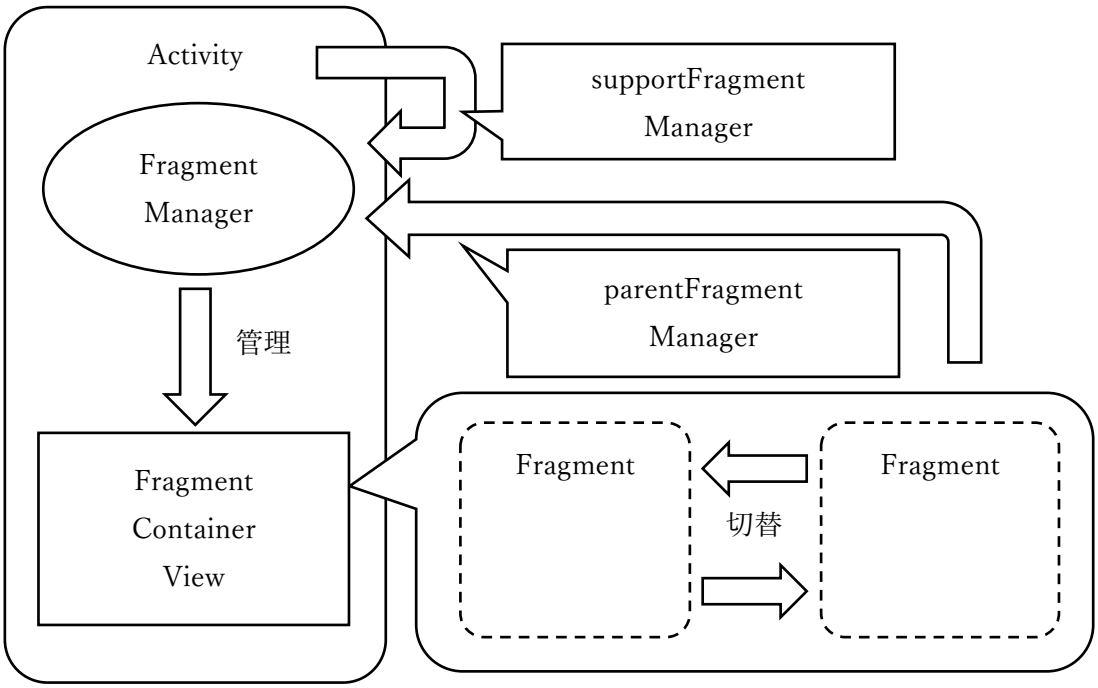
Fragment 応用

Fragment 間の遷移

前回、FragmentManager を使ったフラグメントの切り替え処理を学習しました。
もう少し FragmentManager について学習を進めていきます。

FragmentManager は、アクティビティまたはフラグメントからアクセスできます。
アクティビティからは、getSupportFragmentManager メソッドを通じて、FragmentManager にアクセスできます。フラグメントからホスト(アクティビティ)の FragmentManager にアクセスするには getParentFragmentManager を使用します。
なお、フラグメントは 1 以上の子フラグメントをホストすることが出来て、子フラグメントの FragmentManager にアクセスするには、自フラグメント内で getChildFragmentManager を使用します。

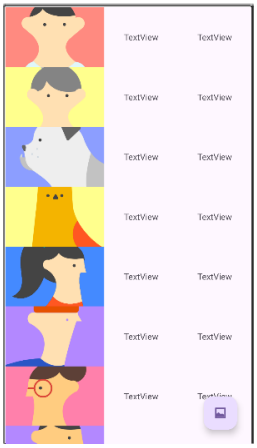
・自フラグメント切替時のフラグメントマネージャーについて

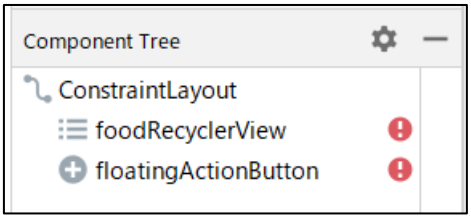


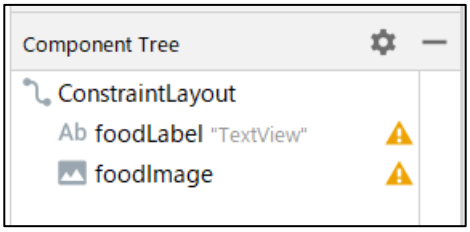
ハンズオン　ハンズオンプロジェクトをフラグメント化する

1. HandsOnProject のプロジェクトを開く
2. フラグメントを 2 つ新規作成する

Fragment Name	FoodListFragment	FoodDetailFragment
Source Language	Kotlin	Kotlin







・FoodListFragment

コンポーネント	属性	設定値
RecyclerView	Id	foodRecyclerView
	listItem	@layout/list_row
floatingActionButton	Id	imageFab
	src	@android:drawable/ic_menu_gallery
	contentDescription	detail

・FoodDetailFragment

コンポーネント	属性	設定値
TextView	Id	foodLabel
	textAppearance	Large
ImageView	Id	foodImage
	srcCompat	@drawable/apple
	Layout_width	0dp
	Layout_height	0dp

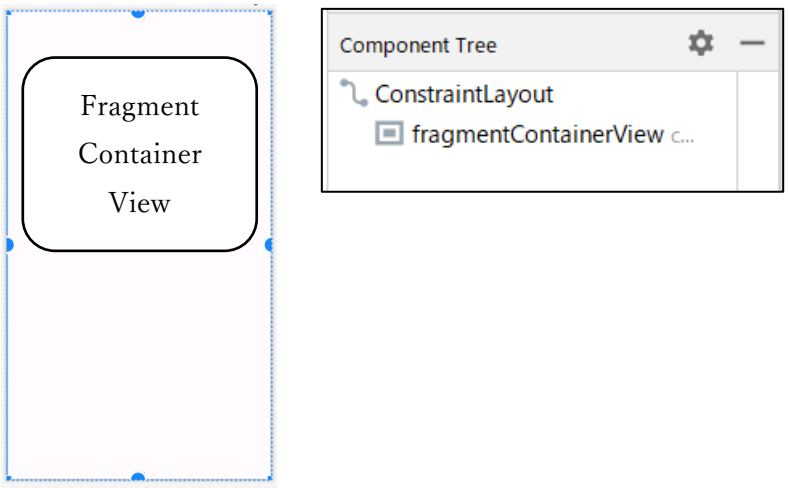
RecycerView の処理を MainActivity から FoodListFragment に移行します。

3. Foods リストを MainActivity から FoodListFragment に移動させる
4. MainActivity の RecycerView 関連のコードを削除する

5. FoodListFragment の onCreateView に RecycerView の設定を行う

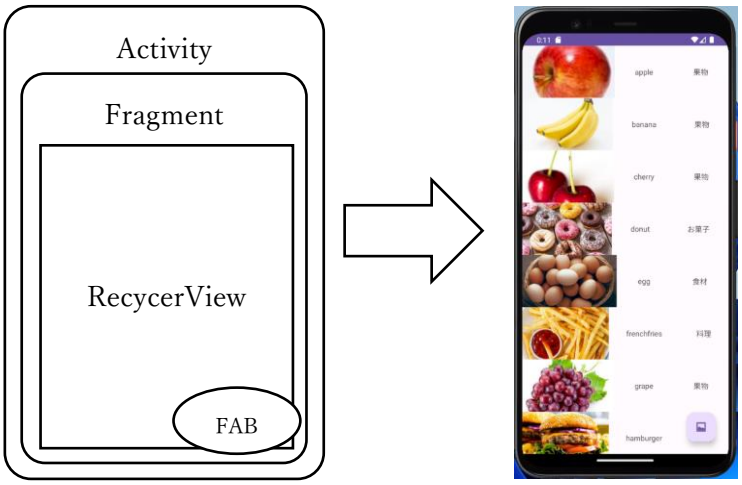
```
val view = inflater.inflate(R.layout.fragment_food_list, container, false)
val foodRv : RecyclerView = view.findViewById(R.id.foodRecyclerView)
foodRv.layoutManager = LinearLayoutManager(context)
foodRv.adapter = FoodAdapter(foods)
return view
```

6. MainActivity のデザインを変更する



7. MainActivity の FragmentManager でフラグメントの切替を行う

```
val foodListFragment = FoodListFragment.newInstance("","")
val transaction = supportFragmentManager.beginTransaction()
transaction.replace(R.id.fragmentContainerView, foodListFragment)
transaction.addToBackStack(null)
transaction.commit()
```



ハンズオン　自フラグメントの切り替えを行う

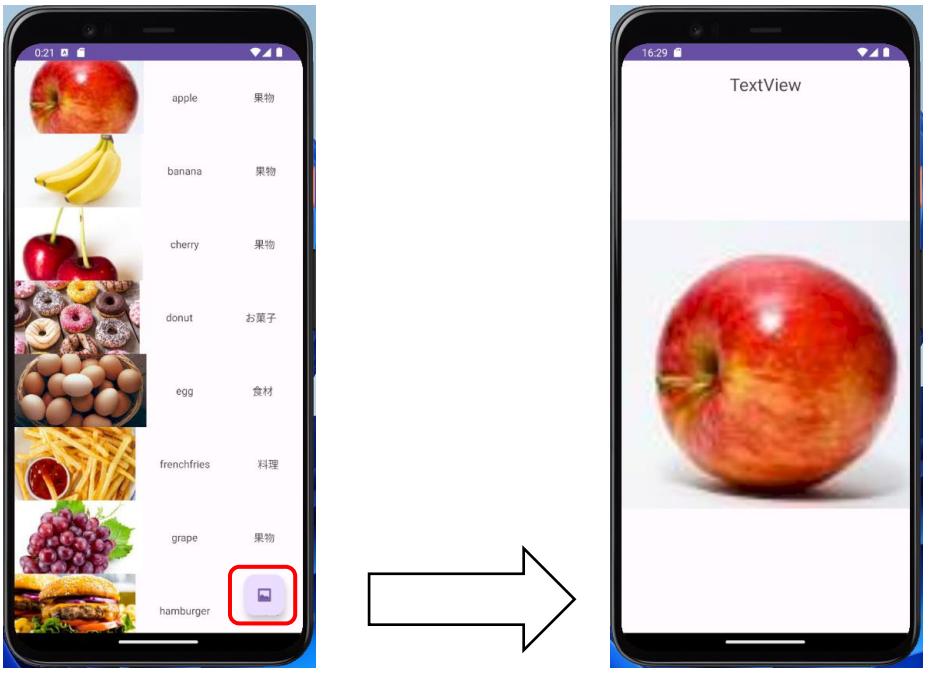
---

1. FoodListFragment の onCreateView メソッドで FloatingActionButton【FAB】のオブジェクト変数を宣言する
2. FAB にクリックイベントリスナーを準備する

```
val imageFab : FloatingActionButton = view.findViewById(R.id.imageFab)
imageFab.setOnClickListener {
}
```

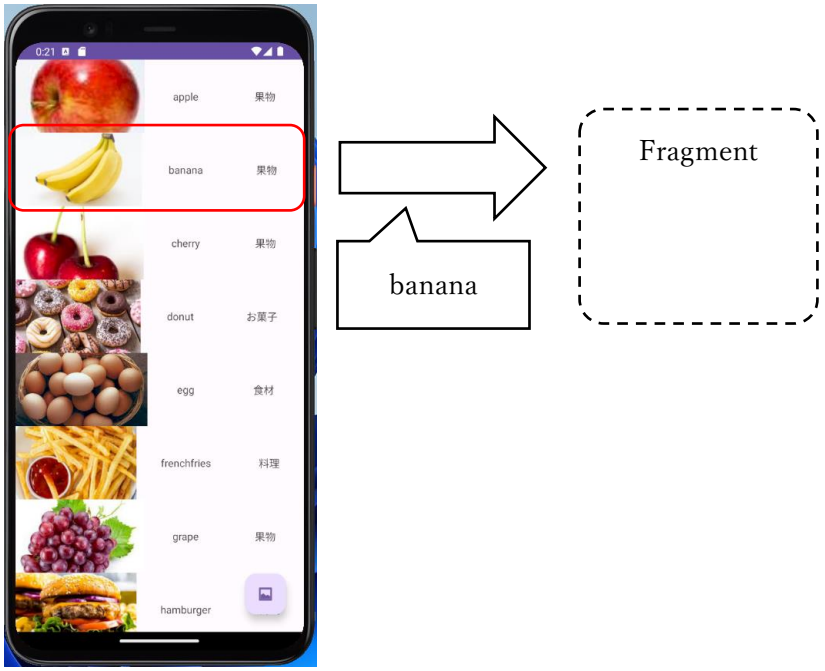
3. クリックイベント内で親 FragmentManager を呼び出し自フラグメントを切替える

```
val foodDetailFragment = FoodDetailFragment.newInstance("", "")
val transaction = parentFragmentManager.beginTransaction()
transaction.replace(R.id.fragmentContainerView, foodDetailFragment)
transaction.addToBackStack(null)
transaction.commit()
```



Fragment のパラメータ

現在、FAB ボタンから DetailFramgent に切り替えを行っています。しかし RecycerView のようにリストの情報ごとにフラグメントの中身を変えたい場合には、フラグメント切替時にパラメータを受け渡す必要があります。



パラメータの受け渡しには、テンプレートで自動生成される newInstance メソッドに適切な引数を設定することで呼び出し元の情報をフラグメントに送れます。

ハンズオン　フラグメントにパラメータを受け渡す

---

1. FoodDetailFragment 生成時の newInstance の第 1 引数を追加する

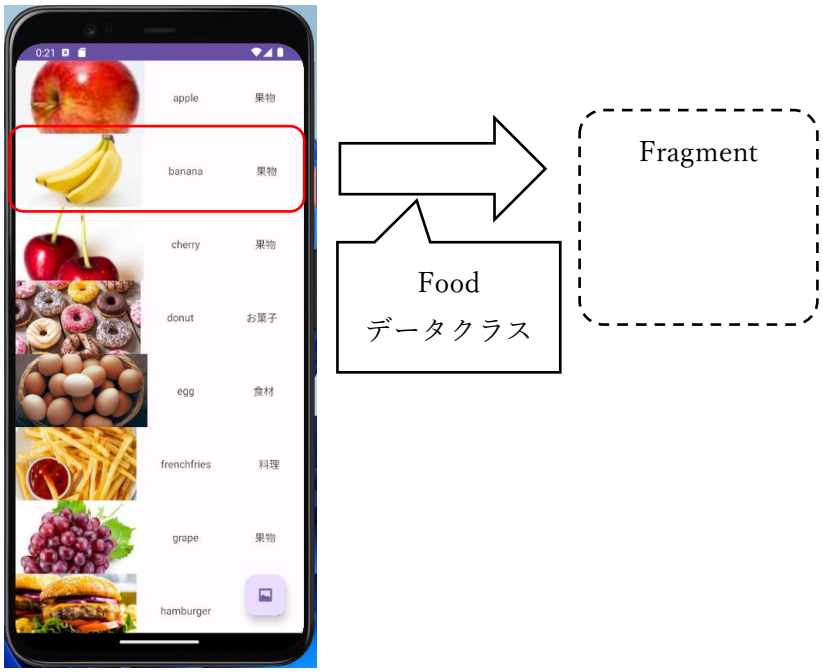
```
val foodDetailFragment = FoodDetailFragment.newInstance("りんご", "")
```

2. FoodDetailFragment の onCreateView で受取ったパラメータを TextView に表示させる

```
val view = inflater.inflate(R.layout.fragment_food_detail, container, false)
val foodLabel:TextView = view.findViewById(R.id.foodLabel)
foodLabel.text = param1
return view
```

Parcelize

アクティビティやフラグメント間の情報をやり取りするときに Bundle クラスを利用して値のやり取りを行います。しかし Bundle クラスには Int や String など基本的なデータ型しか利用できません。「画像 ID」、「商品名」、「カテゴリ」を個別にパラメータにセットしても良いですが、Paracelize 生成ツールを利用することで Food データクラス自体をパラメータにセットすることが可能になります。



ハンズオン　Paracelize 機能を有効化する

---

1. kotlin-parcelize プラグインの追加をモジュールレベル Gradle ファイルの plugins に追記する

```
id("kotlin-parcelize")
```

2. Food データクラスに@Parcelize アノテーションを追記して、Parcelable 対応する

```
@Parcelize
data class Food(
    val imgResID: Int,
    val name: String,
    val category: String
): Parcelable
```

3. FoodDetailFragment の引数を Food データクラスに変更する

```
fun newInstance(param1: Food)
```

4. パラメータセットを putParcelable に変更する

```
arguments = Bundle().apply {
    putParcelable(ARG_PARAM1, param1)
}
```

5. クラス変数のデータ型を Food 型に変更する

```
private var param1: Food? = null
```

6. onCreate メソッド内のパラメータの受け取りを Parcelable に変更する

```
arguments?.let {
    param1 = BundleCompat.getParcelable(it, ARG_PARAM1, Food::class.java)
}
```

7. DetailFramgent のコンポーネントに Food データクラスに対応させる

```
param1?.let{
    foodImage.setImageResource(it.imgResID)
    foodLabel.text = it.name
}
```

Adapter の役割について

RecycerView の学習で Adapter の実装を行いましたが、onBindViewHolder メソッドの中で直接、呼び出し元のクリックイベントを実装しています。この実装方法だと汎用性がない独自 Adapter になりますので、クリックイベント自体は呼び出し元で処理を記述して Adapter はその処理を受け取るようにする必要があります。

ハンズオン　クリックイベントをアダプターから切り離す

---

1. イベント自体を受け取れるように、Adapter に引数を追加する

```
class FoodAdapter(private val foods: List<Food>, private val onFoodClick:(Food) -> Unit) :
RecyclerView.Adapter<FoodAdapter.ViewHolder>() {
```

2. setOnClickListner を引数で渡されるクリックイベントを実装する

```
holder.foodImageView.setOnClickListener {
    onFoodClick(foods[position])
}
```

3. Adapter 呼び出し元で、クリックイベントの実装を行う

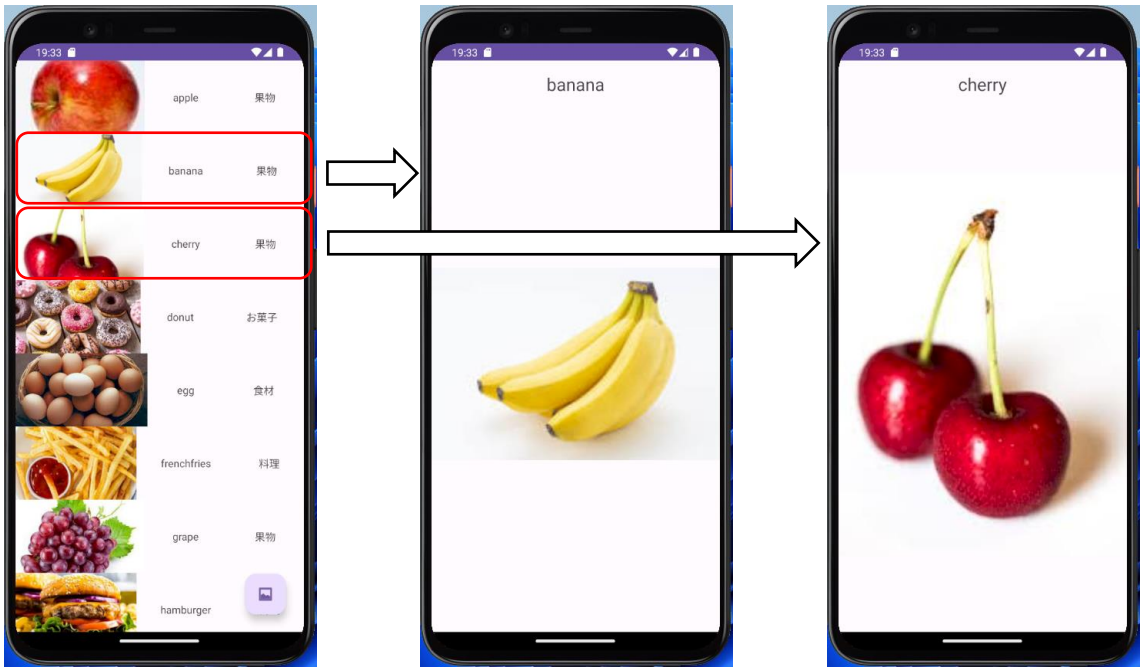
```
// foodRv.adapter = FoodAdapter(foods)
foodRv.adapter = FoodAdapter(foods, onFoodClick = { food ->

})
```

4. クリックイベントでフラグメント切替処理を記述する

```
val foodDetailFragment = FoodDetailFragment.newInstance(food)
val transaction = parentFragmentManager.beginTransaction()
transaction.replace(R.id.fragmentContainerView, foodDetailFragment)
transaction.addToBackStack(null)
transaction.commit()
```

5. Fab ボタン処理は不要になったので処理をコメントアウトする

```
// Fab ボタンは不要のためコメント化する
// val imageFab : FloatingActionButton = view.findViewById(R.id.imageFab)
// imageFab.setOnClickListener{
//     val foodDetailFragment = FoodDetailFragment.newInstance("りんご","")
//     val transaction = parentFragmentManager.beginTransaction()
//     transaction.replace(R.id.fragmentContainerView, foodDetailFragment)
//     transaction.addToBackStack(null)
//     transaction.commit()
// }
```



★Topic★

アニメーションについて

https://developer.android.com/guide/fragments/animate#set-animations

https://developer.android.com/guide/topics/resources/animation-resource?hl=ja#Tween