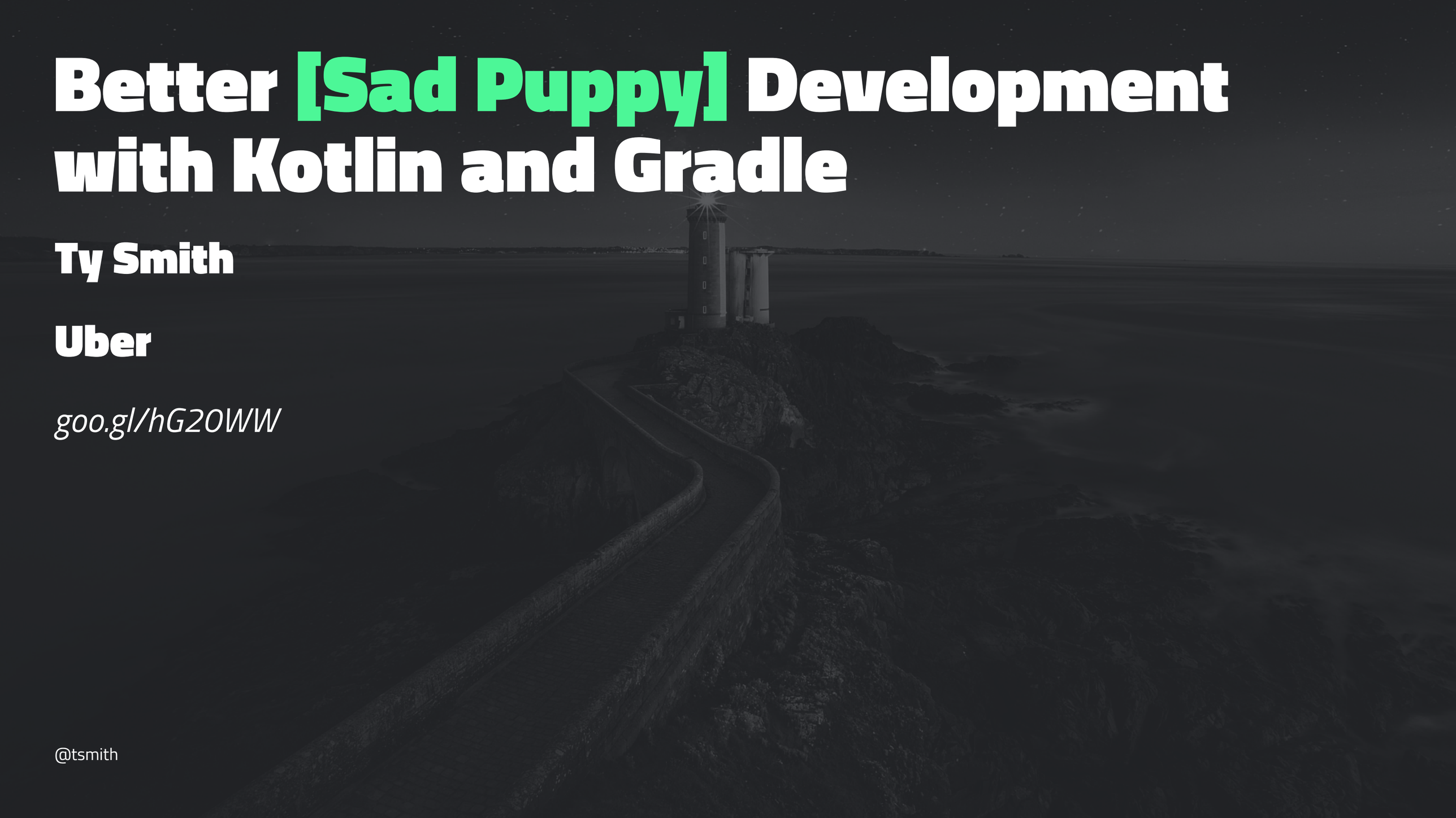
Better [Sad Puppy] Development with Kotlin and Gradle

Ty Smith

Uber

goo.gl/hG20WW



@tsmith

But I like Java!

- We're stuck in a Java purgatory
- Error prone Code (no non-capturing anonymous inner classes)
- General Java language restrictions

How about Groovy?

- Groovy is dynamic
- Harder for IDEs to parse and infer
- Won't ever be consistent with Android

What is Kotlin?

- Built by Jetbrains and Open Source
- Official Gradle support
- Java 6 bytecode compatible
- Small standard library (625kb)
- Statically typed with no runtime overhead
- Modern Language

How can Kotlin help today?

- Reduce common errors
- Reduce boilerplate
- Improve Android API
- Can achieve consistency in the app and build

Kotlin Features

Higher-order functions, properties, mixins and delegation, extension functions, static nullability checking, automatic casts, reified generics, declaration-site variance, modules and build infrastructure, inline-functions (zero-overhead closures), operator overloading, String interpolation, pattern matching, first class IDE support with Java converter, default parameters, infix methods, and more...

Nullability

"I call it my billion-dollar mistake... [which] has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."
— Sir Charles Antony Richard Hoare

Nullability

```
var a: String = "foo" //Reversed Type and name
a = null //Compilation error
```

Nullability

```
var b: String? = "bar"
b = null //ok

val l = b.length //Compilation error
```

Nullability

```
val l = b?.length //Value null

//With a default value
val l = if (b != null) b.length else -1 //Value -1

//or with the Elvis operator
val l = b?.length ?: -1 //Value -1
```

Properties

```
class User(var name: String) {
...
}

fun setName(name: String): User {
  val user = User() // there's no 'new' keyword in Kotlin
  user.name = "Ty" //This is using a method, not a field
  return user
}
```

Properties

```
class User {
  var name: String = ...
    get() = name // Can declare methods inline
    set(value) {
        name = value().toLowerCase()
    }
}
```

Data class

```
data class User(name: String) {
}
```

- equals() and hashCode()
- toString() of the form "User(name=John)",
- copy()

Function Literal (Lambda)

Undeclared function bodies used as an expression (ie, as data).

```
val sum: (Int, Int) -> Int = {x, y -> x+y }
val anAnswer = sum(6,7) //Variable type is inferred


//Or use "it" to get the inferred parameter


val double: (Int) -> Int = {it*2}
val theAnswer = double(21)
```

Higher Order Functions

A higher-order function is a function that takes functions as parameters, or returns a function

```
fun apply(one: Int, two: Int, func: (Int, Int) -> Int) = func(one, two)
//Kotlin has great generic support to simplify this too!

val multiply = apply(6, 7) {x, y -> x * y}
val difference = apply(44, 2) {x, y -> x -y}
```

Extension Functions

Functions added to a type without modifying the original.

```
fun Int.differenceFromTheAnswer(): Int {
    return 42 - this
}

//Or the Single Expression Function version
fun Int.differenceFromTheAnswer(): = 42 - this

//Usage
val difference = 2.differenceFromTheAnswer() //40
```

Extension Functions

Java Interoperability

```
//Example 1:
@file:JvmName("IntUtil")
fun Int.differenceFromTheAnswer(): = 42 - this

//Example 2:
@file:JvmName("Util")
@file:JvmMultifileClass
fun Int.differenceFromTheAnswer(): = 42 - this
```

Extension Functions in Android

```
SharedPreferences.Editor editor;
editor = getSharedPreferences(MODE_PRIVATE).edit();
editor.putString("login_token", token);
editor.apply();
```

Extension Functions in Android

Also Higher Order functions and Function Literals!

```
fun SharedPreferences.edit(func: SharedPreferences.Editor.() -> Unit) {
        val editor = SharedPreferences.Editor()
        editor.func()
        editor.apply()
}

getSharedPreferences(MODE_PRIVATE).edit {
    putString("login_token", token)
}
```

Inline Functions

```
//All the memory allocation
fun SharedPreferences.edit(func: SharedPreferences.Editor.() -> Unit)

//inlines into java byte code, matching the java signature
inline fun SharedPreferences.edit(func: SharedPreferences.Editor.() -> Unit)
```

Builders

Anko

```
verticalLayout {
    padding = dip(16)
    textView("Username:") {
        textSize = 18f
    }.layoutParams { verticalMargin = dip(4) }

    val login = editText()

    button("Sign up") {
        textSize = 20f
        onClick { login(login.text) }
    }.layoutParams { topMargin = dip(8) }
}
```

Kotlin -> Gradle

@tsmith

Installing Kotlin

```
buildscript {
  ext.kotlin_version = '1.0.1'

  repositories { mavenCentral() }
  dependencies {
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
  }
}

apply plugin: 'kotlin'
repositories { mavenCentral() }
dependencies {
  compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
}
```

build.gradle.kts

```
buildscript {
    repositories {
        jcenter()
        gradleScriptKotlin()
    }
    dependencies { classpath("com.android.tools.build:gradle:2.2.0") }
}

apply<AppPlugin>()

android {
    buildToolsVersion("23.0.3")
    compileSdkVersion(23)
    defaultConfigExtension {
        setMinSdkVersion(15)
        setTargetSdkVersion(23)
        applicationId = "com.example.kotlingradle"
        versionCode = 1
        versionName = "1.0"
    }
}
```

build.gradle.kts Extension Functions

```
fun Project.android(setup: AppExtension.() -> Unit) = the<AppExtension>().setup()

fun NamedDomainObjectContainer<BuildType>.release(setup: BuildType.() -> Unit) = findByName("release").setup()

fun AppExtension.defaultConfigExtension(setup: DefaultProductFlavor.() -> Unit) = defaultConfig.setup()

fun AppExtension.buildTypesExtension(setup: NamedDomainObjectContainer<BuildType>.() -> Unit) = buildTypes { it.setup() }

fun DefaultProductFlavor.setMinSdkVersion(value: Int) = setMinSdkVersion(value.asApiVersion())

fun DefaultProductFlavor.setTargetSdkVersion(value: Int) = setTargetSdkVersion(value.asApiVersion())

fun Int.asApiVersion(): ApiVersion = DefaultApiVersion.create(this)
```

The case of API Keys

```
public class SampleApplication extends Application {
    private static final String CONSUMER_KEY = "PRODUCTION_KEY";
    private static final String CONSUMER_SECRET = "PRODUCTION_SECRET";
    @Override
    public void onCreate() {
        super.onCreate();
        SessionConfig authConfig = new SessionConfig(CONSUMER_KEY, CONSUMER_SECRET);
        Sdk.init(authConfig);
    }
}
```

The case of API Keys

```
private static final String CONSUMER_KEY = "PRODUCTION_KEY";
private static final String CONSUMER_SECRET = "PRODUCTION_KEY";


...
final SessionConfig authConfig
        = new SessionConfig(CONSUMER_KEY, CONSUMER_SECRET);
```

Enter Gradle

```
android {
    buildTypes {
        release {
            buildConfigField 'String', 'CONSUMER_KEY', "\"PRODUCTION_KEY\""
            buildConfigField 'String', 'CONSUMER_SECRET', "\"PRODUCTION_SECRET\""
        }
        debug {
            buildConfigField 'String', 'CONSUMER_KEY', '\"DEBUG_KEY\"'
            buildConfigField 'String', 'CONSUMER_SECRET', '\"DEBUG_SECRET\"'
        }
    }
    ...
```

BuildConfig class

```
public final class BuildConfig {
  public static final String CONSUMER_KEY = "PRODUCTION_KEY";
  ...

public class AwesomeApplication extends Application {
    private static final String CONSUMER_KEY = BuildConfig.CONSUMER_KEY;
```

Gradle properties

- Gradle (Project) Properties
- System Variables
- Custom properties file

Provided values

```
def getProp(String key) {
    if (System.properties.get(key) != null) {
        return System.properties.get(key)
    } else if (project.hasProperty(key)) {
        return project.property(key)
    }
    throw new GradleException("${key} is not available")
}
```

Using the provided value

```
//Using provided value via getProp function
release {
    buildConfigField "String", "CONSUMER_KEY", "${getProp("INJECTED_KEY")}"
...
```

custom.properties

```
INJECTED_KEY=PRODUCTION_KEY
INJECTED_SECRET=PRODUCTION_SECRET
...
```

Custom Task

```
task createCustomPropertiesFile << {
    def props = new Properties()
    props.put('INJECTED_KEY', getProp('INJECTED_KEY'))
    props.put('INJECTED_SECRET', getProp('INJECTED_SECRET'))

    file('custom.properties').withOutputStream {
        props.store(it, "")
    }
}
```

Custom Properties file

```
tasks << 'createCustomPropertiesFile' {
    ...
}

$ ./gradlew createCustomPropertiesFile -PINJECTED_KEY=DEV_KEY
```

Gradle plugin

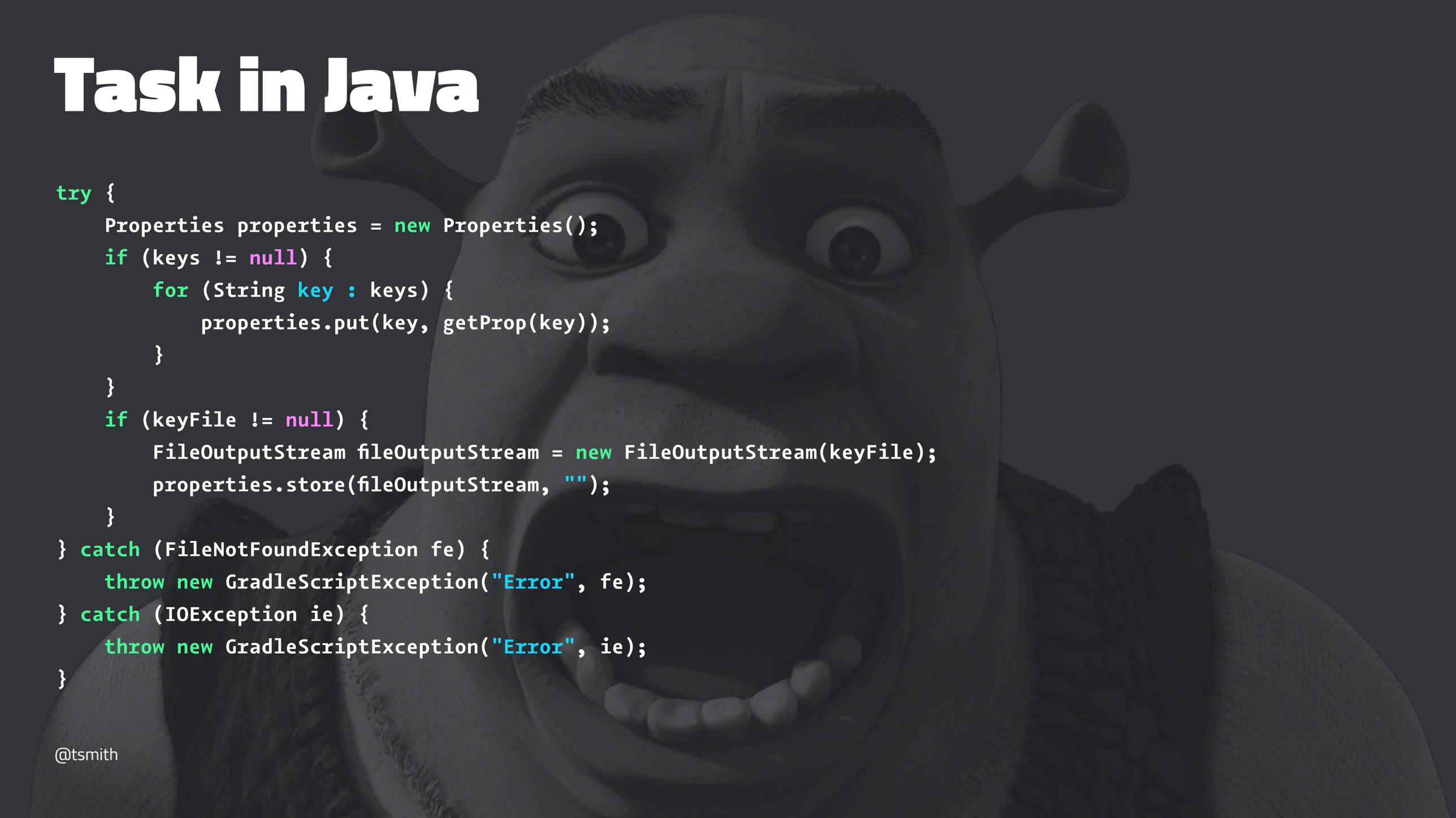
Basic ingredients:

- Task
- Extension
- Plugin

Task

```
class CreatePropertiesFileTask extends DefaultTask {
    @TaskAction
    void createFile() {
        Properties properties = new Properties()
        keys?.each {
            properties.put(it, getProp(it))
        }
        outputFile?.withOutputStream {
            properties.store(it, "")
        }
    }
}
```

Task in Java



```
try {
    Properties properties = new Properties();
    if (keys != null) {
        for (String key : keys) {
            properties.put(key, getProp(key));
        }
    }
    if (keyFile != null) {
        FileOutputStream fileOutputStream = new FileOutputStream(keyFile);
        properties.store(fileOutputStream, "");
    }
} catch (FileNotFoundException fe) {
    throw new GradleScriptException("Error", fe);
} catch (IOException ie) {
    throw new GradleScriptException("Error", ie);
}
```

@tsmith

Task

```
//
    Properties properties = new Properties();
    //
        for (String key : keys) {
            properties.put(key, getProp(key));
        }
    //
    //
        FileOutputStream fileOutputStream = new FileOutputStream(keyFile);
        properties.store(fileOutputStream, "");
    //
//
    //
//
    //
```

Task

```
class CreatePropertiesFileTask extends DefaultTask {
    @Input List<String> keys
    @OutputFile File outputFile

    ...
}
```

Extension

```
class PropertyHolderExtension {

    List<String> keys = []
    File customPropertiesFile
    Project project

    PropertyHolderExtension(Project project) {
        this.project = project
    }
```

Extension

```
class KeyHolderExtension {
    ...
    void keyFile(String filename) {
        keyFile = new File(filename)
    }

    def getProp(String key) {
        if (System.properties.get(key) != null) {
            return System.properties.get(key)
        } else if (project.hasProperty(key)) {
    ...
```

Plugin

```
class PropertyHolderPlugin implements Plugin<Project> {

    @Override
    void apply(Project project) {
        def propertyHolder = project.extensions.create(
                "propertyHolder", PropertyHolderExtension, project)
```

Plugin

```
//Example 1. Without configure
def propertyHolder = project.extensions.create(...)

//Example 2. With configure
project.configure(project) {
    def propertyHolder = extensions.create(...)
```

Plugin

```
project.configure(project) {
    ...
    afterEvaluate {
        if (!propertyHolder.keys.isEmpty()) {
            project.tasks.create("createCustomPropertiesFile",
                                 CreatePropertiesFileTask) {
                it.keyFile = propertyHolder.customPropertiesFile
                it.keys = propertyHolder.keys
            }
        }
    }
    ...
```

Consuming the plugin

```
apply 'custom-properties'

propertyHolder{
    keys = 'CONSUMER_KEY',...
    keyFile 'custom.properties'
}
```

Kotlin Version - Task

```
open class CreatePropertiesFileTask() : DefaultTask() {
    @Input var keys: List<String> = ArrayList()

    @OutputFile var keyFile: File? = null
```

Kotlin Version - Task

```
@TaskAction
fun createFile() {
    val properties = Properties()
    for (key in keys) {
        properties.put(key, getProp(key))
    }

    val fileOutputStream = FileOutputStream(keyFile)
    properties.store(fileOutputStream, "")
}
```

Kotlin Version - Task

```
@TaskAction
fun createFile() {
    val properties = Properties()
    keys.forEach { key -> properties.put(key, getProp(key)) }
    properties.store(keyFile!!.outputStream(), "")
```

Extension - Kotlin version

```
data class PropertyHolderExtension(val project: Project,
                                   var keys: List<String> = ArrayList(),
                                   var customFile: File? = null)
```

Plugin - Kotlin Version

```
class PropertyHolderPlugin : Plugin<Project> {
    override fun apply(project: Project) {
        project.extensions
                .add("kotlinHolder",PropertyHolderExtension(project))
```

Plugin - Kotlin Version

```
class PropertyHolderPlugin : Plugin<Project> {
    override fun apply(project: Project) {
        project.extensions
                .add("kotlinHolder", PropertyHolderExtension(project))

        project.afterEvaluate { project ->
            val extension = project.extensions
                                   .getByName("kotlinHolder")
                                as PropertyHolderExtension
        }
```

Sprinkle it with Extension Functions

```
fun Project.configure(exec: Project.() -> Unit) = exec()

fun Project.extensions(exec: ExtensionContainer.() -> Unit) = extensions.exec()

fun Project.propertyHolder() = extensions.getByName("kotlinHolder") as PropertyHolderExtension
```

Kotlin Version - Plugin

```
//before
project.extensions
        .add("kotlinHolder", PropertyHolderExtension(project))


//after
project.configure {
    extensions {
        add("kotlinHolder", PropertyHolderExtension(project))
    }
```

Kotlin Version - Plugin

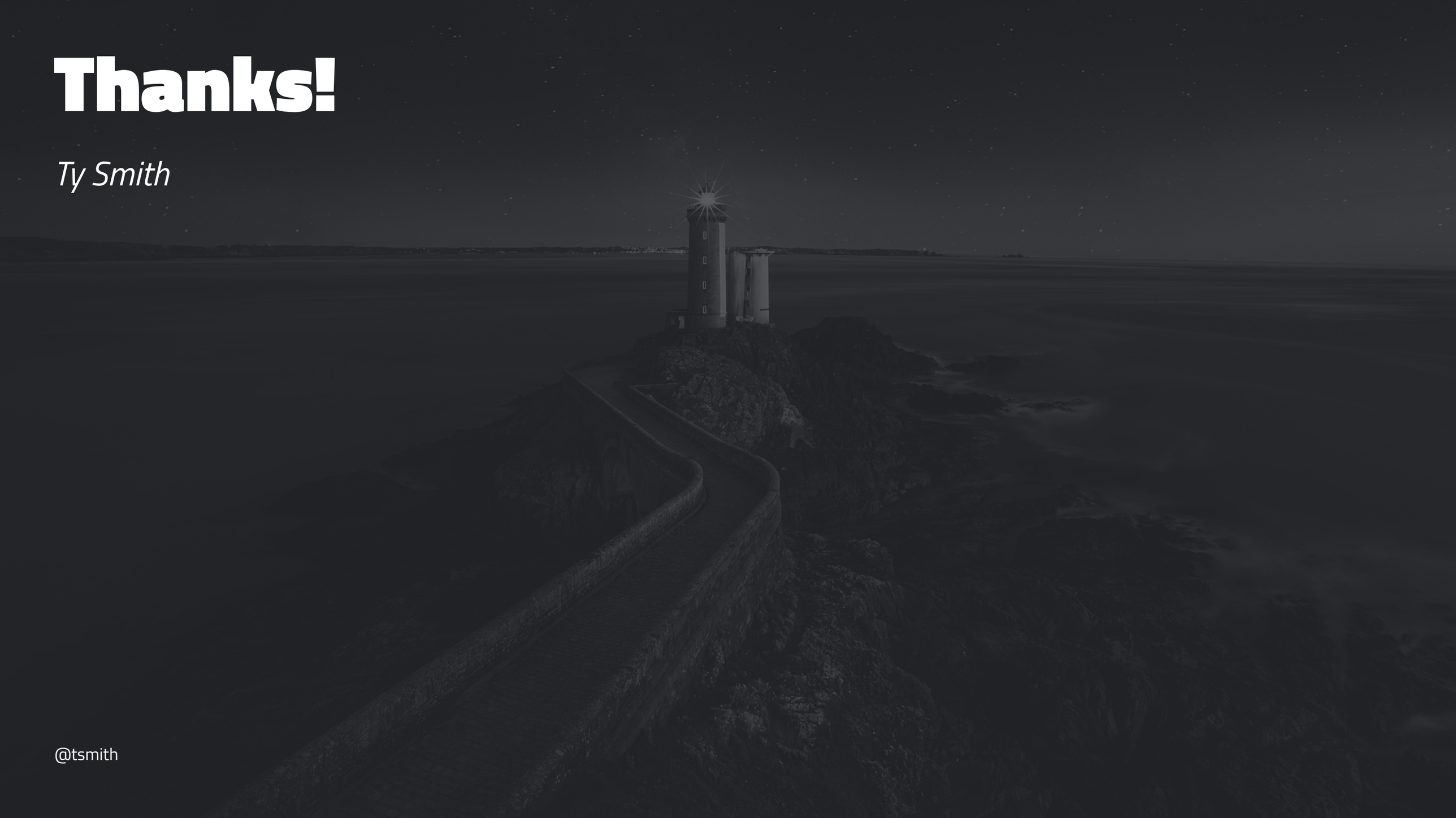
```
project.configure {
    extensions {
        add("kotlinHolder", PropertyHolderExtension(project))
    }
    afterEvaluate { project ->
        if (!propertyHolder().keys.isEmpty()) {
            tasks.create("createPropertiesFile",
                    CreatePropertiesFileTask::class.java) { task ->
                task.keys = propertyHolder().keys
                task.keyFile = propertyHolder().customFile!!
            }
        }
    }
}
```

Resources

- Kotlin docs
- Kotlin Koans
- Anko
- Gradle Script Kotlin
- Advancing Android Development with the Kotlin Language
- This project

Thanks!

Ty Smith



@tsmith