

EXAMPLE UNIVERSITY

INFORMATION SECURITY

INF5021 - EXAMPLE SUBJECT NAME

Example Paper Name

LATEX

Author(s):

Kazumi Harada

Supervisor(s):

Prof. Example NAME

July 16, 2020

Contents

| | |
|--------------------------------------|----------|
| 第1部 Android 基礎 | 2 |
| 第1章 イントロダクション | 3 |
| 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 |
| 第2章 Android アプリを作る | 9 |
| 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 エミュレータをインストールする | 13 |
| 2.4.1 エミュレータとは | 13 |
| 2.4.2 エミュレータのインストール方法 | 13 |
| 2.5 アプリをビルドする | 16 |
| 2.6 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 |
| 第3章 Kotlinにふれる | | 30 |
| 3.1 | この章の目標 | 30 |
| 3.2 | Kotlinとは | 30 |
| 3.2.1 | 関数型プログラミング | 30 |
| 3.2.2 | オブジェクト指向プログラミング | 31 |
| 3.3 | 変数と関数 | 32 |
| 3.3.1 | 変数 | 32 |
| 3.3.2 | 不变(val)と可変(var)の使い分け | 33 |
| 3.3.3 | 関数 | 34 |
| 3.4 | クラス | 35 |
| 3.5 | コレクションクラス | 42 |
| 3.5.1 | List | 42 |
| 3.5.2 | Set | 42 |
| 3.5.3 | Map | 43 |
| 3.6 | 制御構文 | 43 |
| 3.6.1 | 条件分岐 | 44 |
| 3.6.2 | 繰り返し | 46 |
| 3.7 | 例外処理 | 47 |
| 3.8 | クラス設計(抽象的な概念をクラスとして定義する) | 47 |
| 3.9 | まとめ | 49 |
| 第4章 Androidをあやつる | | 51 |
| 4.1 | この章の目標 | 51 |
| 4.2 | Androidライフサイクル | 51 |
| 4.2.1 | Manifest | 51 |
| 4.3 | Activity | 51 |
| 4.4 | Fragment | 51 |
| 4.5 | Service | 51 |
| 4.6 | Intent | 51 |
| 4.6.1 | 明示的Intent | 51 |
| 4.6.2 | 暗黙的Intent | 51 |
| 4.7 | Gradle | 51 |
| 4.8 | 主要なライブラリ | 51 |
| 4.9 | まとめ | 51 |

| | |
|---|-----------|
| 第 II 部 Android 応用 | 52 |
| 第 5 章 思い通りのレイアウトを作る | 54 |
| 5.1 この章の目標 | 54 |
| 5.2 レイアウト | 55 |
| 5.2.1 LinearLayout | 55 |
| 5.2.2 RelativeLayout | 55 |
| 5.2.3 FrameLayout | 55 |
| 5.2.4 ConstraintLayout | 55 |
| 5.3 コンポーネント | 55 |
| 5.3.1 RecyclerView | 55 |
| 5.3.2 AdapterView | 55 |
| 5.3.3 Appbar | 55 |
| 5.3.4 Snackbar | 55 |
| 5.3.5 | 55 |
| 5.4 アニメーション | 55 |
| 5.4.1 RippleEffect | 55 |
| 5.4.2 | 55 |
| 5.5 デザインの基礎知識 | 55 |
| 5.5.1 色相 | 55 |
| 5.5.2 色の意味 | 55 |
| 5.5.3 フォント | 55 |
| 5.5.4 フォントの意味 | 55 |
| 5.5.5 良いレイアウト悪いレイアウト | 55 |
| 5.6 まとめ | 55 |
| 第 6 章 データを管理する | 57 |
| 6.1 この章の目標 | 57 |
| 6.2 Database とは | 57 |
| 6.2.1 SQL の基礎的な使い方 | 57 |
| 6.3 Android における Database の役割 | 57 |
| 6.3.1 ローカル Database | 57 |
| 6.3.2 SharedPreference | 57 |
| 6.4 API 通信 | 57 |
| 6.4.1 HTTP 通信とは | 57 |
| 6.5 Json データの取り扱い | 57 |
| 6.5.1 Gson の使い方 | 57 |
| 6.5.2 リクエストキャッシュ | 57 |
| 6.6 まとめ | 57 |

| | |
|-----------------------------------|-----------|
| 第 III 部 Android 発展 | 58 |
| 第 7 章 Android をあやつる | 60 |
| 7.1 この章の目標 | 60 |
| 7.2 クライアントアーキテクチャ | 60 |
| 7.2.1 MVC | 60 |
| 7.2.2 MVVM | 60 |
| 7.2.3 CleanArchitecture | 60 |
| 7.3 Firebase を使う | 60 |
| 7.3.1 Analytics | 60 |
| 7.3.2 Crashlytics | 60 |
| 7.3.3 RemoteConfig | 60 |
| 7.3.4 CloudMessaging | 60 |
| 7.4 | 60 |
| 7.5 まとめ | 60 |
| 第 8 章 アプリをリリースする | 62 |
| 8.1 この章の目標 | 62 |
| 8.2 Build Variants | 62 |
| 8.3 Proguard | 62 |
| 8.4 リリースの方法 | 62 |
| 8.5 リリース後の運用 | 62 |
| 8.6 まとめ | 62 |
| 第 IV 部 Examples | 63 |
| 第 9 章 Math | 64 |
| 第 10 章 Code | 65 |
| 第 11 章 Citation | 66 |
| 第 12 章 Table | 67 |

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。

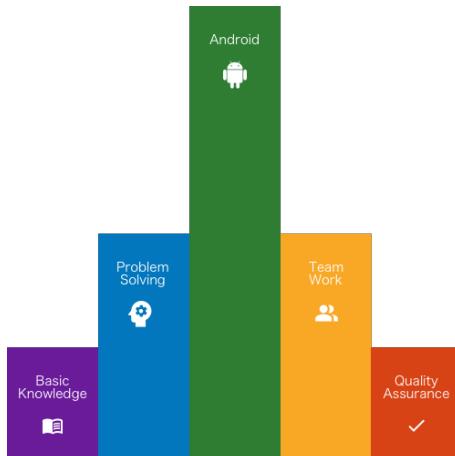


図 1.1: 習得可能なスキルの範囲

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>

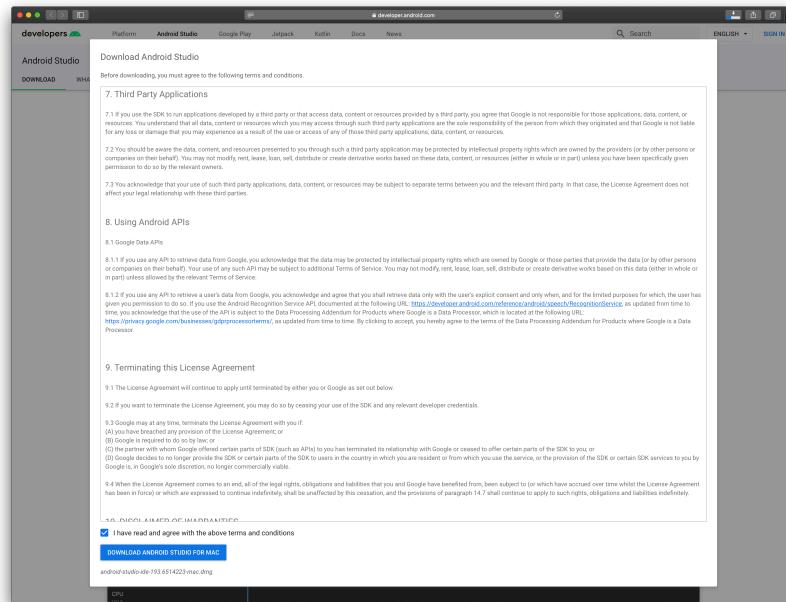


図 2.1: 利用規約に同意してインストール

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。

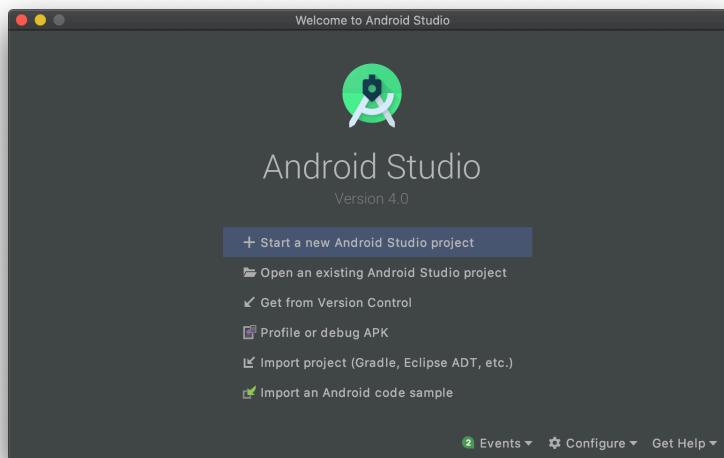


図 2.2: AndroidStudio の初期起動画面

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。

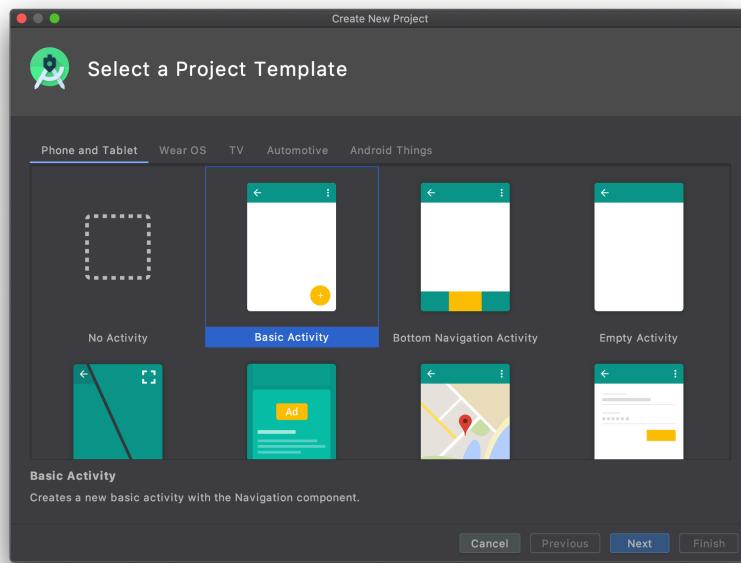


図 2.3: テンプレート選択画面

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

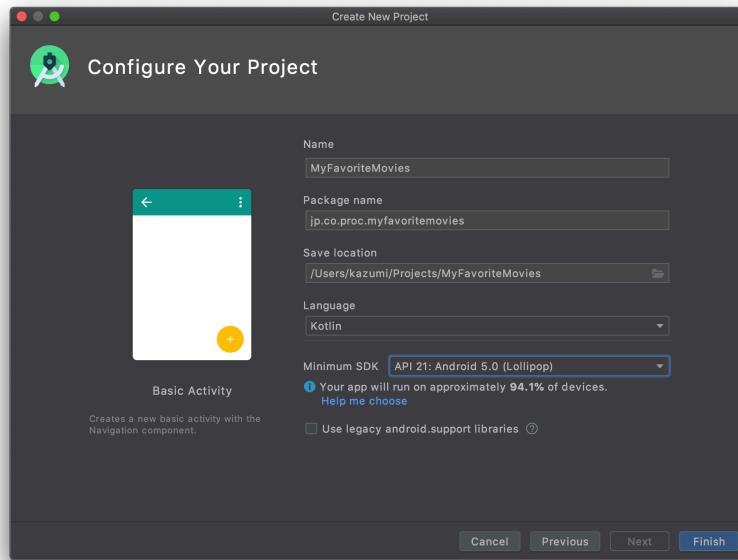


図 2.4: プロジェクト設定画面

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。

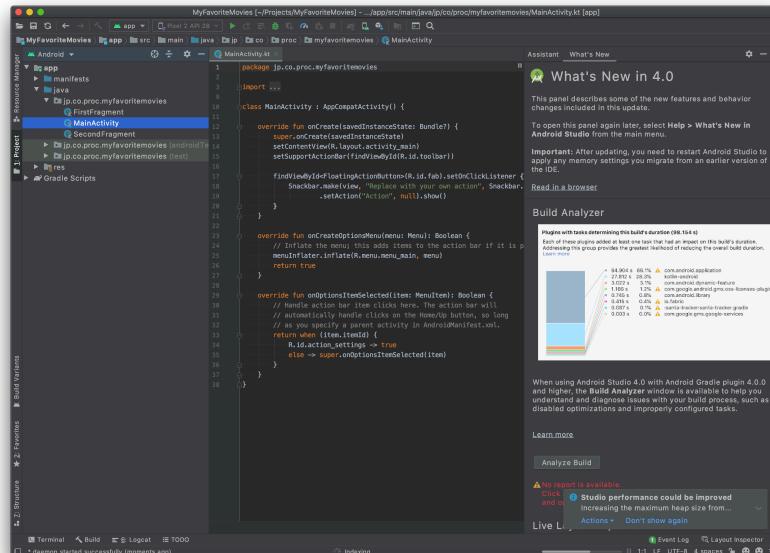


図 2.5: Android アプリ開発画面

2.4 エミュレータをインストールする



最後の準備として、エミュレータ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一度エミュレータ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。

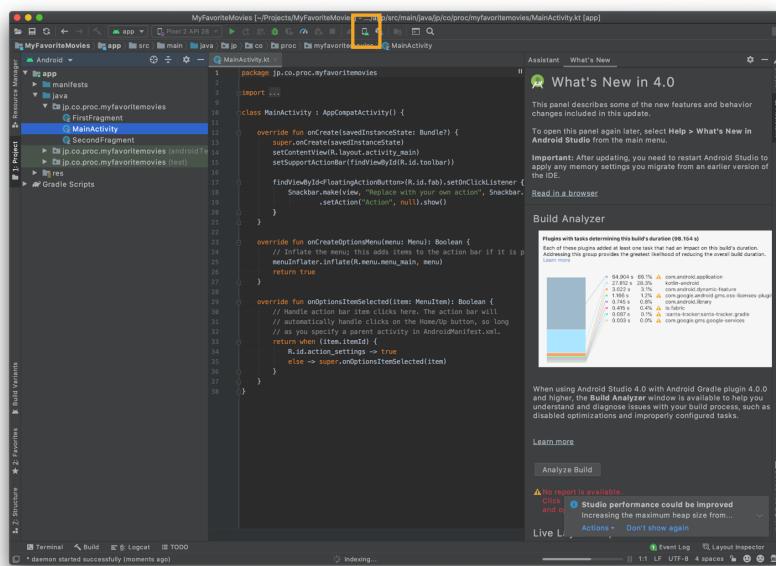


図2.6: AVD Managerボタンをクリック

次に、図2.7のエミュレータを作成ボタンをクリックします。

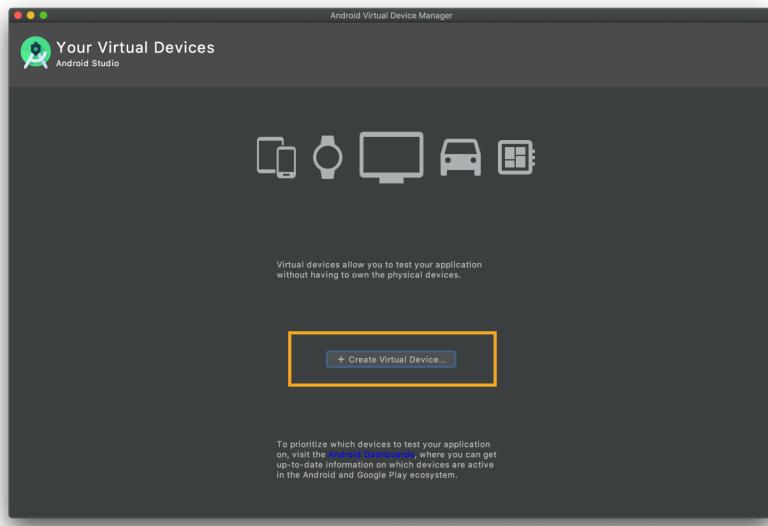


図 2.7: エミュレータを作成

エミュレータを作成するために、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。

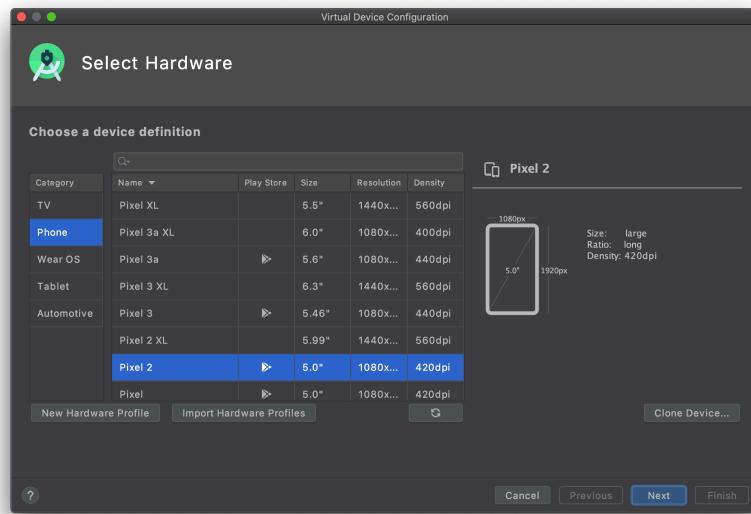


図 2.8: デバイスを選択

次に、図2.9のようにAndroidバーションを選択します。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。

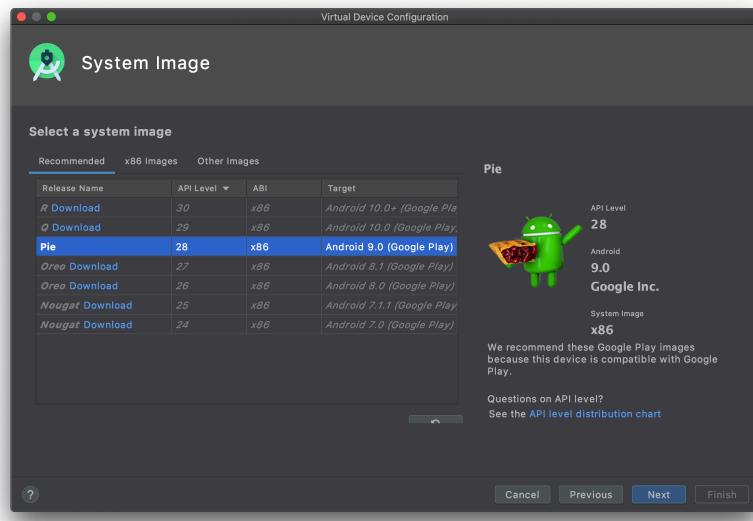


図 2.9: Android バージョンを選択

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。

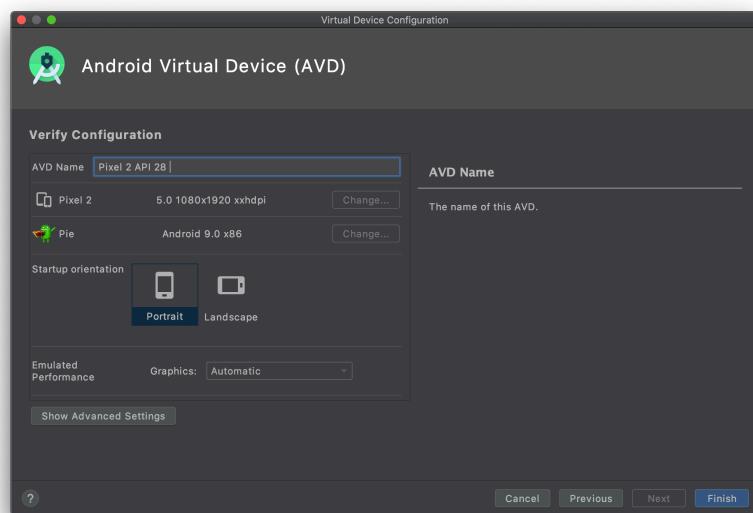


図 2.10: エミュレータ名を設定

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。

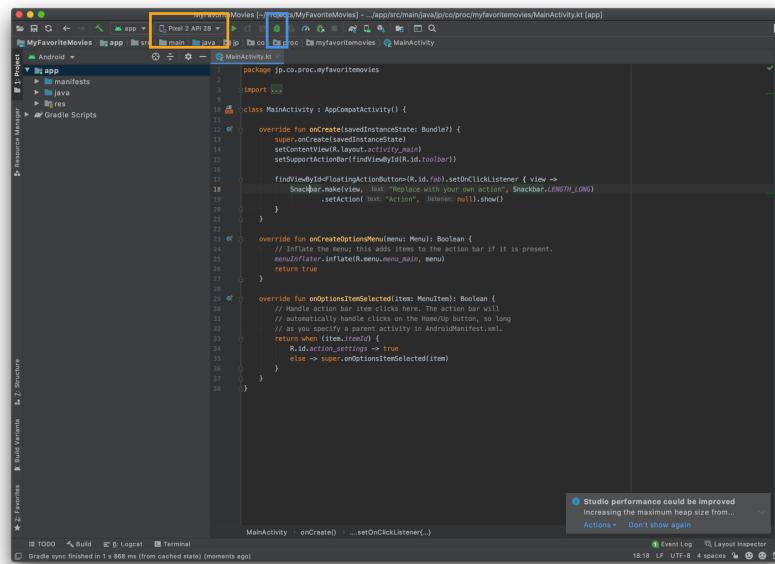


図 2.11: ビルド直前の画面

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。

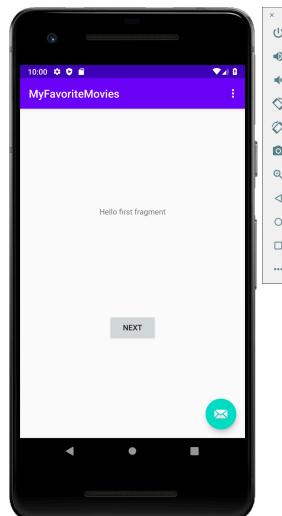


図 2.12: 初期のアプリ画面

2.6 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ことは、この分散型バージョン管理システムの考え方とブランチ管理方法です。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。

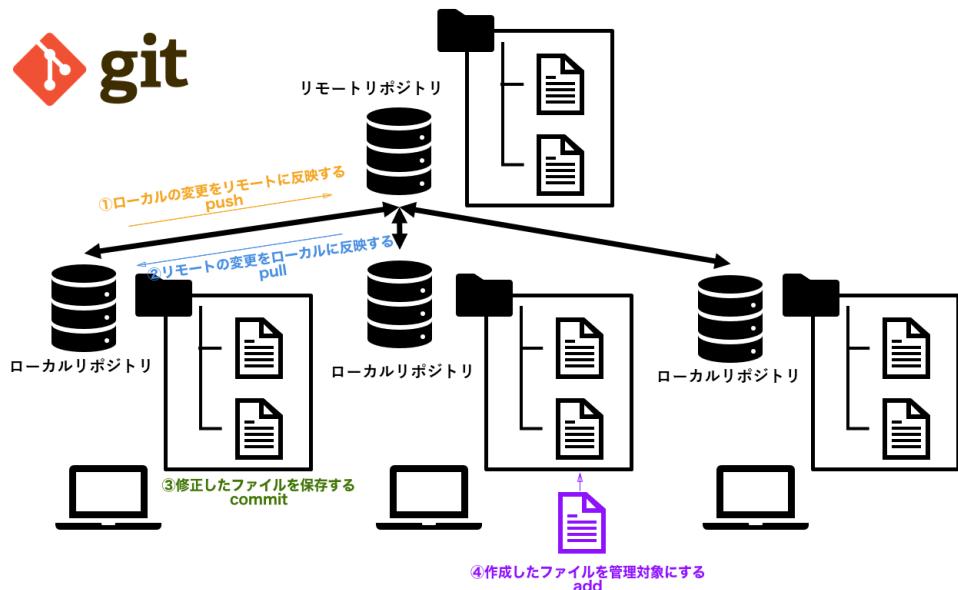


図 2.13: 分散型バージョン管理システム概念図

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。

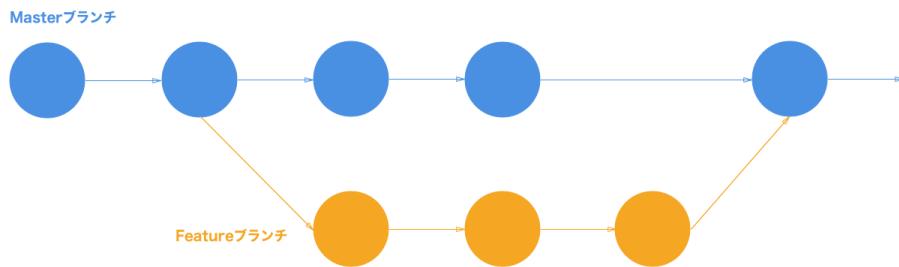


図 2.14: ブランチの概念図

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

Basic Knowledge

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。

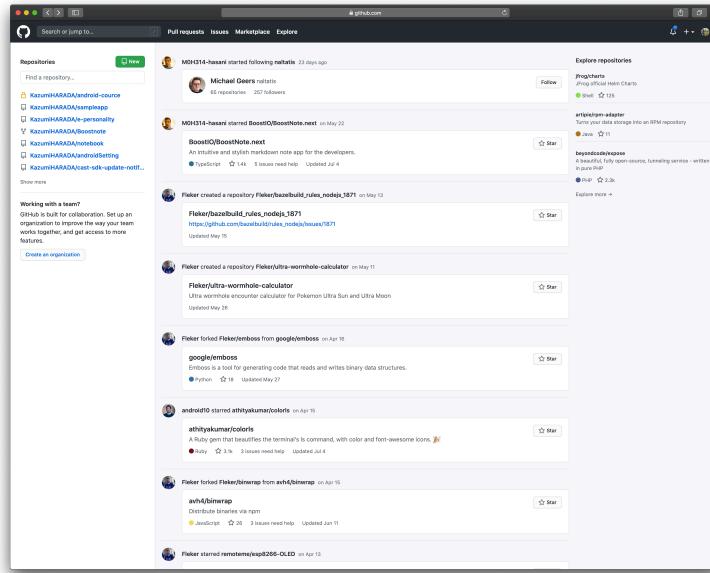


図 2.15: GitHub のトップページ

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。

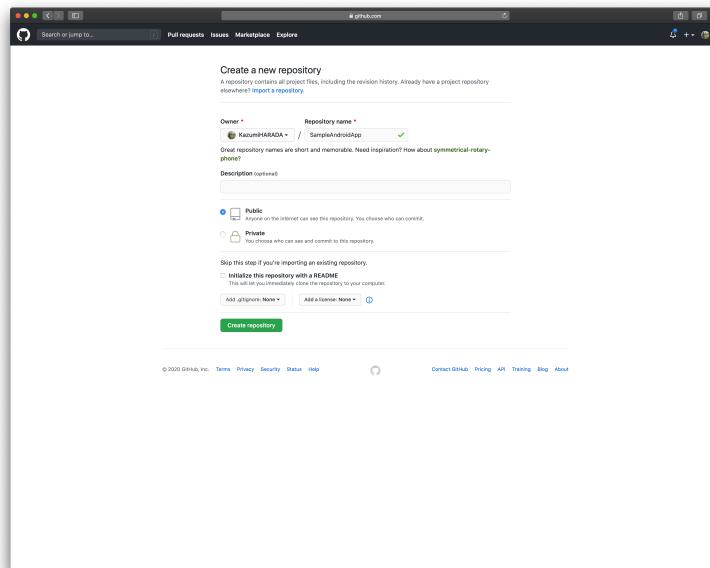


図 2.16: リモートリポジトリ作成画面

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。

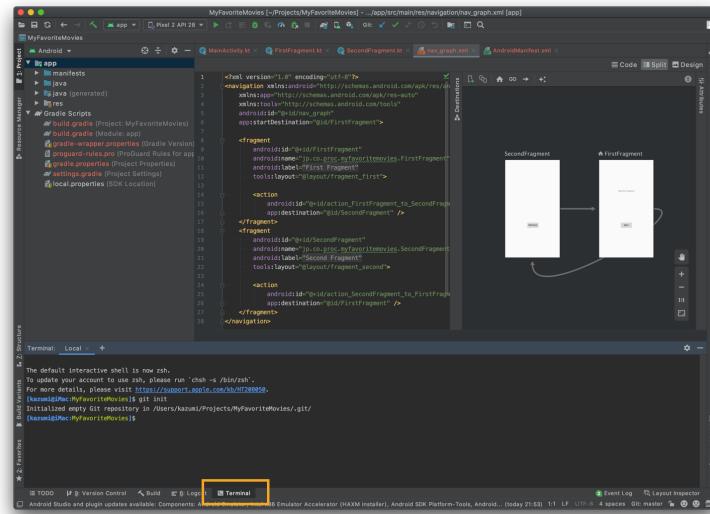


図 2.17: AndroidStudio からターミナルを起動

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 $ git 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。

Quick setup — if you've done this kind of thing before
 Set up in Desktop or [HTTPS](#) [SSH](#) <https://github.com/KazumiHARADA/SampleAndroidApp.git>
Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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。

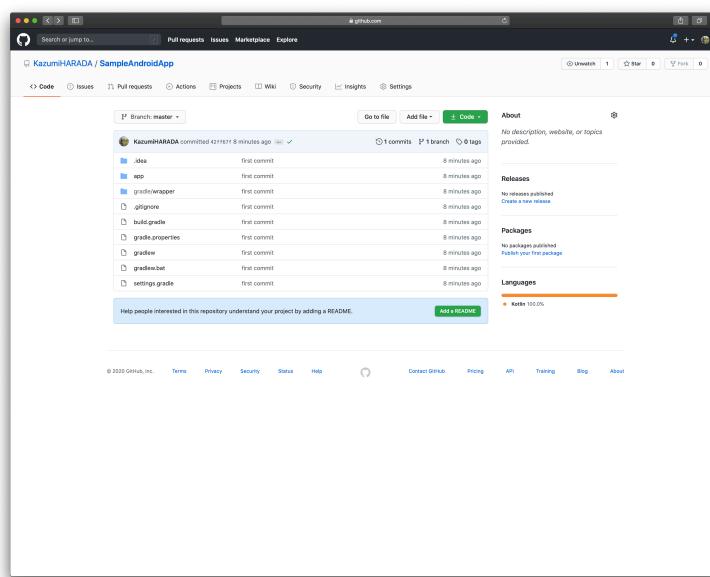


図 2.19: GitHub にプッシュ完了後の表示

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_{first}fragment という名前で設定してある Hello_{first}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 に置き換えていきます。

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

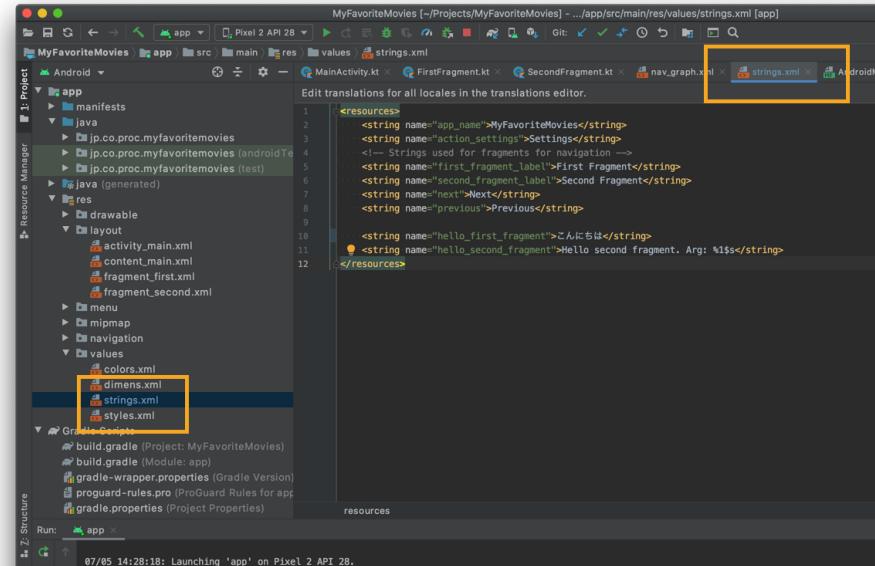


図 2.20: string.xml 修正後の表示

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。

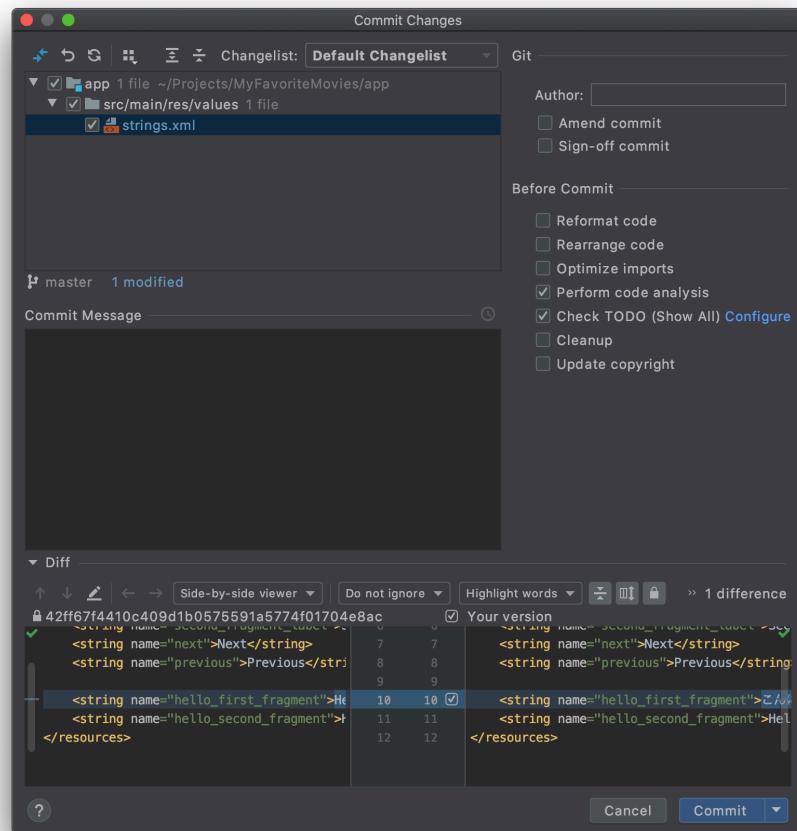


図 2.21: コミット画面

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。

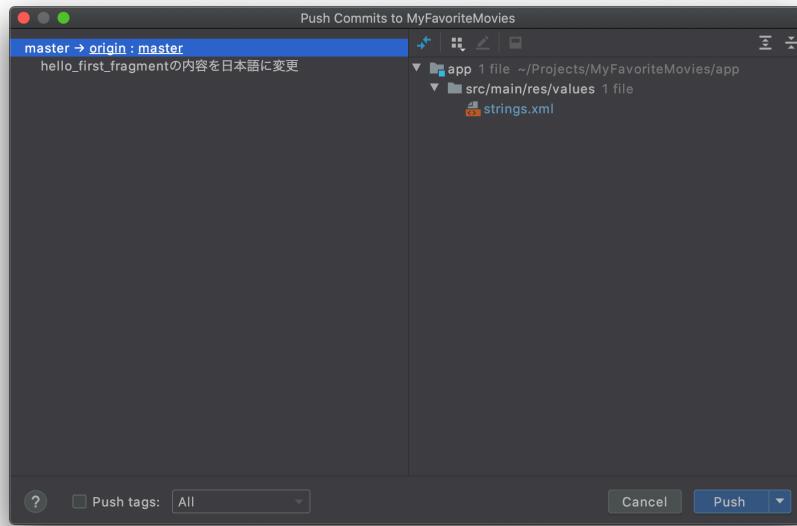


図 2.22: プッシュ画面

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）

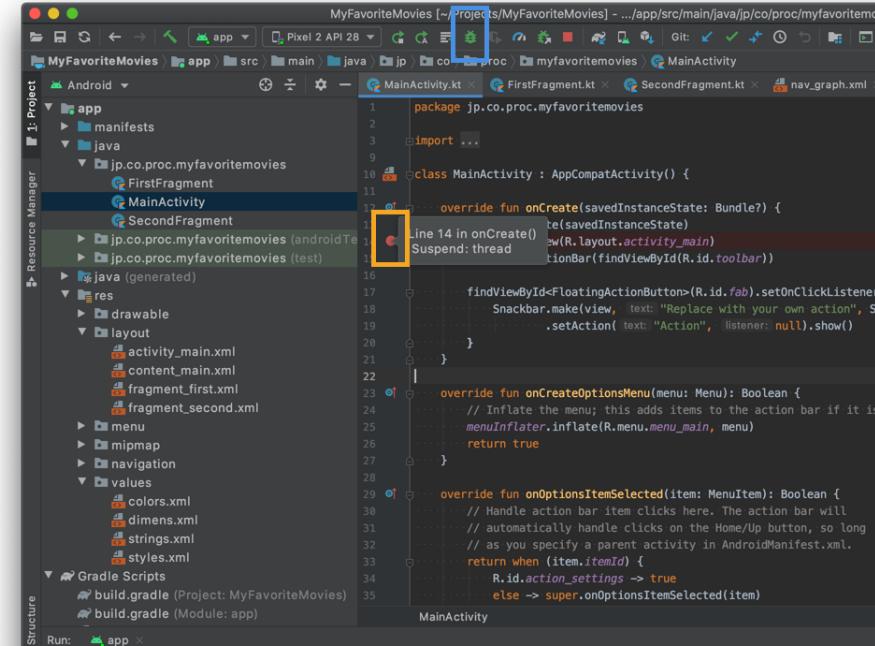


図 2.24: BreakPoint を設定

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)
    }
}
```

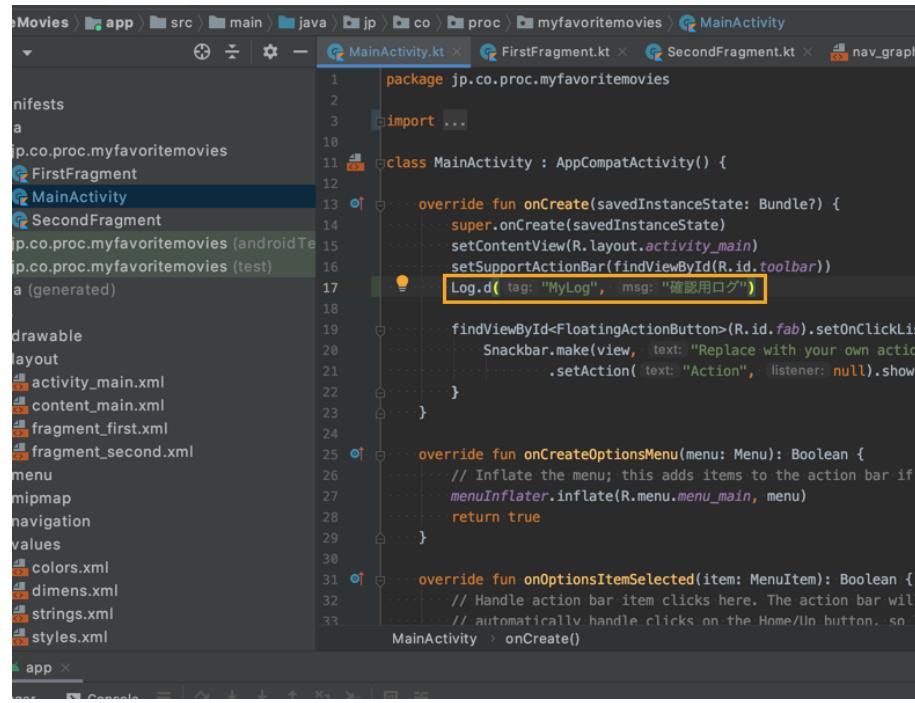
The screenshot shows the Android Studio interface with the code editor open to `MainActivity.kt`. A red circle marks a breakpoint at line 14. The 'Break' button in the toolbar is highlighted with a yellow box. The bottom left corner of the screenshot has a yellow border.

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。

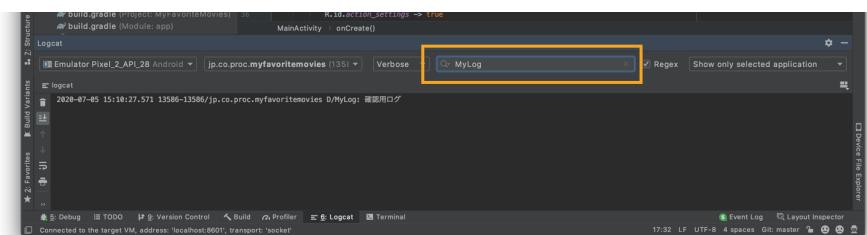
コードの途中に図 2.26 のようにログを挿入し、デバッグ実行します。



```
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("MyLog", "確認用ログ")
        findViewById<FloatingActionButton>(R.id.fab).setOnClickListener {
            Snackbar.make(it, "Replace with your own action", Snackbar.LENGTH_LONG)
                .setAction("Action", 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。



Logcat
Emulator Pixel_2_API_28 Android | jp.co.proc.myfavoritemovies (135) | Verbose | MyLog | Regex | Show only selected application
Connected to the target VM, address: 'localhost:8801', transport: 'socket'
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。
(git clone で検索し、課題を解決してください)

リポジトリ URL <https://github.com/KazumiHARADA/android-source-pub.git>

3. Kotlinにふれる

3.1 この章の目標

- Kotlin の構文を理解する
- 簡単なクラス設計ができるようになる

3.2 Kotlin とは



Kotlin は Android 開発で広く一般的に利用されているプログラミング言語です。元々は Java から派生したもので、Java よりも簡潔かつ安全に記述できるよう設計されています。

Kotlin の特徴は以下の通りです。

- 関数型&オブジェクト指向プログラミング言語
- 静的型付け
- クライアントサイドでもサーバサイドでも使用できる

3.2.1 関数型プログラミング

Kotlin は関数型プログラミングとオブジェクト指向プログラミングの両方の概念をサポートしています。まずは関数型プログラミングのざっくり説明します。

そもそも、関数とはなんなのでしょうか。

例えば、

¹ $1+1=2$

これは式です。1と1を足し合わせると2になるという式になります。

次に、

```
1 f(x) = x + 1
```

これは関数です。xを入力するとxに+1した値が得られます。この場合、もし x に 1 を入力すると 2 が得られます。

では、x に 2 を足した値を得たい場合はどうしたらいいでしょうか？関数型プログラミングでは次のように考えます。

```
1 f(x) = x + 1  
2 j(x) = f(f(x))
```

関数jをx=1で実行すれば、3が得られることが想像できると思います。

関数型プログラミングは、このように小さな機能群を作っていくて、それを組み合わせて一つのものを作ろうという考え方です。

関数型プログラミングの特徴は下記の通りです。

- 第一級関数 - 関数の結果として関数を返すことができます
- イミュータビリティ - 不変なオブジェクトを扱います
- 副作用禁止 - 同じ入力があれば同じ結果を返します

現時点では、完璧にわからなくても大丈夫です。自分で書いていくうちに自然と理解できます。

3.2.2 オブジェクト指向プログラミング

次にオブジェクト指向プログラミングとはどのようなものでしょう？

そもそもオブジェクトとはなんでしょう？

オブジェクト=ものです。

この世に存在する全てのものをオブジェクトと捉えることができます。世の中はこのオブジェクトの相互作用によって成り立っている。それをプログラミングで表現しようと考えたのがオブジェクト指向プログラミングです。

もう少しプログラミング的にいって、変数とか関数とかそのオブジェクトに必要なものを全てまとめたクラスを使ってプログラミングしていきます。クラスはそのオブジェクトの仕様書のようなもので、このクラスから実態であるインスタンスを作成します。

例えば、図3.1のようにクラスとインスタンスを表現することができます。実際に、世の中に存在するもの（インスタンス）は10円、100円、

500円ですが、これらの硬貨には共通のルールがあります。その硬貨という大元の概念がクラスです。

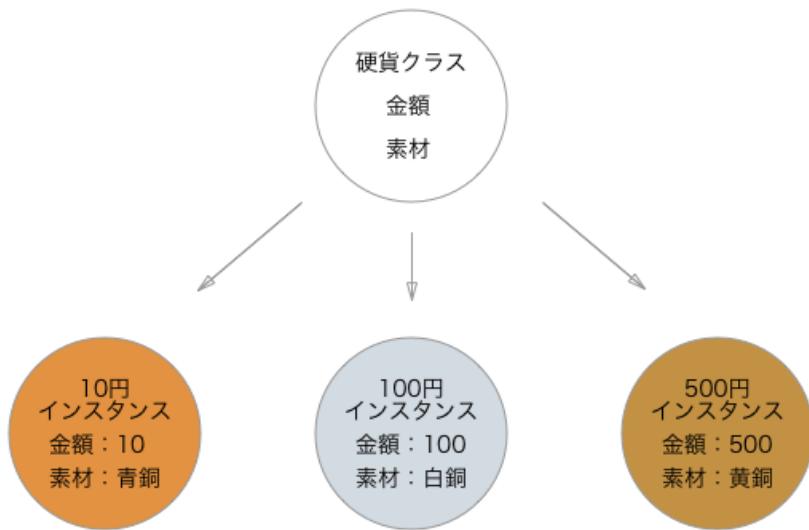


図 3.1: クラスとインスタンスの例

3.3 変数と関数

Basic Knowledge

Kotlin での変数と関数の使い方を理解しましょう。何をやるにしても、必ず使うものなので確実に理解しておきましょう。

3.3.1 変数

Kotlin には 2 種類の変数があります。不変の変数 (val) と可変の変数 (var) です。3.2.1 で示したように、関数型プログラミングでは不変の変数を使用することが推奨されます。

必要な場合をのぞいて、可能な限り val を使用するようにしましょう。

では実際にコードを書いて試してみましょう。MainActivity を開いて、onCreate 関数の setSupportActionBar の直下に、下記の変数を定義してみましょう。

```
1 val result
```

※コードは実際に書いて、ビルドして確認しましょう

このままではエラーになってしまいます。不変なオブジェクトは必ず定義時に初期化する必要があります。

```
1 val result = 0
```

エラーが消えたと思います。次にこの result に計算結果を入れましょう。

```
1 val result = 1 + 1
```

実際にログを追記して、アプリをビルドしてみてください。

```
1 val result = 1 + 1
2 Log.d("MyLog", "$result")
```

ログを絞り込んで、結果を確認してください。

```
1 D/MyLog: 2
```

と表示されていれば、正しく動作しています。

次に可変の変数 var を使ってみましょう。まずは先ほどの val で定義した変数を処理の途中で変更してみましょう。

```
1 val result = 1 + 1
2 Log.d("MyLog", "$result")
3 result = 10 // ここを追加
```

すると、Val cannot be reassigned というエラーが発生します。その名の通り val は定義後、2度と変更できません。

続いて、val を var に変更してみましょう。

```
1 var result = 1 + 1 // ここを変更
2 Log.d("MyLog", "$result")
3 result = 10
4 Log.d("MyLog", "$result")
```

ログに 2 と 10 が表示されていると思います。このようにして var は値をいつでも変更することができます。

3.3.2 不変 (val) と可変 (var) の使い分け

ここまで学んでいれば、プログラミングは覚えることも考えることも多く大変だ感じる方が大半だと思います。ただ、それはどんなエンジニアでも共通の課題です。エンジニアこれから学ぶ多くのことを考慮に入れながら 1つのものを作っていく必要があります。そうすると、必然的に考えることを減らしたいという思いが生まれてきます。そこで活躍するのが val という不变の変数です。不变であるということは、一度内容が分かってしまえば、途中で変更される可能性を捨てることができます。これこそがプログラムの読みやすさにつながっていくのです。なので可能な限り val を使い、どうしようもない場合のみ var を使用するようにしましょう。

3.3.3 関数

続いて関数を使ってみましょう。

MainActivity を開いて、onCreate 関数と onCreateOptionsMenu 関数の間に AddOne 関数を追加してみてください。

```
1  override fun onCreate(savedInstanceState: Bundle?) {
2      super.onCreate(savedInstanceState)
3      setContentView(R.layout.activity_main)
4      setSupportActionBar(findViewById(R.id.toolbar))
5      var result = 1 + 1
6      Log.d("MyLog", "$result")
7      result = 10
8      Log.d("MyLog", "$result")
9
10     findViewById<FloatingActionButton>(R.id.fab).apply {
11         setOnClickListener { view ->
12             Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
13                 .setAction("Action", null).show()
14     }
15
16     // ここを追加
17     fun addOne(x: Int): Int {
18         return x + 1
19     }
20
21     override fun onCreateOptionsMenu(menu: Menu): Boolean {
22         // Inflate the menu; this adds items to the action bar if it is present.
23         menuInflater.inflate(R.menu.menu_main, menu)
24         return true
25     }
```

関数には引数と戻り値というものがあります。そもそも関数とは、何か値を受け取って、何か値を返すものです。その受け取る値を引数。返す値を戻り値と呼びます。

| 関数名 | 引数 | 戻り値の型 |
|--|----|-------|
| <code>fun addOne(x: Int): Int {</code> | | |
| <code> return x + 1</code> | | |
| <code>}</code> | | |

図 3.2: 関数の詳細

関数が定義できたら関数を呼び出してみましょう。

```

1   override fun onCreate(savedInstanceState: Bundle?) {
2       super.onCreate(savedInstanceState)
3       setContentView(R.layout.activity_main)
4       setSupportActionBar(findViewById(R.id.toolbar))
5       var result = 1 + 1
6       Log.d("MyLog", "$result")
7       result = 10
8       result = addOne(result) // ここを追加
9       Log.d("MyLog", "$result")
10
11      findViewById<FloatingActionButton>(R.id.fab).apply {
12          setOnClickListener { view ->
13              Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
14                  .setAction("Action", null).show()
15      }
16  }

```

関数が呼び出されて、2回目の result が 11 になっているはずです。

3.4 クラス



続いて、クラスの作成方法についてです。3.2.2で説明したクラスを実際に作成してみましょう。

最終成果物には必要なくなるクラスなので、わかりやすいように別パッケージを作成しておきましょう。パッケージを作成したい部分（今回は、jp.co.proc.myfavoritemovies の直下）を右クリックして図 3.3 のように階層を進めて、パッケージを作成しましょう。名前はあとで消すことがわかれればなんでも大丈夫です。

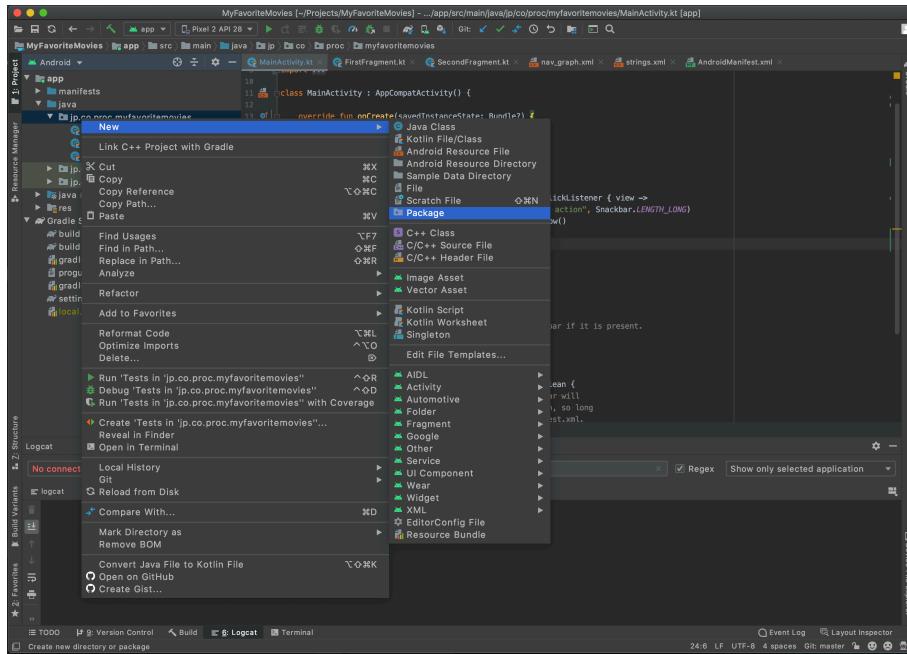


図 3.3: 新規パッケージの作成

次に Coin クラスを作成しましょう。

先ほど作成したパッケージを右クリックし「Kotlin File/Class」を作成します。Kotlin Class を選択し、名前に Coin を入力します。図 3.4 のようにクラスが作成されていれば完了です。

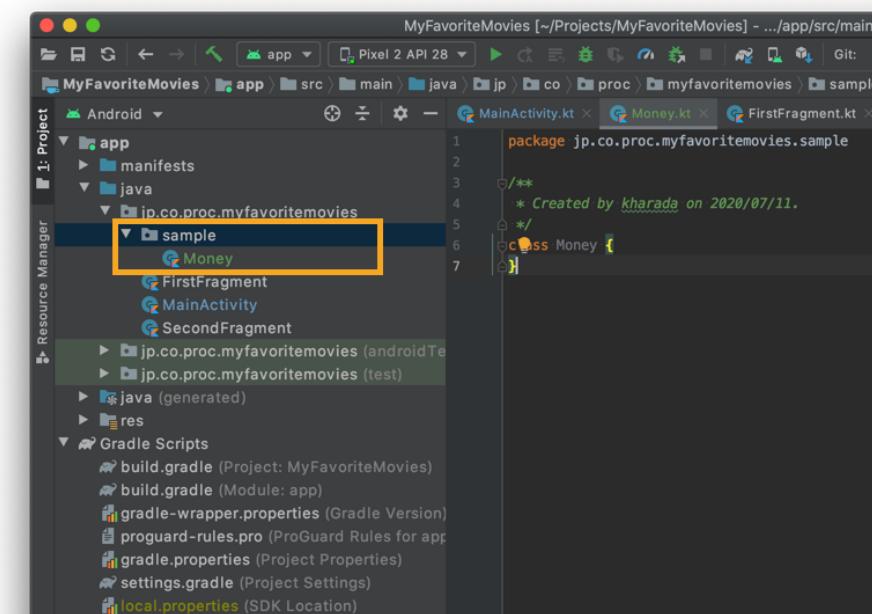


図 3.4: 新規クラスの作成

クラスが作成できたら Coin クラスを下記のように変更しましょう。

```
1 class Coin {  
2     var amount : Int = 0  
3     var material : String = ""  
4 }
```

図3.1で示したように、金額と素材を硬貨クラスに設定しています。この金額と素材のことをプロパティといいます。

では実際にこの作成したクラスを使ってみましょう。MainActivity に下記のコードを追加してみてください。

```
1 override fun onCreate(savedInstanceState: Bundle?) {  
2     super.onCreate(savedInstanceState)  
3     setContentView(R.layout.activity_main)  
4     setSupportActionBar(findViewById(R.id.toolbar))  
5     val result = 1 + 1  
6     Log.d("MyLog", "$result")  
7  
8     // ここから追加-----  
9     var tenCoin = Coin()  
10    tenCoin.amount = 10  
11    tenCoin.material = "bronze"  
12  
13    var hundredCoin = Coin()  
14    hundredCoin.amount = 100  
15    hundredCoin.material = "white copper"  
16  
17    var fivehundredCoin = Coin()  
18    hundredCoin.amount = 500  
19    hundredCoin.material = "brass"  
20    // ここまで追加-----  
21  
22    findViewById<FloatingActionButton>(R.id.fab).  
23        .setOnClickListener { view ->  
24            Snackbar.make(view, "Replace with your own action",  
25                Snackbar.LENGTH_LONG)  
26                .setAction("Action", null).show()  
27        }  
28}
```

今追加したコードは先ほど作成した Coin クラスを使って 10 円インスタンス、100 円インスタンス、500 円インスタンスを作成しています。

一行ごとの解説は下記の通りです。

```
1 var tenCoin = Coin() // 円インスタンスを作成10
2 tenCoin.amount = 10 // 円インスタンスの金額にを設定1010
3 tenCoin.material = "bronze" // 円インスタンスの素材に10を設定bronze
```

ただこの Coin クラスには多くの問題点があり、このクラスをみた時に、多くのエンジニアは恐怖を感じます。

恐怖を感じる原因は下記のような可能性が考えられるためです。

- 金額と素材の設定を忘れる人がいるかもしれない
- 処理の途中で金額や素材が変更されてしまうかもしれない
- 素材をタイプミスしてしまうかもしれない
- 素材にどんな内容が入ってくるかわからない

それでは問題を少しづつ改善していってみましょう。

まず、金額と素材に値が設定されない可能性があります。そんな硬貨は現実的にあり得るのでしょうか？現実とすればその分コードは読みにくくなってしまいます。では、金額と素材の設定を強制するよう変更しましょう。

まずは Coin クラスを下記のように変更します。

```
1 // 変更前
2 class Coin {
3     var amount : Int = 0
4     var material : String = ""
5 }
6
7 // 変更後
8 class Coin(var amount: Int, var material: String) {
9 }
```

変更すると今度は MainActivity でエラーが発生しています。MainActivity を下記のように変更しましょう。

```
1 // 変更前
2 var tenCoin = Coin()
3 tenCoin.amount = 10
4 tenCoin.material = "bronze"
5
6 var hundredCoin = Coin()
7 hundredCoin.amount = 100
8 hundredCoin.material = "white copper"
9
10 var fivehundredCoin = Coin()
11 hundredCoin.amount = 500
12 hundredCoin.material = "brass"
13
14 // 変更後
15 var tenCoin = Coin(10, "bronze")
16 var hundredCoin = Coin(100, "white copper")
17 var fivehundredCoin = Coin(500, "brass")
```

随分とすっきりしました。変更前も変更後も同じことを表現していますが、表現方法が少し違います。変更後はコンストラクタという機能を使ってインスタンス作成時に、金額と素材の設定を必須にしています。

コンストラクタ

```
class Money(var amount: Int, var material: String){}
```

図 3.5: コンストラクタの書き方

コンストラクタのおかげで、金額と素材が設定されていないかもしれないという恐怖から解放されました。

次に、処理の途中で金額や素材が変更されてしまう可能性を消しましょう。10円はいつまで経っても10円です。物価や、消費税が変更されて10円で買えるものに変更はあるかもしれません、10円という金額、素材は変更されません。それをコードで表現してみましょう。

```
1 // 変更前
2 class Coin(var amount: Int, var material: String) {
3 }
4
5 // 変更後
6 class Coin(val amount: Int, val material: String) {
7 }
```

`var` を `val` に変えるだけです。これでもう 2 度と変更することはできなくなります。不变の変数 `val` のおかげで、金額と素材が途中で変更されてしまうかもしれないという恐怖から解放されました。

最後に、素材をタイプミスしてしまう可能性と素材に何が入ってくるかわからない可能性を消しましょう。これは `Enum` クラスというものを使います。

クラスファイルを作った時と同じように、パッケージを右クリックし「Kotlin Class/File」をクリックしてください。今回は `Enum Class` を選択し、名前を `Material` に設定してください。

`Material` クラスが作成できたら、下記のように修正します。

```
1 // 変更前
2 enum class Material {
3 }
4
5 // 変更後
6 enum class Material {
7     BRONZE,
8     WHITE_COPPER,
9     BRASS
10 }
```

設定可能な素材を、一覧で記載します。慣習として `Enum` クラスのエントリーは大文字で記載します。続いて `Coin` クラスを今作成した `Material` を受け取るよう変更しましょう。

```
1 // 変更前
2 class Coin(var amount: Int, var material: String) {
3 }
4
5 // 変更後
6 class Coin(var amount: Int, var material: Material) {
7 }
```

変更すると今度は MainActivity でエラーが発生しています。MainActivity を下記のように修正します。

```
1 // 変更前
2 var tenCoin = Coin(10, "bronze")
3 var hundredCoin = Coin(100, "white copper")
4 var fivehundredCoin = Coin(500, "brass")
5
6 // 変更後
7 var tenCoin = Coin(10, Material.BRONZE)
8 var hundredCoin = Coin(100, Material.WHITE_COPPER)
9 var fivehundredCoin = Coin(500, Material.BRASS)
```

これで、素材をタイプミスしてしまう恐怖と素材に何が入ってくるかわからない恐怖から解放されました。

Enum クラスを引数として受け取っているため、タイプミスする可能性はありません。厳密にはタイプミスした場合、エラーになるのでアプリを実行できません。また、Enum クラスに設定可能な値は全てまとまっているため、何が設定されるかひとめでわかります。

ここまで来れば、Coin クラスとしては及第点です。

3.5 コレクションクラス

Basic Knowledge

Kotlinにはコレクションクラスというクラスをまとめて扱うクラスが大きく分けて3つ用意されています。それらの使い方を説明します。

3.5.1 List

順序つきのリストです。下記のように記載します。

```
1 var coins = mutableListOf(tenCoin, hundredCoin, fiveHundredCoin)
```

図3.6のように簡に順番が付けられて保存されているようなイメージを持つと理解しやすいです。順番があるので、仮に同じものがあったとしても別物として扱うことができます。



図3.6: リストのイメージ

3.5.2 Set

順序なしのリストです。下記のように記載します。

```
1 var coins = mutableSetOf(tenCoin, hundredCoin, fiveHundredCoin)
```

図3.7のように袋の中に保存されているようなイメージを持つと理解しやすいです。袋に適当に詰め込むので同じものがあると、どれがどれだかわからなくなってしまいます。そのため、Setは重複を禁止しています。

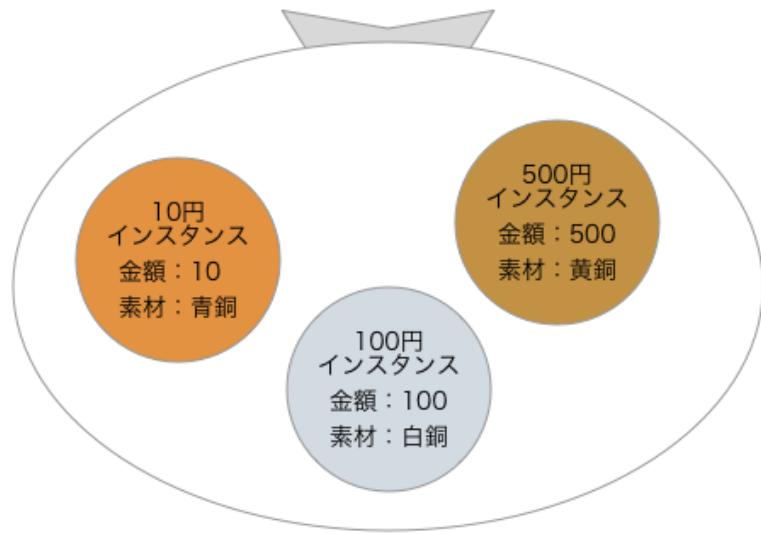


図 3.7: セットのイメージ

3.5.3 Map

キーバリューのリストです。リストと同じようなイメージですが、それに名前が付けられるようになったものです。

```
1 var coins = mutableMapOf("円10" to tenCoin, "百円" to hundredCoin, "500" to fiveHundredCoin)
```



図 3.8: マップのイメージ

3.6 制御構文

Basic Knowledge

Kotlin の制御構文について学びましょう。制御構文はどのプログラミング言語にも存在するものですが、Kotlin 特有の構文もあります。実際、コードを書く際には制御構文を使って条件分岐や繰り返し処理を実現します。必ず使用するものなので確実に理解しましょう。

3.6.1 条件分岐

Kotlin の条件分岐は if, when の 2 つの書き方があります。

まずは if から理解しましょう。if は下記のように書きます。

MainActivity を開いて、下記のように硬貨の種類をログに出力する関数を書きましょう。

```
1 fun coinChecker(coin : Coin) { // 関数定義
2     if (coin.material == Material.BRONZE) { // もし素材が青銅だったら
3         Log.d("MyLog", "円です10") // 円ですとログに出力する10
4     }
5     if (coin.material == Material.WHITE_COPPER) {
6         Log.d("MyLog", "円です100")
7     }
8     if (coin.material == Material.BRASS) {
9         Log.d("MyLog", "円です500")
10    }
11 }
```

このように条件を設定して処理を分岐することができます。条件部分に記載している==を演算子と呼び、特別な意味を持ちます。

条件分岐で使用可能な演算子は表 3.1 の通りです。

表 3.1: 使用可能な演算子

| 演算子 | 意味 | 例 |
|------|---------------------|-----------------------------------|
| == | 左右が同じ値かどうか | coin.material == Material.BRONZE |
| != | 左右が異なる値かどうか | coin.material != Material.BRONZE |
| ==== | 左右が参照するものが同じかどうか | coin.material === Material.BRONZE |
| !== | 左右が参照するものが異なるかどうか | coin.material !== Material.BRONZE |
| > | 左の方が大きいかどうか | coin.amount > 5 |
| < | 右の方が大きいかどうか | coin.amount < 5 |
| >= | 左の方が大きいか、もしくは同じかどうか | coin.amount >= 5 |
| <= | 左の方が小さいか、もしくは同じかどうか | coin.amount <= 5 |

ただ、先ほど記載した coinChecker 関数にはいくつか問題があります。

coinChecker 関数の問題点

- 無駄な判定をしている
- 実装が煩雑

これから上記の問題点を解決していきます。

まず、無駄な判定をしている点です。今の実装のままでは、例えば10円インスタンスが引数に渡された場合、10円ですと出力すればいいだけですが、必ず100円かどうか、500円かどうかを確認しています。10円の場合は100円と500円である可能性はないので無駄な処理になっています。いくつか、解法はあります、今回は ElseIf 構文を使用します。

```
1 // 変更前
2 fun coinChecker(coin :Coin) { // 関数定義
3     if (coin.material == Material.BRONZE) { // もし素材が青銅だったら
4         Log.d("MyLog", "円です10") // 円ですとログに出力する10
5     }
6     if (coin.material == Material.WHITE_COPPER) {
7         Log.d("MyLog", "円です100")
8     }
9     if (coin.material == Material.BRASS) {
10        Log.d("MyLog", "円です500")
11    }
12 }
13 // 変更後
14 fun coinChecker(coin :Coin) { // 関数定義
15     if (coin.material == Material.BRONZE) { // もし素材が青銅だったら
16         Log.d("MyLog", "円です10") // 円ですとログに出力する10
17     } else if (coin.material == Material.WHITE_COPPER) {
18         Log.d("MyLog", "円です100")
19     } else if (coin.material == Material.BRASS) {
20         Log.d("MyLog", "円です500")
21     }
22 }
```

else if を使用すると1つの条件にマッチすると後の条件は確認しません。すなわち10円インスタンスの場合は最初の確認で処理が終了します。

ただ Kotlin の場合はよりシンプルに条件分岐を書くことができます。この場合は When という構文を使います。

```
1 // 変更前
2 fun coinChecker(coin :Coin) { // 関数定義
3     if (coin.material == Material.BRONZE) { // もし素材が青銅だったら
```

```

4         Log.d("MyLog", "円です10") // 円ですとログに出力する10
5     } else if (coin.material == Material.WHITE_COPPER) {
6         Log.d("MyLog", "円です100")
7     } else if (coin.material == Material.BRASS) {
8         Log.d("MyLog", "円です500")
9     }
10    }
11 // 変更後
12 fun coinChecker(coin : Coin) {
13     when (coin.material) {
14         Material.BRONZE -> Log.d("MyLog", "円です10")
15         Material.WHITE_COPPER -> Log.d("MyLog", "円です100")
16         Material.BRASS -> Log.d("MyLog", "円です500")
17     }
18 }
```

When は 1 つの値で複数の条件分岐を書く時に特に効果的です。

課題：関数を実行して動作を確認しましょう。

3.6.2 繰り返し

繰り返し処理は例えばリストの全てに対して何か処理を行いたい時や、複数回同じ処理を行いたい時に記載します。

繰り返し処理は、大きく分けて 3 つの方法 (for, while, do-while) がありますが、実際使用する可能性があるものは for だけなので for に絞って説明します。

MainActivity でリストを作つて繰り返し処理をしてみましょう。

```

1 var tenCoin = Coin(10, Material.BRONZE)
2 var hundredCoin = Coin(100, Material.WHITE_COPPER)
3 var fiveHundredCoin = Coin(500, Material.BRASS)
4
5 var coins = mutableListOf(tenCoin, hundredCoin, fiveHundredCoin)
6
7 for (coin in coins) {
8     Log.d("MyLog", "${coin.amount}")
9 }
```

for は coins というリストの要素を一つずつ coin という変数に設定して繰り返し処理を行っています。

課題：処理を実行して動作を確認しましょう。

3.7 例外処理



Kotlin の例外処理について学びましょう。例外処理はどのプログラミング言語にも存在するもので、Android 開発の場合は特に使用するケースが多いので使えるようになっておきましょう。

例外処理には、try, catch, finally という構文があります。

それぞれ下記のような意味があります。

- try - 例外が起きててもいいと待ち構える範囲。
- catch - 捕まえる範囲。何かが起きた時、どうするかを記載する。
- finally - 最後にやること。

では、実際に MainActivity に処理を書いてみましょう。

```
1 var tenCoin = Coin(10, Material.BRONZE)
2 var hundredCoin = Coin(100, Material.WHITE_COPPER)
3 var fiveHundredCoin = Coin(500, Material.BRASS)
4
5 var coins = mutableListOf(tenCoin, hundredCoin, fiveHundredCoin)
6
7 try {
8     for (coin in coins) {
9         Log.d("MyLog", "${coin.amount}")
10        throw Exception("サンプルエラーメッセージ") // 句を用いて例外を投げるthrow
11    }
12 } catch (e:Exception) {
13     Log.d("MyLog", e.message)
14 } finally {
15     Log.d("MyLog", "最終処理")
16 }
```

処理終了後のログの順番を確認してみてください。1度目のループ処理が呼び出された後、例外処理に入っていることが確認できます。

課題：throw Exception("サンプルエラーメッセージ") の行を削除してどのような動作になるか確認しましょう。

3.8 クラス設計（抽象的な概念をクラスとして定義する）



先ほど作成した Coin クラスを使って、金額計算できるお金クラスを作成してみましょう。今のクラスのままでも金額計算はできますが、下記のようになってしまいます。

```

1  var tenCoin = Coin(10, Material.BRONZE)
2  var hundredCoin = Coin(100, Material.WHITE_COPPER)
3  var fiveHundredCoin = Coin(500, Material.BRASS)
4
5  var result = tenCoin.amount + hundredCoin.amount + ←
    fiveHundredCoin.amount

```

これはあまり直感的ではありません。なぜなら 10 円という硬貨を使う時にいちいち金額という概念を気にしないためです。例えば、10 円 + 100 円 = 110 円をシンプルに表現できるのが正しいクラスの形です。お金という新しいクラスを作つて金額計算の機能を実現しましょう。

- ステップ 1 : 空の Money クラスを作成する
- ステップ 2 : Money クラスをどのように使いたいか考える
- ステップ 3 : Money クラスに必要なプロパティ・関数を考える
- ステップ 4 : 実装する

ステップ 1 : 空の Money クラスを作成する

パッケージを右クリックして、「Kotlin Class/File」をクリックしてください。ファイル名に Money と入力して、Class を選択しましょう。

ステップ 2 : Money クラスをどのように使いたいか考える

amount への参照をさけて、そのまま演算できるようにしたいので、下記のような使用方法が理想です。

```

1  var result = tenYen + hundredYen + fiveYen

```

ステップ 3 : Money クラスに必要なプロパティを考える

ステップ 2 の理想に向かって、何が Money クラスに必要でしょうか？

- 合計金額を保持する
- + の演算子を金額計算ように変更する

実際に実装したクラスが下記になります。まずはこの通りクラスを作成してみましょう。理解できない部分は調べてみてください。

```

1 class Money {
2
3     enum class Operator {
4         PLUS,
5     }
6

```

```

7   var amount = 0 // 合計金額を保持
8   var coins = mutableListOf<Coin>() // 使用した硬貨を保持
9
10  constructor(coins: MutableList<Coin>, operator: Operator) {
11      // 演算子メソッドからのみ呼び出されるコンストラクタ
12      for (coin in coins) {
13          when (operator) {
14              Operator.PLUS -> amount += coin.amount
15          }
16      }
17  }
18  this.coins = coins
19 }
20
21 constructor(coin: Coin) { // 外部から呼び出されるコンストラクタ
22     amount = coin.amount
23     coins = mutableListOf(coin)
24 }
25
26 operator fun plus(other: Money): Money { // 演算子のオーバーロード処理
27     this.coins.addAll(other.coins)
28     return Money(this.coins, Operator.PLUS)
29 }
30 }
```

課題：上記のクラスに引き算の機能を追加してみましょう。

3.9 まとめ

- Kotlin の構文を理解する簡単な変数や関数、クラスの使い方を理解しました。正しく使えるよう、コードの意味を再確認しておきましょう。
- 簡単なクラス設計ができるようになる Coin クラス、Money クラスを修正することで大まかなクラス設計を理解しました。

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

第II部

Android應用

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     def __init__(self, item):
7         """Initialize class for the given item."""
8
9         self.__queue_item = item
10
11        item.mount("http://", requests.HTTPAdapter)
12        item.mount("https://", requests.HTTPAdapter)
13
14    @staticmethod
15    def hello():
16        """Print an example message."""
17
18        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 tellus¹.

¹testcitation.

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 | Opening Meeting 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 | Documentation Documentation about scoping, planning, implementing, monitoring and reviewing. |
| 12:00 | Information security management system (ISMS) This meeting will be held with security officers as well as the process manager. |
| 21:00 | Closing Remarks (Audit Follow-Up) 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. |