

EXAMPLE UNIVERSITY

INFORMATION SECURITY

INF5021 - EXAMPLE SUBJECT NAME

---

## Example Paper Name

---

**LATEX**

*Author(s):*

Kazumi Harada

*Supervisor(s):*

Prof. Example NAME

July 5, 2020

# Contents

<b>第1部 Android 基礎</b>	<b>2</b>
<b>第1章 イントロダクション</b>	<b>3</b>
1.1 本コースの目標 . . . . .	3
1.2 本コースの特徴 . . . . .	3
1.3 Android エンジニアに求められるスキル . . . . .	3
1.4 本コースの進め方 . . . . .	4
1.5 Android エンジニアを取り巻く状況 . . . . .	5
1.6 Android とは . . . . .	6
1.7 Android アプリでできること . . . . .	7
1.7.1 AOSP . . . . .	7
1.7.2 センサプログラミング . . . . .	7
1.8 iPhone と Android の違い . . . . .	8
<b>第2章 Android アプリを作る</b>	<b>9</b>
2.1 この章の目標 . . . . .	9
2.2 AndroidStudio のインストール . . . . .	9
2.3 新規プロジェクトを作成する . . . . .	11
2.4 エミュレーターをインストールする . . . . .	13
2.4.1 エミュレーターとは . . . . .	13
2.4.2 エミュレーターのインストール方法 . . . . .	13
2.5 アプリをビルドする . . . . .	16
2.6 作られたファイルの役割を理解する . . . . .	17
2.7 自分のコードを管理する . . . . .	17
2.7.1 分散型バージョン管理システム . . . . .	18
2.7.2 ブランチ管理 . . . . .	19
2.8 実際に Git を操作する . . . . .	19
2.8.1 リモートリポジトリを作成する . . . . .	20
2.8.2 ローカルリポジトリ設定する . . . . .	21
2.8.3 自分のコードを追加する . . . . .	22

2.9	本当に動いているのか確かめる . . . . .	25
2.9.1	BreakPoint を使用する . . . . .	26
2.9.2	ログ出力を使用する . . . . .	27
2.10	まとめ . . . . .	28
2.10.1	課題 . . . . .	29
<b>第3章</b>	<b>Kotlinに触れる</b>	<b>30</b>
3.1	この章の目標 . . . . .	30
3.2	Kotlinとは . . . . .	30
3.3	変数と関数 . . . . .	30
3.4	クラス . . . . .	30
3.5	制御構文 . . . . .	30
3.6	例外処理 . . . . .	30
3.7	コレクションクラス . . . . .	30
3.8	クラス設計1 . . . . .	30
3.9	クラス設計2 . . . . .	30
3.10	まとめ . . . . .	30
<b>第II部</b>	<b>Android応用</b>	<b>31</b>
<b>第4章</b>	<b>Androidをあやつる</b>	<b>33</b>
4.1	この章の目標 . . . . .	33
4.2	Androidライフサイクル . . . . .	33
4.2.1	Manifest . . . . .	33
4.3	Activity . . . . .	33
4.4	Fragment . . . . .	33
4.5	Service . . . . .	33
4.6	Intent . . . . .	33
4.6.1	明示的Intent . . . . .	33
4.6.2	暗黙的Intent . . . . .	33
4.7	Gradle . . . . .	33
4.8	主要なライブラリ . . . . .	33
4.9	まとめ . . . . .	33
<b>第5章</b>	<b>思い通りのレイアウトを作る</b>	<b>35</b>
5.1	この章の目標 . . . . .	35
5.2	レイアウト . . . . .	36
5.2.1	LinearLayout . . . . .	36
5.2.2	RelativeLayout . . . . .	36
5.2.3	FrameLayout . . . . .	36

5.2.4	ConstraintLayout . . . . .	36
5.3	コンポーネント . . . . .	36
5.3.1	RecyclerView . . . . .	36
5.3.2	AdapterView . . . . .	36
5.3.3	Appbar . . . . .	36
5.3.4	Snackbar . . . . .	36
5.3.5	... . . . .	36
5.4	アニメーション . . . . .	36
5.4.1	RippleEffect . . . . .	36
5.4.2	... . . . .	36
5.5	デザインの基礎知識 . . . . .	36
5.5.1	色相 . . . . .	36
5.5.2	色の意味 . . . . .	36
5.5.3	フォント . . . . .	36
5.5.4	フォントの意味 . . . . .	36
5.5.5	良いレイアウト悪いレイアウト . . . . .	36
5.6	まとめ . . . . .	36
<b>第6章</b>	<b>データを管理する</b>	<b>38</b>
6.1	この章の目標 . . . . .	38
6.2	Database とは . . . . .	38
6.2.1	SQL の基礎的な使い方 . . . . .	38
6.3	Android における Database の役割 . . . . .	38
6.3.1	ローカル Database . . . . .	38
6.3.2	SharedPreference . . . . .	38
6.4	API 通信 . . . . .	38
6.4.1	HTTP 通信とは . . . . .	38
6.5	Json データの取り扱い . . . . .	38
6.5.1	Gson の使い方 . . . . .	38
6.5.2	リクエストキャッシュ . . . . .	38
6.6	まとめ . . . . .	38
<b>第III部</b>	<b>Android 発展</b>	<b>39</b>
<b>第7章</b>	<b>Android をあやつる</b>	<b>41</b>
7.1	この章の目標 . . . . .	41
7.2	クライアントアーキテクチャ . . . . .	41
7.2.1	MVC . . . . .	41
7.2.2	MVVM . . . . .	41
7.2.3	CleanArchitecture . . . . .	41

7.3	Firebase を使う . . . . .	41
7.3.1	Analytics . . . . .	41
7.3.2	Crashlytics . . . . .	41
7.3.3	RemoteConfig . . . . .	41
7.3.4	CloudMessaging . . . . .	41
7.4	.....	41
7.5	まとめ . . . . .	41
<b>第8章</b>	<b>アプリをリリースする</b>	<b>43</b>
8.1	この章の目標 . . . . .	43
8.2	Build Variants . . . . .	43
8.3	Proguard . . . . .	43
8.4	リリースの方法 . . . . .	43
8.5	リリース後の運用 . . . . .	43
8.6	まとめ . . . . .	43
<b>第IV部</b>	<b>Examples</b>	<b>44</b>
<b>第9章</b>	<b>Math</b>	<b>45</b>
<b>第10章</b>	<b>Code</b>	<b>46</b>
<b>第11章</b>	<b>Citation</b>	<b>47</b>
<b>第12章</b>	<b>Table</b>	<b>48</b>

# 第I部

## Android 基礎

# 1. イントロダクション

## 1.1 本コースの目標

Android エンジニアとして働くようになることを目指します。

## 1.2 本コースの特徴

現在、ほとんどのエンジニアスクールは Android アプリを作れるようになることを目標に置いています。しかし、実業務では Android アプリを作れることは当然のスキルであり、 $+ \alpha$ の能力が求められます。さらに業務で求められる「アプリケーションを開発する」ということは、ただ作ればいいわけではなく、「動き続けるものを作る」必要があります。すなわち、机上の知識+実務の知識が実際に Android エンジニアとして働くためには必要です。

よって、本コースでは、Android の技術+実業務において必要な技術をバランス良く説明していきます。

## 1.3 Android エンジニアに求められるスキル

一般的に Android エンジニアは下記のスキルを求められます。(TopTal: Android Developer Job Description Template から抜粋)

- Android への理解
- 問題解決能力
- 他チームとの協業力
- 品質管理能力
- 一般的な技術への広い理解

具体的にしたもののが表 1.1 です。

表 1.1: Android エンジニアに求められるスキル

スキル	具体例
Android への理解	純粋にコードを書く力があるか
問題解決能力	問題が発生した時にどうやって対処するか
他チームとの協業力	他人と協力してサービスを作れるか。他のエンジニアやデザイナーと会話できる程度の理解があるか
品質管理能力	成果物を正しく管理できるか。継続して安定した成果を上げられるか
一般的な技術への広い理解	誰でも知ってるレベルの知識があるか

ここまで読めば、Android アプリを作れるだけでは、実務者としての能力が不足していることが理解できるでしょう。上記の懸念点から、本コースは図 1.1 のバランスでスキルを習得していきます。

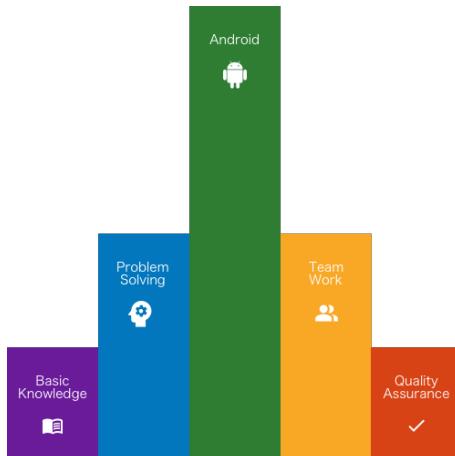


図 1.1: 習得可能なスキルの範囲

## 1.4 本コースの進め方

オリジナルのYoutube 再生アプリを開発します。アプリを都度追記・修正しながら、Android に関する知識や業務の進め方を把握していきます。実際に使用するであろうツールを使いつつ、最終的にはリリースできるレベルものを開発します。

## 1.5 Android エンジニアを取り巻く状況

内容に入っていく前に、2020年時点でのAndroidエンジニアを取り巻く環境について理解しておきましょう。まず、アプリの調査会社として世界で広く利用されるApp Annieの予測では、世界全体のスマートフォンアプリの市場規模は、2022年に2017年の2倍近くになるとしています。(図1.2)



図 1.2: 2022 年のアプリ市場予測

また、Android端末のスペックの向上により1つでiOSアプリもAndroidアプリも開発できるハイブリッドアプリケーションの技術が少しづつ注目されてきています。しかし、現在ハイブリッドアプリケーション開発はまだ多くの選択肢があり、デファクトスタンダードが決まっていない状況です。その状態で1つのハイブリッドアプリケーション開発に絞ることは、もしその技術が廃れてしまった場合、非常に大きなリスクになります。

上記のことから、まずは現在需要のあるネイティブアプリケーションの技術を学び、ハイブリッドアプリケーションの状況にも気を配っておくことが重要です。

## 1.6 Android とは

Android アプリの開発に入る前に、そもそも Android がなんなのかを大まかに理解しておく必要があります。

Android = 携帯端末に搭載している OS(オペレーティングシステム)の名前

すなわち Windows や MacOS、iOS と同じ立ち位置のものになります。携帯端末は小さなパソコンです。端末を構成するハードウェアは PC とほとんど同じで、各種のセンサーが搭載されている部分がパソコンとは異なります。Windows8,10 や MacOS X と同じように Android にもコードネームが設定されており、今までのコードネームは表 1.2 の通りです。

表 1.2: Android のコードネーム

コードネーム	Android バージョン
Cupcake(カップケーキ)	1.5
Donut(ドーナツ)	1.6
Froyo(フローズンヨーグルト)	2.2
Gingerbread(ジンジャーブレッド)	2.3
Honeycomb(ハニカム)	3.0
Ice Cream Sandwich(アイスクリームサンドウイッチ)	4.0
Jelly Bean(ジェリービーン)	4.1
KitKat(キットカット)	4.4
Lollipop(ロリポップ)	5.0
Marshmallow(マシュマロ)	6.0
Nougat(ヌガー)	7.0
Oreo(オレオ)	8.0
Pie(パイ)	9.0

アルファベット順になっているのが特徴です。

## 1.7 Android アプリでできること

### 1.7.1 AOSP

Android はフリーの OS であることが大きな特徴です。その気になれば、自分で OS 自体をカスタマイズして携帯端末にインストールすることもできます。AndroidOS を修正するプロジェクトのことを AOSP(Android Open Source Project) といい Android で不具合を発見した場合は自ら修正して、取り込んでもらうことができます。



図 1.3: Huawei 社が開発している Android ベースの独自 OS

### 1.7.2 センサプログラミング

Android 端末の各種センサを用いてプログラミングすることができます。主にユーザのジェスチャーや周囲の状況を計測します。全ての端末にセンサが搭載されているわけではなく、実装時にどのようなセンサが使用可能か確認する必要があります。

表 1.3: 主なセンサ

センサ	概要	用途
加速度センサ	端末をどの程度の速さで動かしているか	シェイクアクション
ジャイロセンサ	端末をどの程度傾けたか	画面の自動回転
心拍センサ	心拍数計測	ヘルスケアアプリ
温度センサ	周囲の温度を計測	ヘルスケアアプリ
照度センサ	周りの明るさを計測	画面の明るさ調整

## 1.8 iPhone と Android の違い

見た目の違いはもちろんですが主にエンジニア視点から比較すると、表 1.4 のような違いがあります。

表 1.4: iPhone と Android の違い

ポイント	Android	iOS
開発元	Google	Apple
開発言語	Java → Kotlin	Objective-C → Swift
シェア率 (世界, 日本)	87%, 40%	13%, 60%
需要	世界的高需要	日本内高需要
制約	ほとんどなし	Apple の要求に常に従う必要がある
アプリリリース	自由	Apple の審査あり
端末依存	大 (端末ごとの動作に大きな差分あり)	小 (検証端末が限られている)
運用コスト	1 度の Developer 登録のみ	年ごとに Developer 登録を更新
マーケット	PlayStore	AppleStore

Android 開発の大きなメリットとしては、障壁の低さにあります。iOS アプリに比べ、ランニングコストが掛からず端末も比較的安価なものからあります。ただし、端末ごとの差分が多く、ハードウェア（カメラやセンサ類）を使用した時は特にテストのコストが大きくなります。その場合は、AWS や Firebase のオンラインテストツールを使いある程度コストを下げるることができます。

## 2. Androidアプリを作る

### 2.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

### 2.2 AndroidStudio のインストール

AndroidStudio とは Android アプリ開発に使用する IDE（統合開発環境）です。Android アプリを開発するほとんどのエンジニアはこのアプリケーションを使用しています。Google でも公式に推奨しているアプリケーションなので、特別な理由がないかぎりこちらを使用するよう にしましょう。

下記 URL からイメージファイルをダウンロードします。（実際のダウンロード画面：図 2.1）

<https://developer.android.com/studio>

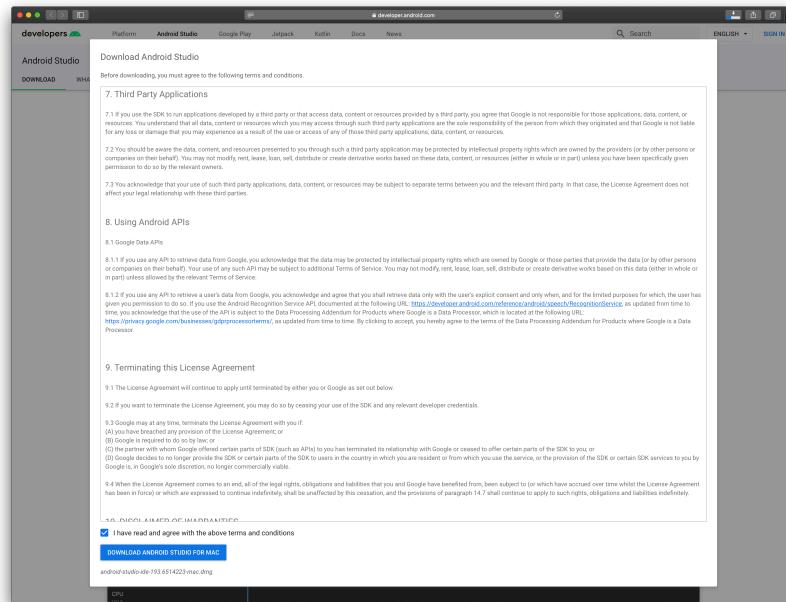


図 2.1: 利用規約に同意してインストール

ダウンロードしたイメージファイルをクリックして、AndroidStudio をインストールします。

インストール完了後、AndroidStudio を起動して、図 2.2 のような画面が表示されれば、AndroidStudio を使用する準備は完了です。

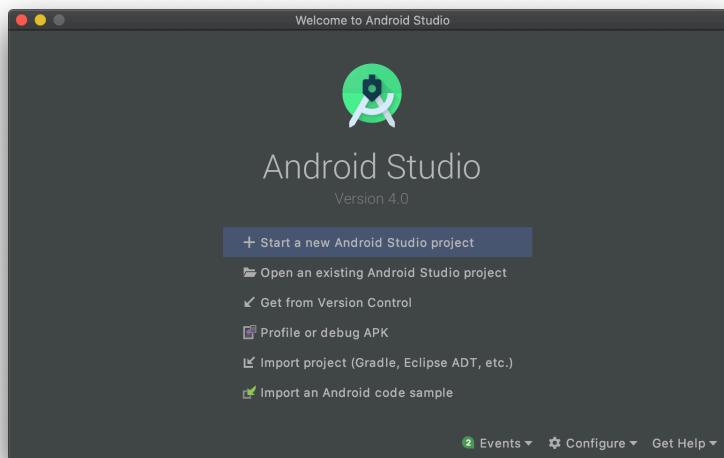


図 2.2: AndroidStudio の初期起動画面

## 2.3 新規プロジェクトを作成する

インストールが完了したら、早速プロジェクトを作成しましょう。Android アプリは一般的に 1 アプリ 1 プロジェクトで開発します。

初期起動画面で Start a new Android Studio project を選択してください。すると図 2.3 のようなテンプレート選択画面が起動します。今回はここで Basic を選択してください。

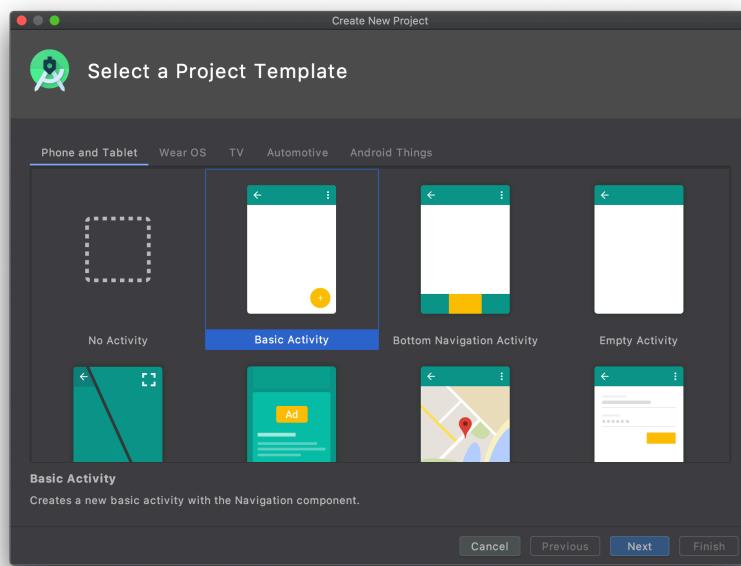


図 2.3: テンプレート選択画面

テンプレート選択後、図 2.4 のような、プロジェクトの基本情報を設定する画面が表示されます。

入力が必要な項目は下記の通りです。

**Name** プロジェクト名 - 後から変更可能なのでなんでも OK

**Package name** パッケージ名 - 一般的に保有するドメインの逆順で入力する。PlayStore の URL に組み込まれるので、誰に見られても恥ずかしくないものを指定する

**Save location** 保存場所 - 特に意図がなければ変更の必要はない

**Language** 使用する言語 - Java or Kotlin が選択可能。必要がないかぎり Kotlin を選択する

**Minimum SDK** サポートする下限の Android バージョン - 下げれば下げるほどアプリをインストールできるデバイスは増えるが、テストの範囲は増え、最新の機能は使えなくなる

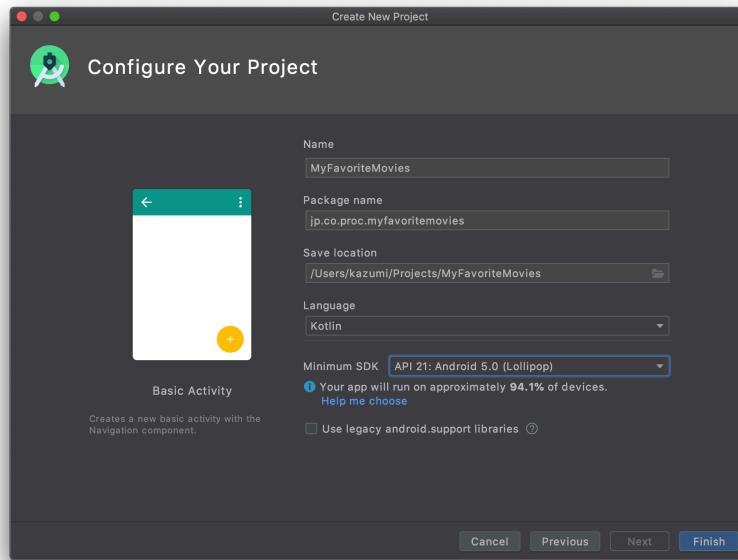


図 2.4: プロジェクト設定画面

必要な項目を入力後、Finish ボタンを押下し、ライブラリ等のインストール完了を待ちます。最終的に図 2.5 のような画面が表示されればコードを書く準備は完了です。

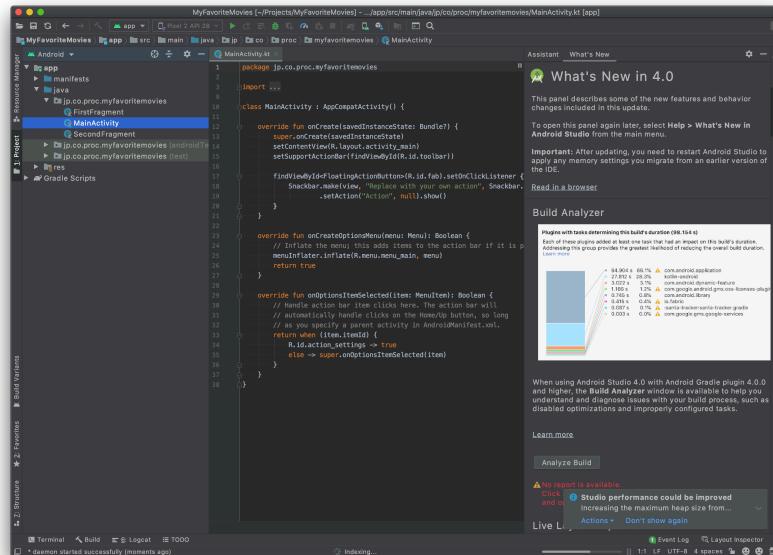


図 2.5: Android アプリ開発画面

## 2.4 エミュレーターをインストールする

最後の準備として、エミュレーターをインストールしましょう。実機をもっている場合は、本セクションを飛ばしても構いませんが、エミュレーターのインストール方法を知らない場合は、基本的な知識なので1度エミュレーターのインストールを試しておきましょう。

### 2.4.1 エミュレーターとは

通称エミュと呼ばれ、自分のPC上でAndroidモバイル端末を擬似的に動作させます。ただし、擬似的に再現しているだけなので実機のみで発生する問題を再現できないケースがあります。特に動画の再生は、搭載されているCPUやモジュールに依存した問題が多いので、実機での確認が必須になります。開発時点ではエミュレーターの方が便利ですが、テストは可能な限り実機でやる方が安全です。

### 2.4.2 エミュレーターのインストール方法

まず、図2.6の上部にあるAVD Managerボタンをクリックしてください。

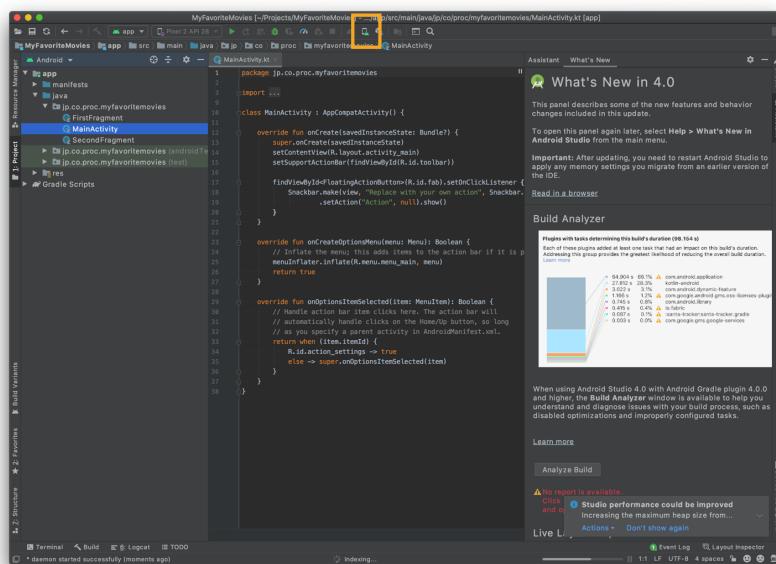


図 2.6: AVD Manager ボタンをクリック

次に、図2.7のエミュレーターを作成ボタンをクリックします。

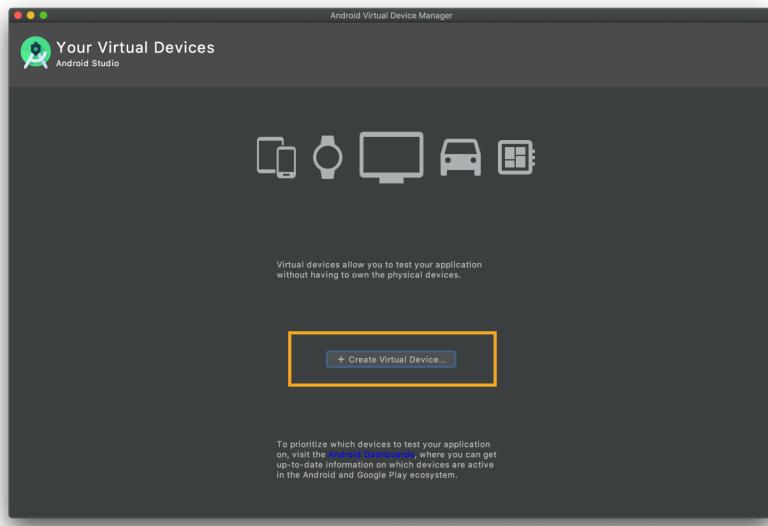


図 2.7: エミュレーターを作成

エミュレーターを作成するために、まず 図 2.8 のようにデバイスを選択します。好きなもので OK ですが、ドキュメントでは Pixel2 を選択しています。

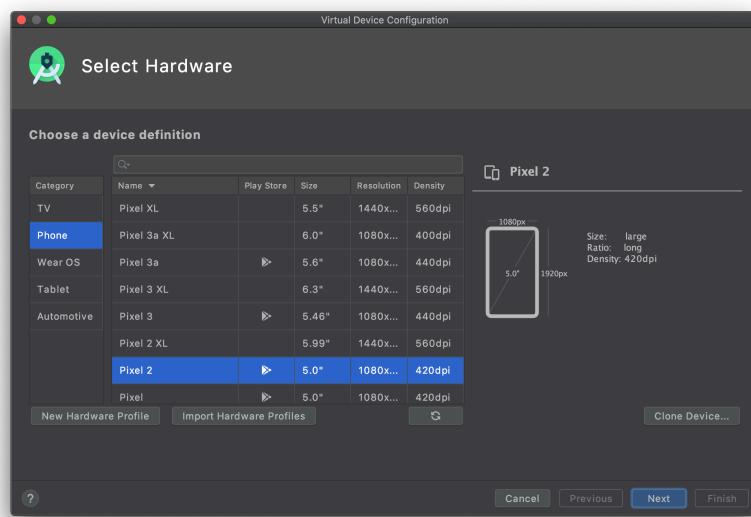


図 2.8: デバイスを選択

次に、図 2.9 のように Android バージョンを選択します。初回は、AndroidOS のイメージをダウンロードする必要があるので、エミュレーターを作成したい Android バージョンの右側にある Download ボタンをクリックします。

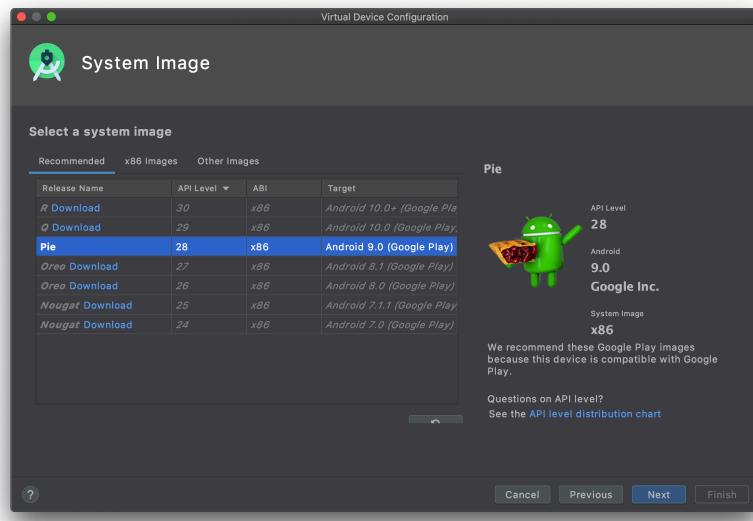


図 2.9: Android バージョンを選択

最後にエミュレーターの名前を入力して完了です。入力完了後、画面を終了します。

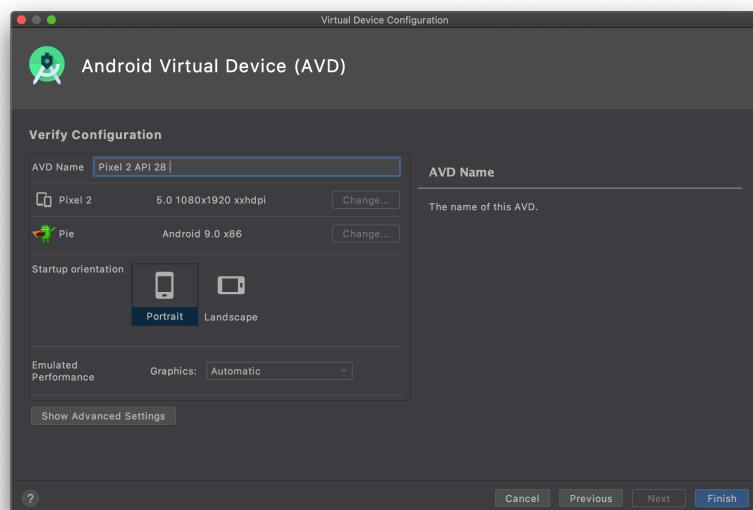


図 2.10: エミュレータ名を設定

## 2.5 アプリをビルドする

Android 開発の全ての準備が整ったので、Android アプリをビルドします。エミュレーターの作成が完了している場合は、図 2.11 のオレンジの枠のように作成したエミュレーターがデフォルトで表示されているはずです。もし実機をつないでいた場合はここをクリックすることで、どこにアプリをインストールするか決められます。

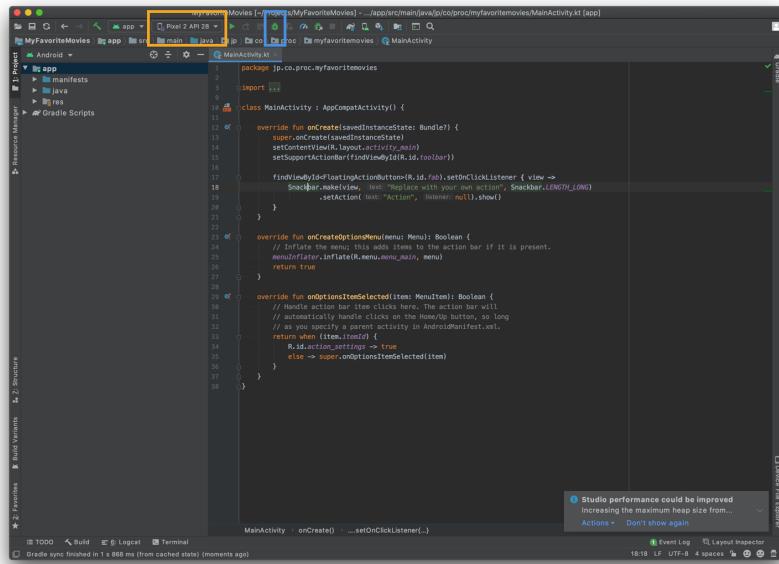


図 2.11: ビルド直前の画面

問題なさそうであれば、図 2.11 のブルーの枠にある、デバッグ実行ボタンを押します。しばらく待てば、ビルドが完了し、エミュレーターにアプリがインストールされます。

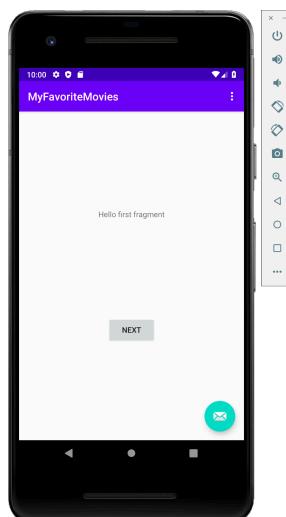


図 2.12: 初期のアプリ画面

## 2.6 作られたファイルの役割を理解する

新規プロジェクトを作成した時に、自動生成されたコードがあります。それらのコードは Android アプリを動かすために最低限必要なものでありそれぞれ明確な役割があります。

どこに何をすればどう動くのか理解するために、まずは各フォルダの役割を理解しましょう。

表 2.1: 自動生成されるフォルダ

フォルダ名	概要
manifest	アプリがどのようなものなのか OS に伝えるためのもの。使用する権限や OS からどのように呼び出せるかを記載する
java	ソースコードを記載する
java(androidTest)	UI テストコードを記載する
java(test)	Unit テストコードを記載する
res/drawable	画像ファイルの保管場所
res/layout	レイアウトファイル
res/menu	メニューファイル
res/mipmap	主にアイコンファイルを設置する
res/navigation	画面遷移元、遷移先を指定する
res/values	静的値を記載する。(文字列、カラーコード、共通のレイアウト設定等)
Gradle Scripts	必要なライブラリの指定や、ビルドの設定を記載する

実装時に最も使用するフォルダは java になるはずです。実際の開発の流れは、java フォルダ配下に Kotlin のコードを記載しビルド・デバッグ後、問題があれば修正となります。各フォルダの詳しい役割は、本格的に使用する際に詳細を説明するので、今は各フォルダの役割をざっくり理解していれば大丈夫です。

## 2.7 自分のコードを管理する

Android アプリを開発する準備が整いましたが、最後の準備として開発したものを管理する方法を説明します。

Android アプリ開発に止まらず、一般的にソースコードは Git というツールを使って管理することが多いです。Git は別名、分散型バージョン管理システムとも呼ばれ、類似ツールには Mercurial というツールがあります。概念は共通なので、本コースでは Git に絞って説明していきます。

### 2.7.1 分散型バージョン管理システム

Git で最低限、理解しておく必要がある概念は、この分散型バージョン管理システムとブランチです。分散型バージョン管理システムがどういうものなのか説明します。

このシステムは複数人で一つのものを作成することを念頭に置いており、各人の作業が円滑進むように構築されています。概念図を図 2.13 に記載します。

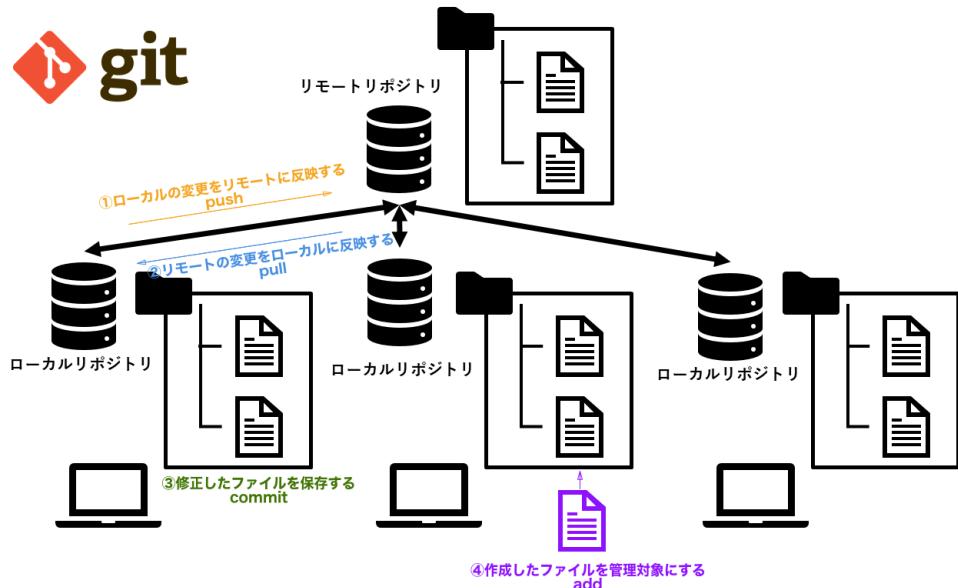


図 2.13: 分散型バージョン管理システム概念図

**リモートリポジトリ** 複数人で 1 つのものを開発するので、最終的には 1 つのものとして成果物を管理する必要があります。その唯一の管理対象がリモートリポジトリと呼ばれています。リモートリポジトリは基本的に 1 つしか存在しません。

**ローカルリポジトリ** リモートリポジトリのコピーを自分の PC に特定の方法で保存したものです。開発はローカルリポジトリで行い、切りのいいタイミングでリポートリポジトリに反映します。

表 2.2: 主要なコマンド

コマンド名	概要
push	ローカルの変更をリモートに反映する
pull	リモートの変更をローカルに反映する
commit	修正したファイルを保存する
add	作成したファイルを管理対象にする

## 2.7.2 ブランチ管理

Git にはブランチという機能があり、実際の開発はこのブランチ機能を利用して、他の人の開発とバッティングしないように自分の開発作業を進めています。ブランチのワークフローを図 2.14 に記載します。

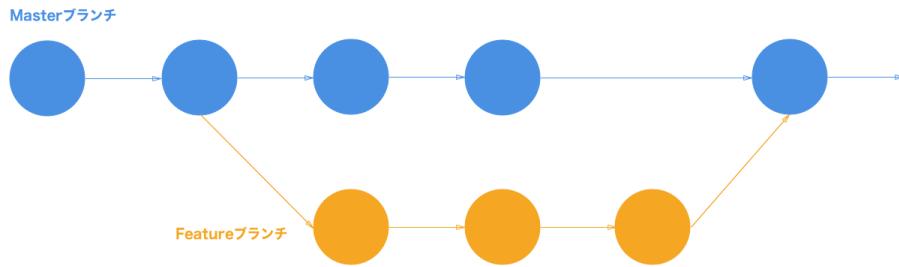


図 2.14: ブランチの概念図

図 2.14 の●はコミットを示しており、開発では大きく分けて二つのブランチを使用します。

Master ブランチ実装完了したものを追加する。常に動作する状態を保つておく

Feature ブランチ機能開発用ブランチ。主に機能ごとに作成し、開発が完了しだい Master ブランチに取り込む (merge)

## 2.8 実際に Git を操作する

この章では実際に Git を操作できるよう設定し、一通りの操作を行います。

## 2.8.1 リモートリポジトリを作成する

<https://github.com> にアクセスしアカウントを作成する。

作成完了後、図 2.15 のような画面にアクセスできるようになります。

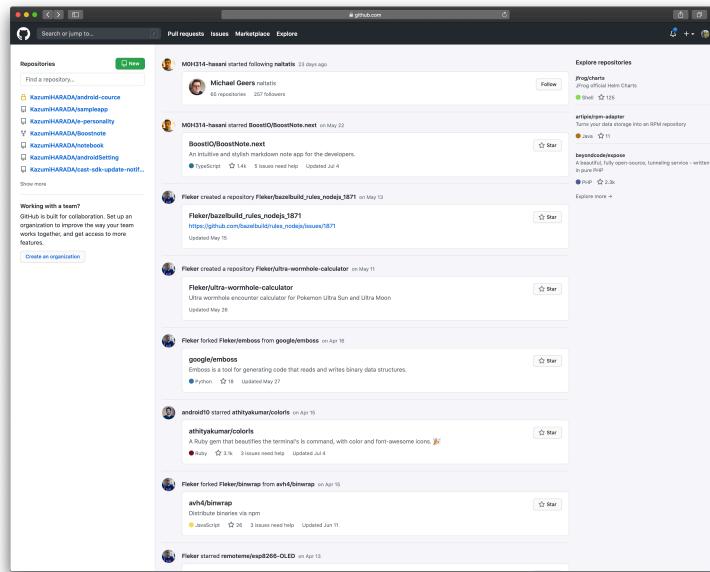


図 2.15: GitHub のトップページ

Github のトップページにアクセスできたら、左上にある New ボタンから新規リモートリポジトリを作成します。Repository Name には自分にわかりやすい名前を入力し、Public リポジトリとして作成します。

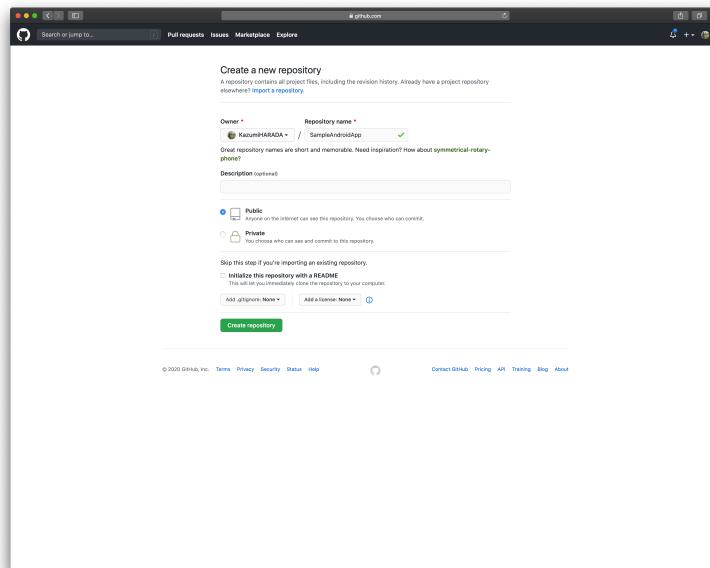


図 2.16: リモートリポジトリ作成画面

リポジトリ作成完了後、ローカルのプロジェクトとリモートリポジトリを紐づける必要があります。まずは、ローカルリポジトリを初期化します。コマンドの操作は、図 2.17 のように AndroidStudio から Terminal を起動して行ってください。

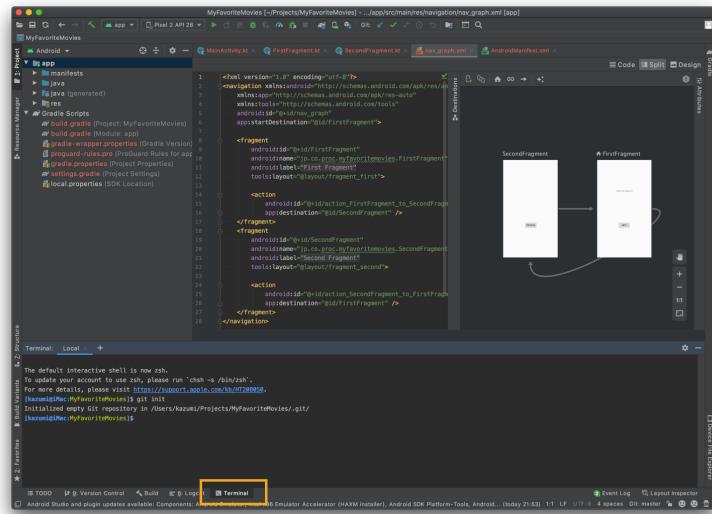


図 2.17: AndroidStudio からターミナルを起動

### 2.8.2 ローカルリポジトリ設定する

ローカルのプロジェクトをローカルリポジトリとして初期化します。

```
1 | $ git init
```

管理対象として全てのコードを追加します。

```
1 | $ git add --all
```

最初のコードをコミットします。最初は全てのコードを追加するので下記コマンドを実行してください。

```
1 $ qit commit -m "first commit"
```

コミットが完了したらリモートリポジトリとの紐付け設定を行います。Githubのリポジトリ作成完了画面から、自分のリモートリポジトリのURLをコピーしておくと簡単です。



図 2.18: GitHub からリモートリポジトリの URL を取得

```
1 $ git remote add origin 自分のリモートリポジトリのURL
```

最後にローカルの修正をリモートリポジトリにプッシュします。

```
1 $ git push -u origin master
```

プッシュが完了すると、リモートリポジトリ上で自分のコードが追加されていることを確認できます。

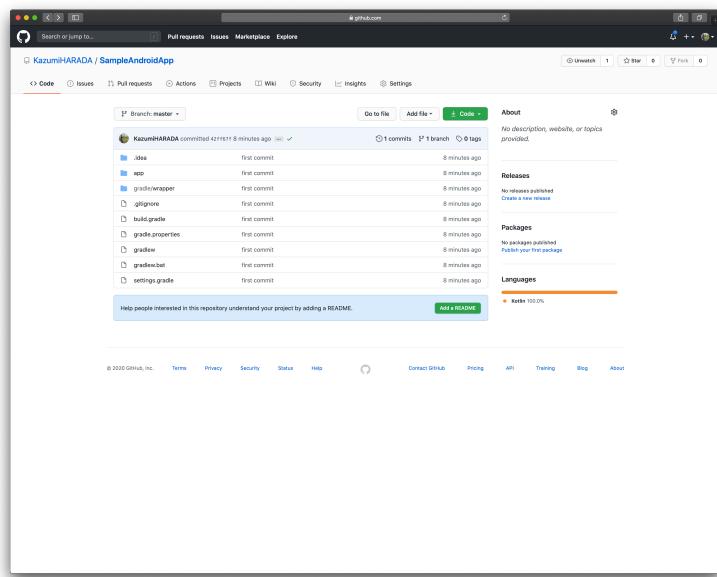


図 2.19: GitHub にプッシュ完了後の表示

### 2.8.3 自分のコードを追加する

Git と Github の設定が完了したところで、試しに自分のコードを一度書いてみましょう。

res/values 配下にある string.xml ファイルを開いてください。hello<sub>first</sub>fragment という名前で設定してある Hello<sub>first</sub>fragment を好きな内容に置き換えます。

今回は Hello first fragment を こんにちは に置き換えています。

```
1 <!-- 変更前 -->
2 <string name="hello_first_fragment">Hello first ←
3     fragment</string>
4 <!-- 変更後 -->
5 <string name="hello_first_fragment" こんにちは
6     ></string>
```

AndroidStudio でビルドしてみると変更が反映されていることがわかります。

そして今回着目していただきたい点は、図 2.20 のオレンジ色の枠の部分です。先ほど修正したファイル名が青色になっていると思います。これは**未コミットの修正**を示しています。ファイル上は保存されていても、Git に保存されていない状態のため、切りのいいタイミングでコミットしておきましょう。

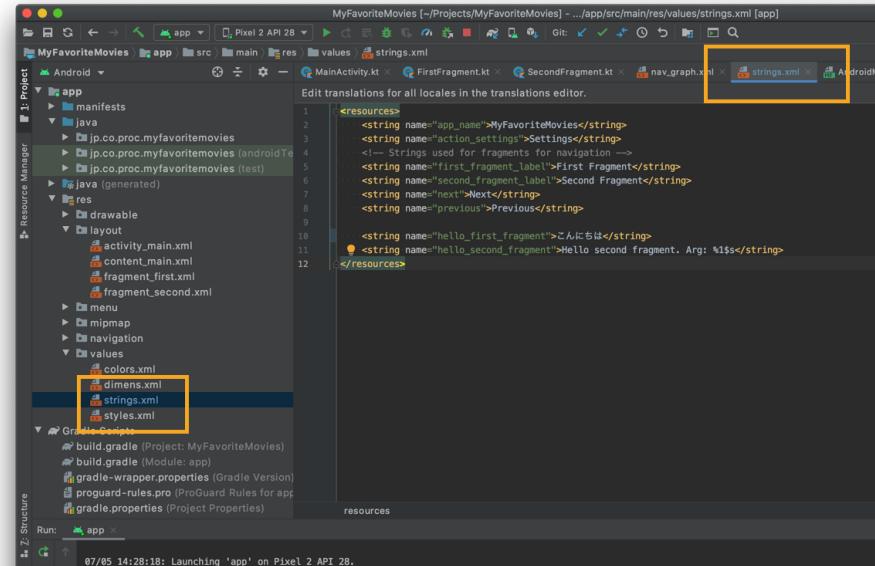


図 2.20: string.xml 修正後の表示

それでは、修正をコミットします。

Ctrl+V ボタンを押下して、VCS Option を表示します。

コミットを選択し、図 2.21 の画面を表示します。

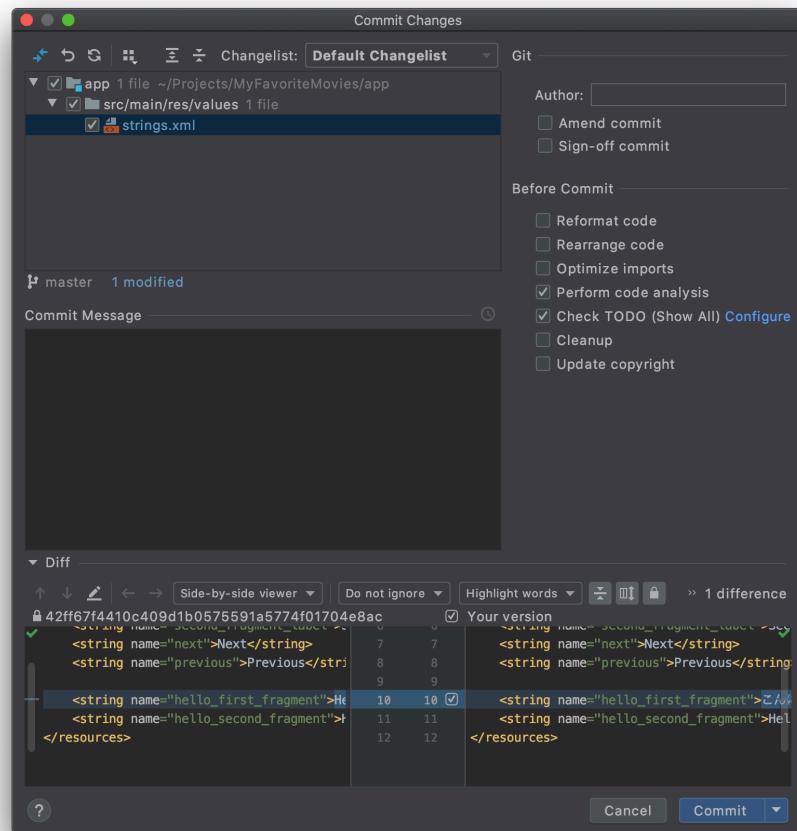


図 2.21: コミット画面

コミットメッセージを入力してコミットボタンを押します。

コミットが完了したらプッシュしてみましょう。

コミットと同じように Ctrl+V ボタンを押下して、VCS Option を表示します。

プッシュを選択し、図 2.22 の画面を表示します。

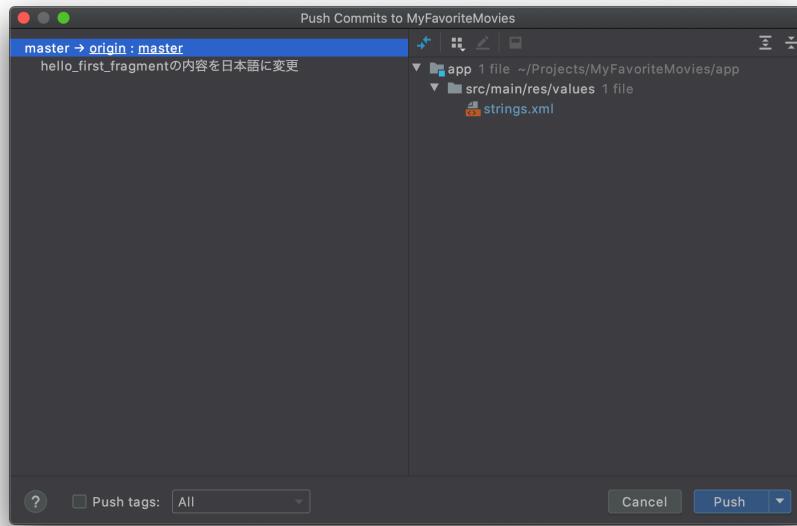


図 2.22: プッシュ画面

プッシュボタンを押します。もし、下記のようなダイアログが表示された場合は、Github のユーザ名とパスワードを入力しましょう。



図 2.23: リモートリポジトリ認証情報入力画面

最後に Push が完了したら、自分の Github のページを確認してみてください。変更が反映されているはずです。

## 2.9 本当に動いているのか確かめる

この章の最後に、開発時に最も必要なデバッグの方法について説明します。

Android のデバッグは主に BreakPoint か、Log 出力を利用します。

## 2.9.1 BreakPoint を使用する

BreakPoint は、処理を止めたい部分に設定します。止める部分の左側にあるコードの行数当たりをクリックすると、赤い丸印がつきます。これが、BreakPoint の設定です。（図 2.24 のオレンジの枠）

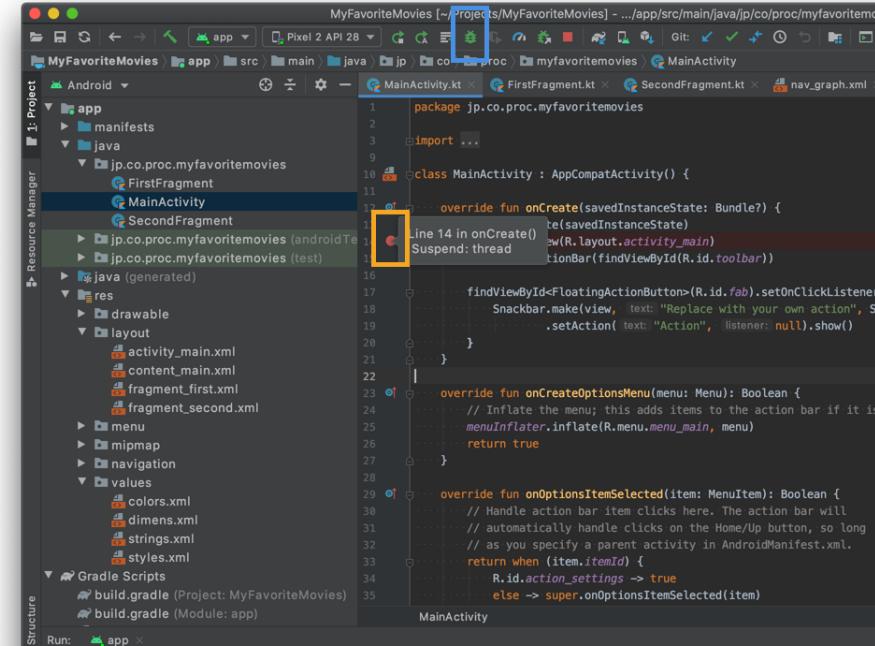
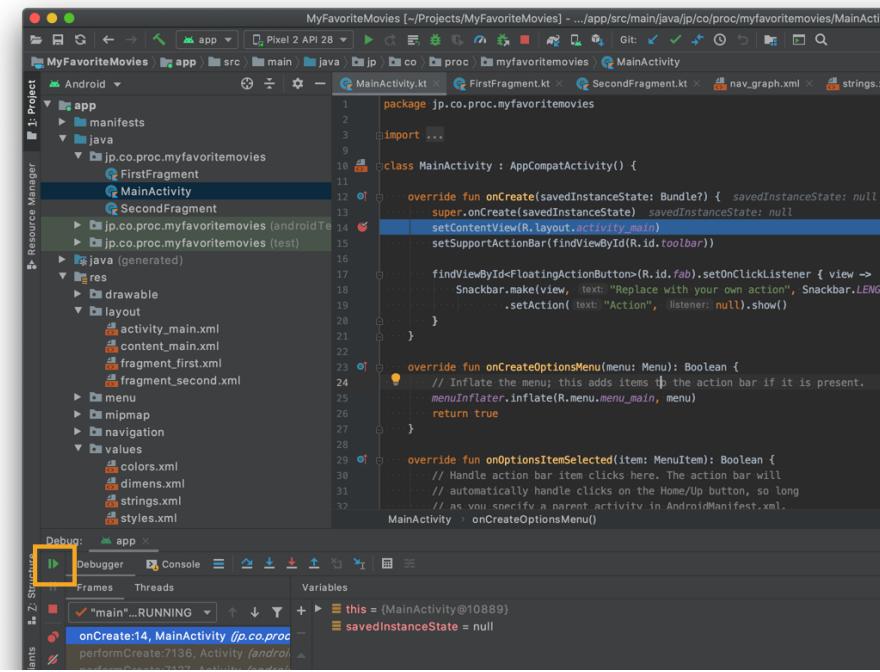


図 2.24: BreakPoint を設定

実際に、処理が停止するか試す場合は、図 2.24 のブルーの枠にあるデバッグ実行ボタンを押して処理を開始してください。設定した BreakPoint の部分で処理が一時的にストップすると思います。このように処理を止めて、その時に何が起きているのかを詳細に確認することができます。（図 2.25）停止後に処理を再開する場合は図 2.25 のオレンジの枠にあるスキップボタンをクリックします。



```
MyFavoriteMovies [~/Projects/MyFavoriteMovies] - .../app/src/main/java/jp/co/proc/myfavoritemovies/MainActivity.kt
MainActivity.kt x FirstFragment.kt x SecondFragment.kt x nav_graph.xml x strings.xml
package jp.co.proc.myfavoritemovies
import ...
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setSupportActionBar(findViewById(R.id.toolbar))
        ...
    }
    ...
    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        // Inflate the menu; this adds items to the action bar if it is present.
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }
    ...
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        return super.onOptionsItemSelected(item)
    }
}

```

図 2.25: BreakPoint を使ってデバッグ実行

## 2.9.2 ログ出力を使用する

基本的には BreakPoint で充分、開発可能ですが、時々処理を止められない箇所でバグが発生することがあります。その時には、処理の途中にログを埋め込み、実行後にログを確認します。使う頻度は少ないかもしれません、必要な方法なので説明しておきます。

コードの途中に図 2.26 のようにログを挿入し、デバッグ実行します。

```
package jp.co.proc.myfavoritemovies
import ...
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setSupportActionBar(findViewById(R.id.toolbar))
        Log.d(tag: "MyLog", msg: "確認用ログ")
        findViewById<FloatingActionButton>(R.id.fab).setOnClickListener {
            Snackbar.make(it, text: "Replace with your own action", duration: Snackbar.LENGTH_LONG).setAction(text: "Action", listener: null).show()
        }
    }
    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        // Inflate the menu; this adds items to the action bar if it is available.
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // MainActivity > onCreate()
    }
}
```

図 2.26: 処理の途中にログを挿入

プロジェクトの最下部にある Logcat をクリックして、ログを表示してください。おそらくすごい勢いで Log が表示されていると思います。このままだと、ログを出しても必要な情報にたどり着けないので、先ほど入力したログの Tag で絞り込みます。

```
2020-07-05 15:10:27.571 13586-13586/jp.co.proc.myfavoritemovies D/MyLog: 確認用ログ
```

図 2.27: 挿入したログを表示

このようにして、出力したログを確認し、問題を特定します。

## 2.10 まとめ

この章では Android の基本的な開発方法を学びました。さらに開発に必要な Git の使用方法も理解できたと思います。よりこれらの理解を深めるために課題に取り組みましょう。

## 2.10.1 課題

1. 2.9.2 で追加したコードをコミットしてプッシュしてみましょう。  
また、完了後は GitHub で反映されていることを確認しましょう。
2. 下記プロジェクトをクローンして PDF ファイルを入手しましょう。

```
<!-- 変更前 -->
<string name="hello_first_fragment">Hello ←
    first fragment</string>
<!-- 変更後 -->
<string name="hello_first_fragment">こんにちは
    </string>
```

# 3. Kotlinに触れる

## 3.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 3.2 Kotlin とは

### 3.3 変数と関数

### 3.4 クラス

### 3.5 制御構文

### 3.6 例外処理

### 3.7 コレクションクラス

### 3.8 クラス設計 1

### 3.9 クラス設計 2

### 3.10 まとめ

# **第II部**

## **Android應用**

subfiles

# 4. Androidをあやつる

## 4.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 4.2 Android ライフサイクル

### 4.2.1 Manifest

### 4.3 Activity

### 4.4 Fragment

### 4.5 Service

### 4.6 Intent

#### 4.6.1 明示的 Intent

#### 4.6.2 暗黙的 Intent

### 4.7 Gradle

### 4.8 主要なライブラリ

### 4.9 まとめ

subfiles

# 5. 思い通りのレイアウトを作る

## 5.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 5.2 レイアウト

5.2.1 LinearLayout

5.2.2 RelativeLayout

5.2.3 FrameLayout

5.2.4 ConstraintLayout

## 5.3 コンポーネント

5.3.1 RecyclerView

5.3.2 AdapterView

5.3.3 Appbar

5.3.4 Snackbar

5.3.5 ...

## 5.4 アニメーション

5.4.1 RippleEffect

5.4.2 ...

## 5.5 デザインの基礎知識

5.5.1 色相

5.5.2 色の意味

5.5.3 フォント

5.5.4 フォントの意味

5.5.5 良いレイアウト悪いレイアウト

## 5.6 まとめ

subfiles

# 6. データを管理する

## 6.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 6.2 Database とは

### 6.2.1 SQL の基礎的な使い方

## 6.3 Android における Database の役割

### 6.3.1 ローカル Database

### 6.3.2 SharedPreference

## 6.4 API 通信

### 6.4.1 HTTP 通信とは

## 6.5 Json データの取り扱い

### 6.5.1 Gson の使い方

### 6.5.2 リクエストキャッシュ

## 6.6 まとめ

# **第III部**

# **Android 発展**

subfiles

# 7. Androidをあやつる

## 7.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 7.2 クライアントアーキテクチャ

### 7.2.1 MVC

### 7.2.2 MVVM

### 7.2.3 CleanArchitecture

## 7.3 Firebaseを使う

### 7.3.1 Analytics

### 7.3.2 Crashlytics

### 7.3.3 RemoteConfig

### 7.3.4 CloudMessaging

## 7.4 ...

## 7.5 まとめ

subfiles

# 8. アプリをリリースする

## 8.1 この章の目標

- Android アプリを動かせるようになる
- Android のコードの大まかな役割を理解する
- ソースコードの管理ができるようになる

## 8.2 Build Variants

## 8.3 Proguard

## 8.4 リリースの方法

## 8.5 リリース後の運用

## 8.6 まとめ

## 第IV部

### Examples

## 9. Math

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam a orci ornare nibh tincidunt molestie sed nec tellus. Morbi non sapien id lorem posuere pretium. Vestibulum commodo cursus purus, a elementum sem imperdiet sit amet.

$$\begin{array}{ccccccc} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ 0 & 0 & 1 & -2 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 0 & 0 & \cdots & 1 & -2 \end{array}$$

Phasellus posuere dolor dignissim aliquam tempus. Morbi egestas felis in lorem varius, ac egestas ante lacinia. Nulla sed ultrices dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras rutrum nisl ut ligula tristique gravida. Fusce augue enim, elementum in euismod a, mollis et nunc. Duis vel ipsum eros. Pellentesque odio nisl, bibendum non ex in, sodales laoreet enim. Curabitur eu leo ac urna scelerisque dictum at non tellus.

$$\begin{aligned} \text{codeword} &= m(x) \cdot g(x) \\ &= (1 + x + x^3) \cdot (1 + x + x^3) \\ &= 1 + x + x^3 + x + x^2 + x^4 + x^3 + x^4 + x^6 \\ &= 1 + 2x + x^2 + 2x^3 + 2x^4 + x^6 \\ &= (1, 0, 1, 0, 0, 0, 1) \text{ mod } 2 \end{aligned}$$

# 10. Code

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam a orci ornare nibh tincidunt molestie sed nec tellus. Morbi non sapien id lorem posuere pretium. Vestibulum commodo cursus purus, a elementum sem imperdiet sit amet. Phasellus posuere dolor dignissim aliquam tempus. Morbi egestas felis in lorem varius, ac egestas ante lacinia. Nulla sed ultrices dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras rutrum nisl ut ligula tristique gravida. Fusce augue enim, elementum in euismod a, mollis et nunc. Duis vel ipsum eros. Pellentesque odio nisl, bibendum non ex in, sodales laoreet enim. Curabitur eu leo ac urna scelerisque dictum at non tellus.

Listing 10.1: A sample Python listing

```
1 import requests
2
3 class Scanner:
4     """The Scanner scans items on sandbox escaping. ↵
5         """
6
7     def __init__(self, item):
8         """Initialize class for the given item."""
9
10    self.__queue_item = item
11
12    item.mount("http://", requests.HTTPAdapter)
13    item.mount("https://", requests.HTTPAdapter ↵
14        )
15
16    @staticmethod
17    def hello():
18        """Print an example message."""
19
20        print("Hello_World!")
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit (see 10.1).

# 11. Citation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam a orci ornare nibh tincidunt molestie sed nec tellus. Morbi non sapien id lorem posuere pretium. Vestibulum commodo cursus purus, a elementum sem imperdiet sit amet. Phasellus posuere dolor dignissim aliquam tempus. Morbi egestas felis in lorem varius, ac egestas ante lacinia. Nulla sed ultrices dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras rutrum nisl ut ligula tristique gravida. Fusce augue enim, elementum in euismod a, mollis et nunc. Duis vel ipsum eros. Pellentesque odio nisl, bibendum non ex in, sodales laoreet enim. Curabitur eu leo ac urna scelerisque dictum at non tellus<sup>1</sup>.

---

<sup>1</sup>**testcitation.**

## 12. Table

The schedule 12.1 may be subject to changes on short notice. These activities are the *do* stage of the Deming Wheel.

表 12.1: List of audit activities to perform

Date	Activity
01-01-2018	On-site audit activities:
09:00	<b>Opening Meeting</b> Establish personal contact with the auditee, confirm the plan for carrying out the audit, explain and confirm the activities, roles and responsibilities of those involved in the audit, confirm communication arrangements and reporting requirements and provide an opportunity for the auditee to clarify issues and ask any questions.
10:00	<b>Documentation</b> Documentation about scoping, planning, implementing, monitoring and reviewing.
12:00	<b>Information security management system (ISMS)</b> This meeting will be held with security officers as well as the process manager.
21:00	<b>Closing Remarks (Audit Follow-Up)</b> The primary purpose of this meeting is to present the audit findings and conclusions, ensure a clear understanding of the results, and agree on the timeframe for corrective actions. This meeting can be held at the end of the audit.