rgi | GEOSPATIAL

Going Cross Platform with Kotlin

FOSS4G St. Louis

Jenifer Cochran
Software Developer

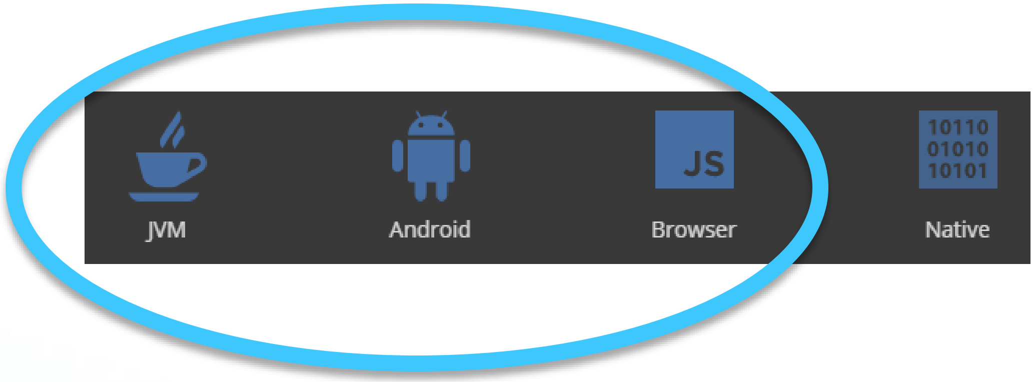
Jenifer.Cochran@rgi-corp.com

About Me

- High School Math Teacher turned developer
- Software developer at RGi for 3+ years
- Soccer Captain/Coach to Team Merkator
- First time presenter at FOSS4G
- Enjoys puns and dad jokes

What is Kotlin

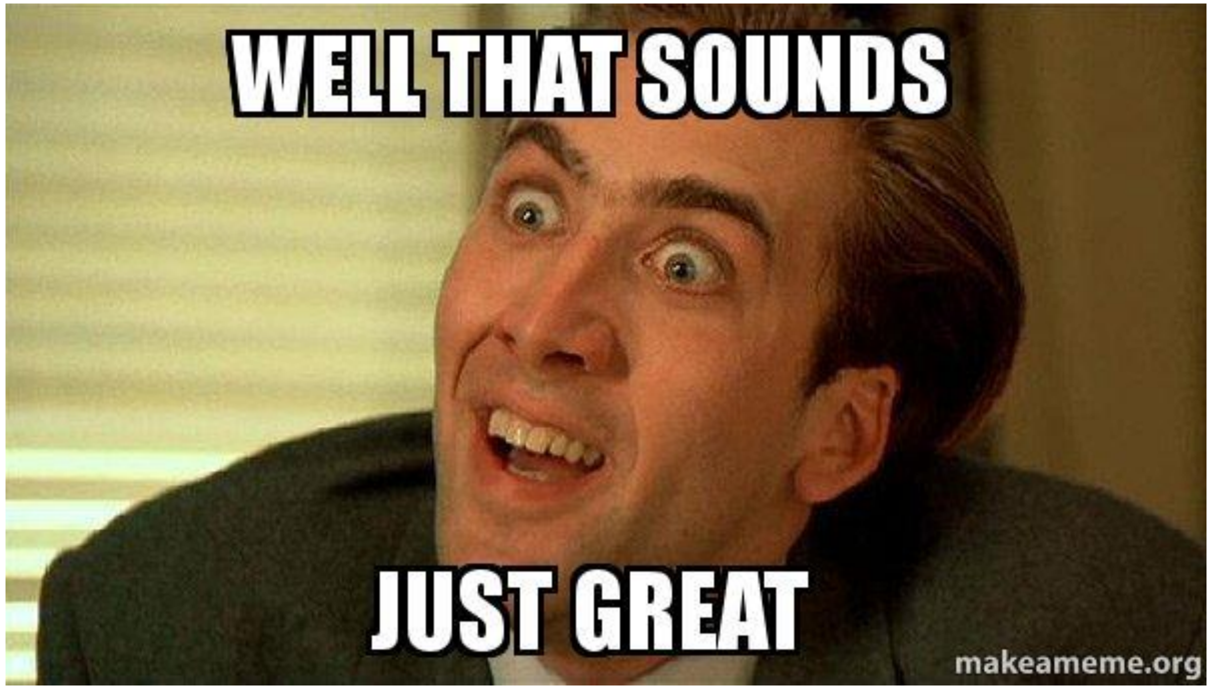
- Statically typed programming language for multiplatform applications



```
package hello

fun main(args: Array<String>)
{
        println("Hello World!")
}
```

Why choose Kotlin?

- Tool Friendly
- Concise
- Safe
- Interoperable



Tool Friendly

USE
Android Studio
Bundled with Studio 3.0, plugin available for earlier versions

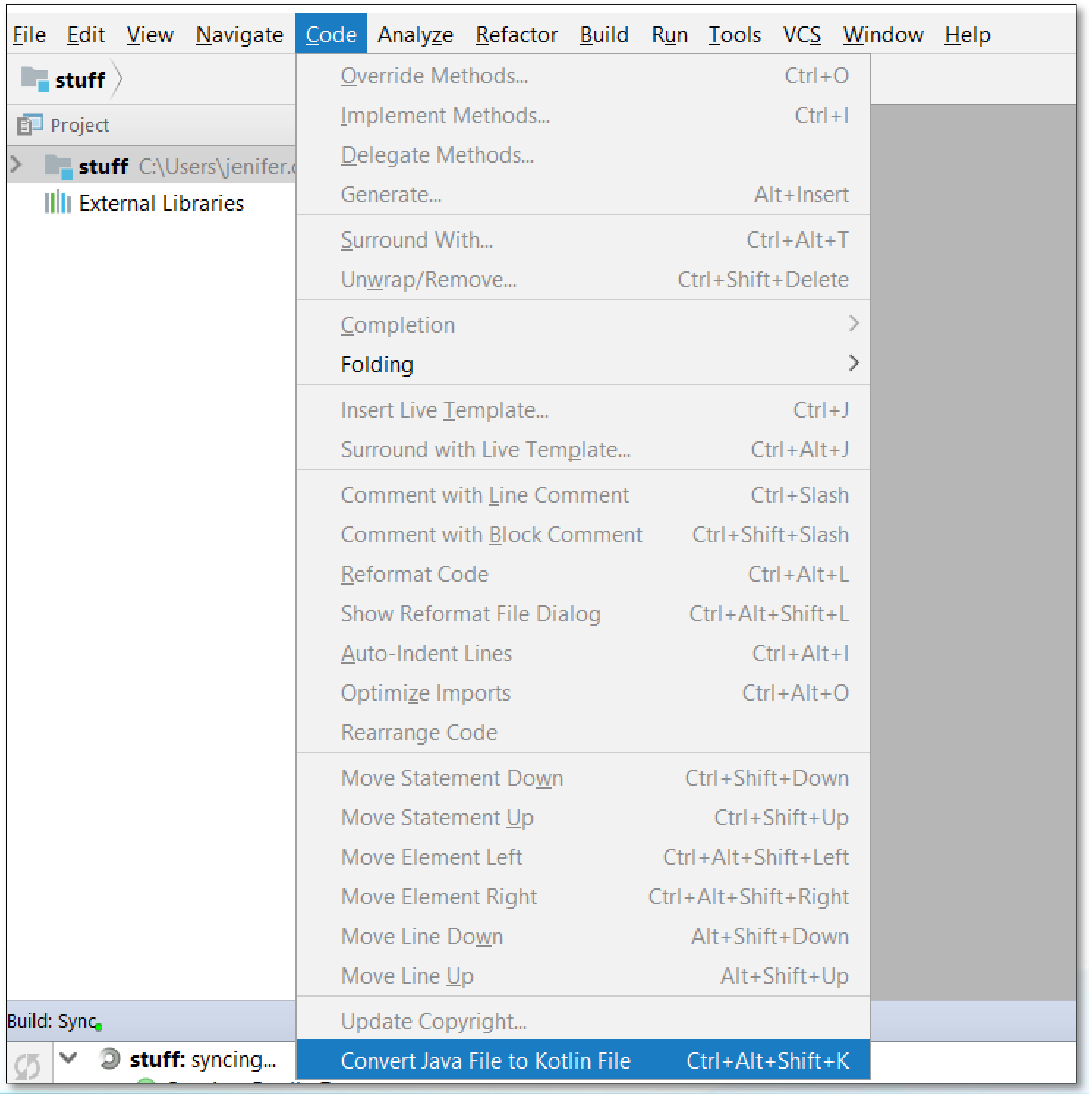
USE
Eclipse
Install the plugin from the Eclipse Marketplace

USE
IntelliJ
Bundled with Community Edition or IntelliJ IDEA Ultimate

USE
Command Line
Use any editor and build from the command line

Interoperability Tools

- Java to Kotlin converter tool
  - Built into Intellij
- Typescript to Kotlin converter tool
  - https://github.com/Kotlin/ts2kt


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
stuff
Project
stuff C:\Users\jenifer.
External Libraries
Override Methods... Ctrl+O
Implement Methods... Ctrl+I
Delegate Methods...
Generate... Alt+Insert
Surround With... Ctrl+Alt+T
Unwrap/Remove... Ctrl+Shift+Delete
Completion
Folding
Insert Live Template... Ctrl+J
Surround with Live Template... Ctrl+Alt+J
Comment with Line Comment Ctrl+Slash
Comment with Block Comment Ctrl+Shift+Slash
Reformat Code Ctrl+Alt+L
Show Reformat File Dialog Ctrl+Alt+Shift+L
Auto-Indent Lines Ctrl+Alt+I
Optimize Imports Ctrl+Alt+O
Rearrange Code
Move Statement Down Ctrl+Shift+Down
Move Statement Up Ctrl+Shift+Up
Move Element Left Ctrl+Alt+Shift+Left
Move Element Right Ctrl+Alt+Shift+Right
Move Line Down Alt+Shift+Down
Move Line Up Alt+Shift+Up
Update Copyright...
Convert Java File to Kotlin File Ctrl+Alt+Shift+K
Build: Sync
stuff: syncing...

Interoperable

- Java and Kotlin are 100% compatible
  - Call Kotlin from Java
  - Call Java from Kotlin
- Javascript dependencies can be pulled into Kotlin (More on how later)
- Call Inline Javascript from Kotlin
  - Otherwise known as a bad idea
- Kotlin can be exported as a .jar, .js, .apk

Java

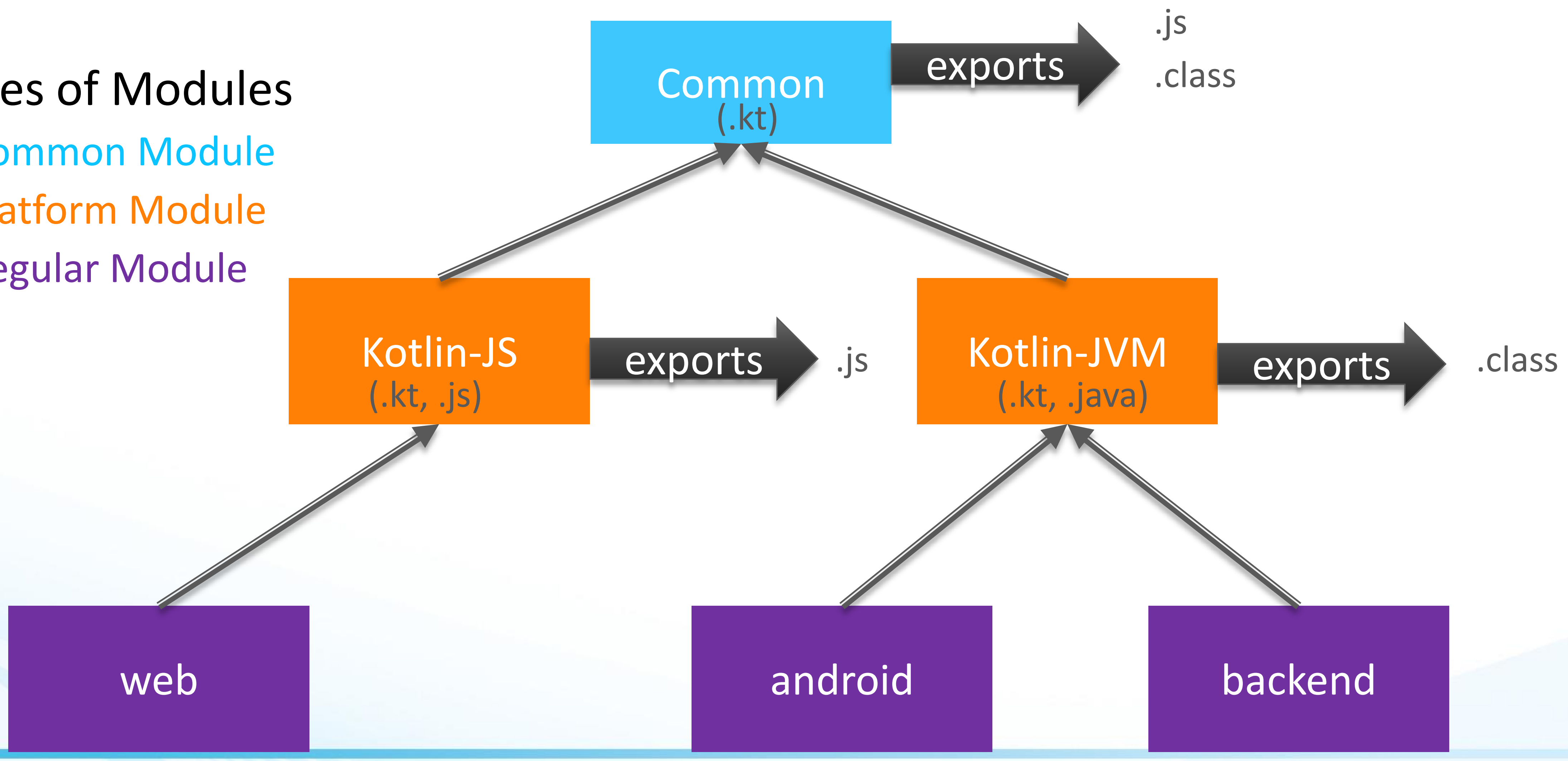
```java
public class Customer
{
    private String firstName;
    private String lastName;
    // standard setters and getters
}
```

Kotlin

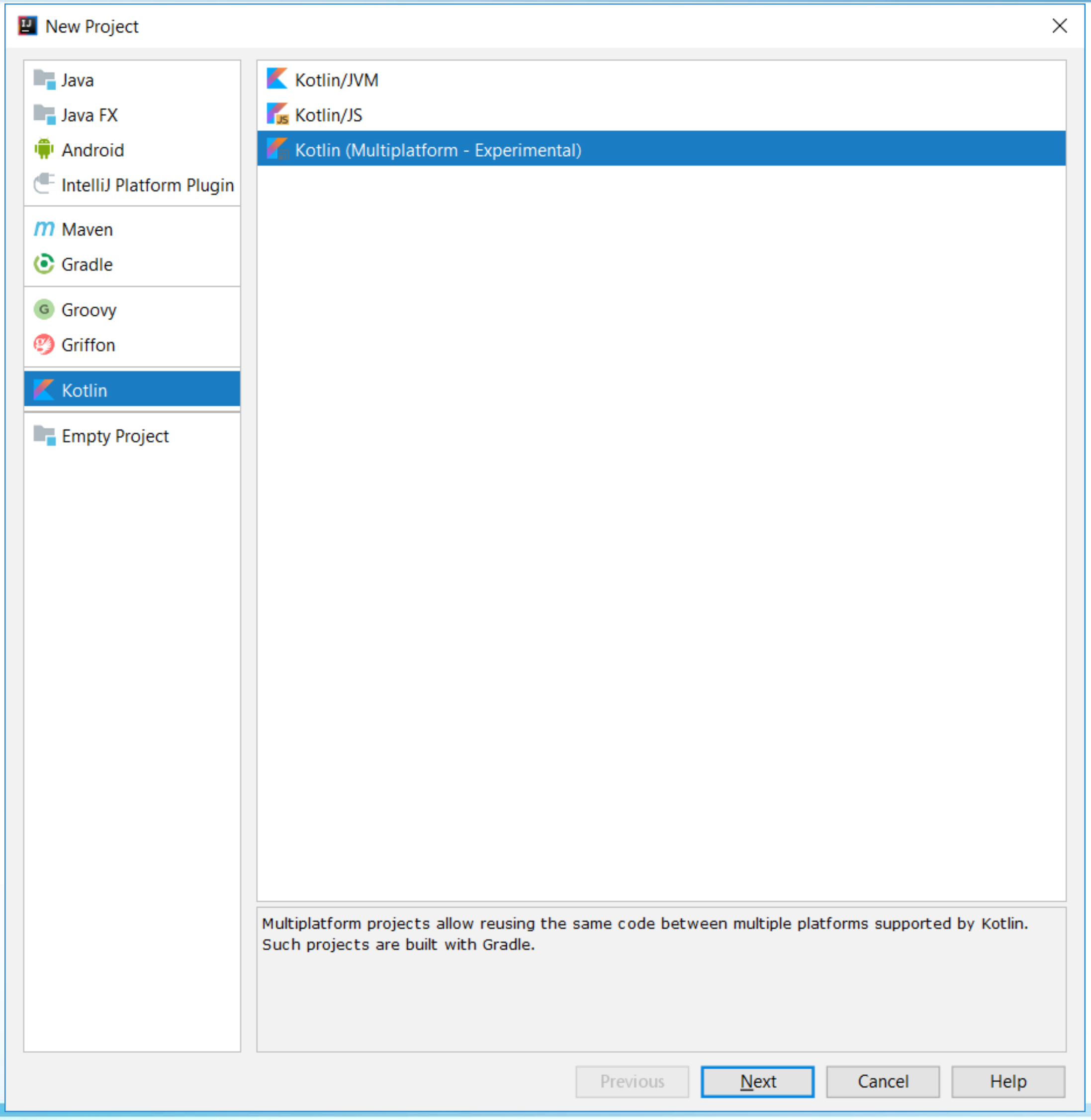
```kotlin
val customer = Customer()

customer.firstName = "Frodo"
customer.lastName = "Baggins"
```

Types of Modules

- Types of Modules
  - Common Module
  - Platform Module
  - Regular Module



Intellij sets it up for you!!!! phewwwww



Multiplatform projects allow reusing the same code between multiple platforms supported by Kotlin. Such projects are built with Gradle.

Common Module

- Kotlin Standard Library
  - Collections and sequences
  - Functional operations on collections (filter, map, etc)
  - Higher-order utility functions (with, apply, etc)
  - Exception
- Testing
- Higher-level infrastructure libraries

How do you build connections between Common Module and the Platform specific code?

Keywords expect & actual

- Expect keyword
  - Defines what the Common Module *expects* to use
  - Like Java's abstract class
  - No implementation needed in expect
- Actual keyword
  - Implementation in the Platform Specific Module
  - Package names must match

Common Module

```
package org.jetbrains.foo

expect class Foo(bar: String)
{
        fun frob()
}
```

Common Module's main class

```
fun main(args: Array)
{
        Foo("Hello").frob()
}
```

Example 1: Connection to Common Module

Common Module

```
package org.jetbrains.foo

expect class Foo(bar: String)
{
        fun frob()
}
```

```
package org.jetbrains.foo

actual class Foo actual constructor(val bar: String)
{
        actual fun frob()
         {
                println("$bar from JVM.")
         }
}
```

JVM Module

```
package org.jetbrains.foo

actual class Foo actual constructor(val bar: String)
{
        actual fun frob()
         {
                println("$bar from Javascript.")
         }
}
```

JS Module

Example 2: Reuse existing implementations

Common Module

```
package multiplatform.expected

expect class MpDate
{
      constructor()
}
```

```
package multiplatform.expected

actual typealias MpDate = java.util.Date;
```

JVM Module

```
package multiplatform.expected

actual typealias MpDate = kotlin.js.Date;
```

JS Module

How do you include Platform Specific dependencies?

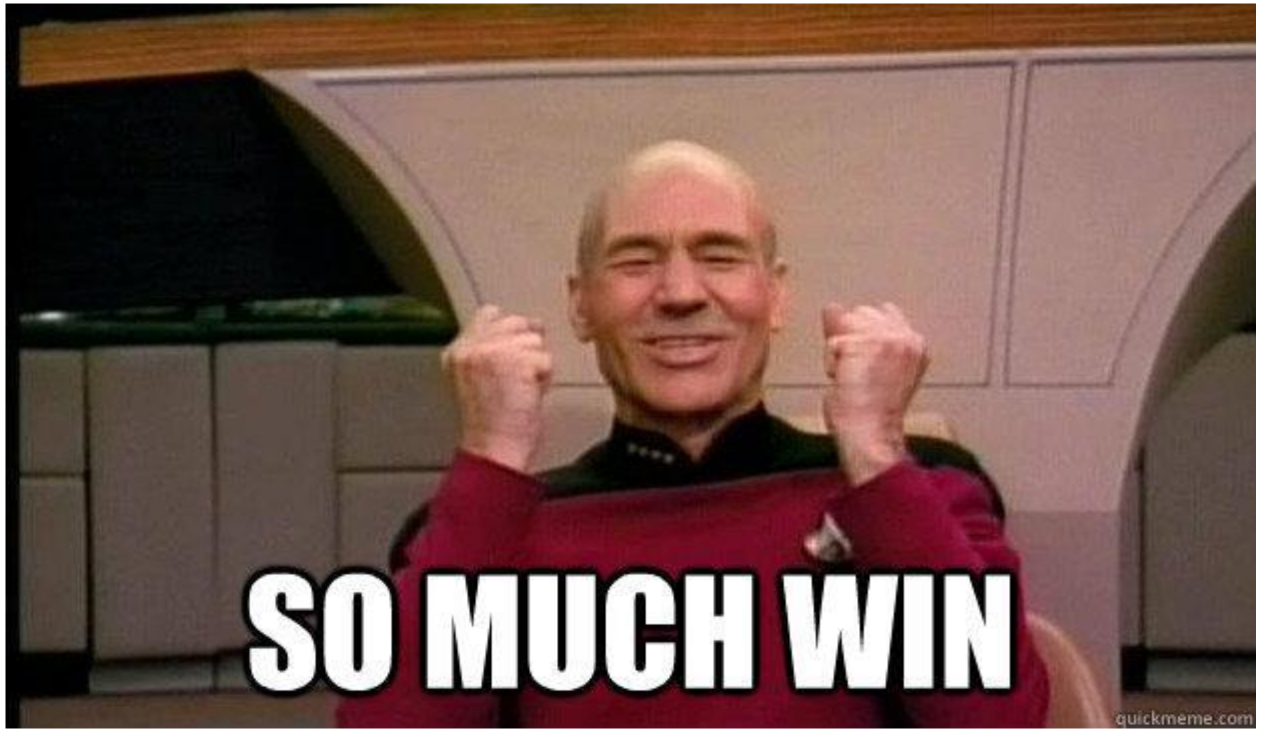
Kotlin-JVM module dependencies

How do you include Java Dependencies?

Let’s talk about dependencies

- JVM Module using platform specific APIs

- Simply include any dependencies in the gradle file for the JVM module



Kotlin-JS module dependencies

How do you include Javascript dependencies?

What about Javascript Dependencies?

- Cannot include .js files as a single instance
- Can call inline Javascript from Kotlin using **js(String value)**
  - Input is a string and it is not checked!
- Can call Javascript node packages from Kotlin*
  - Anything on NPM
- Can use the dynamic keyword
  - No type checking!

Kotlin File

```
js("console.log('Calling JavaScript')")
```

*May involve headaches

How to use Javascript Dependencies from NPM

- Create a class “interface” of the classes and methods you want to use from the Javascript Dependency written in Kotlin
- Use **@file:JsModule** and **@file:JsNonModule** keywords to indicate it is a Javascript Dependency
- Use the **external** keyword to indicate the implementation is not included
- Match the methods/classes to the Javascript dependency

Example of Javascript Dependency used in Kotlin

Kotlin File

```
@file:JsModule("leaflet")
// should match module name in the Gradle file
@file:JsNonModule()
// allows this to be used in CommonJS types as well
package leaflet

//Name of the package is CRS
external class CRS
{
//companion object means you do not have to have an
//instance of the class to call the following functions
companion object{
// the name of the function must match the one in JS
// The parameters and return values must match
// Any: if you do not want to strictly type the object
    fun scale(zoom: Any) : Any {
    }
  }
}
```

Javascript: Leaflet Module

```
export var CRS = {

// @method scale(zoom: Number):
Number

scale: function (zoom) {

  return 256 * Math.pow(2, zoom); }

}
```

Calling the Javascript dependency in Kotlin

Kotlin File

```kotlin
package js

import leaflet.CRS

fun main(arguments: Array<String>)
{
    var value = CRS.scale(1)
    println("Scale: " +  value)
}
```

JS Module Gradle file

```
apply plugin: 'kotlin2js' // converts Kotlin to JavaScript
apply plugin: 'org.jetbrains.kotlin.frontend' // for dependencies on
JavaScript modules and Bundling capabilities
// Configure bundle and JavaScript Dependencies
kotlinFrontend
{ // all dependencies hosted on NPM that this module needs when exported to
JavaScript
    npm
    {
        replaceVersion("kotlin-js-library", "1.1.0")
            dependency("leaflet") // JavaScript Module Dependency
    }
    // Bundles JavaScript dependencies with the exported JavaScript file
    webpackBundle
    {
        bundleName = "emp3-web" // name of the bundle JavaScript file
        publicPath = "/"         // web prefix
        port = 8080             // developer server port
        proxyUrl = ""           // URL to be proxied
    }
}
```

What about something with less boilerplate?

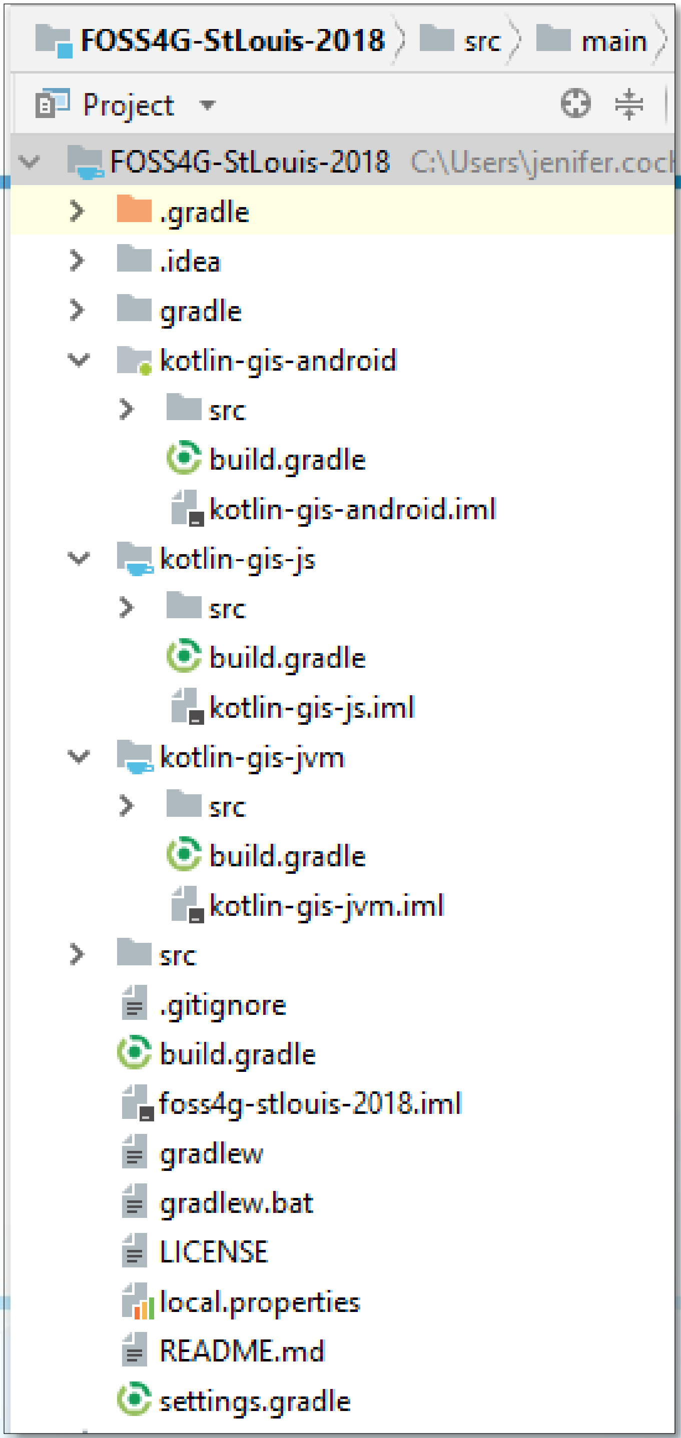
Calling Javascript code: unchecked

Kotlin File

```
val response: dynamic = loadJson("example.com/api")

val text = response.messages[0].text
```

Multiplatform Project Structure

- Root
  - **Platform JS Module**: Kotlin-js
    - src/main/kotlin
  - **Platform JVM Module**: Kotlin-jvm
    - src/main/kotlin
  - **Common Module**: src/main/kotlin
  - **Regular Module**: Kotlin-android
    - src/main/kotlin

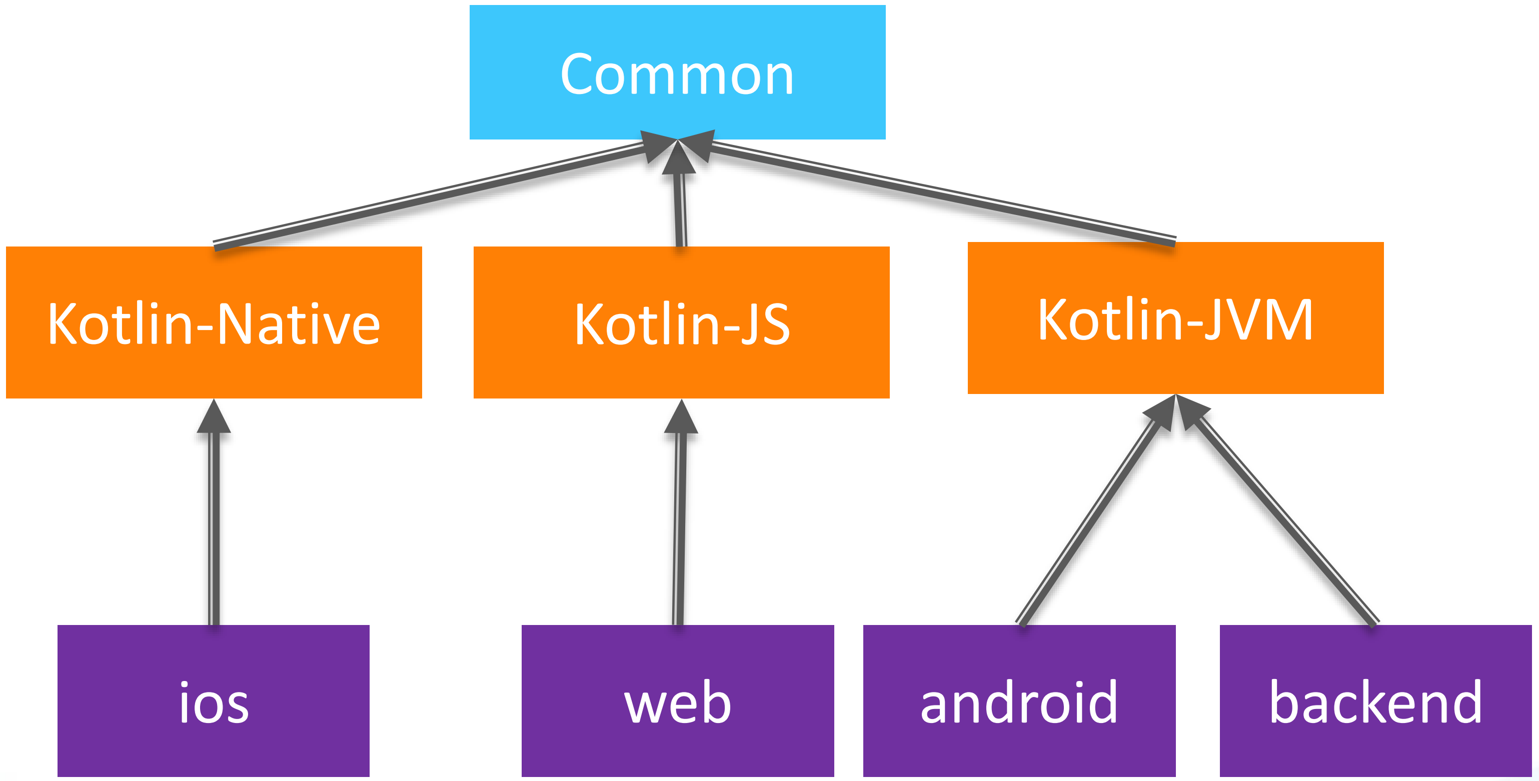


```
FOSS4G-StLouis-2018  C:\Users\jenifer.coc
  .gradle
  .idea
  gradle
  kotlin-gis-android
    src
    build.gradle
    kotlin-gis-android.iml
  kotlin-gis-js
    src
    build.gradle
    kotlin-gis-js.iml
  kotlin-gis-jvm
    src
    build.gradle
    kotlin-gis-jvm.iml
  src
  .gitignore
  build.gradle
  foss4g-stlouis-2018.iml
  gradlew
  gradlew.bat
  LICENSE
  local.properties
  README.md
  settings.gradle
```

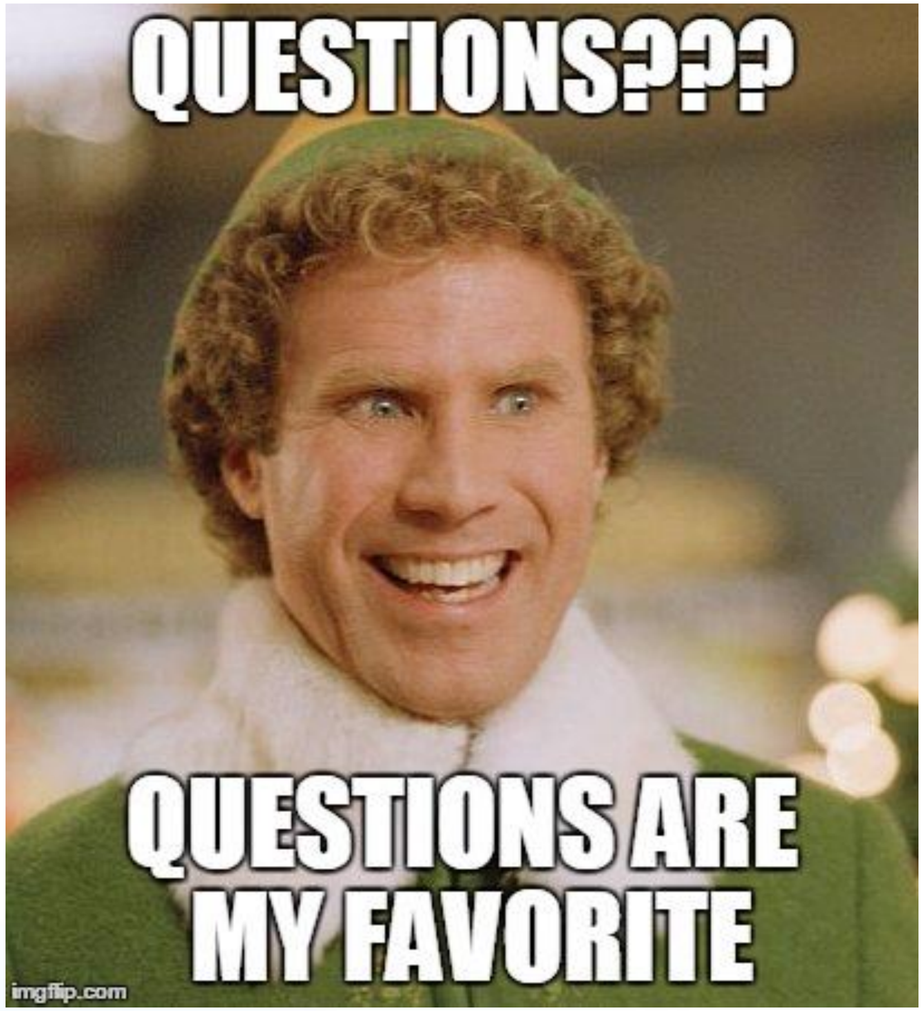
What other things I learned?

Kotlin Native

- technology for compiling Kotlin to native binaries that run without any VM
- Supported Platforms
  - Windows (x86_64 only at the moment)
  - Linux (x86_64, arm32, MIPS, MIPS little endian)
  - MacOS (x86_64)
  - iOS (arm64 only)
  - Android (arm32 and arm64)
  - WebAssembly (wasm32 only)
- Future:
  - Support for multiplatform project structure!



Questions? Thanks for listening!!



Jenifer Cochran
Software Developer

Jenifer.Cochran@rgi-corp.com

rgi | GEOSPATIAL

Slides & Code: https://github.com/GitHubRGI/FOSS4G-StLouis-2018