・kotlinの基礎知識\_基本構文 ・Android studioのデバッグ方法

・Activityのライフサイクル

本日の内容

2年生からは、IT基礎演習IIで学んだ内容を活かしながら、次のステップへ進みます。

言語をKotlinに変更し、内容も応用編に近くなります。

まずはKotlin特有の記述方法に慣れることから始めましょう。

■Kotlin基礎知識

・Kotlinとは

IntelliJ IDEAで有名なJetBrainsが開発した**オブジェクト指向プログラミング言語**

2017年に**Android公式開発言語**に追加

・特徴

|  |
| --- |
| **KotlinとはJavaとの相互運用性が高い** |
| KotlinからJavaを呼び出すことも、JavaからKotlinを呼び出すことも可能 |
| **Null安全（Null Safety）を採用** |
| Kotlinの言語仕様で、Null参照による実行時エラーを未然に防止する仕組み |
| **変数宣言にvalとvarの2つがある** |
| valは定数のような用途、varは変数のような用途で使い分ける |
| **行末の;（セミコロン）を省略できる** |
| Javaでコーディングする場合には行末に「;」を付ける必要がありましたが、  Kotlinなら行末の「;」は不要です。 |
| **型宣言は後ろに置く** |
| 構文　(var 変数名: 型 = リテラル)  記述例 var num: Int = 1 |
| **ifが式である** |
| プログラミングにおける「if」はif文として利用されていますが、  Kotlinではifが式になっていますので、三項演算子と同じ動きになります。 |
| **when式** |
| whenではswitchと同様の処理を記述することができますが、  Javaのswitchの条件分岐よりも  Kotlinのwhenの方がシンプルでわかりやすいコードを記述することができます。 |
| **ラムダ式** |
| Kotlinの標準ライブラリではラムダ式を多用していることから、  ラムダ式によってコレクションの操作がしやすくなっています。 |

等々…他にも細かい特徴は沢山ありますが、紹介はここまでとしておきます。

■Kotlin言語に触れてみる

project作成

まずはAndroid Studioで動かしながら慣れていきましょう

グラフィカル ユーザー インターフェイス, アプリケーション

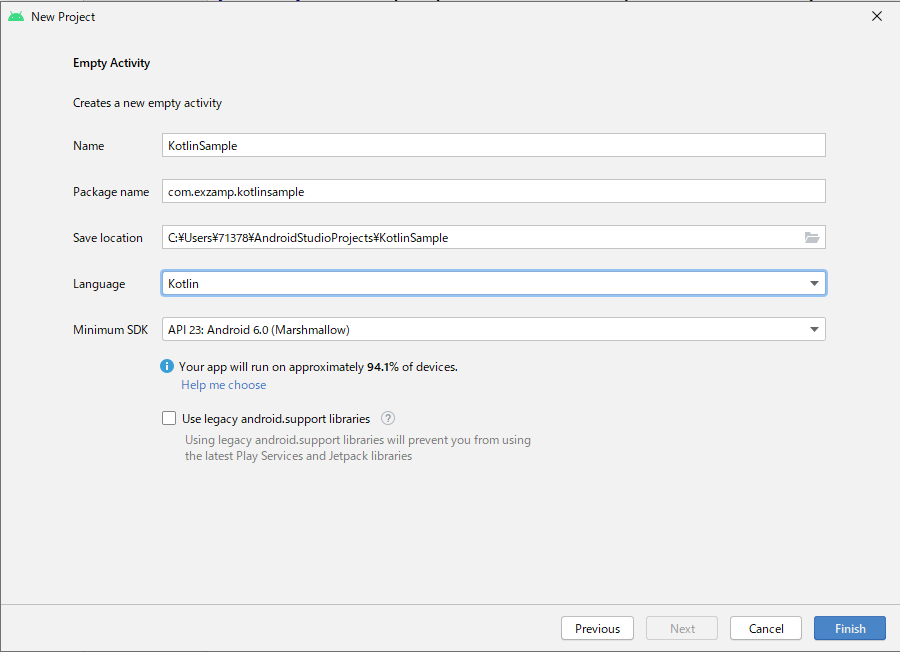
自動的に生成された説明

1. 新規プロジェクトを作成する際、EmptyViewsActivityを選択

②Nameを今回は[KotlinSample]と記入

③Languageを[**Kotlin**]に変更

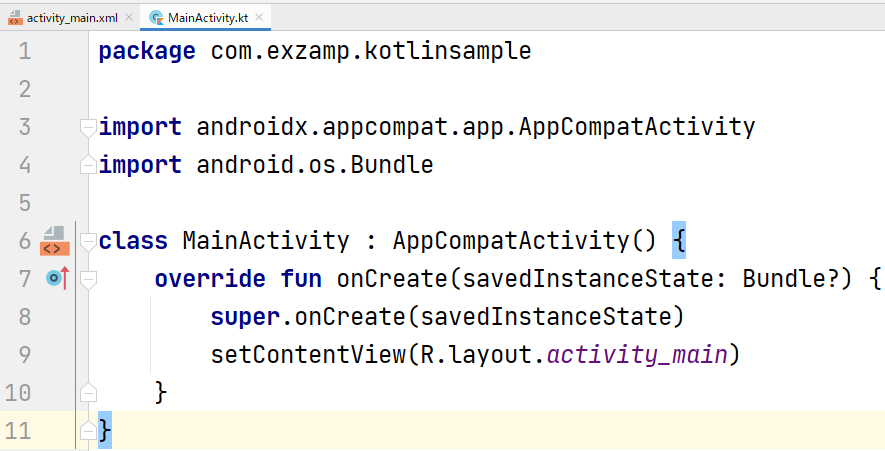
④Finishをクリック



ビルドに少し時間が掛かります。

生成されたファイル名が.ktとなっていますね！

これがKotlinファイルの拡張子となります。



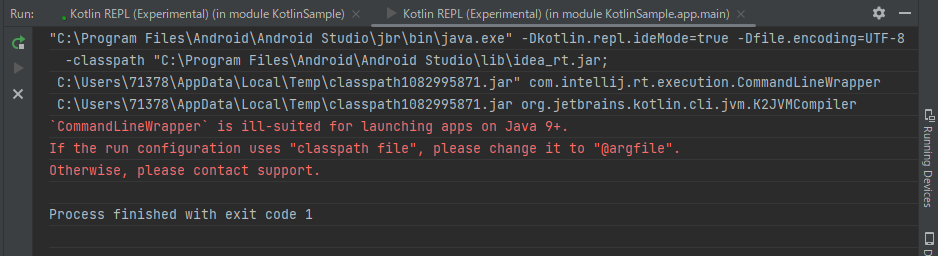
事前準備OKです

ここから、言語の構文を学んでいきますが、毎回Androidに書き込んで動作確認を行うと

処理が重いため、KotlinREPLという機能でKotlin言語を学びます。

■KotlinREPLの起動

上部タブの「Tools」 > 「Kotlin」 > 「Kotlin REPL」をクリック



初回はエラーが出るので、このwindowを選択している状態で再度

上部タブの「Tools」 > 「Kotlin」 > 「Kotlin REPL」をクリック

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

この作成したプロジェクトの一番上のKotlinSampleを選択

テキスト

自動的に生成された説明

上記のようになっていればOKです。

■Kotlinの基本的な構文

ではここからは、実際にプログラミングしながら、構文と動きを確認していきましょう。

ステップ 1: 数値演算子を調べる

他の言語と同様に、Kotlinは、プラス、マイナス、時間、除算を使用します。

Kotlinは+-\*/Int、Long、Double、Floatなどのさまざまな数値タイプもサポートしています。

REPL に次の式を入力します。結果を確認するには、それぞれの後にControl+Enter

(Macの場合：Command+Enter)を押します。

1.1 試しに、５つの式を順番に実行してみましょう

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

演算の結果はオペランドの型を保持するため、1/2 = 0 ですが、1.0/2.0 = 0.5 であることに注意してください。

1.2 整数と10進数の異なる組み合わせでいくつかの式を試してください。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

1.3 数値に対していくつかのメソッドを呼び出します。

Kotlinは数値をプリミティブとして保持しますが、数値のメソッドをオブジェクトであるかのように呼び出すことができます。

グラフィカル ユーザー インターフェイス

中程度の精度で自動的に生成された説明

ステップ 2: 型の使用を練習する

Kotlinは暗黙的に数値型を変換しないので、long変数に直接short値を代入したり、.NET変数に直接aを代入したりすることはできません。 これは、暗黙的な数値変換がプログラムにおけるエラーの一般的な原因であるためです。異なる型の値を代入するには、常にキャストする必要があります。

2.1 可能なキャストのいくつかを見るために、REPL.Intで型の変数を定義する。



2.2 新しい変数を作成し、上に示した変数名の後に.toを入力する。



補完可能なリストが表示されるのでtoBye()を選択

モニター画面に映る文字のスクリーンショット

自動的に生成された説明

変数を出力してみましょう





2.4 異なる型の変数に値を割り当てる。

テキスト

自動的に生成された説明

2.5 エラーを返した代入については、代わりにキャストしてみてください。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

2.6 長い数値定数を読みやすくするために、Kotlinでは数字の中にアンダースコアを入れることができます。異なる数値定数を入力してみてください。

テキスト

自動的に生成された説明

ステップ3：変数タイプの値を学ぶ

Kotlinは、変更可能な変数と変更不可能な変数の2種類をサポートしています。

valを使えば、一度だけ値を代入することができます。もう一度値を代入しようとするとエラーになる。

varを使えば、値を代入し、後でプログラムの中で値を変更することが出来ます。

3.1 val varを使用して変数を定義し、新しい値を代入する。

図形

自動的に生成された説明



fishはvar定義されているので、値を代入し、新しい値を代入することができます。

aquariumはvalで定義されているので、新しい値を代入しようとするとエラーになります。

変数に格納する型は、コンパイラが文脈から割り出すことができる場合に推測されます。

必要であれば、コロン記法を使って変数の型を明示的に指定することもできます。

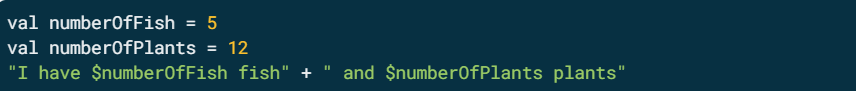
3.2 いくつかの変数を定義し、型を明示的に指定する。



一度、コンパイラによって型が割り当てられると、その型を変更することは出来ません。

ステップ4：Stringsについて学ぶ

Kotlinの文字列は、他のプログラミング言語の文字列とほとんど同じように、文字列と単一文字に対して機能し、演算子を使って文字列を連結することができる。変数名は値を表すテキストに置き換えられます。これは変数補間と呼ばれます。





他の言語と同様、値は式の結果であることが出来ます。中括弧を使って式を定義します。





ステップ5 タスク 条件とブール値の比較

他の言語と同様に、

Kotlinにもブーリアンと、less than、equal to、greater thanなどのブーリアン演算子があります。

テキスト

中程度の精度で自動的に生成された説明



5.1 ステートメントで範囲を試す

Kotlinでは、テストする条件にも範囲を使うことができます。

テキスト

中程度の精度で自動的に生成された説明\



より複雑な条件の場合は、論理と論理または論理を使用します。

他の言語と同様に、 if && || else ifを使用することで複数のケースを持つことができます。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

5.2 ステートメントを使ってみよう。

他の言語のステートKotlinでは、if elseをより簡潔に書く方法があります。それはwhenステートメントを使用する方法です。このwhenは、他の言語のswitchのようなものです。また、whenの条件では範囲も使用できます。従って、他の言語のif-else if-elseのような条件分岐も、Kotlinではwhenを使って表現することが可能です。

グラフィカル ユーザー インターフェイス, テキスト

自動的に生成された説明



ステップ6 : null 値の許容について学習する

null変数と非null変数について学ぶ。NULLを含むプログラミング・エラーは数え切れないほどのバグの原因となってきました。Kotlinでは、nullableでない変数を導入することで、バグを減らそうとしています。

6.1 null 値の許容について

デフォルトでは、変数をnullにすることはできません。

Int型のnullを宣言して割り当てます。





型の後にクエスチョンマーク演算子 を使って、変数がnullになる可能性があることを示す。



6.2 ？と演算子について学ぶ

？演算子を使ってテストできるので、多くのnullチェック文を書く手間が省けます。

従来の書き方

テキスト

自動的に生成された説明

？演算子を使ったKotlinの書き方を見てみよう。



この演算子を使ってnullのテストを連鎖させることもできる。



これは、nullでない場合は、デクリメントして使用します。

それ以外の場合は、 値 (0) を使用します。

not-nullアサーション演算子 (!!)

値が の場合は例外をスローします。



ステップ7 : 配列、リスト、ループ

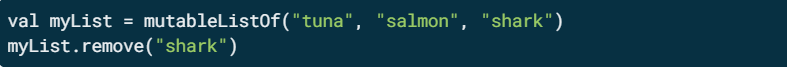
7.1 リストを作成する

リストはKotlin の基本的なタイプであり、他の言語のリストに似ています。





7.2 mutableListOfを使用して変更できるリストを宣言します。アイテムを削除します。

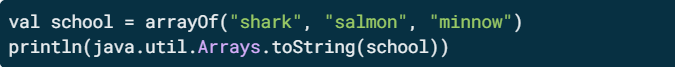




7.3 配列を作成する

他の言語と同様に、Kotlinには配列があります。

変更可能なバージョンと不変バージョンがある Kotlin のリストとは異なり、Array配列を作成すると、サイズは固定されます。新しい配列にコピーする場合を除き、要素を追加または削除することはできません。





arrayOfで宣言された配列には要素に関連付けられた型がないため、型を混在させることができ、便利です。異なる型の配列を宣言します。



すべての要素に対して 1 つの型で配列を宣言することもできます。

intArrayOf()を使用して整数の配列を宣言します。

他の型の配列に対応するビルダーまたはインスタンス化関数があります。



7.4 2つの配列を+演算子で結合します。

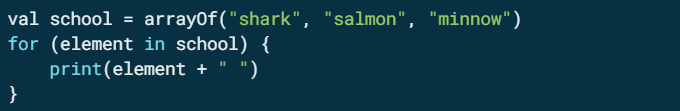
テキスト

自動的に生成された説明



ステップ8 ループを作成する

8.1 配列を作成します。forループを使用して配列を反復処理し、要素を出力します。





8.2 Kotlin では、要素とインデックスを同時にループ処理できます。

テキスト

自動的に生成された説明

図形

低い精度で自動的に生成された説明

8.3 まざまなステップサイズと範囲を試してください。数字または文字の範囲をアルファベット順に指定できます。また、他の言語と同様に、1ずつ前進する必要はありません。downToを使用して後方にステップできます。

テキスト

自動的に生成された説明

8.4 いくつかのループを試してみましょう。

他の言語と同様に、Kotlin にはwhileループ、do...whileループ、++および--演算子があります。

Kotlinにはrepeatループもあります。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

■Activityのライフサイクル

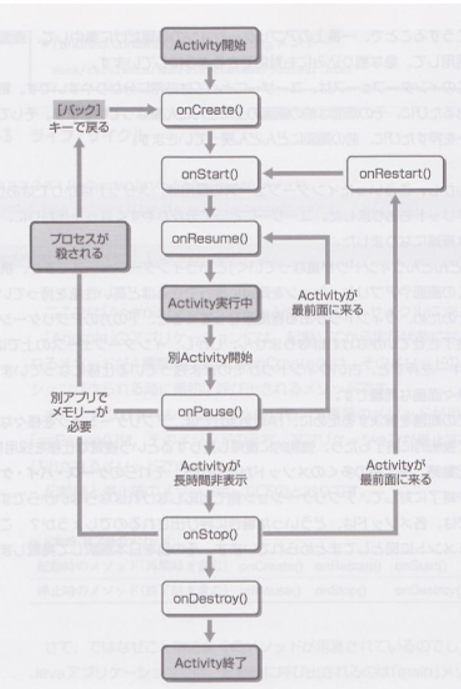
Androidは画面遷移やバックグラウンドへの移動、アプリキルなど様々な動作により

Activityの生成と消滅を繰り返しています。

このような１つ１つのActivityが生成されてから破棄されるまでの工程をライフライクルといいます。

◆Activityライフサイクルのコールバック

Activityのライフサイクルの各段階を移動するために、Activityクラスから提供されているコールバックが7つ



各コールバックの説明

◆onCreate( )

Activityが生成される際、最初に呼ばれる。

ライフサイクルの中で1回のみ実行するものなので

基本的な初期設定やパラメータsavedInstanceStateの受け取りを行うことが多い

◆onStart( )

Activityがフォアグラウンドに移動し、操作可能な状態になるまでの間に呼ばれる。

◆onResume( )

Activityが前面に表示され、ユーザとやりとりが可能になる直前に呼び出される

そのため、表示に必要な処理や初期化をここで処理することが多い

◆onPause( )

アプリがバックグラウンドに移動したことを示すために呼び出される。

そのため、続行しない操作を停止、続行処理はバッググラウンドにあることを想定して調整を行う

なお、実行は非常に短時間で終了するため、状態の保存には適していない

◆onStop( )

アプリが完全にバックグラウンドに移動した時に呼び出される

Activityは非表示になり、停止している。

そのため、高負荷の状態保存はここで行う(アプリデータの保存、ネットワークの呼び出し)

◆onRestart( )

Activityの再表示の際に呼び出される

◆onDestoroy( )

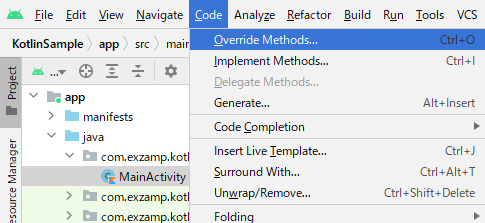
以下の内容でActivityが破棄される前に呼び出される

・アプリキルなどしてActivityが終了した時

・画面の回転、テーマ変更、マルチウィンドウモードによりActivityが一時的に破棄される

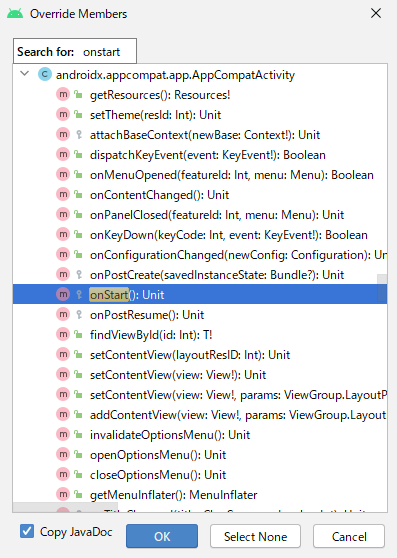
◆各コールバックの実装、および動作タイミングの確認

①CodeタブのOverride Methods..をクリック

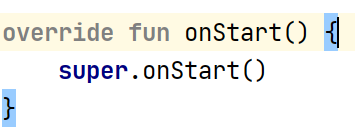


②出てきたwindow内で検索したい文字を入力[例:onStart]しコールバックメソッドを探す

③OKをクリック



指定したコールバック関数が追加されました



同様に

・onResume()

・onPause()

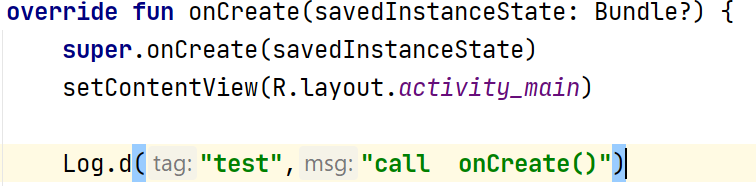
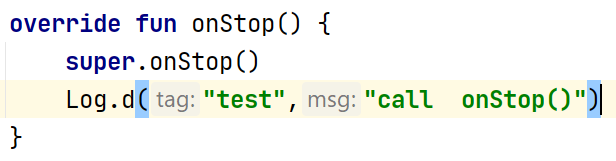
・onStop()

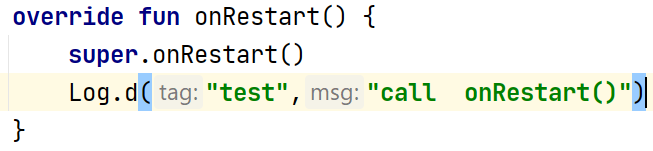
・onRestart()

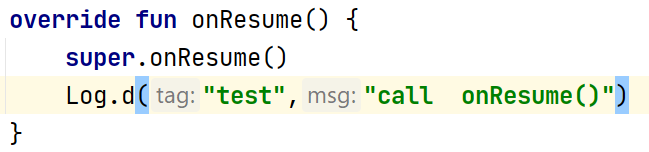
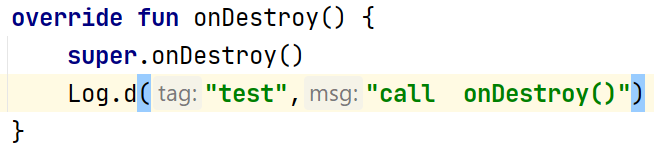
・onDestroy()

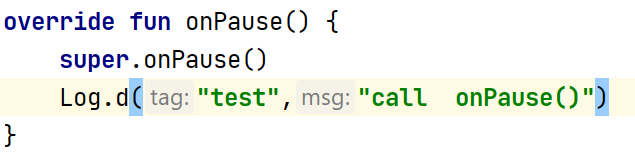
も追加しましょう。

◆各コールバックに確認用のログを出力するように処理を追加





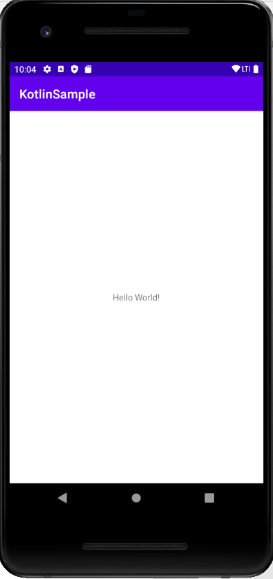




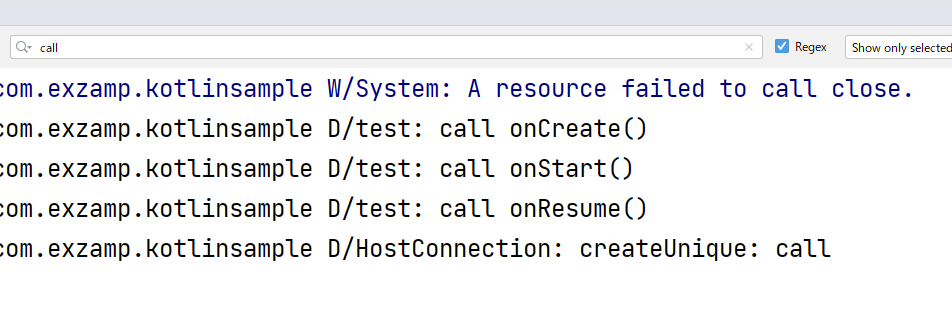
◆アプリ起動時のライフライクル

では実行してログを確認していきます。

まずは実行してみましょう。

　　　　　　　　　　　　　ログはそのままだと見つけにくいので

検索窓に「call」と入力して文字を絞り込んでいます



アプリ起動時のonCreate()

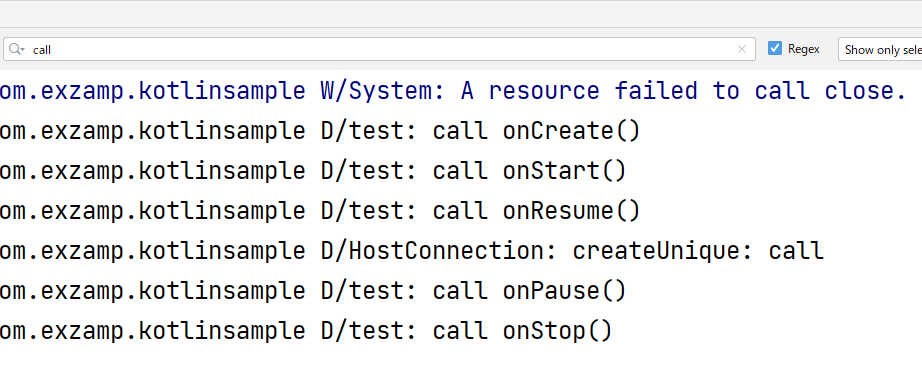
アプリ表示のonStart()

ユーザ操作可能のonResume()

のタイミングでログが出力されていますね！

◆ホームボタン押下のライフサイクル

ではホームボタンを押下してアプリをバッググラウンド状態にさせてみましょう



バッググラウンドへ移動開始時のonPause()

バッググラウンド移動完了時のonStop()

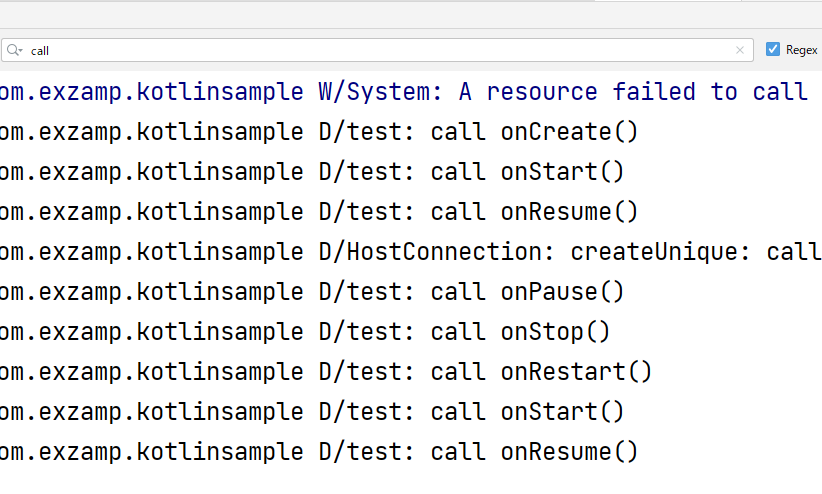
のタイミングでログが出力されていますね！

◆アプリ復帰時のライフサイクル

では□ボタンを押下した状態で、アプリを選択することで

再度アプリをフォアグランドに戻しましょう







アプリの再表示開始時のonResutart()

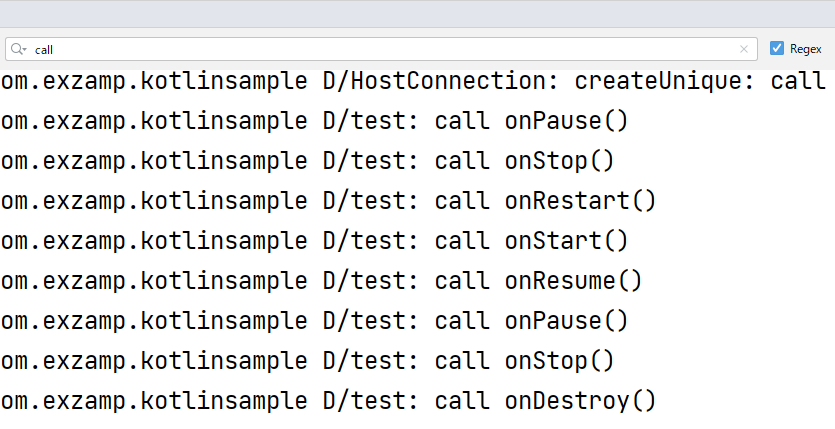
アプリ表示時のonStart()

ユーザ操作可能のonResume()

のタイミングでログが出力されていますね！

◆アプリキル時のライフサイクル

では□ボタンを押下して、更にアプリを上にスワイプすることでアプリキルになります。



バッググラウンドへ移動開始時のonPause()

バッググラウンド移動完了時のonStop()

アプリがキルされる直前のonDestroy()

のタイミングでログが出力されていますね！

デバッグについて

■デバッグ方法：その1.ログで確認

デバッグとは、コンピュータプログラムに潜む欠陥を探し出して取り除くこと。

エラー内容を確認するだけでなく、プログラムの途中で変数に何が入っているかの確認も

デバッグ機能で確認することが可能です。

■デバッグ方法：その2.ブレイクポイントで止めて確認する

ログに出力するのではなく、特定行の時点で変数には何が格納されているのかを

Android Studioでは確認することが出来ます。

特定の行で処理を止める為に「ブレイクポイント」を設定します。

一時的に止めたい処理の行の場所をクリックすることでブレイクポイントを設定できます

ブレイクポイントを設定した状態でデバッグ実行をします

デバッグ実行は虫アイコンをクリックで行います

