

みんなでKotlinを始めよう！

日本Androidの会 浜松支部
第74回ミーティング



2017年7月8日(土)
まーくん@macha1972

自己紹介

氏名 市川 雅明(まーくん@macha1972)

会社 株式会社オルトア

開発 組み込み/Windows/Androidアプリ開発
C/C++, Java, PHPなど
Kotlin初心者

所属 日本Androidの会 浜松支部

特技 スべること(自称：氷上のプログラマ)

今日のミッション

- Kotlin概要
- Kotlin環境構築
- Hello Kotlin!

Kotlinの概要

- KotlinとはJetBrains社が開発したオブジェクト思考プログラミング言語です。
- Goole I/O 2017でKotlinがAndroidの公式言語になることが発表されました。

- JVM(Java仮想マシン)上で動作する
JVM言語です。
- Android, JavaScriptで動作します。

Kotlinの特徴

- オブジェクト思考
- JVM言語(JVMで動作)
- 簡潔な記述(文末にセミコロン不要)
- null安全
- 静的型付け
- ラムダ式
- インライン展開
- デフォルト引数
など

JVM言語

- JVMマシンで動作します。

- KotlinとJavaと相互運用できます。

Javaで書かれたコードをKotlinで呼び出すことも、Kotlinで書かれたコードをJavaで呼び出すこともできます。

- JVM言語なのでKotlin→Javaにデコンパイル
できます。

[Tools] – [Kotlin] – [Show Kotlin Bytecode]
を選択して[Decompile]ボタンを押下します。

基本の型

型	種類	ビット数	リテラル
Double	浮動小数点	64	123.5 123.5e10
Float	浮動小数点	32	123.5f 123.5F
Long	整数	64	123L
Int	整数	32	123
Short	整数	16	123
Byte	整数	8	0x0f 0b00001011
Boolean	真偽値	-	true false
Char	文字	-	'a' '0'
String	文字列	-	"Hello"

※Javaのint, doubleなどはない。

<https://kotlinlang.org/docs/reference/basic-types.html>

基本的なコード

- 文末にセミコロンは不要

```
var a: String = "Hello" // セミコロン不要  
val b: Int = 123
```

- コメントは、// or /* */

```
// 1行コメント
```

```
/*  
    ブロックコメント  
*/
```

- 変数の宣言

varは変更可能変数

valは変更不可変数(再割り当て不可)

→1回のみ初期化できる

→Javaのfinalに相当

```
var a: String = "Hello" // 変更可能
```

```
val b: Int = 123 // 変更不可
```

- if – else

```
var c: String = ""  
if (b == 10) {  
    c = "abc"  
} else {  
    c = "xyz"  
}
```

- 式として扱えるので最後に評価した値を代入できる

```
var c: String = if (b == 10) {  
    "abc"  
} else {  
    "xyz"  
}
```

<https://kotlinlang.org/docs/reference/control-flow.html#if-expression>

- while, do - whileループ

```
var d:Int = 10
while (d > 0) {
    d--
}
```

```
do {
    val e = hoge()
} while (e == true)
```

- forループ

```
for (item in items) {  
    println(item)  
}
```

- Javaのfor (int i = 0; i < 100; i++)

```
for (i in 0 until 100) {  
    println(i)  
}
```

- when(Javaのswitch)
breakは不要
defaultはelseに該当
カンマ(,)区切りで複数条件

```
when (x) {  
    1 -> print("x == 1")  
    2,3 -> print("x == 2 or x == 3")    // 複数条件  
    else -> {                          // defaultに該当  
        print("otherwise")  
    }  
}
```

- fun(関数宣言)

fun 関数名(引数):戻り値

戻り値がない場合は、戻り値の型をUnitと記述

[戻り値あり]

```
fun double(x: Int): Int {  
    return 2 * x  
}
```

[戻り値なし] (Unitは省略可)

```
fun printHello(name: String): Unit {  
    println("Hello, ${name}")  
}
```


静的型付け

- 型推論により型の記述を省略できます。

```
val a = "Hello"           // String
val b = 123                // Int
```

クラス

- 変数はProperty(setter/getterを合わせたもの)

```
class Person {
```

```
    val name: String  
    var age: Int
```

← Property

```
    constructor(name: String, age: Int) {  
        this.name = name  
        this.age = age  
    }
```

↑
コンストラクタでProperty初期化

```
    fun getPerson(): String {  
        return name + " " + age  
    }
```

← メンバ関数

```
}
```

- インスタンス生成はnew不要

```
val person:Person = Person("santa", 20)
```

- Propertyはsetter/getterを合わせたもので
以下のようにアクセスすることができる。

```
val age = person.age  
person.age = age + 1
```

- コンストラクタでPropertyを同時に宣言
デフォルト引数が設定可能

Propertyを同時に宣言

```
class Person(val name: String, var age: Int = 0) {  
    fun getPerson(): String {  
        return name + " " + age  
    }  
}
```

デフォルト引数

インスタンス生成(age省略)

```
val person: Person = Person("santa")
```

- クラスの初期化で処理を行いたいときはinitブロックに記述する。
Propertyのチェックをするなどに使用する。
→require()でPropertyを評価して
IllegalArgumentExceptionを発生させる。

```
class Person(val name: String, var age: Int = 0) {  
    init {  
        require(age >= 0) { "Count must be > 0, was ${age}" }  
    }  
}
```

↑
評価結果がfalseなら例外発生

- クラスを継承するには継承元にopenを付ける
コロン(:)の後に継承元クラスを記述する

継承元

```
open class Person(val name: String, var age: Int = 0) {  
}
```

継承先

```
class Student(val name: String) : Person(name) {  
  
}
```

null安全

- nullの可能性がある箇所はコンパイル時にエラーになります。

型?で宣言すればnullを代入できます。

```
val s:String = null           // コンパイルエラー  
val s:String? = null         // OK
```

※実行前にnullのリスクの箇所が分かるが、
万能ではないことに注意してください。

- メンバ関数にアクセスもチェックされます。
変数?.メンバ関数でコンパイルは通ります。
このとき変数がnullならばメンバ関数は
実行されない。(内部でnullチェック)

```
val s:String? = null
val len = s.length           // コンパイルエラー
val len = s?.length          // OK
```

- 変数!!.メンバ関数で回避できるが、
nullにならないと保証できる場合のみとする。

Kotlin動作環境

- Android Studio 2.2以降で動作
 - ※Android Studio 3.0から標準
(2017年7月時点では3.0はβ版)
- Kotlin Android Pluginをインストール
 - ※Android Studio 3.0 β 以前の場合

Kotlin開発環境構築

- 環境構築の前にはお願いします。

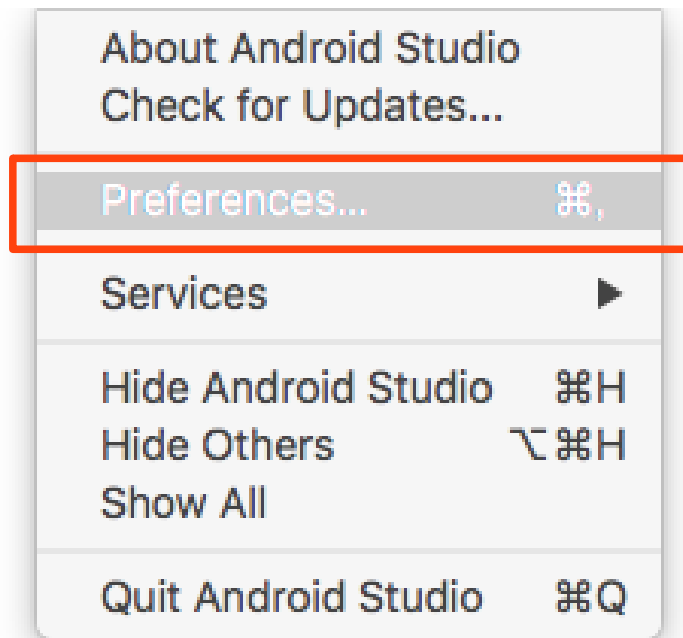
Macベースで説明します。

Windowsは適宜読み替えてください。

不明点は、遠慮なく質問してください。

Kotlin Android Pluginインストール

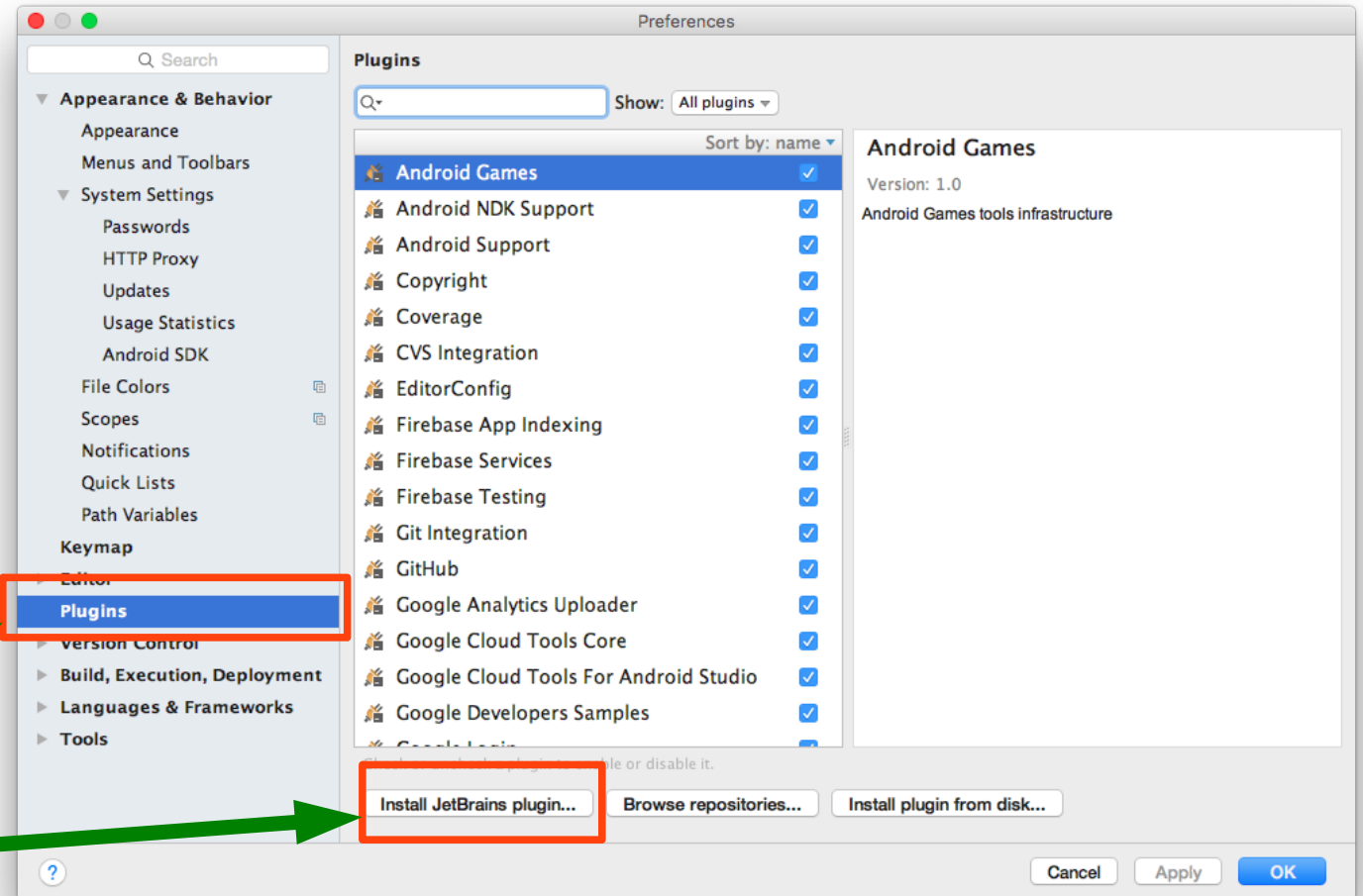
- Android Studioを起動します。
- [Android Studio] – [Preferences]を押下します。



- [Plugins] - [Install JetBrains Plugins]
ボタンを押下します。

1)ここを選択

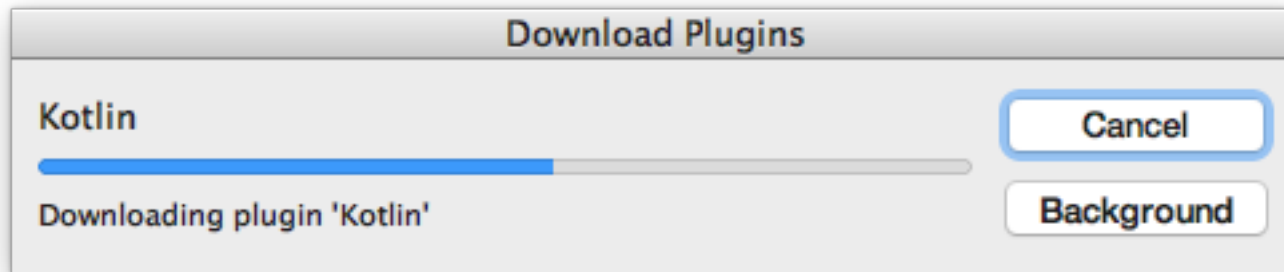
2)ここを押下



- 左上に”Kotlin”と入力します。
- リストから”Kotlin”を選択し、右側の[Install]を押下します。

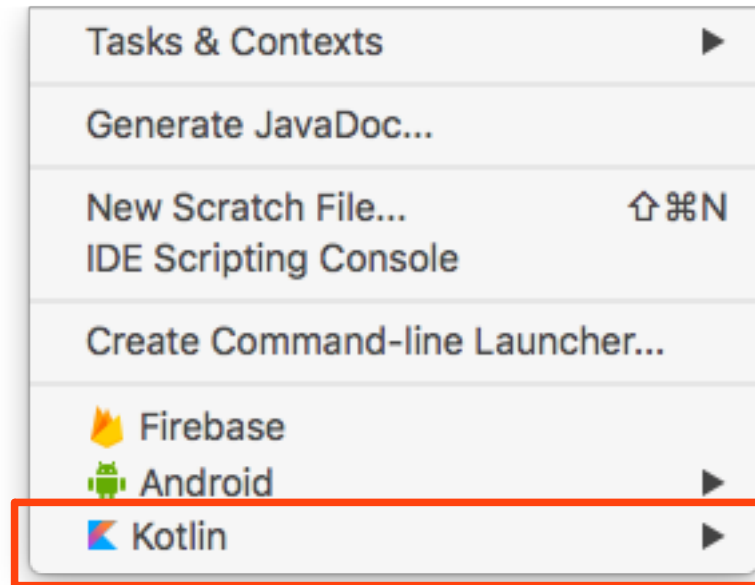


- Kotlin Pluginがインストールされます。



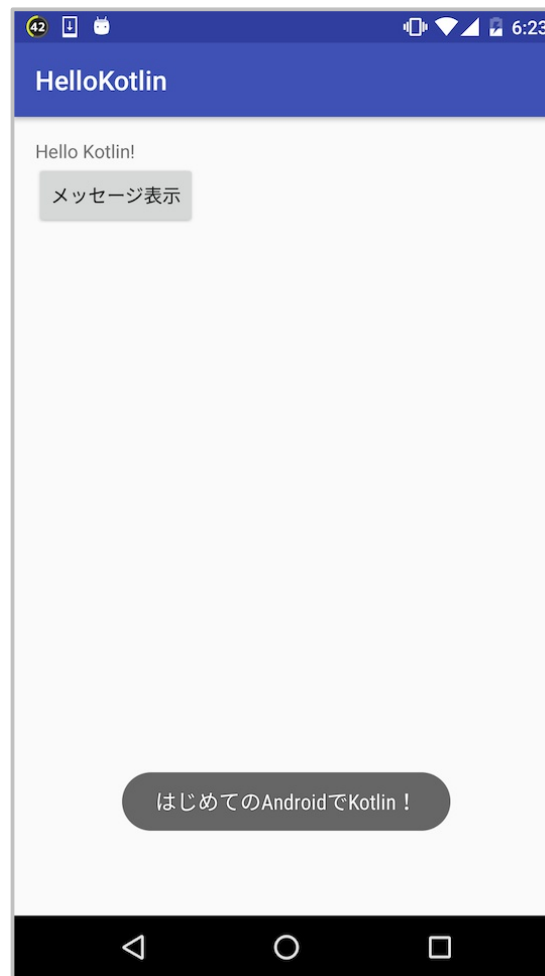
- インストール後にAndroid Studioを再起動します。

- [Tools] – [Kotlin]が追加されていれば、インストール完了です。



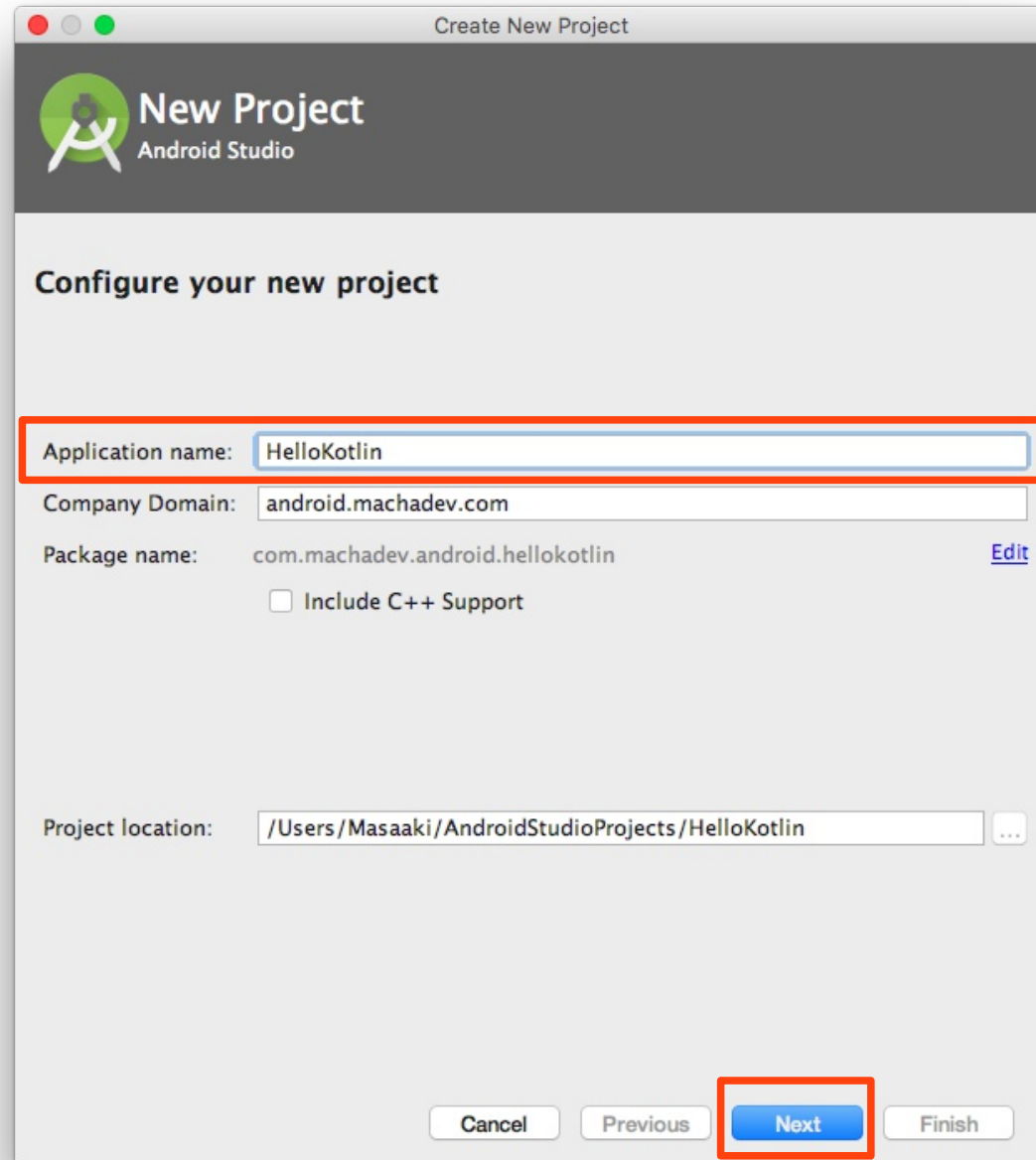
Hello Kotlin!

- Hello Kotlin!をテキストに表示し、Toastでメッセージ表示するアプリを作ります。

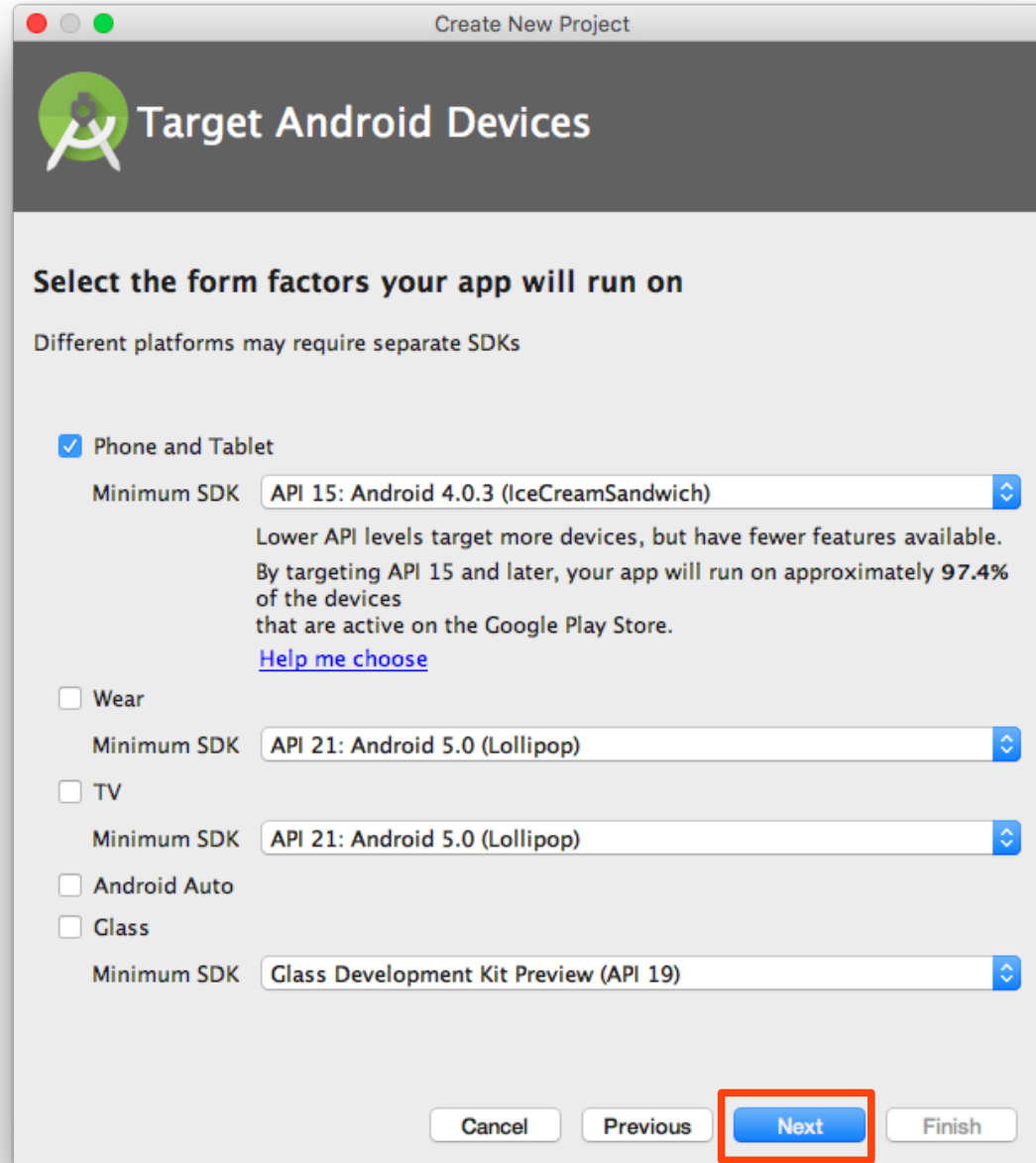


- プロジェクトを開いていないならば、
[Start a new Android Studio project]を
選択します。
- プロジェクトを開いているならば、
[File] – [New] – [New Project]
選択します。


- Application nameにHelloKotlinと入力し、[Next]ボタンを押下します。



- Minimum SDKはAPI 15で
[Next]ボタンを押下します。



Create New Project

 Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK

Lower API levels target more devices, but have fewer features available.
By targeting API 15 and later, your app will run on approximately 97.4%
of the devices
that are active on the Google Play Store.
[Help me choose](#)

☐ Wear

Minimum SDK

☐ TV

Minimum SDK

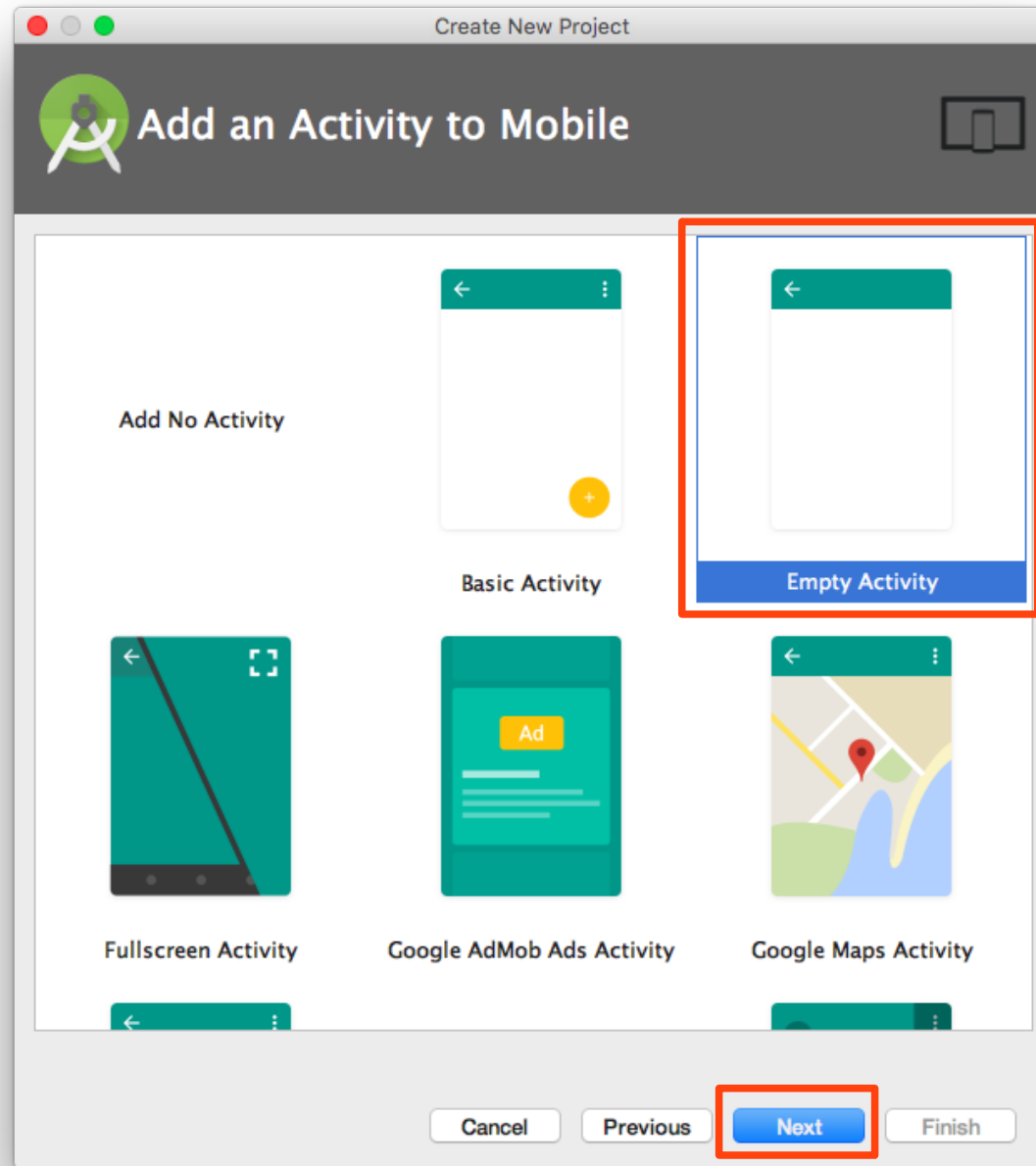
☐ Android Auto

☐ Glass

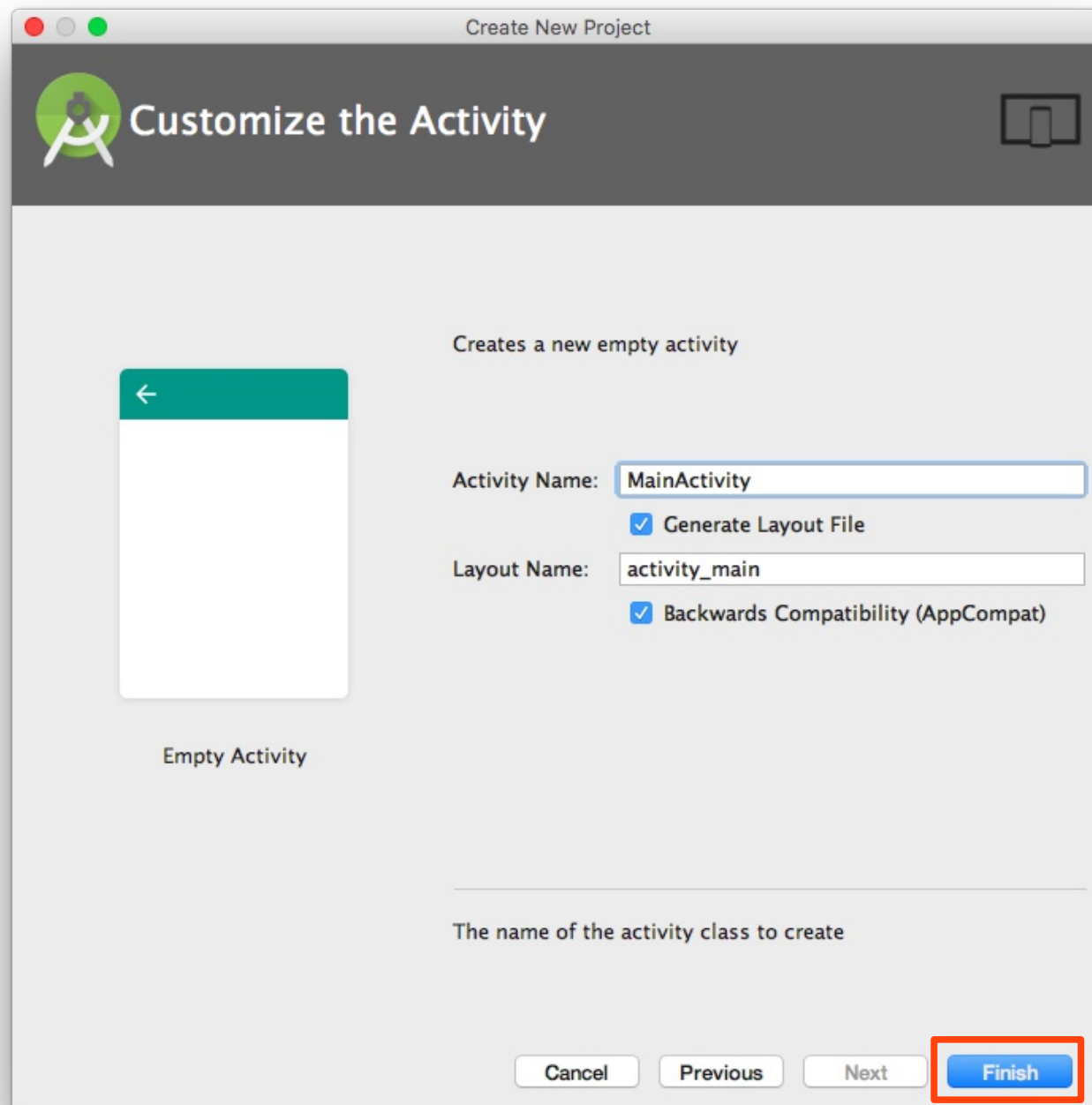
Minimum SDK

Cancel Previous **Next** Finish

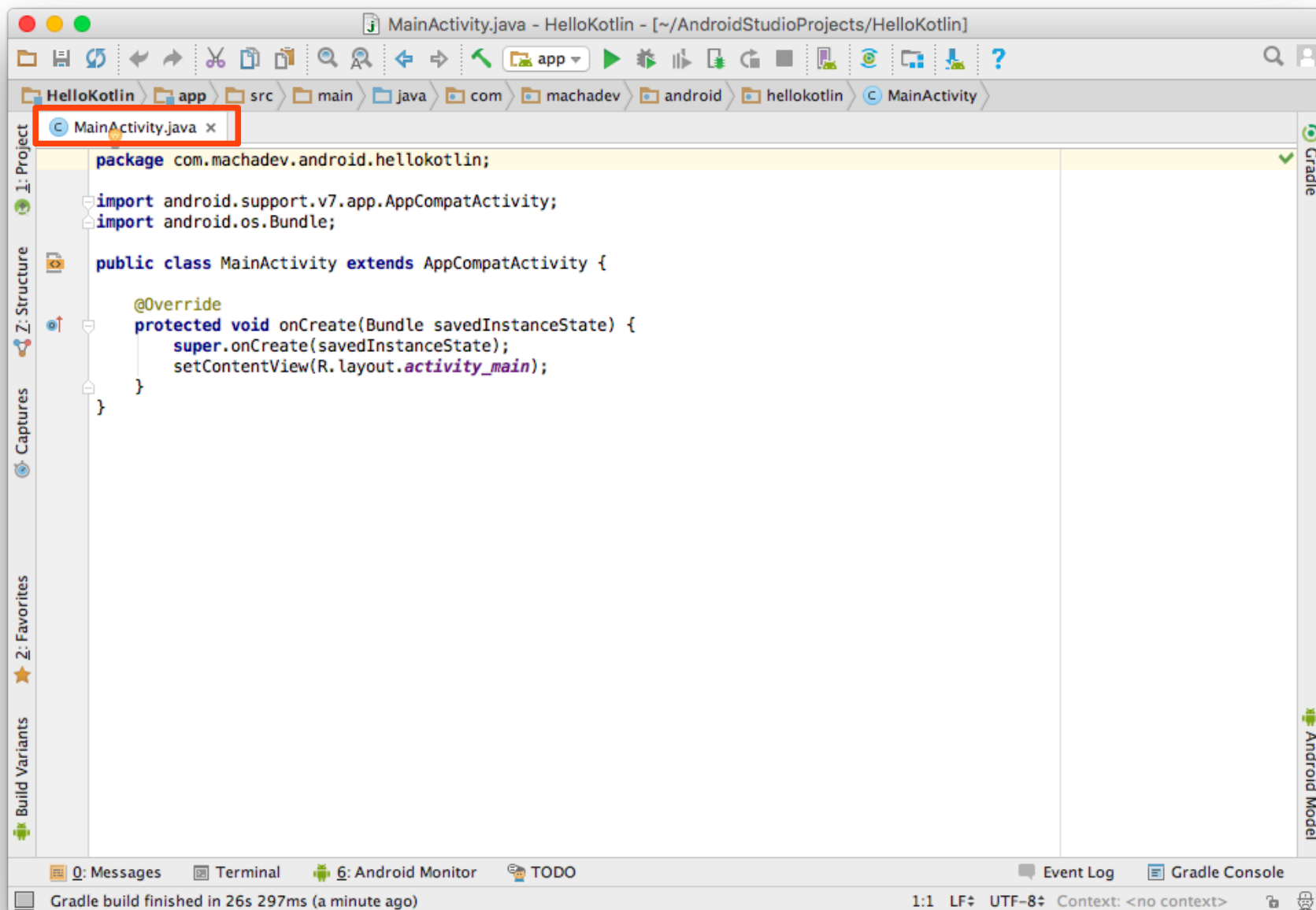
- Empty Activityを選択し、
[Next]ボタンを押下します。



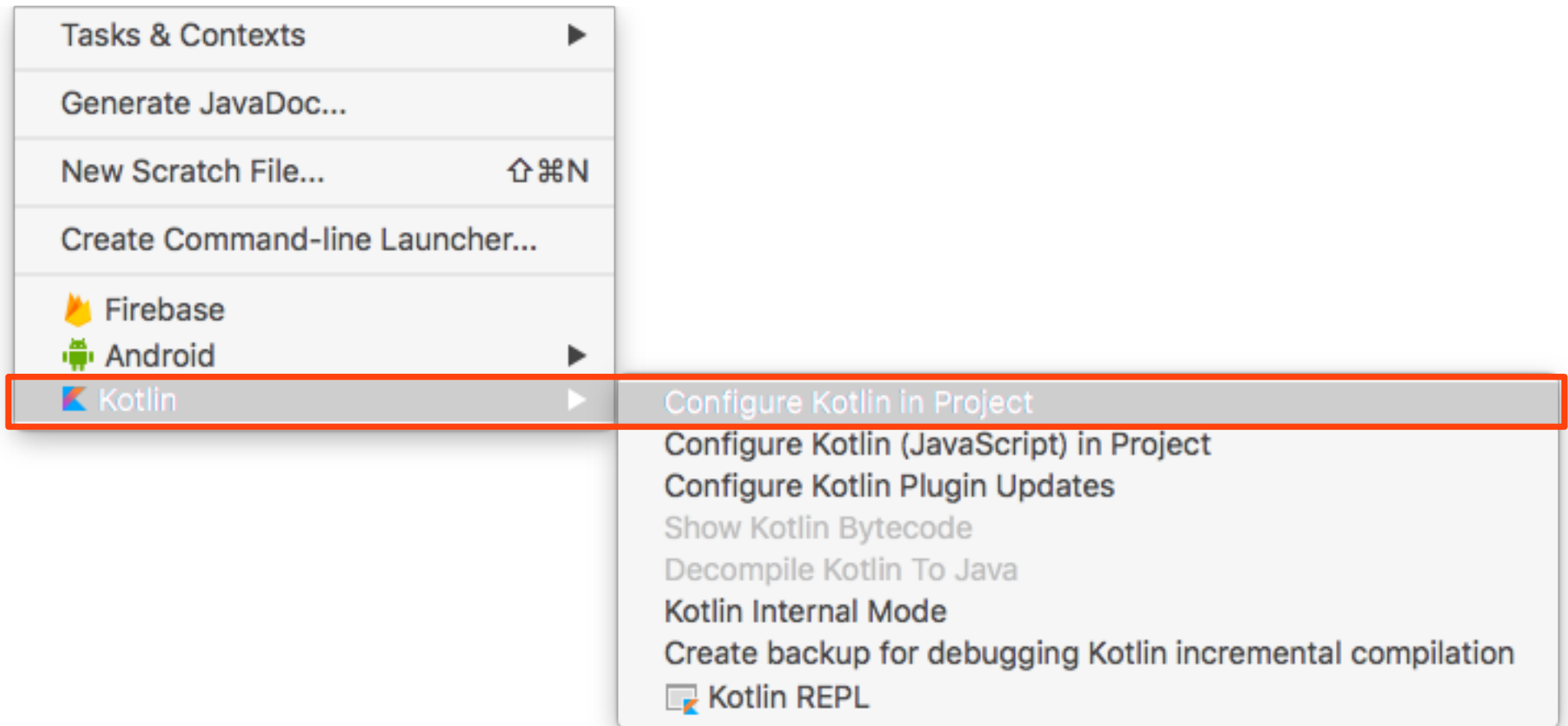
- [Finish]ボタンを押下します。



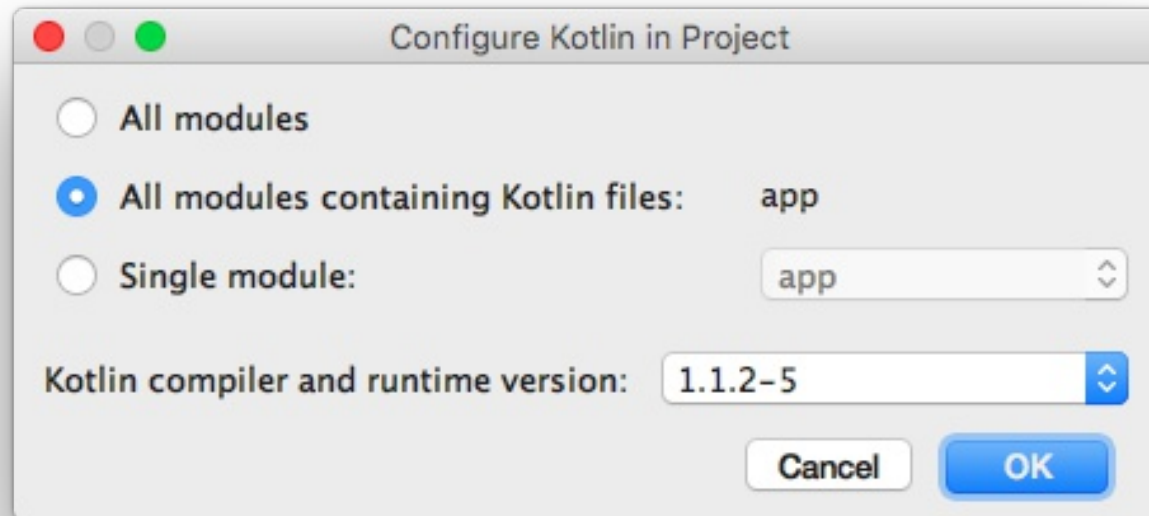
- Project作成後にMainActivityを表示した状態
まだJavaです。



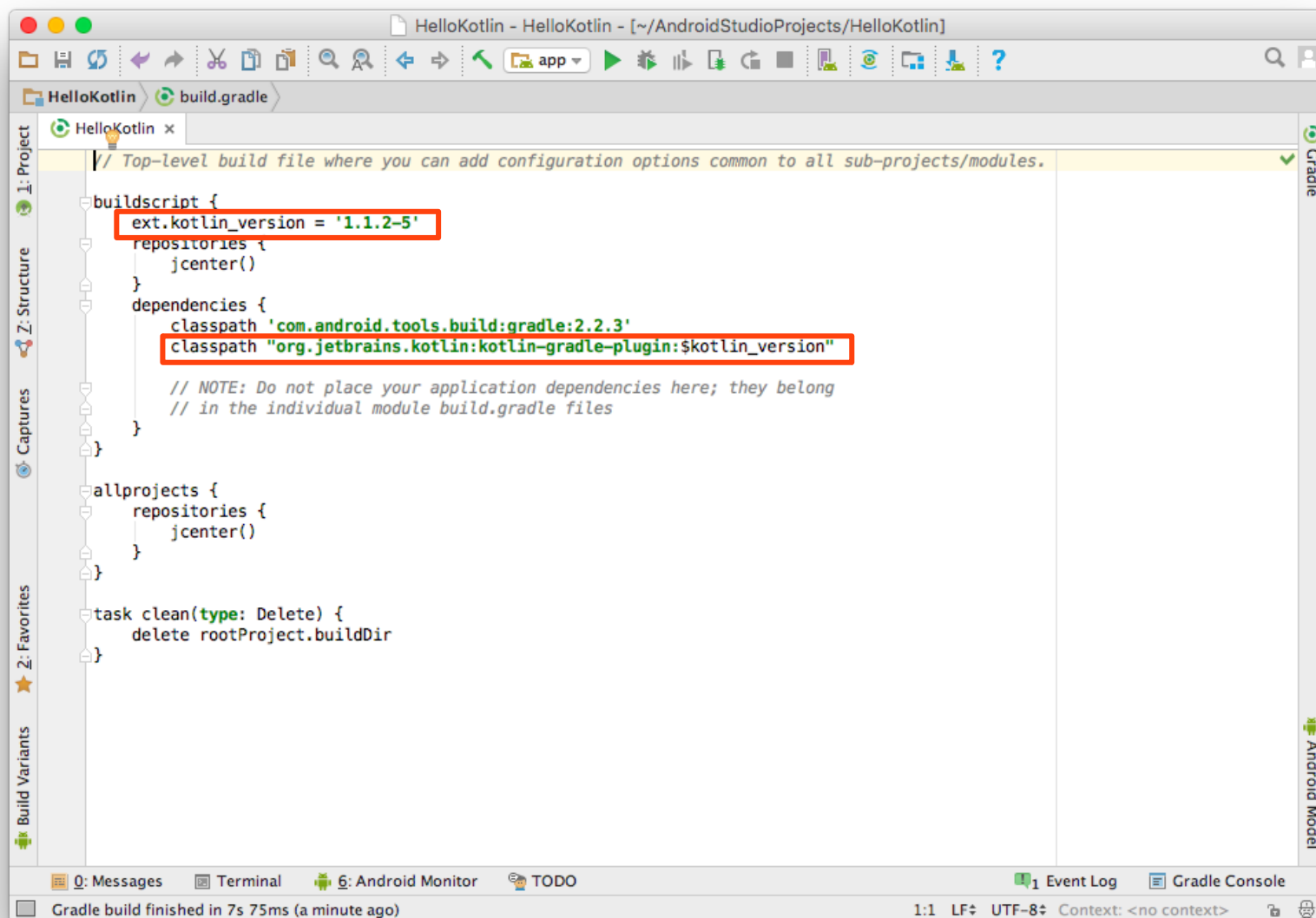
- [Tools]-[Kotlin]-[Configure Kotlin in Project]
を選択します。



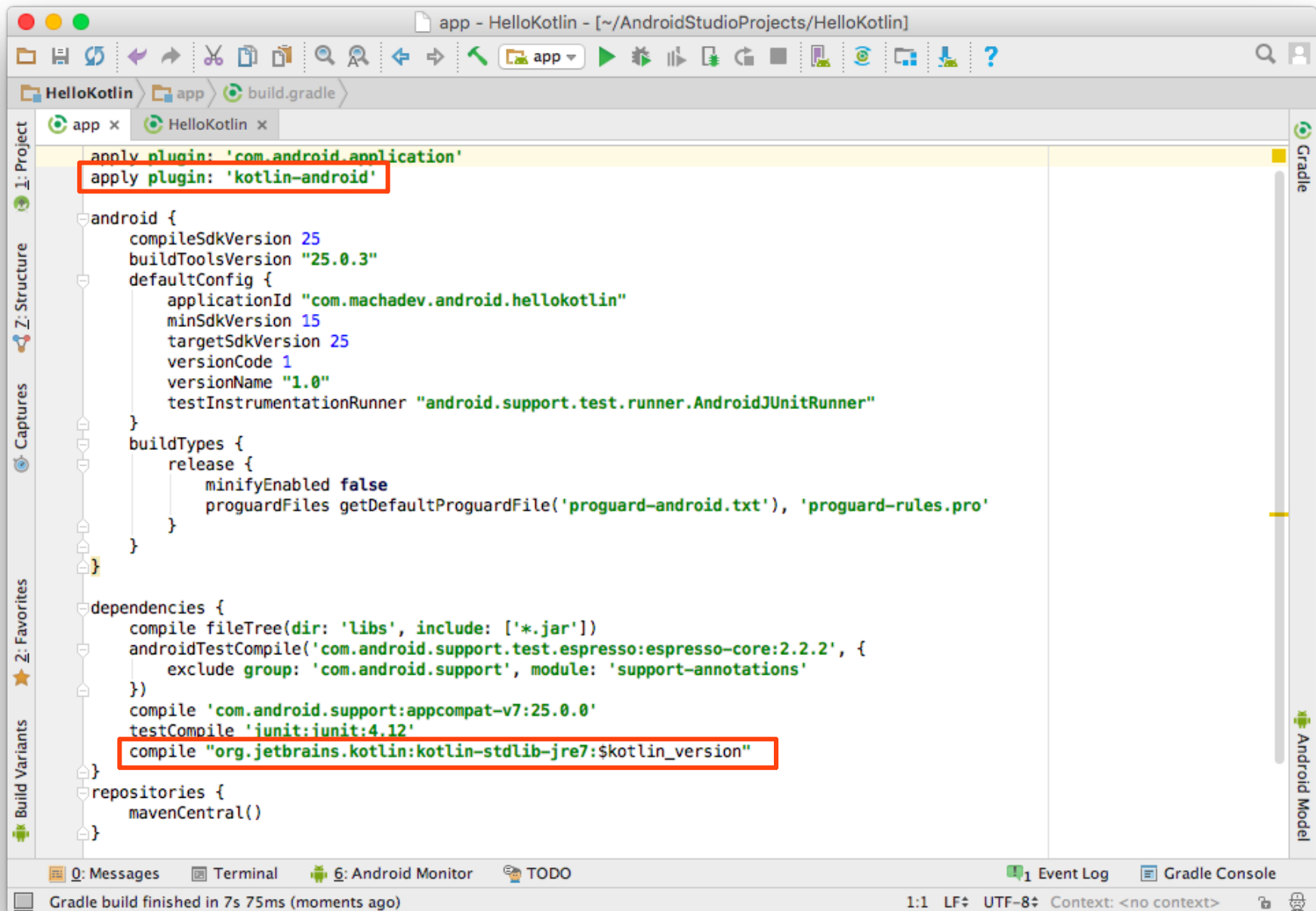
- [All modules containing Kotlin files]を選択し
[OK]ボタンを押下します。



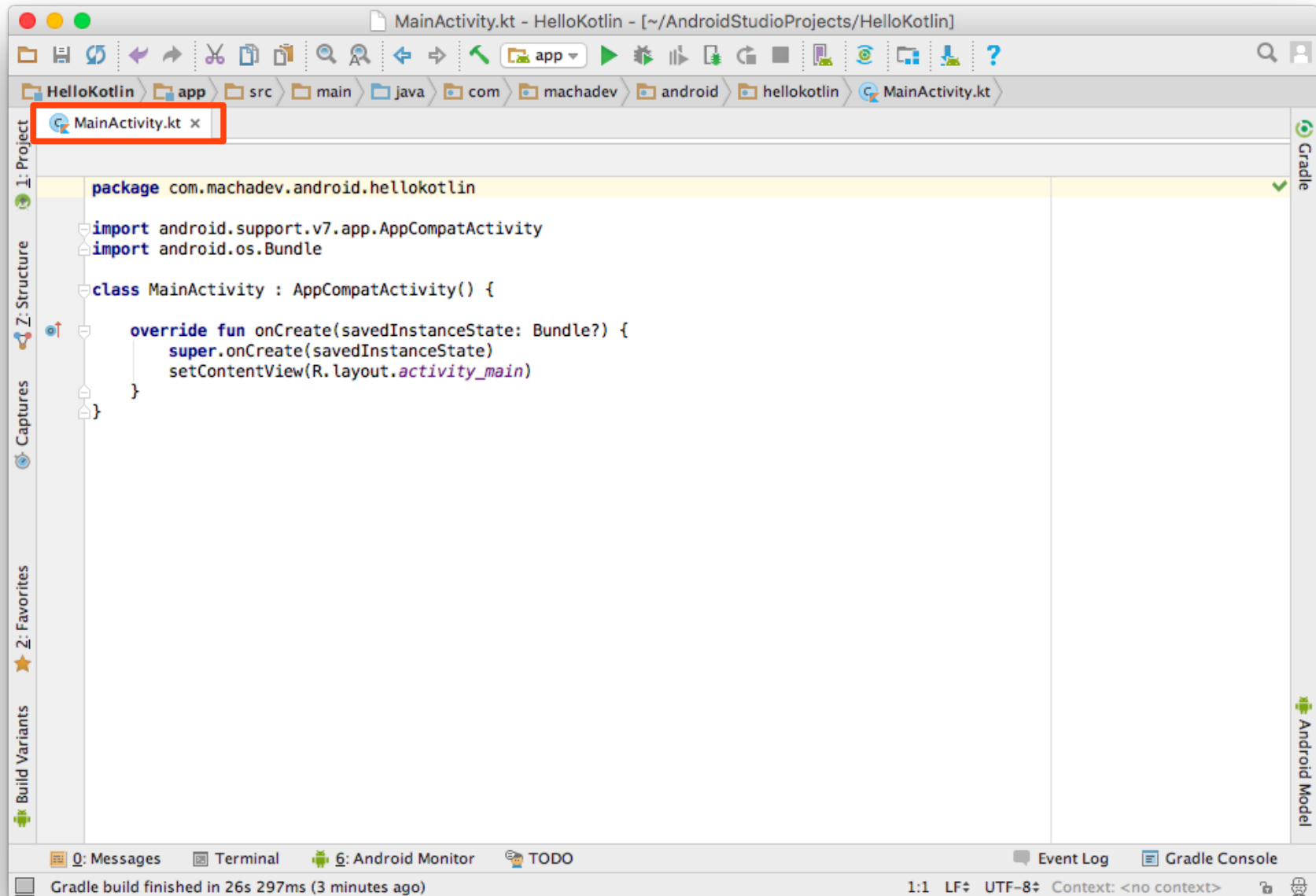
- build.gradle(module/app)にKotlinの設定が追加されます。
build.gradle(module)



- build.gradle(app)



- [Code] – [Convert Java File to Kotlin file]で Kotlinファイルに変換します。(拡張子はkt)



- JavaとKotlinを並べてみます。

[Java]

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

[Kotlin]

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
}
```

- res/layout/activity_main.xmlを開きます。
以下のように追加します。



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.machadev.android.hellokotlin.MainActivity">

    <TextView
        android:id="@+id/tvMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:id="@+id/btnExec"
        android:layout_below="@+id/tvMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="メッセージ表示" />

</RelativeLayout>
```

- src/MainActivity.ktを開いて追加します。

```
MainActivity.kt x
MainActivity
package com.machadev.android.hellokotlin

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    var tvMessage:TextView? = null
    var btnExec:Button? = null

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

        tvMessage = findViewById(R.id.tvMessage) as TextView
        btnExec = findViewById(R.id.btnExec) as Button
        btnExec.setOnClickListener { View ->
            tvMessage?.text = "Hello Kotlin!"
            showMessage("はじめてのAndroidでKotlin!")
        }

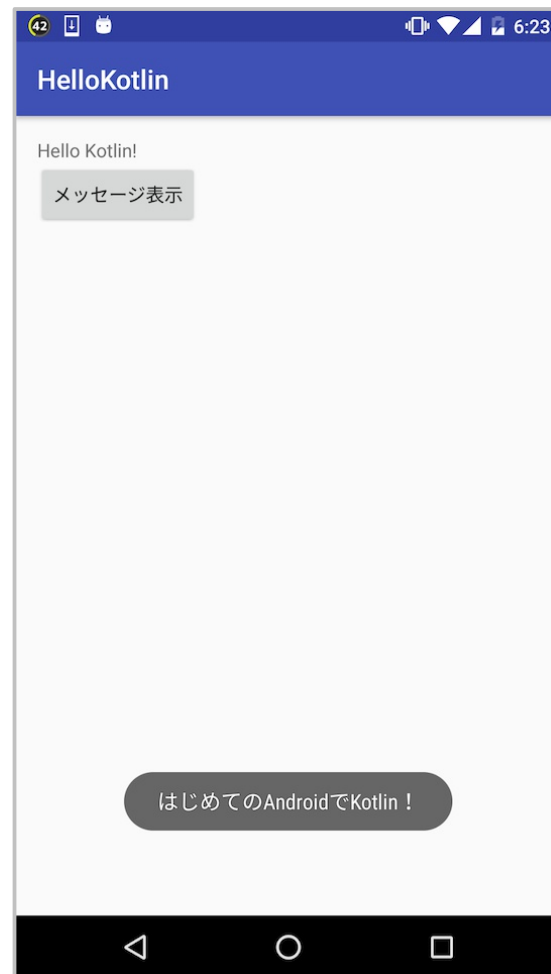
        fun showMessage(message: String) {
            Toast.makeText(this, message, Toast.LENGTH_LONG).show()
        }
    }
}
```

追加

追加
コントロール取得
ボタンイベント設定

追加
メッセージ表示関数

- コンパイルしてアプリを起動します。
- メッセージ表示ボタンを押下して
以下のように表示されれば完成です。



おわりに

- KotlinはAndroid開発の標準言語になり、Kotlinのプロジェクトも増えると思います。
- 簡潔なコードやnull安全は魅力的です。過信はせずに適切な使い方をしましょう。
- これから始める方は、小さなプロジェクトからKotlinを導入してみると良いと思います。

リンク集

- 公式サイト(Android)

<https://developer.android.com/kotlin/index.html>

- 公式サイト(Kotlin)

<https://kotlinlang.org/>

- JavaプログラマのためのKotlin入門

<http://qiita.com/koher/items/bcc58c01c6ff2ece658f>

- Try Kotlin(WebでKotlinコーディング&実行できる)

<https://try.kotlinlang.org/>

- プログラマに優しい現実指向JVM言語 Kotlin入門

<http://gihyo.jp/dev/serial/01/jvm-kotlin/0001>

ご清聴ありがとうございました。