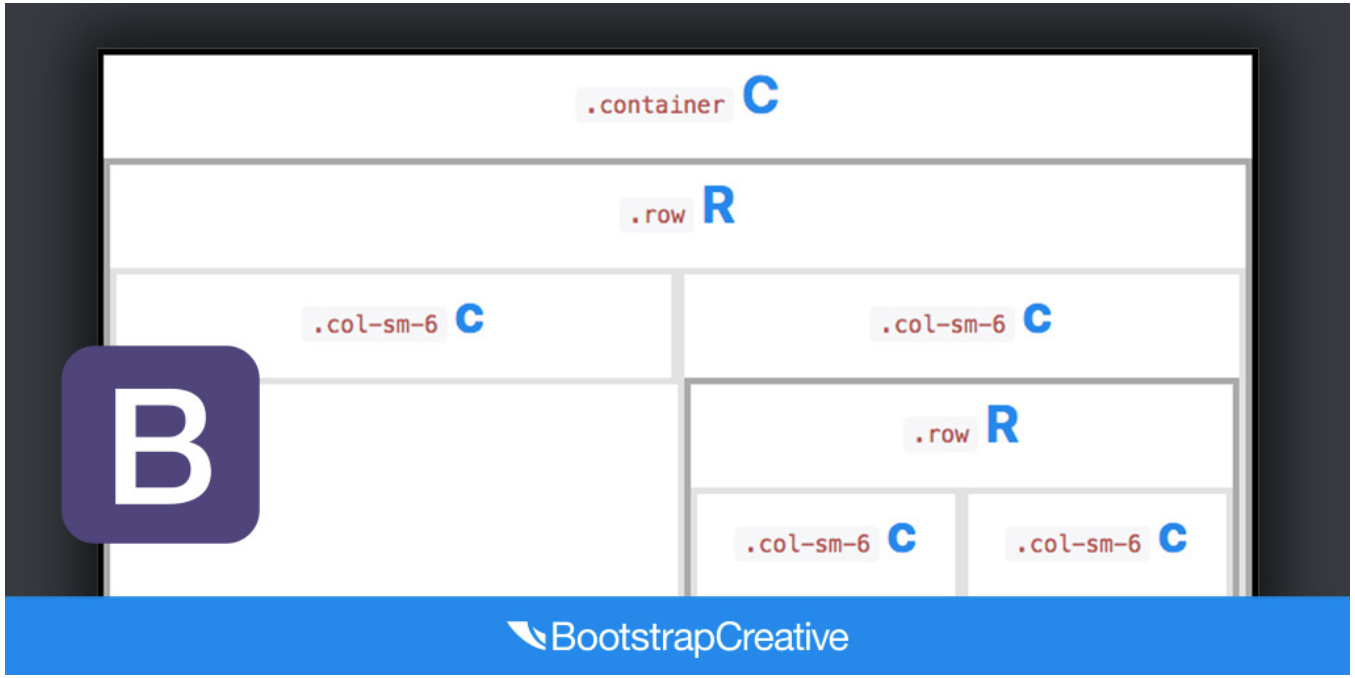
	Pixel Size	Viewport
Phones		
Nexus 6P	1440 x 2560	411 x 731
Nexus 5X	1080 x 1920	411 x 731
Google Pixel	1080 x 1920	411 x 731
Google Pixel XL	1440 x 2560	411 x 731
Google Pixel 2	1080 x 1920	411 x 731
Google Pixel 2 XL	1440 x 2560	411 x 731
Samsung Galaxy Note 5	1440 x 2560	480 x 853
LG G5	1440 x 2560	480 x 853
One Plus 3	1080 x 1920	480 x 853
Samsung Galaxy S9	1440 x 2960	360 x 740
Samsung Galaxy S9+	1440 x 2960	360 x 740
Samsung Galaxy S8	1440 x 2960	360 x 740
Samsung Galaxy S8+	1440 x 2960	360 x 740
Samsung Galaxy S7	1440 x 2560	360 x 640
Samsung Galaxy S7 Edge	1440 x 2560	360 x 640
Tablets		
Nexus 7 (2013)	1200 x 1920	600 x 960
Nexus 9	1536 x 2048	768 x 1024
Samsung Galaxy Tab 10	800 x 1280	800 x 1280
Chromebook Pixel	2560 x 1700	1280 x 850

See also: https://www.gsmarena.com/

Credits: mediagenesis

Bootstrap 4

- It is a framework for mobile-first responsive web design
- The main strengths is the use of ‘flex-box’ CSS layout, which defines how a container adapts to viewport



BS4 Breakpoints

xs	Extra small <576px	portrait mobile
sm	Small ≥576px	landscape mobile
md	Medium ≥768px	portrait tablets *navbar collapse*
lg	Large ≥992px	landscape tablets
xl	Extra large ≥1200px	laptops, desktops, TVs

Hybrid app

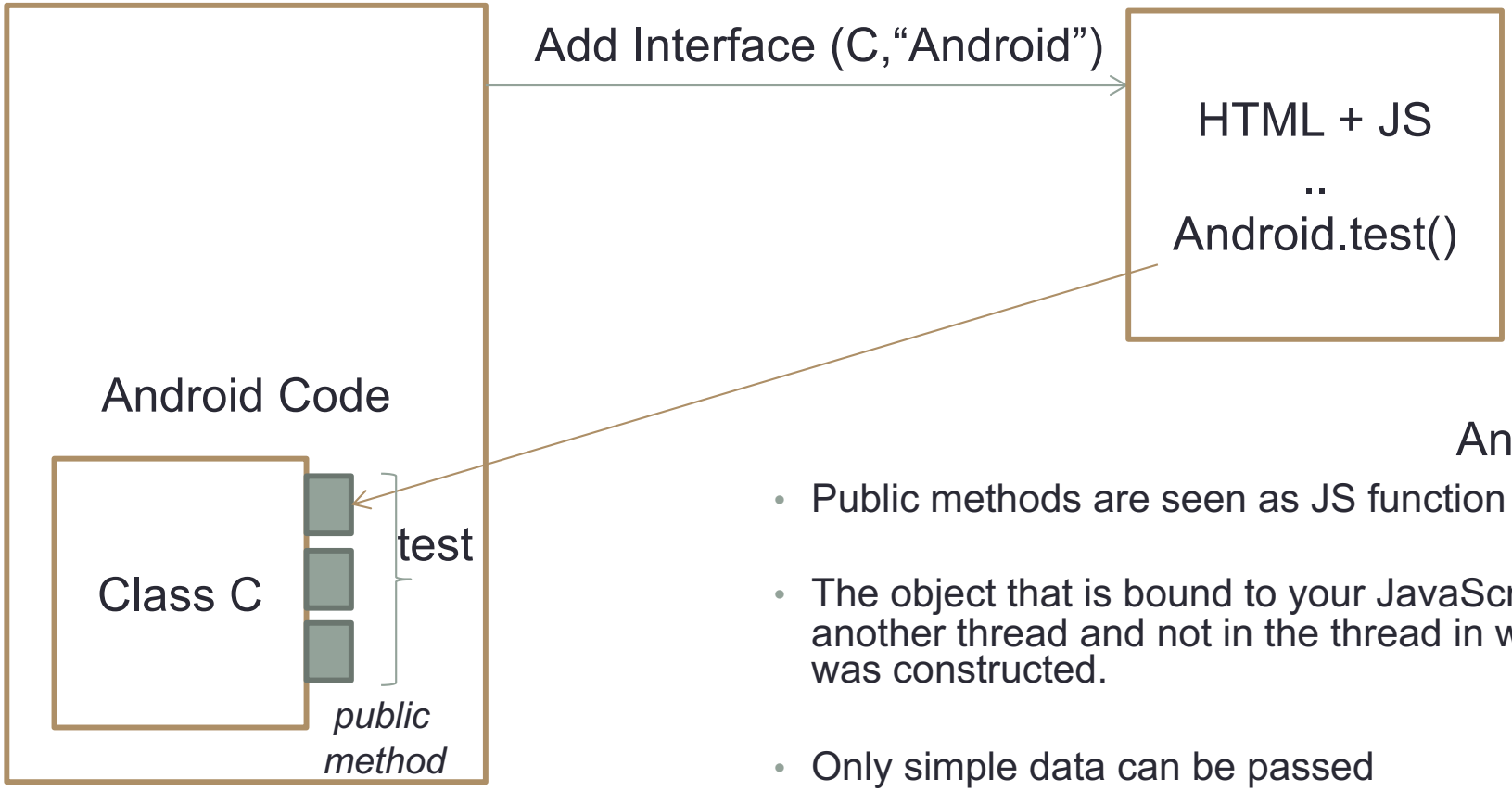
- Android method mapped to JS functions, i.e., available JS functions can be extended so that..
- Device features can be exported in JavaScript
- And, android code can call JS function
- Also, Apps can be made *crossplatform (similar behaviour in other platform)*

Binding JavaScript code to Android code

- It is possible to create interfaces between JavaScript code and Android code.

- In other words, JavaScript code can call a method implemented in ‘android’

Binding JavaScript code to Android code



Android.test();

- Public methods are seen as JS function
- The object that is bound to your JavaScript runs in another thread and not in the thread in which it was constructed.
- Only simple data can be passed
- JSON can be useful in this respect

Example (android side)

```
public class Android{
    Context mContext;
    Android(Context ctx) { mContext = ctx; }

    @JavascriptInterface
    public void showToast(String toast){
        Toast.makeText(mContext,toast,Toast.LENGTH_SHORT).show();
    }
    @JavascriptInterface
    public String sayHello() { return "Hello JavaScript from Android!"; }
}
```

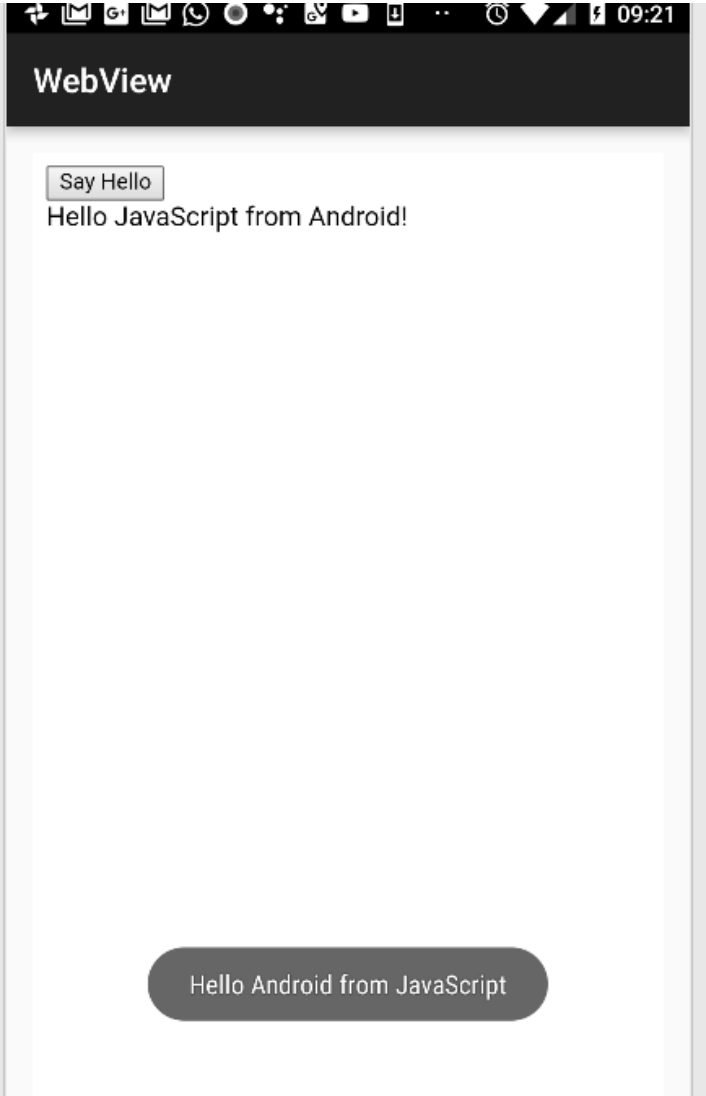
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    WebView webView = (WebView) findViewById(R.id.webview);
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl("file:///android_asset/hello.html");
```

```
webView.addJavascriptInterface(new Android( ctx: this), name: "Android");
```

Example (JS side)

```
<html>
<script type="text/javascript">
    function CallAndroid(toast){
        Android.showToast(toast);
        msg=Android.sayHello();
        document.getElementById("mDiv").innerHTML=msg;
    }
</script>


    <body>
    <input  type="button"
            value="Say Hello"
            onClick="CallAndroid('Hello Android from JavaScript')"/>

    <div id="mDiv"></div>
```

Example

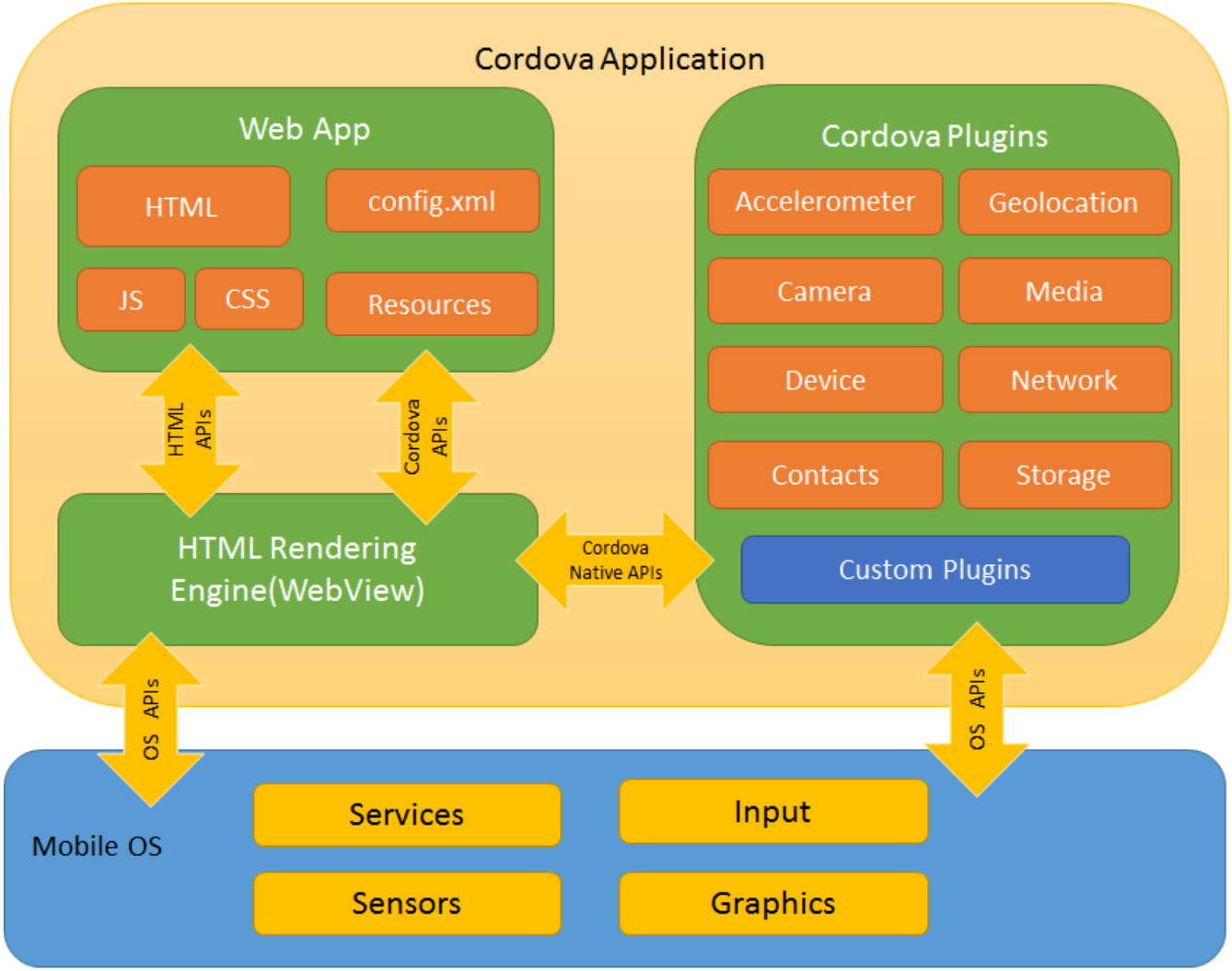

09:21
WebView
Say Hello
Hello JavaScript from Android!
Hello Android from JavaScript

Injecting JS code in the view

- Starting from Api 19 it is possible to asynchronously injecting javascript code in a loaded page…
- So one can call JS functions

```
String javascript = "";
webView.evaluateJavascript(javascript, new ValueCallback<String>() {
        @Override
        public void onReceiveValue(String value) {
            Log.i("onReceiveValue! " + value);
        }
    });
```

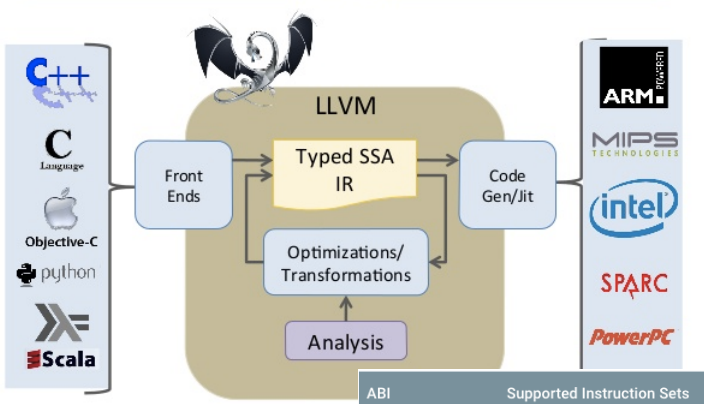
Hybrid apps: Cordova



NATIVE LIBRARIES AND NDK

What is NDK?

- Native Development Kit (NDK) is a set of tools that allow to embed native code into Android applications
- It exploits the **LLVM** project, and in particular: LLVM Compiler Infrastructure



ABI	Supported Instruction Sets
armeabi-v7a	• armeabi • Thumb-2 • VFPv3-D16
arm64-v8a	• AArch64
x86	• x86 (IA-32) • MMX • SSE/2/3 • SSSE3
x86_64	• x86-64 • MMX • SSE/2/3 • SSSE3 • SSE4.1, 4.2 • POPCNT

- **CLANG** C/C++ *cross* compiler
  - 32-bit ARM, AArch64, x86, and x86-64.
  - C library: **bionic**
  - C++ library: **LLVM libc++**
  - NDK libraries: **openGL, Vulkan, Neural Network ,..**

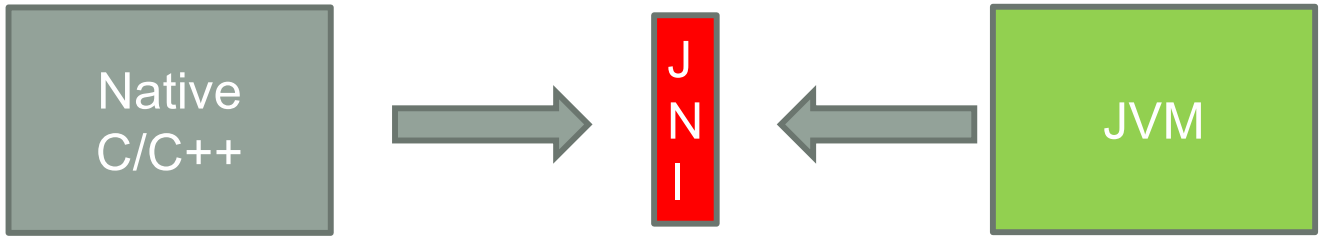
- **CMake** as build tool

What is NDK?

- NDK uses **Java Native Interface** to mix Java/Kotlin code with C/C++ code
- Application written in Java/Kotlin can access C/C++ code and vice versa
- **Native Activity**: Entire activities are implemented in C/C++
  - Example: Flutter

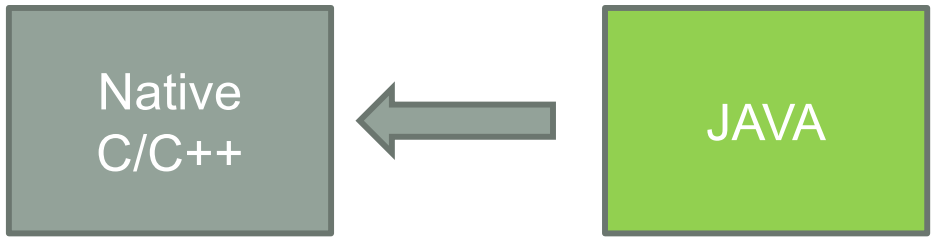
JNI

- Implements the Foreign Function Invocation mechanism
- For example, JNI allows to C code to:
  - **Create**, **inspect**, and **update** Java objects (including arrays and strings)
  - Call Java methods
  - Catch and throw exceptions
- It also allows Java code to call C/C++ code



JNI (java → C/C++)

- Java code sees **external** symbols (functions) defined in a shared library (result of the native code compilation)
  - Recall that a .so is an *ELF* file with code + meta-info that makes code in the file accessible



JNI (java → C/C++)

- Java/Kotlin loads the **shared library** at **run-time**:
  - **System.loadLibrary("native-lib")**

- Extern functions are retrieved from the library

```
external fun stringFromJNI(): String

companion object {

    // Used to load the 'native-lib' library on application startup.
    ini {
        System.loadLibrary( libname: "native-lib")
    }
}
```

JNI (C/C++ → java)

- A native method always receives two arguments
- a pointer to the JNI interface, **JNIEnv *env**
- a pointer to *a* java object bounded to the function

Native
C/C++

```
env->
CallNonvirtualCharMethodA(jobject obj, jclass clazz, jmethodID methodID, const…   jchar
CallNonvirtualCharMethodV(jobject obj, jclass clazz, jmethodID methodID, va_li…   jchar
CallNonvirtualDoubleMethod(jobject obj, jclass clazz, jmethodID methodID, ....   jdouble
CallNonvirtualDoubleMethodA(jobject obj, jclass clazz, jmethodID methodID, c…   jdouble
CallNonvirtualDoubleMethodV(jobject obj, jclass clazz, jmethodID methodID, v…   jdouble
CallNonvirtualFloatMethod(jobject obj, jclass clazz, jmethodID methodID, ...)    jfloat
CallNonvirtualFloatMethodA(jobject obj, jclass clazz, jmethodID methodID, con…   jfloat
CallNonvirtualFloatMethodV(jobject obj, jclass clazz, jmethodID methodID, va_…   jfloat
CallNonvirtualIntMethod(jobject obj, jclass clazz, jmethodID methodID, ...)       jint
CallNonvirtualIntMethodA(jobject obj, jclass clazz, jmethodID methodID, const j…   jint
CallNonvirtualIntMethodV(jobject obj, jclass clazz, jmethodID methodID, va_list…   jint
CallNonvirtualLongMethod(jobject obj, jclass clazz, jmethodID methodID, ...)      jlong
CallNonvirtualLongMethodA(jobject obj, jclass clazz, jmethodID methodID, const…   jlong
CallNonvirtualLongMethodV(jobject obj, jclass clazz, jmethodID methodID, va_li…   jlong
CallNonvirtualObjectMethod(jobject obj, jclass clazz, jmethodID methodID, ....   jobject
CallNonvirtualObjectMethodA(jobject obj, jclass clazz, jmethodID methodID, c…   jobject
CallNonvirtualObjectMethodV(jobject obj, jclass clazz, jmethodID methodID, v…   jobject
CallNonvirtualShortMethod(jobject obj, jclass clazz, jmethodID methodID, ...)    jshort
CallNonvirtualShortMethodA(jobject obj, jclass clazz, jmethodID methodID, con…   jshort
CallNonvirtualShortMethodV(jobject obj, jclass clazz, jmethodID methodID, va_list args)
^↓ and ^↑ will move caret down and up in the editor >>
```

JNI data transfer

- A key part of JNI is a support to translate **primitive** native data types to java primitive data types

Java Type	Native Type	Description
boolean	**jboolean**	unsigned 8 bits
byte	**jbyte**	signed 8 bits
char	**jchar**	unsigned 16 bits
short	**jshort**	signed 16 bits
int	**jint**	signed 32 bits
long	**jlong**	signed 64 bits
float	**jfloat**	32 bits
double	**jdouble**	64 bits
void	**void**	N/A

JNI [data manipulation example]

- env->NewStringUTF(«hello»);

```
env->New
f NewBooleanArray(jsize length)                                   jbooleanArray
f NewByteArray(jsize length)                                        jbyteArray
f NewCharArray(jsize length)                                        jcharArray
f NewDirectByteBuffer(void *address, jlong capacity)                   jobject
f NewDoubleArray(jsize length)                                     jdoubleArray
f NewFloatArray(jsize length)                                       jfloatArray
f NewGlobalRef(jobject obj)                                            jobject
f NewIntArray(jsize length)                                         jintArray
f NewLocalRef(jobject ref)                                             jobject
f NewLongArray(jsize length)                                        jlongArray
f NewObject(jclass clazz, jmethodID methodID, ...)                     jobject
f NewObjectA(jclass clazz, jmethodID methodID, const jvalue *args)     jobject
f NewObjectArray(jsize length, jclass elementClass, jobject initialElemen…  jobjectArray
f NewObjectV(jclass clazz, jmethodID methodID, va_list args)           jobject
f NewShortArray(jsize length)                                       jshortArray
f NewString(const jchar *unicodeChars, jsize len)                      jstring
f NewStringUTF(const char *bytes)                                      jstring
f NewWeakGlobalRef(jobject obj)                                          jweak
f ThrowNew(jclass clazz, const char *message)                            jint
^↓ and ^↑ will move caret down and up in the editor >>                   π
```

Name convention

- The name of a native function is the concatenation of
  - the prefix Java_
  - the mangled fully-qualified class name
  - the **method name**

```
Java_com_example_macc_nativefirst_MainActivity_stringFromJNI(JNIEnv*
env,jobject /* this */) {

 return env->NewStringUTF("hello");


}
```

Example 1: understanding the template

This function is exported

```
#include <jni.h>
#include <string>
```

returned type

body of the function

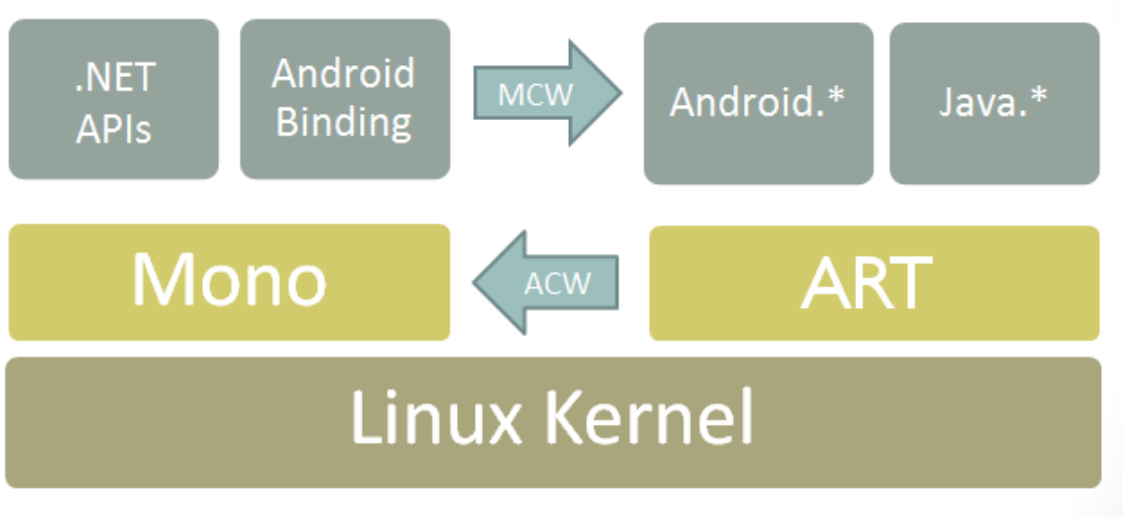
```
extern "C" JNIEXPORT jstring JNICALL

Java_com_example_macc_nativefirst_MainActivity_stringFromJNI(
JNIEnv* env,
jobject /* this */) {

 std::string hello = "Hello from C++";
 return env->NewStringUTF(hello.c_str());

}
```

Example 2: adding hidden native code

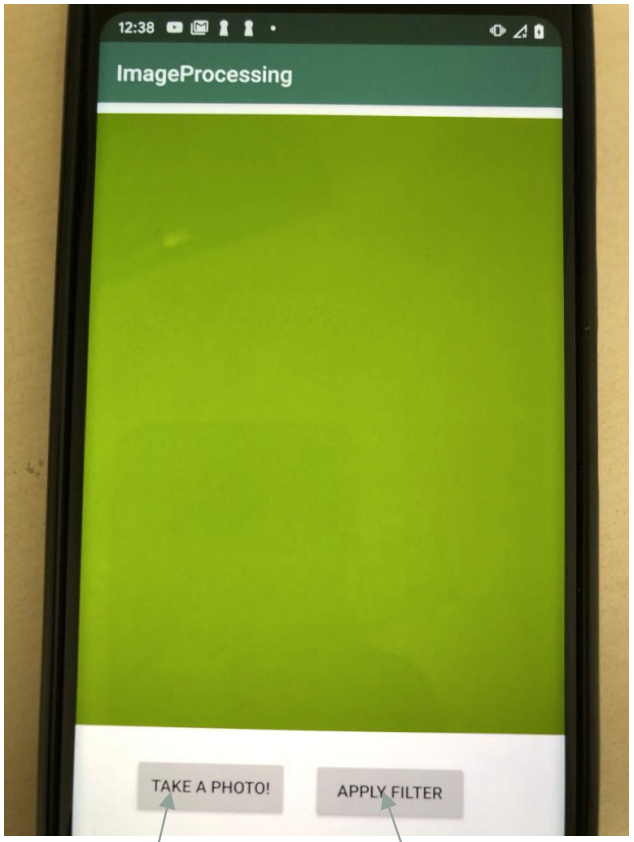
```
#include <jni.h>
#include <string>

int hidden_worker( ) {
    return 2;
}

extern "C" JNIEXPORT jstring JNICALL
Java_com_example_macc_nativefirst_MainActivity_stringFromJNI(
JNIEnv* env,
jobject /* this */) {

    int a = hidden_worker();
    std::string res = std::to_string(a);
    return env->NewStringUTF(res.c_str());

}
```

Xmarine (basic idea)

ACW = Android Callable Wrapper
MCW = Mono Callable Wrapper



An example: openCV



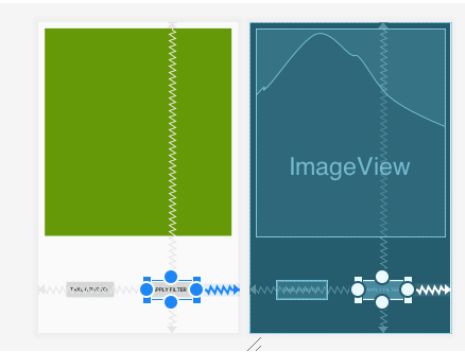
Take of a photo **Apply a filter (use OpenCV)**

- Step 1:
- Use cam to take a photo
- Set the taken photo as ImageView

- Permission to use the camera
- Intent for the camera
- Getting back the result

- Step 2:
- Install OpenCV
- Apply a blur filter to the image

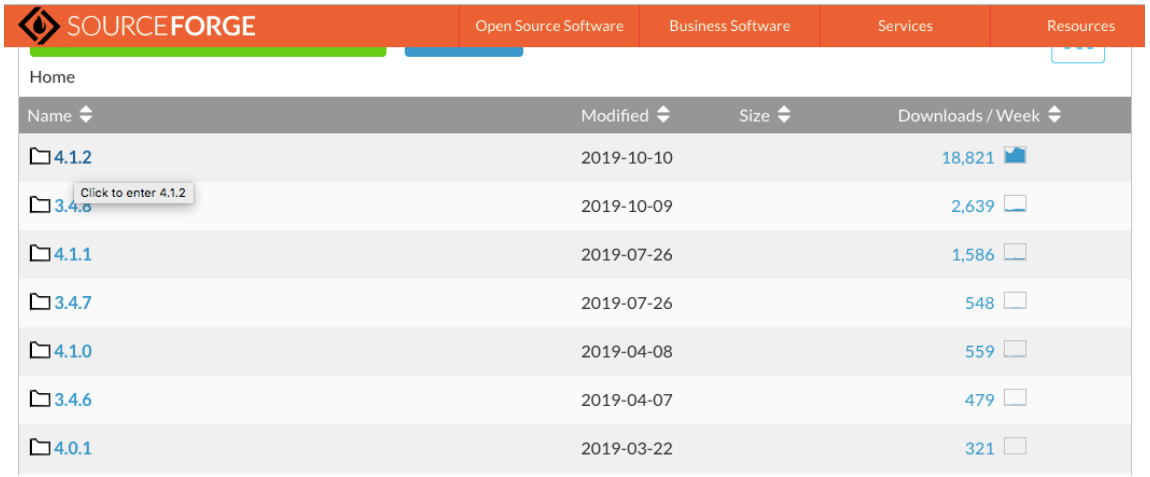
Step 1



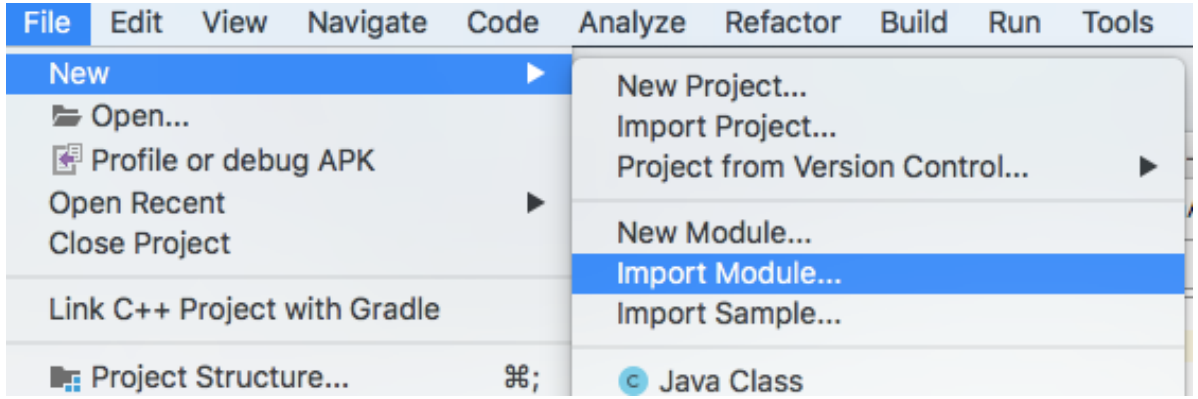

```
class MainActivity : AppCompatActivity() {


    val MY_CAMERA_REQUEST_CODE = 1

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (checkSelfPermission(Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED)
            arrayOf(Manifest.permission.CAMERA),MY_CAMERA_REQUEST_CODE)
    }

    fun takePhoto(v : View) {
        val takePicture = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        startActivityForResult(takePicture,MY_CAMERA_REQUEST_CODE)
    }

    fun applyFilter(v : View) {
        if ((v as Button).text=="Apply Filter") {
            v.setBackgroundColor(Color.BLUE)
            (v as Button).text="Remove Filter"
            return
        }
        (v as Button).text="Apply Filter"
        v.setBackgroundColor(Color.GRAY)

    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == MY_CAMERA_REQUEST_CODE && resultCode == RESULT_OK) {
            val extra = data?.extras
            val  bitmap :Bitmap =  extra?.get("data") as Bitmap
            imageView.setImageBitmap(bitmap)
        }

}
```

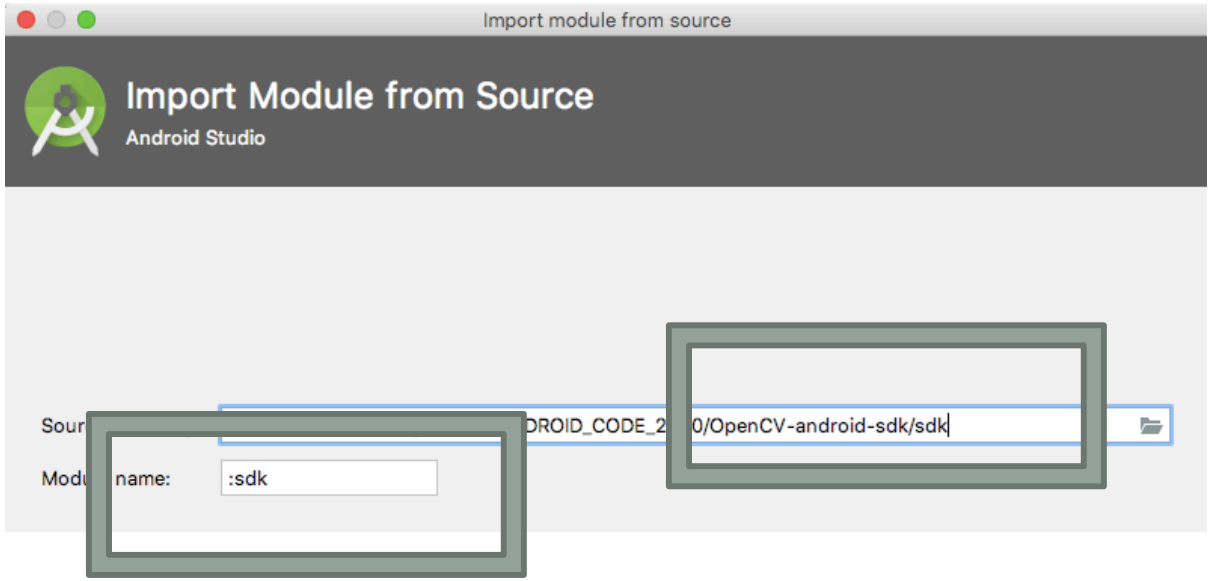
Step 2: 1/4 [download OpenCV sdk]

- Download and unzip OpenCV
- https://sourceforge.net/projects/opencvlibrary/files/4.1.2/

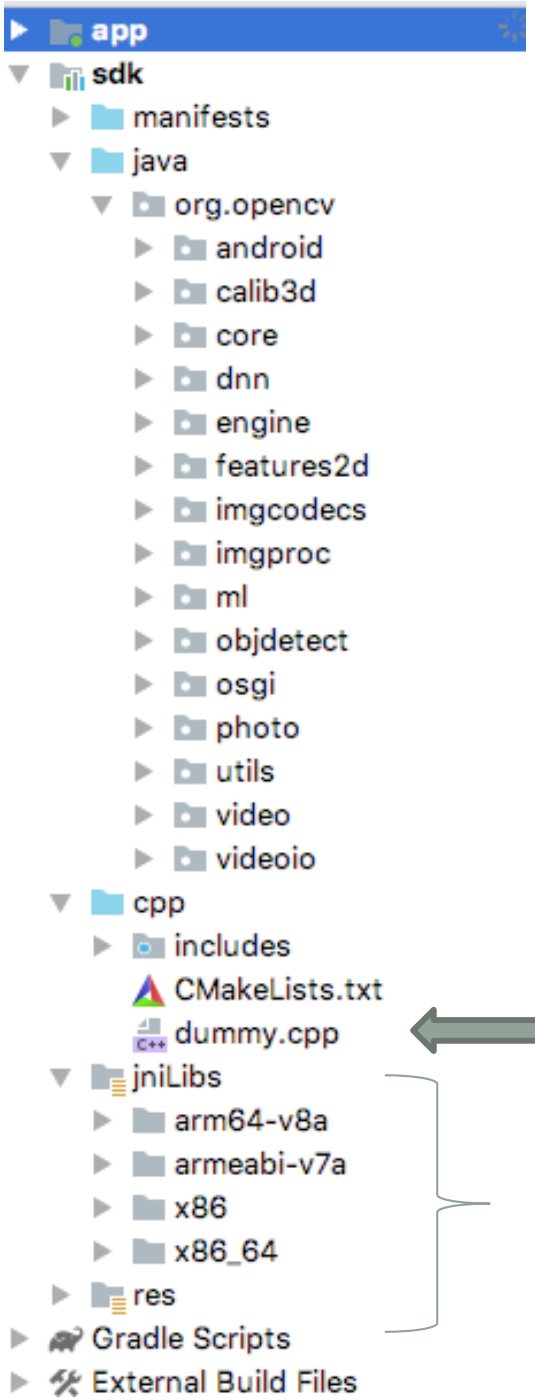


Step 2: 2/4 [adding OpenCV as a module]





Check result of the build



java packages

- Java to native method mapping
- Additional methods (e.g. load library. etc)

placeholder for additional native code

.so library compiled for 4 ABI

Step 2: 3/4 [change build.gradle of app]

build.gradle (Module: app)

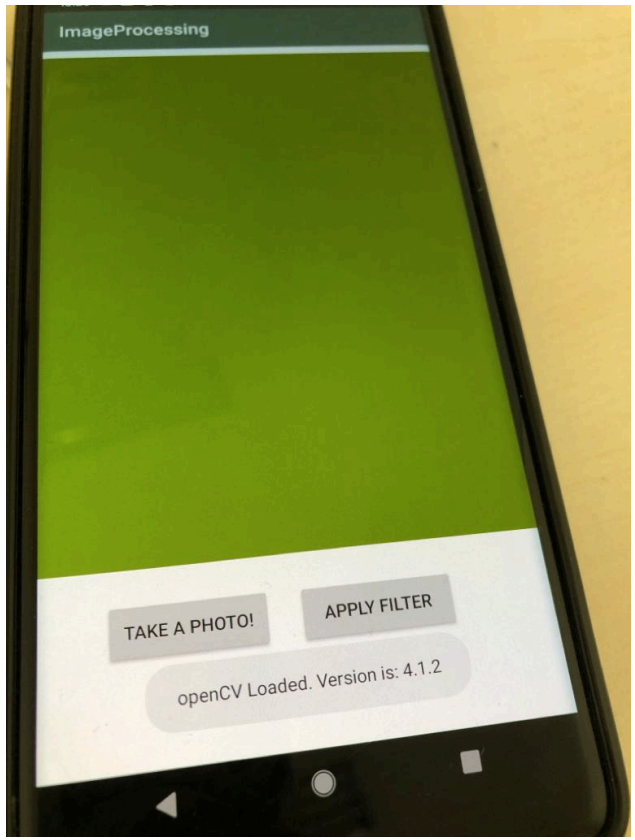
```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation project(':sdk')
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
```

```
import org.
class
    va
```

- opencv (org.opencv)
- apache (org.apache)
- intellij (org.intellij)
- jetbrains (org.jetbrains)
- json (org.json)

Step 2: 4/4 [load library]

```
if (OpenCVLoader.initDebug()){
//OK… print OpenCVLoader.OPENCV_VERSION
 }
```



Step 2: applying the blur filter

```
fun applyFilter(v : View) {
        var  src :Mat = Mat()
        var  dst :Mat = Mat()
        Utils.bitmapToMat(photo,src)
        val size = Size( width: 45.0,  height: 45.0)
        val point = Point( x: 20.0,  y: 30.0)
        Imgproc.blur(src, dst, size, point, Core.BORDER_DEFAULT)
        Utils.matToBitmap(dst,photo)
}
```