



# Android勉強会 #1

2012/11/16



# TDDで Android開発する



# Getting started

# Application

- ・日記帳アプリを作ります
- ・起動したら、日記の一覧が出てくる
- ・メニュー ボタンのメニューから日記追加ができる
- ・一覧を長押しすると、日記が削除できる
- ・削除する前に確認画面が出てくる

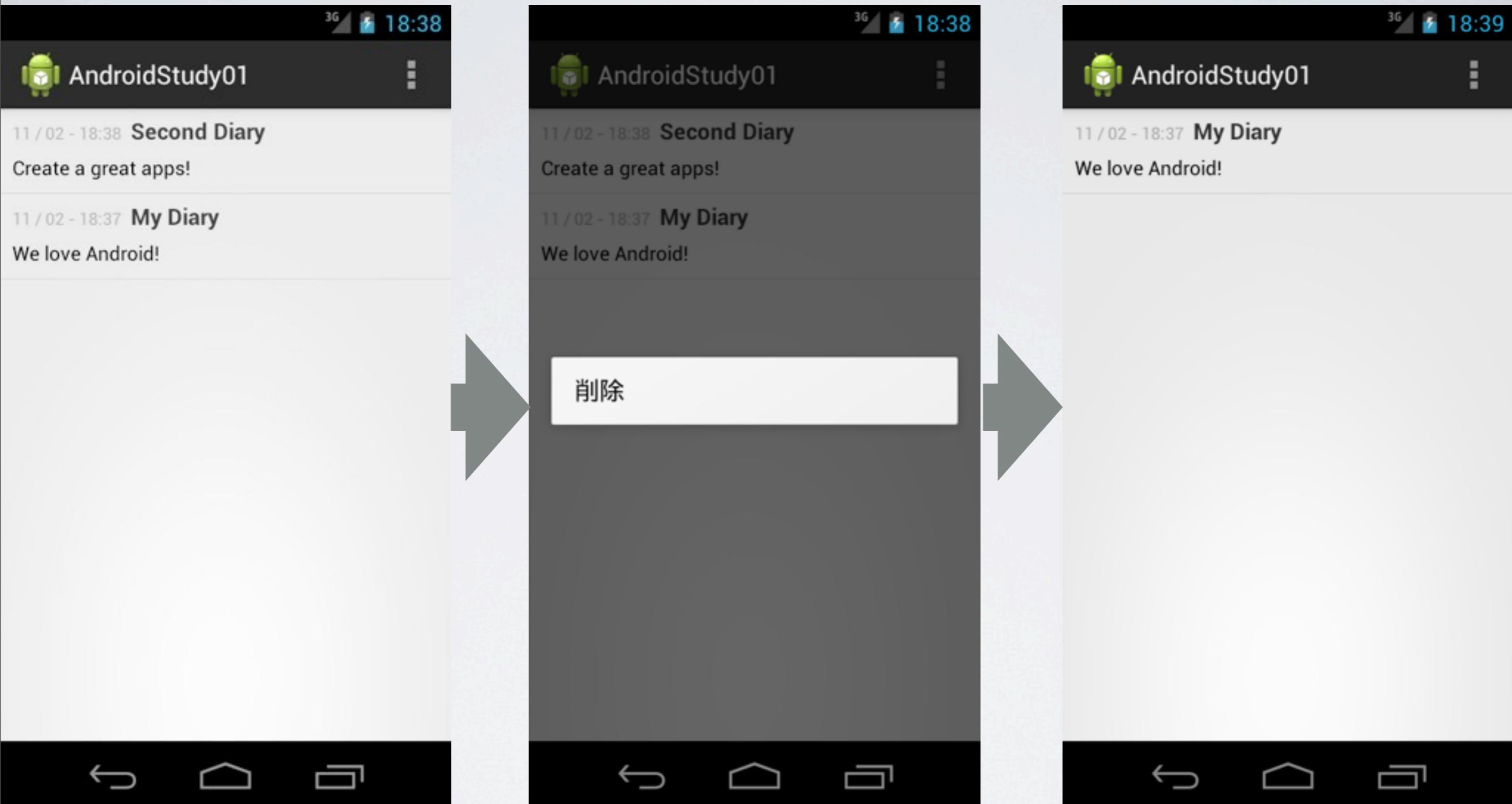
Getting Started

# Application



Getting started

# Application



Getting started



ENGINE  
START

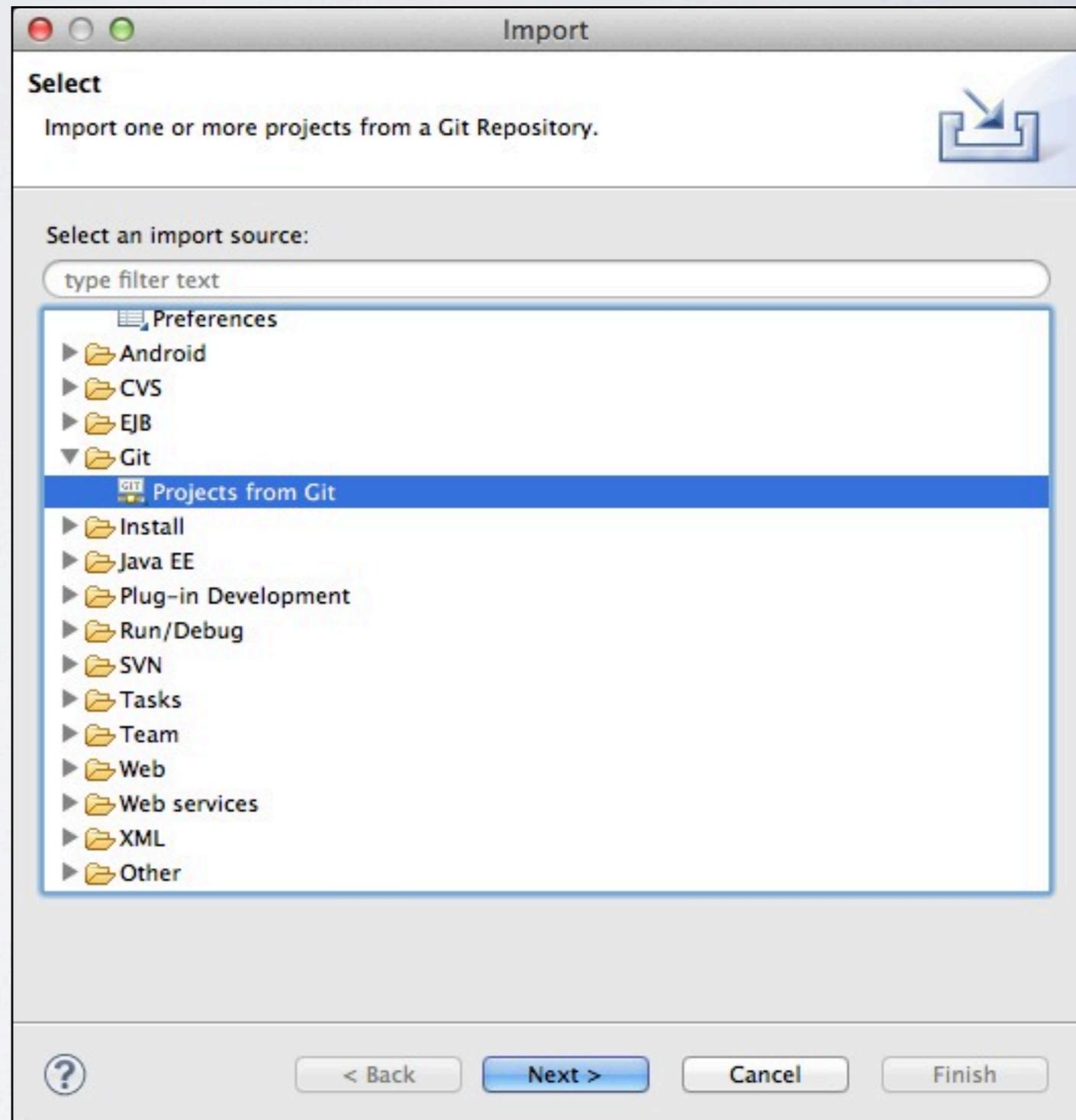
# Clone projects

<http://bit.ly/RJyhwN>

Getting started

# Import projects from git

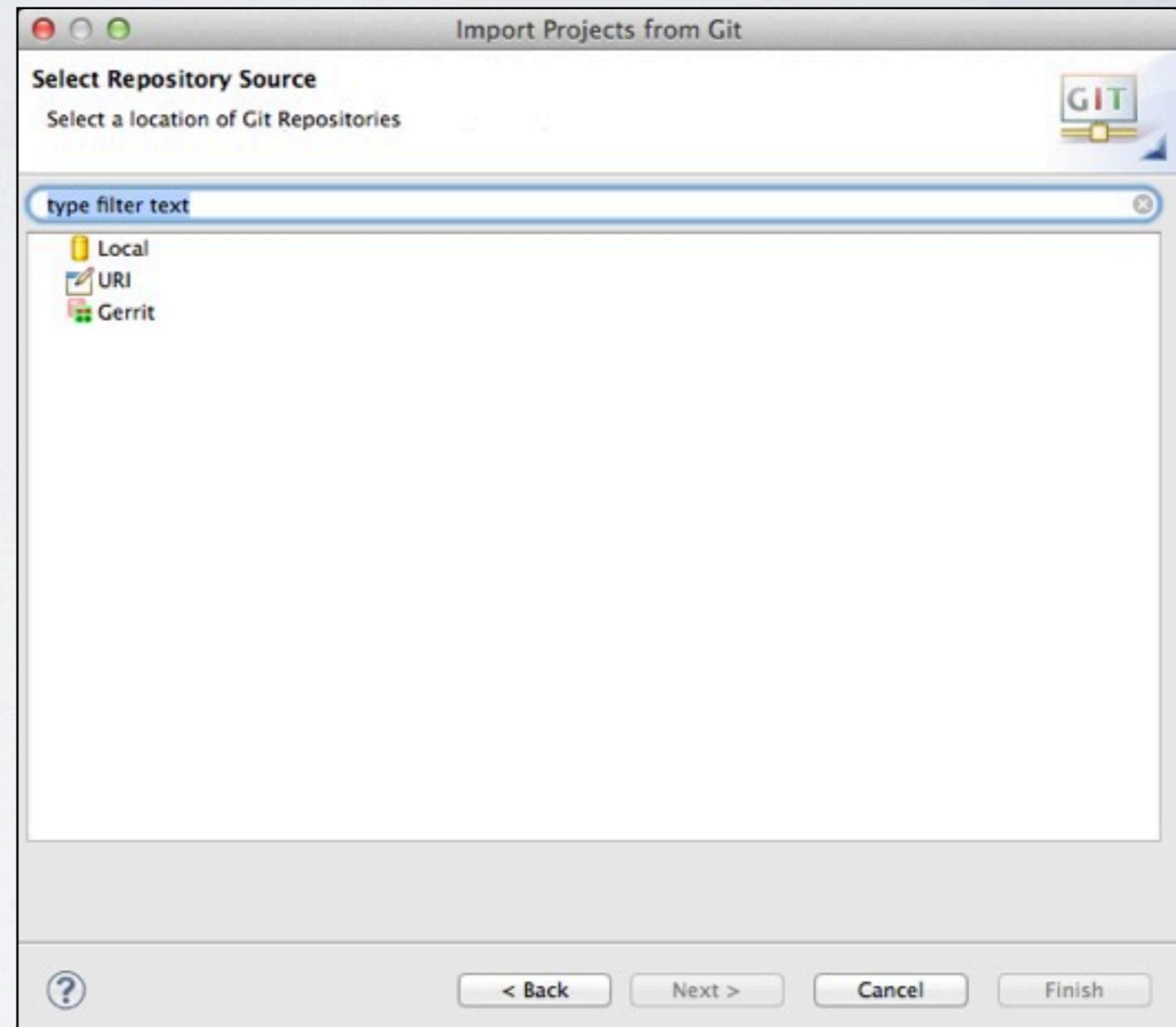
- File > Import...
- Git > Projects from Git



Gett

# Import projects from git

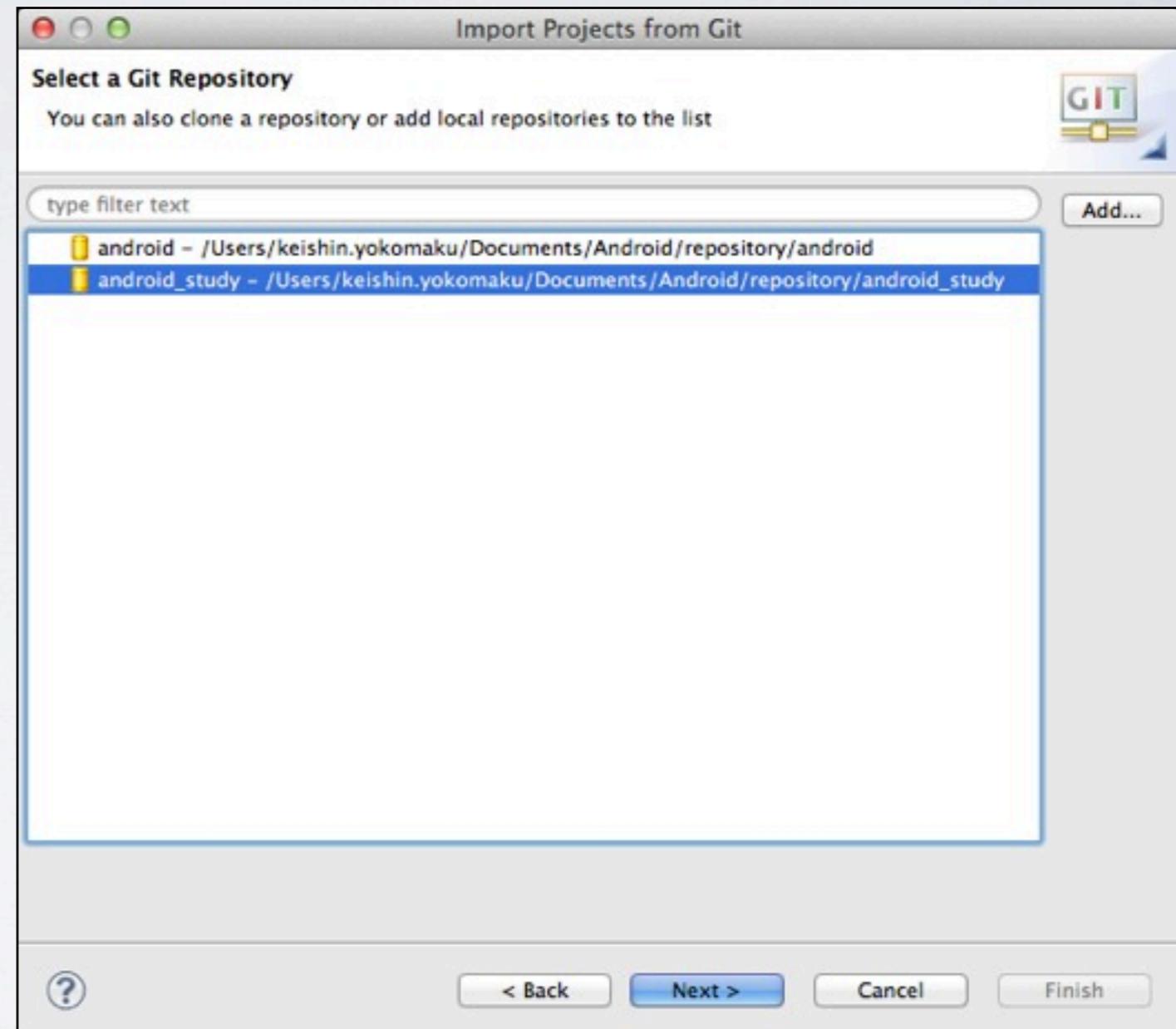
- File > Import...
- Git > Projects from Git
- Localから



Getting started

# Import projects from git

- File > Import...
- Git > Projects from Git
- Localから
- Cloneしたrepository
- あとはNextとFinish



Getting started

# Check Point

- 赤いビックリマークが出る時
  - プロジェクトの設定
    - Javaのコンパイラを1.5以上にする
    - 文字コードをUTF-8にする
  - Project > Clean...

Getting started

# Building Layout



# Building Layout

- res/layout ディレクトリ
  - activity\_compose\_diary.xml



# Building Layout

- **RelativeLayout**

- Viewを相対位置で配置する

- **LinearLayout**

- Viewを順番に並べる

- **TableView**

- Viewを表形式で並べる

- **AbsoluteView**

- @Deprecated



# Building Layout

- **View**

- 全てのViewの親

- **横**(android:layout\_width)

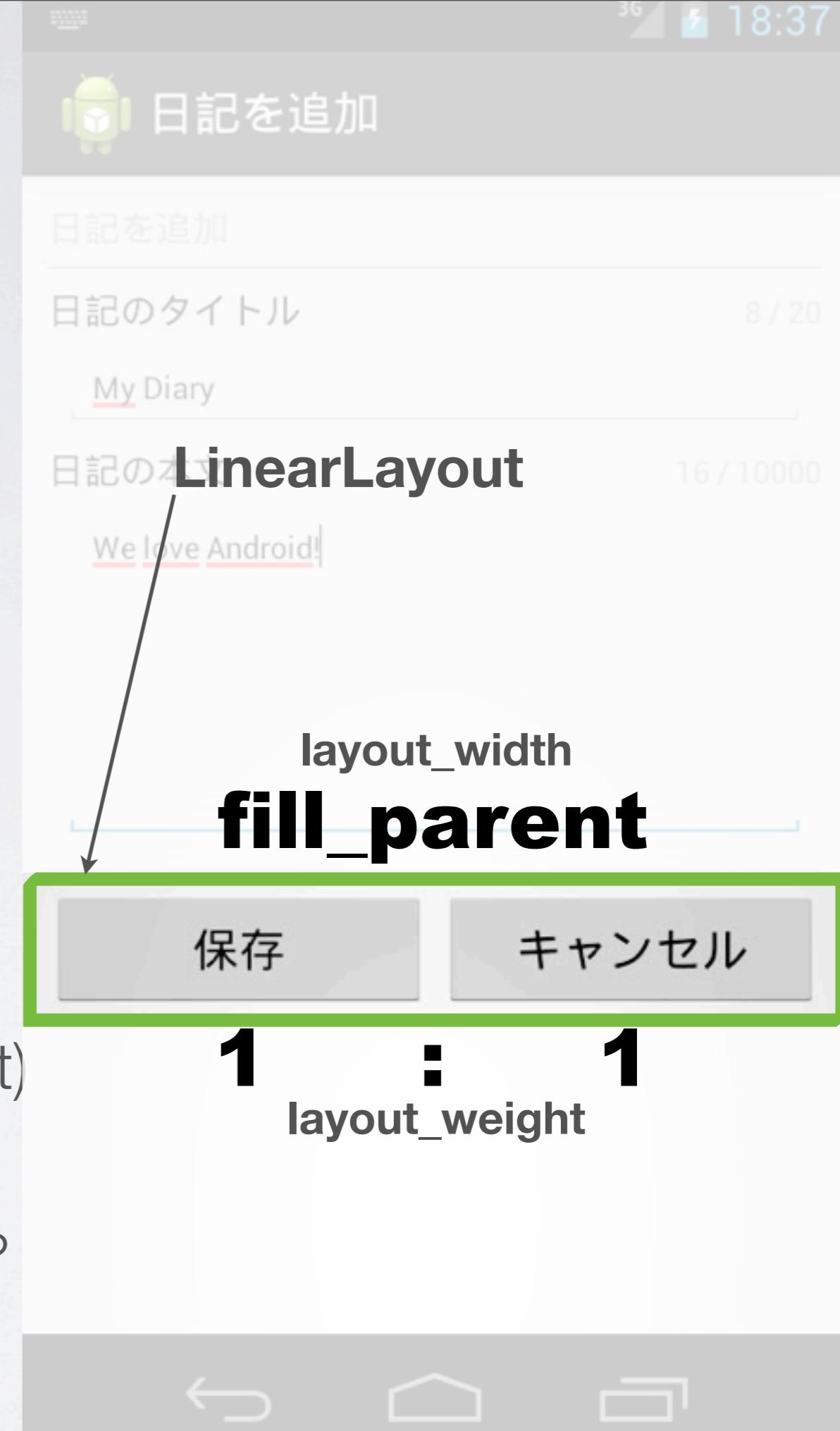
- wrap\_content: 中身に合わせる
  - fill\_parent: 親一杯に広げる

- **縦**(android:layout\_height)

- 横と同じ

- **ウェイト**(android:layout\_weight)

- 1, 2, 3...
  - LinearLayoutの子だけが設定できる



# Building Layout

- **View**

- 全てのViewの親

- マージン(android:layout\_margin)

- 1dp, 2dp, 3dp...

- 他のViewとの距離を取る

- パディング(android:padding)

- 1dp, 2dp, 3dp...

- 自分のViewの中にスペースを取る



# Building Layout

- **View**

- 全てのViewの親

- **右側**(android:layout\_toRightOf)
- **左側**(android:layout\_toLeftOf)
- **下**(android:layout\_below)
- **上**(android:layout\_above)
  - RelativeLayoutの子だけ設定できる
  - Viewのidを設定する



# Building Layout

## • ScrollView

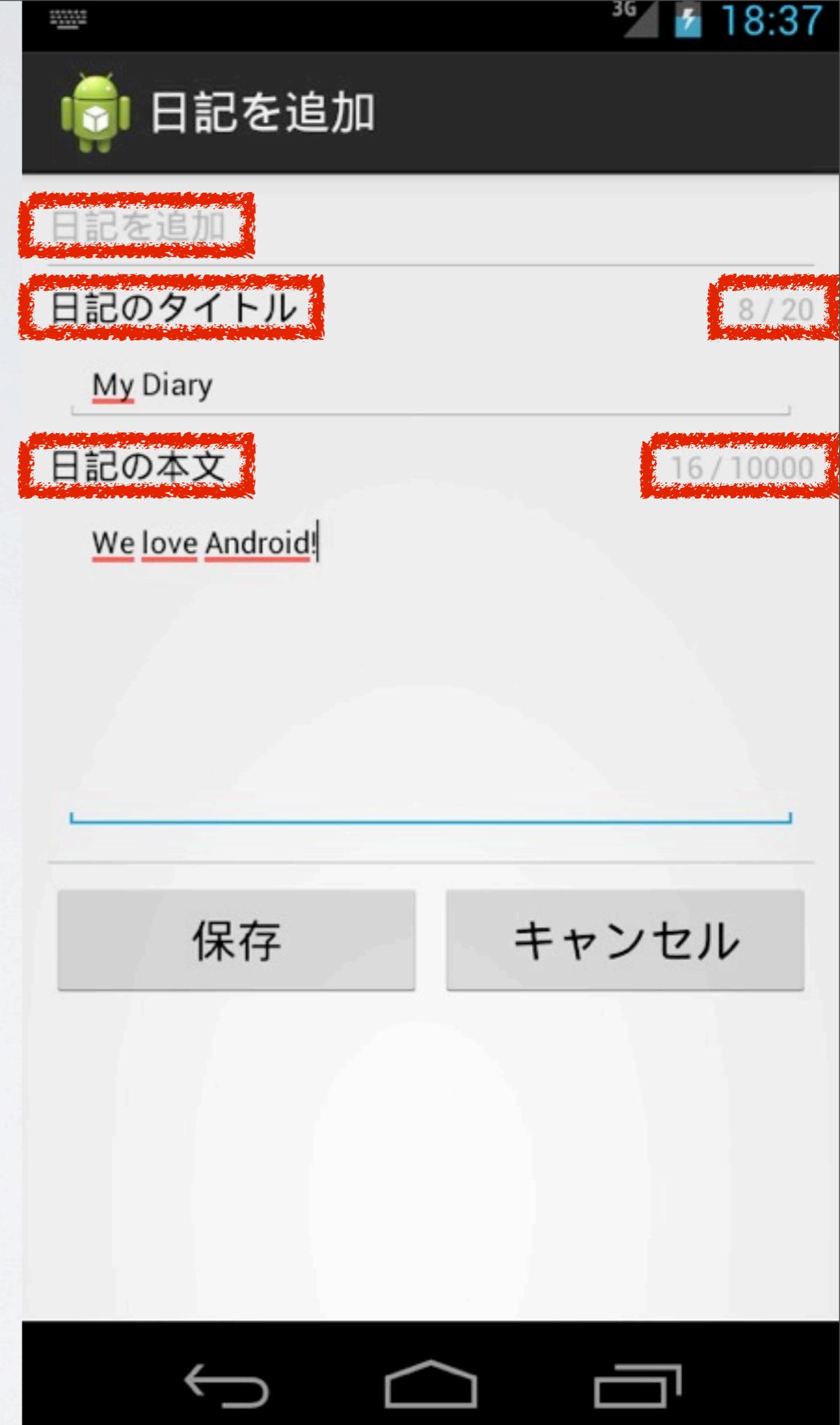
- 中身をスクロール可能に
  - 子要素は一つだけ
    - 二つ以上入れると怒られる
  - 子要素のandroid:layout\_height
    - wrap\_content
  - ソフトウェアキーボード
    - キーボードも画面の要素
    - 表示する分だけ狭くなる
    - もしScrollViewがないと...



# Building Layout

- **TextView**

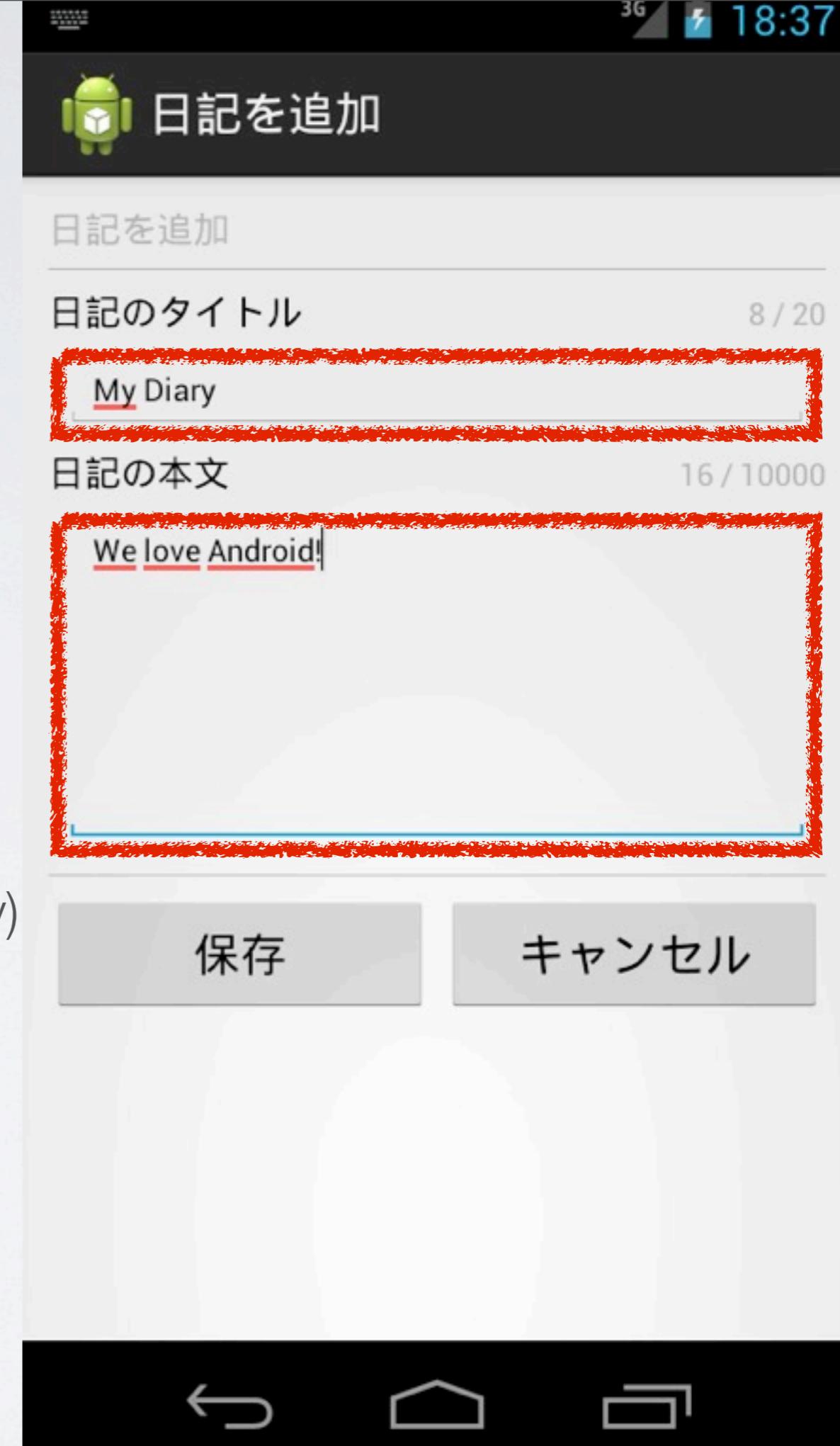
- 文字列を表示するView
  - 色(android:textColor)
    - #333333
    - @android:color/black
  - スタイル(android:textStyle)
    - bold, italic
    - 端末依存あり
  - サイズ(android:textSize)
    - 12sp, 13sp, 14sp...
    - 8spだと小さすぎて見えない



# Building Layout

- **EditText**

- 文字を入れるView
  - 行数(android:lines)
    - 1, 2, 3...
  - ヒント(android:hint)
    - 文字列を渡してプレースホルダに
    - AndroidLintの警告対象(Accessibility)



# Building Layout

- **Button**

- ボタンView
  - イベント処理(android:onClick)
    - クリックしたら呼ぶメソッド名
    - Activityにpublicで定義しておくこと



# Building Layout

- <ScrollView>
  - <RelativeLayout>
    - <TextView>
    - <View>
    - <TextView>
    - <TextView>
    - <EditText>
    - <TextView>
    - <TextView>
    - <EditText>
    - <View>
    - <LinearLayout>
      - <Button>
      - <Button>



# Building Layout

- Sample

- <http://bit.ly/TlqwsY>



# Activity



日記を追加

日記を追加

日記のタイトル

8 / 20

My Diary

日記の本文

16 / 10000

We love Android!

保存

キャンセル



# Activity

## • 文字数カウントする

- EditText

- addTextChangedListener()

- TextWatcherを登録しよう！

- Activity#onStart()で実行する

- ユーザの入力を受け付ける前準備

- Activity#onStop()で解除する

- コールバックの登録=参照を渡す

- コールバックの解除=参照を切る

- 参照が残る=GCできない

- これを忘れるメモリリークする



# Activity

## • 文字数カウントする

- EditText

- TextWatcher

- onTextChanged()

- EditTextの文字列が変わったら呼ばれる

- ここで文字列の長さを取得する

- 文字列の長さはlength()メソッドで

- TextViewにlength()は無いので...

- 長さを表示するためのメソッド

- setTitleLengthIndicator()

- setBodyLengthIndicator()



# Activity

## • ボタンのイベント処理

- android:onClick
- XMLで書いたものと同じ名前のメソッドを用意しよう
  - onSaveClick()
  - onCancelClick()
  - 引数にViewオブジェクトを取る
    - イベントを発火したViewオブジェクト



# Activity

## • ボタンのイベント処理

- onSaveClick

- 入力した結果をオブジェクトに

- 今回はDiaryEntityオブジェクト

- ビルダーパターンで入力を  
オブジェクトに突っ込んでいく

- オブジェクトをIntentに入れる

- Intent#putExtra(String, Parcelable)

- Activity#setResult(int, Intent)で、  
呼び出し元ActivityにIntentを飛ばせる

- Activity#finish()でActivityが終わる



# Activity

- Sample

- <http://bit.ly/ZrQtl9>

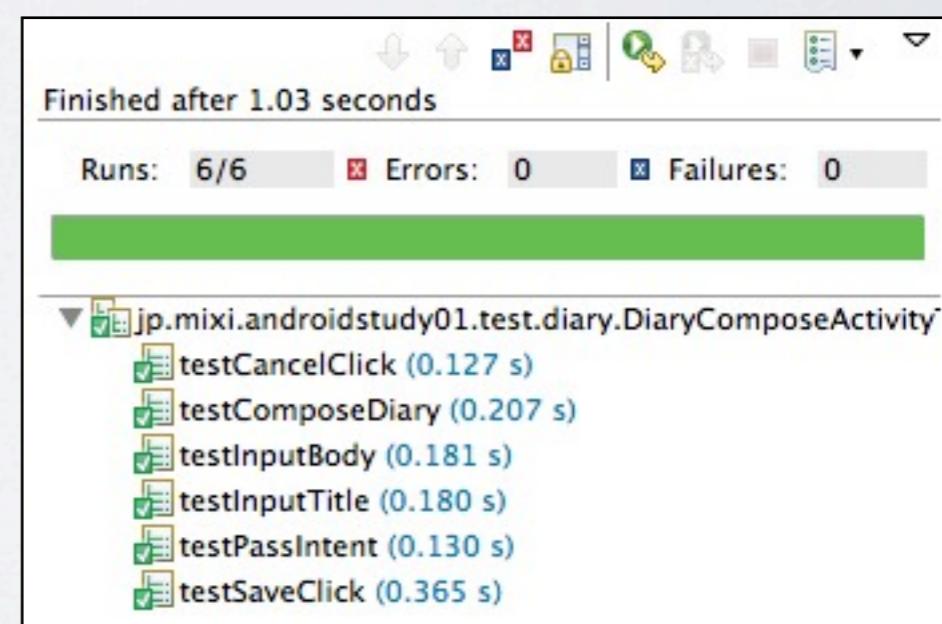
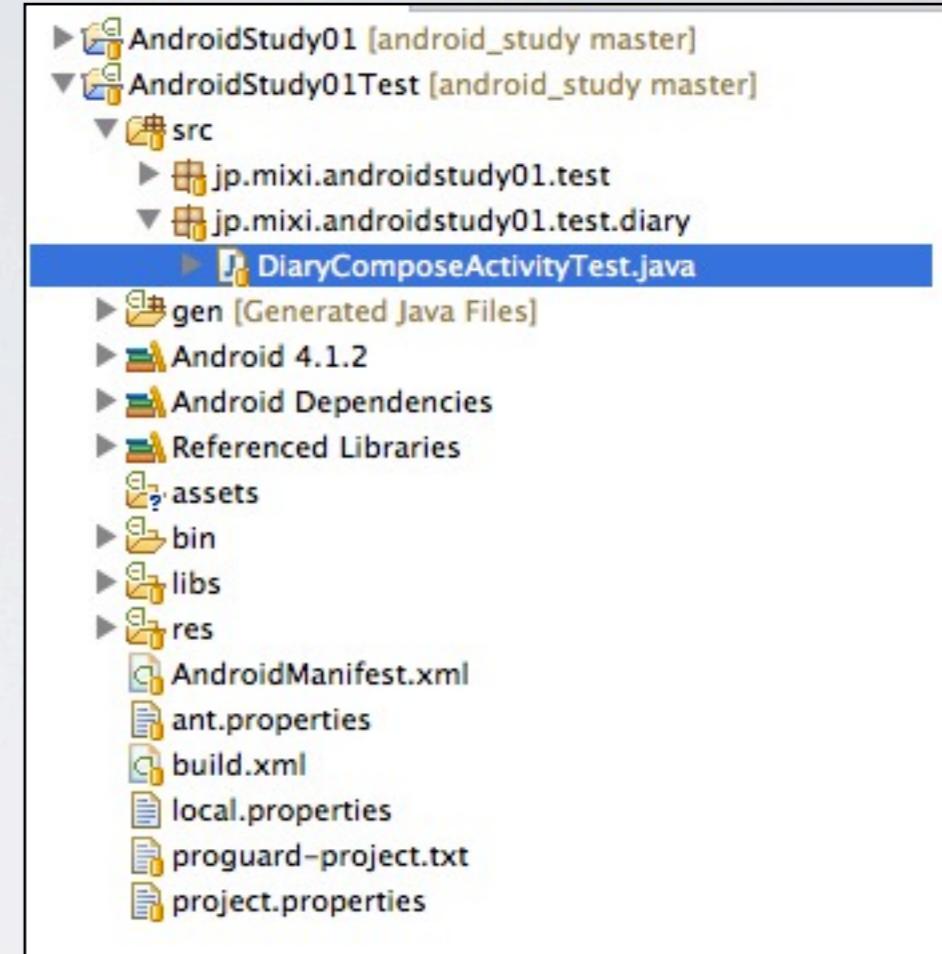


# Test

- テストを動かす
- 画面の動作テスト
- ActivityInstrumentationTestCase2<T extends Activity>
  - 自分で操作しなくても、プログラム書いたら勝手に動いてくれます
  - 動かす前に、端末のロックを解除するのを忘れずに

# Test

- テストを動かす
- DiaryComposeActivityTest
  - 選択して、実行ボタン
  - 全部緑なら成功
  - 赤い×はエラー
  - 青い×は失敗



# Activity



AndroidStudy01

:

日記を追加

何もありません



# Activity

- MainActivity

- **ListView**

- Listが空の時の表示(EmptyView)

- ListView#setEmptyView(View)
- 好きなViewオブジェクトを渡して  
空の時の表示をカスタマイズ可能



# Activity

- MainActivity

- Menuキーで出すメニュー

- Activity#onCreateOptionsMenu()

- res/menuディレクトリの下のxmlを読み込んで、メニュー一覧を作る

- MenuInflater#inflate(int, Menu)で読み込む



# Activity

- MainActivity

- Menuキーで出すメニュー

- Activity#onOptionsItemSelected()

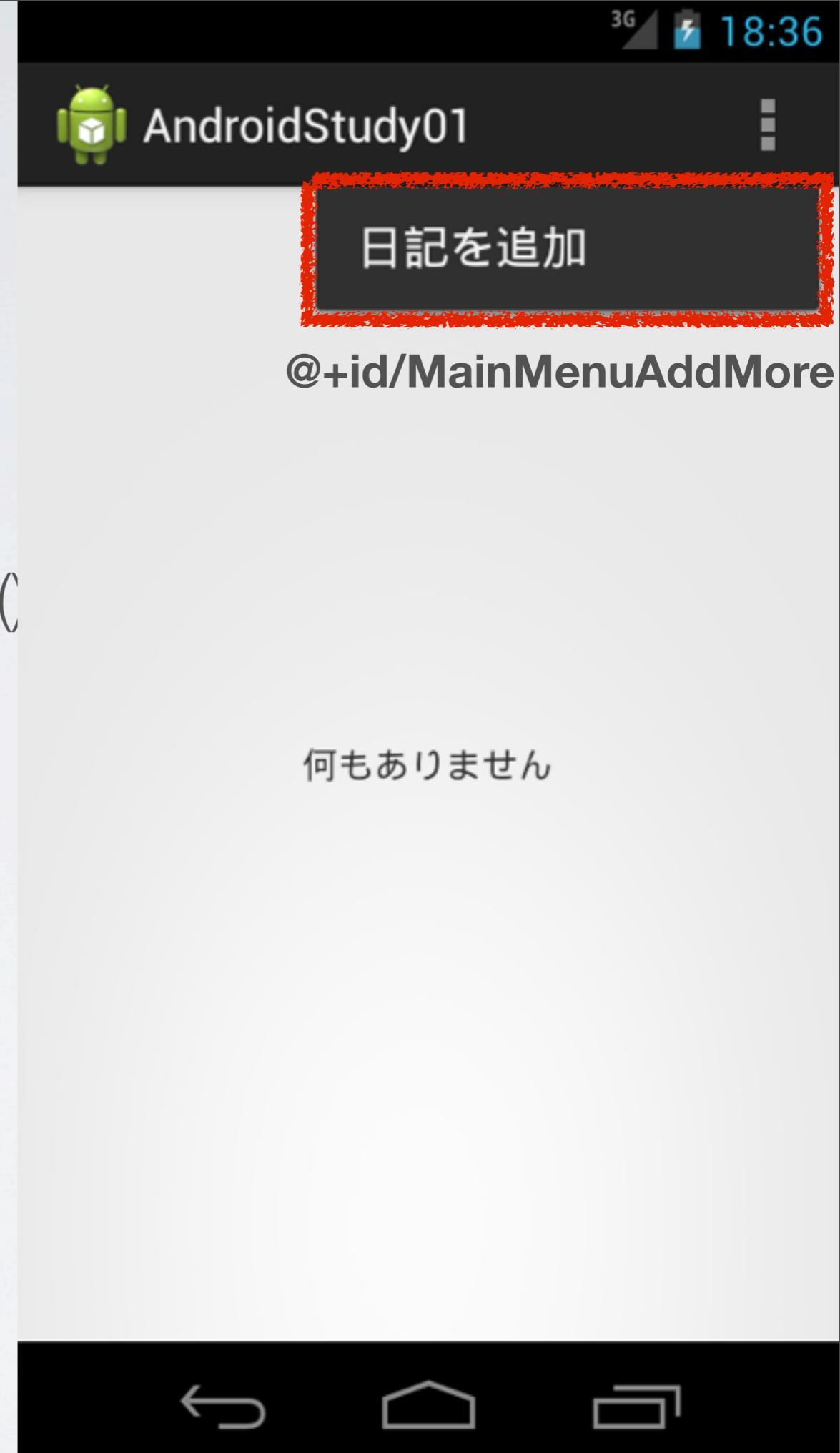
- メニューのどれかが選択された

何もありません

- 対応する処理を呼び出す

- メニューにはidがついています

- idを見て...



# Activity

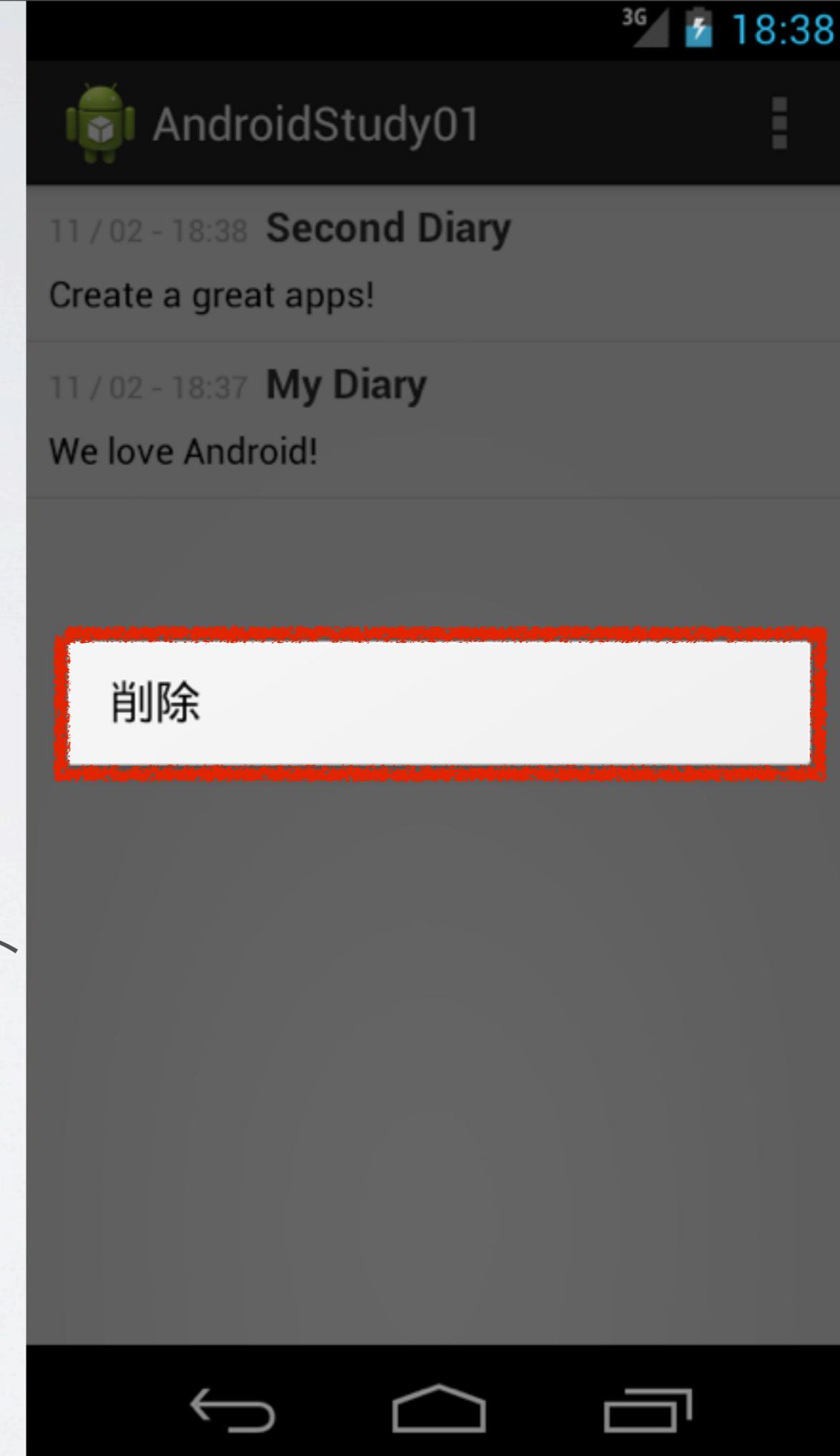
- MainActivity

- 長押しメニュー

- ContextMenuとも言う

- Activity#  
registerForContextMenu(View)

- Viewオブジェクトに長押しついイベント  
ハンドラーを登録する



# Activity

- MainActivity

- 長押しメニュー

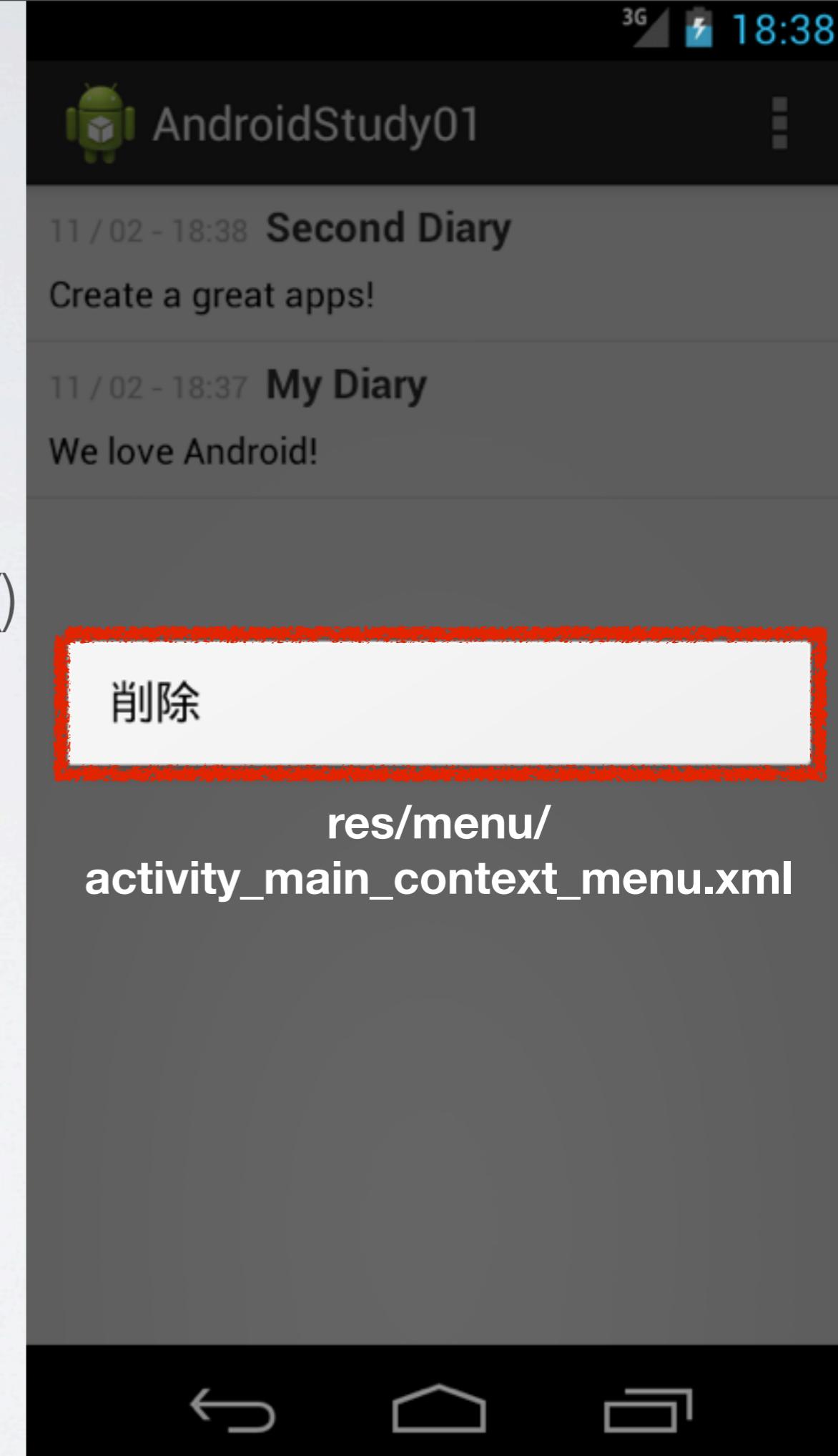
- Activity#onCreateContextMenu()

- OptionsMenuと同じく  
res/menu以下のxmlを読み込んで  
表示する

- menuのxml

- <item>要素が1項目に相当

- android:idでidを降る
      - android:titleで表示する文字列を設定



# Activity

- MainActivity

- 長押しメニュー

- Activity#onContextItemSelected

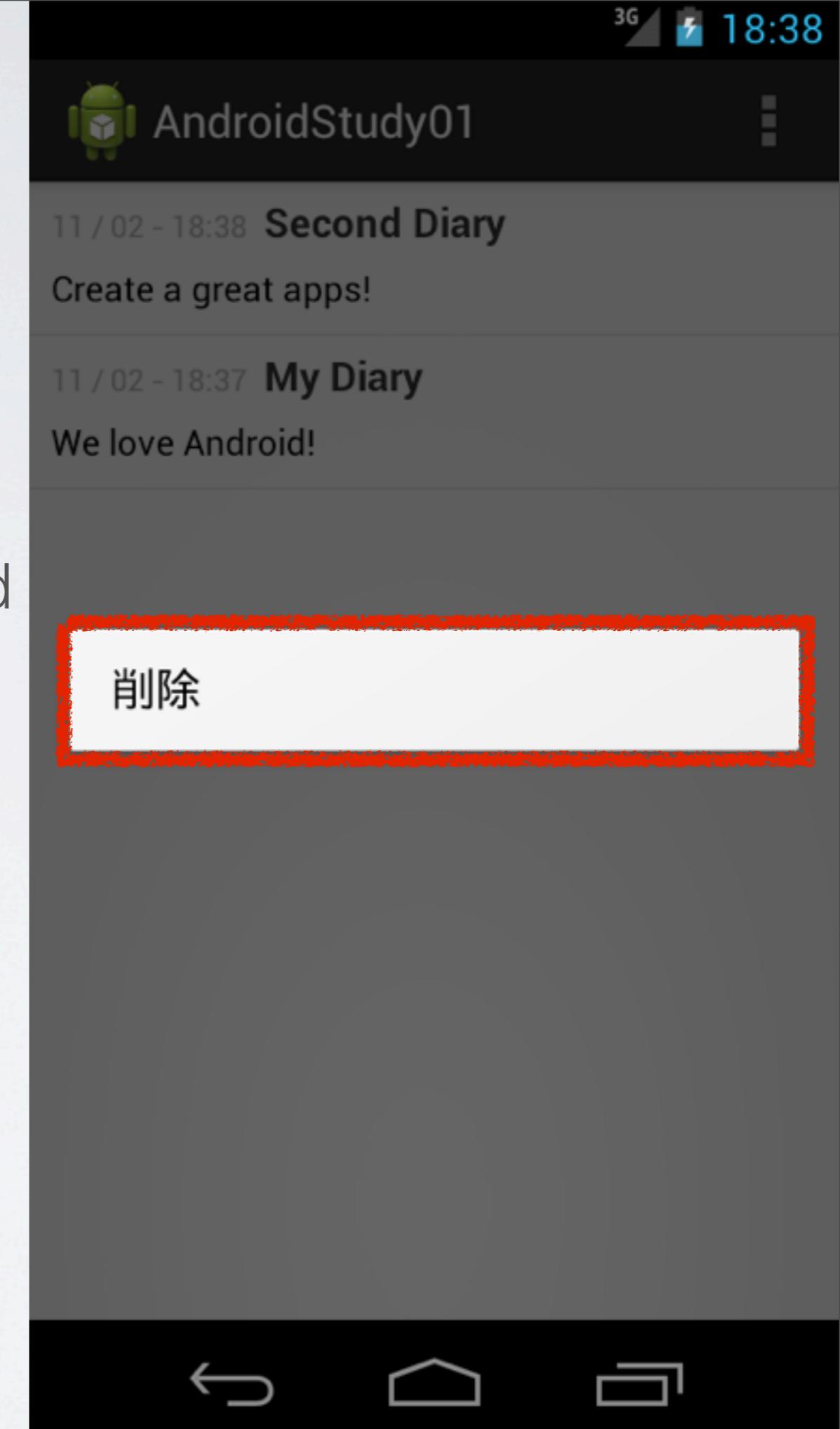
- 長押しメニューが選択された時

- ListViewなので、Listのどの位置が長押しされたか知りたい

- ListView用のMenuItemInfoオブジェクトを引数のMenuItemオブジェクトから取得

- AdapterView.AdapterContextMenuInfo

- 他はOptionsMenuと同じように...



# Fragment

- DialogFragment

- ダイアログを表示する

- タイトル
- メッセージ
- ポジティブボタン
- ネガティブボタン
  - ポジティブ・ネガティブの位置はOSバージョンで違いがある



# Fragment

- DialogFragment

- ダイアログを表示する

- onCreateDialog(Bundle)

- ここでAlertDialogオブジェクトを作ります

- AlertDialog.Builder

- setTitle(int)
    - setMessage(int)
    - setPositiveButton(int, OnClickListener)
    - setNegativeButton(int, OnClickListener)



# Fragment

- DialogFragment

- ダイアログを表示する
- コールバックしよう
  - 削除するボタンを押したことを FragmentからActivityに伝える
  - Fragment#getActivity()で自分が紐付いているActivityを取得
  - Activityはコールバック用の Interfaceを持っているので...



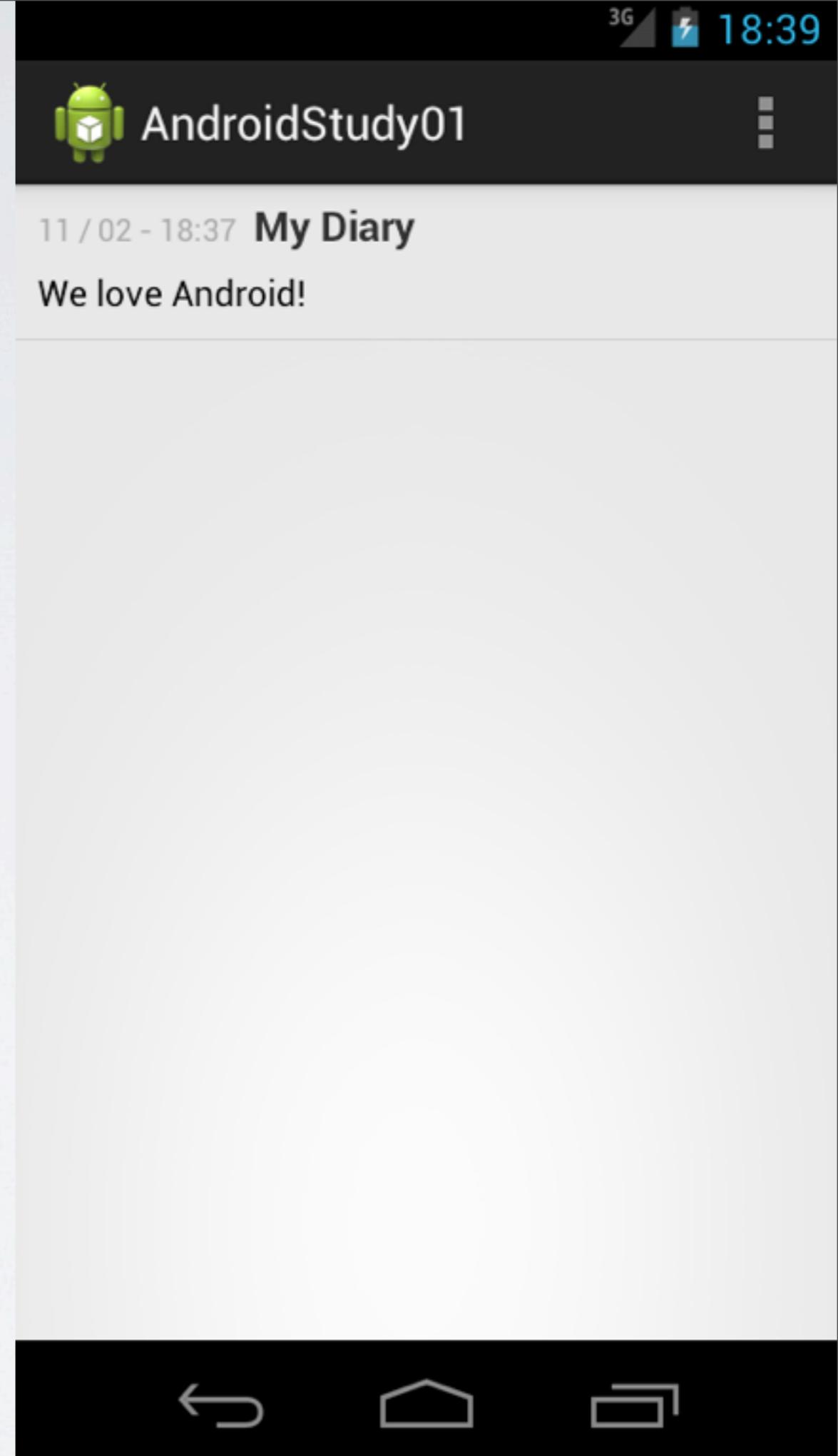
AlertDialog

# Activity

- MainActivity

- コールバックを受ける

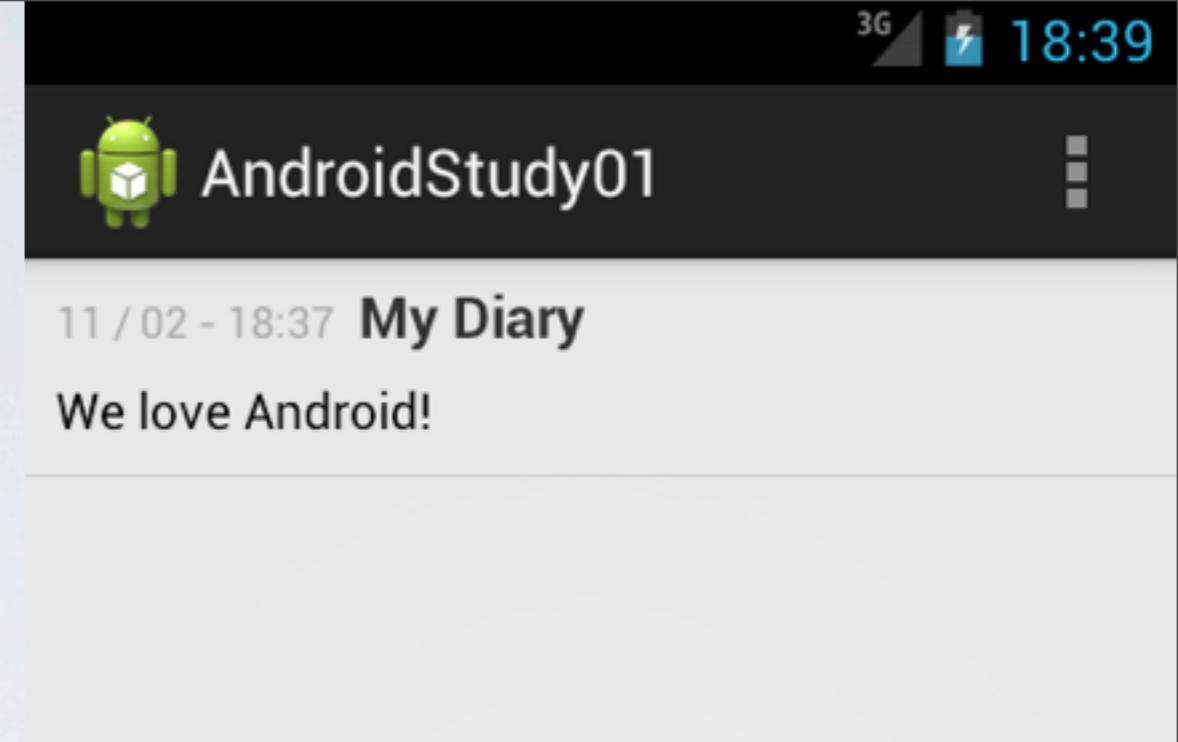
- onPositiveClick(int)
  - 削除する位置が引数にあります
  - ArrayAdapter#remove(int)



# Activity

- MainActivity

- 画面回転すると...

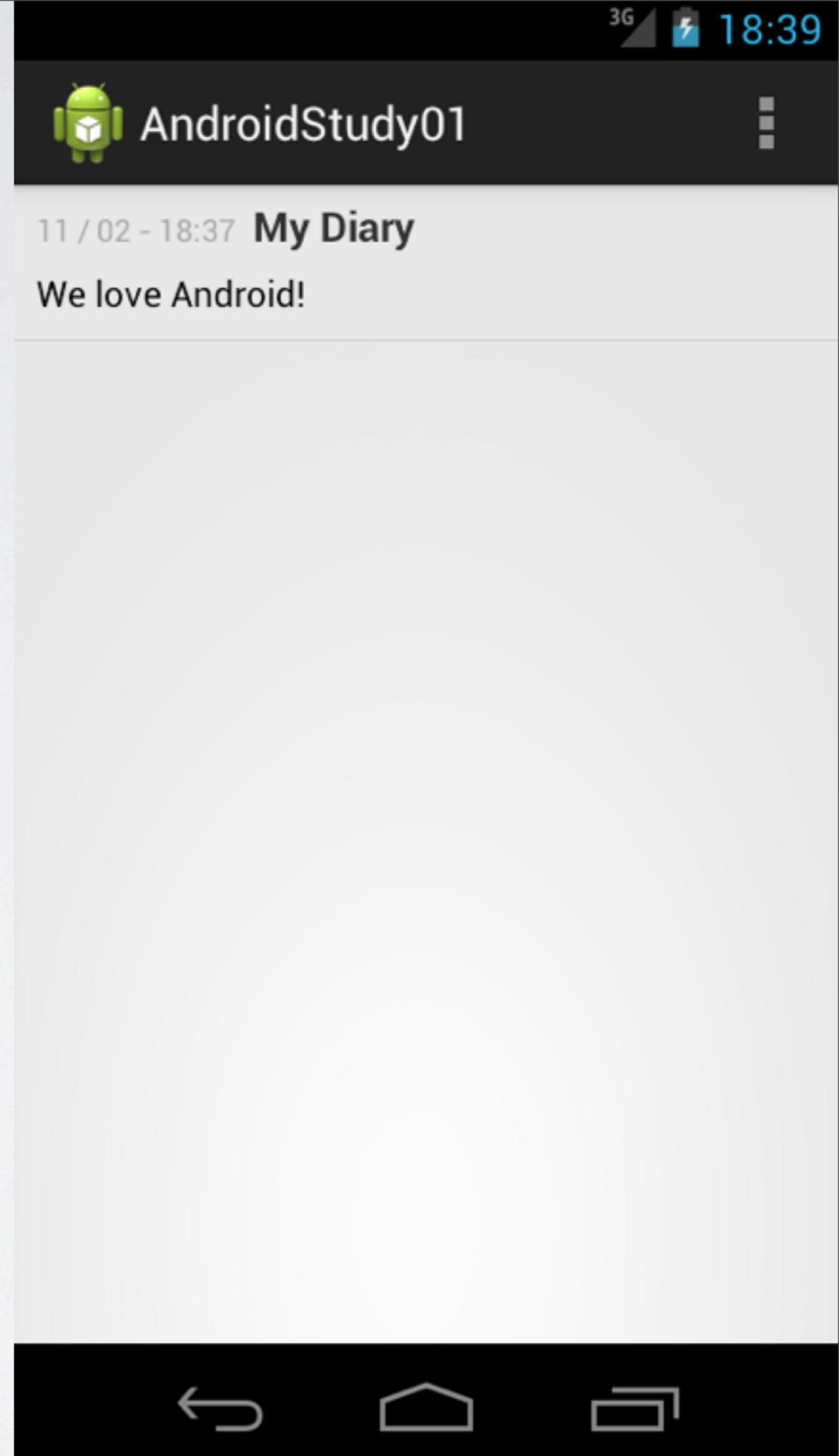


# Activity

- MainActivity

- 画面回転で状態を引き継ぐ

- onSaveInstanceState(Bundle)
  - 画面回転前に、状態を保存する
  - 今回は、Adapterが持っているListを取り出して、Bundleに入れる
- onRestoreInstanceState(Bundle)
  - 画面回転後に、状態を復帰する
  - Bundleに入れたListを取り出し、Adapterにセットする



# Test

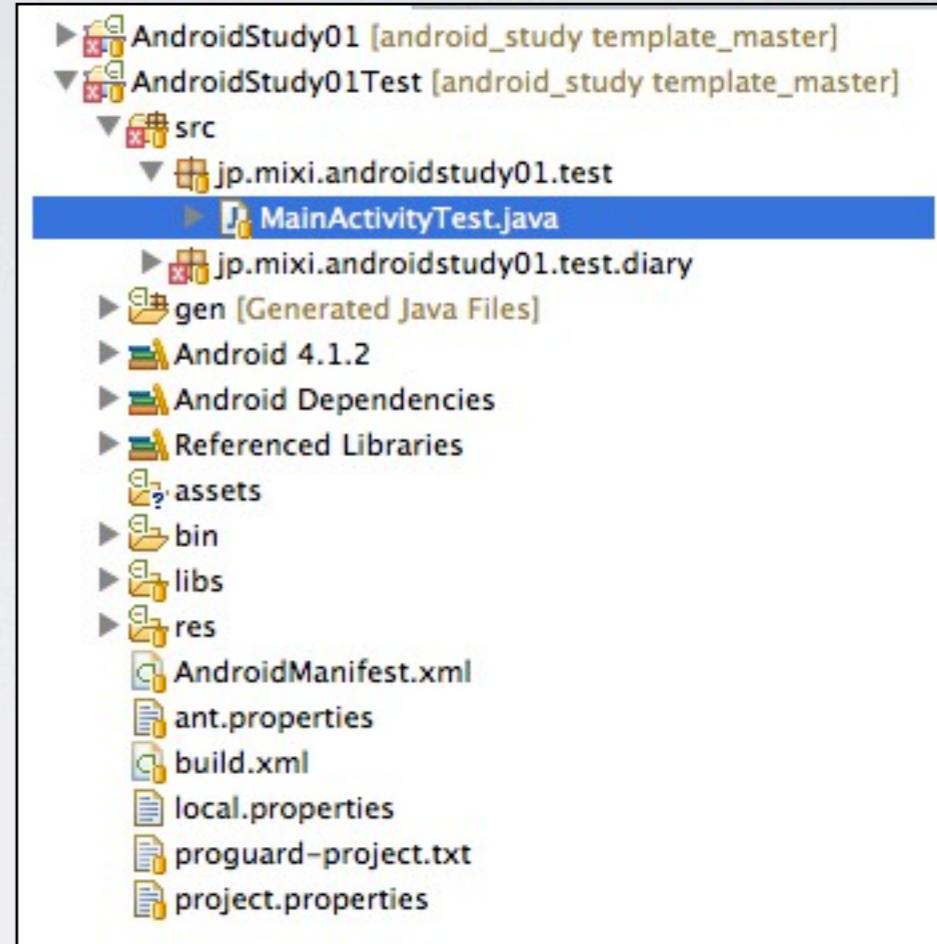
- テストを動かす

- MainActivityTest

- 選択して、実行ボタン



- DiaryComposeActivityTestとは  
すこし違うやり方をしていきます



# Robotium

- 画面のテストをするフレームワーク
- 特徴
  - 実際のユーザの操作をプログラムする感覚で書ける
    - ○○と書いてあるボタンを押す、とか、キーボードのMを押す、とか
  - Android標準フレームワークで出来ないテストが書ける
    - ListViewの何番目を押す、とか、ダイアログのボタンを押す、とか
  - Jenkinsの自動テストでも動かせる
    - Androidのテストの仕組み（JUnit3）上で動いてくれる

# Goal!



# まとめ

# Check Point

- 画面を作るときに気をつけること
  - メモリから破棄する準備をちゃんとする
  - **割り当てたリソースの開放**を忘れないこと
    - Activityそのものがリークするので結構大きな領域を取ってしまう
    - AsyncTask等で別スレッド作ったら、**破棄前に止める**こと
      - メイン(UI)スレッドに戻った時にNullPointerExceptionが出ることも

Activity & Fragment

# Check Point

- 画面を作るときに気をつけること
- 重い処理は別スレッドでやる
  - HTTPリクエストやストレージのR/W等は**非同期処理**で
  - ただし、非同期処理の中でUIを触ってはいけない
  - UIを触っていいのはメイン(UI)スレッドだけ

Activity & Fragment

おしまい

# Enquete

**<http://svy.mk/TCaK0p>**

**11/21(水)まで**

# **Appendix**

# **Activity & Fragment**

# Activity

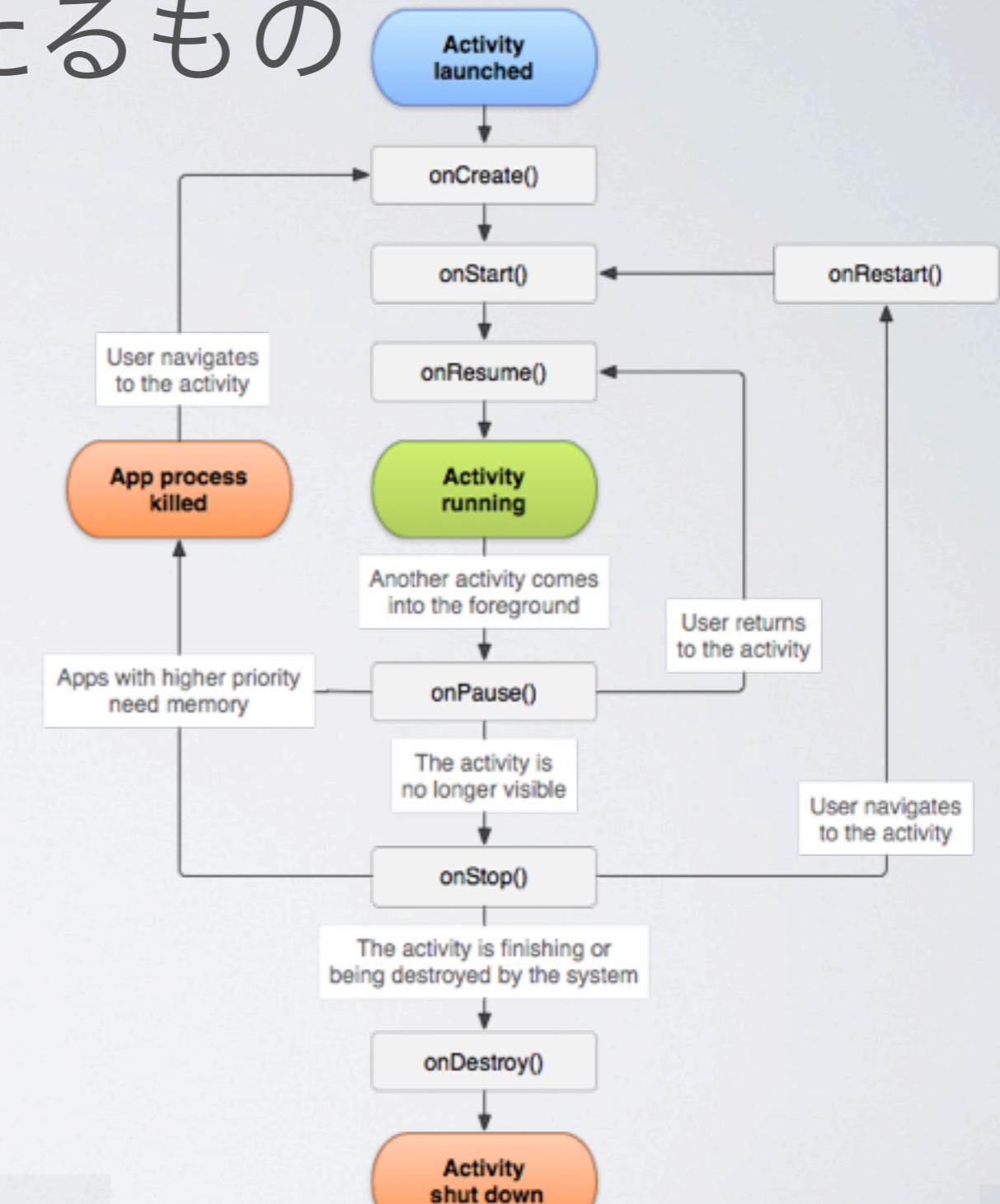
- MVCのControllerに当たるもの

- ライフサイクルがある

- 画面遷移したり

- 画面回転したり

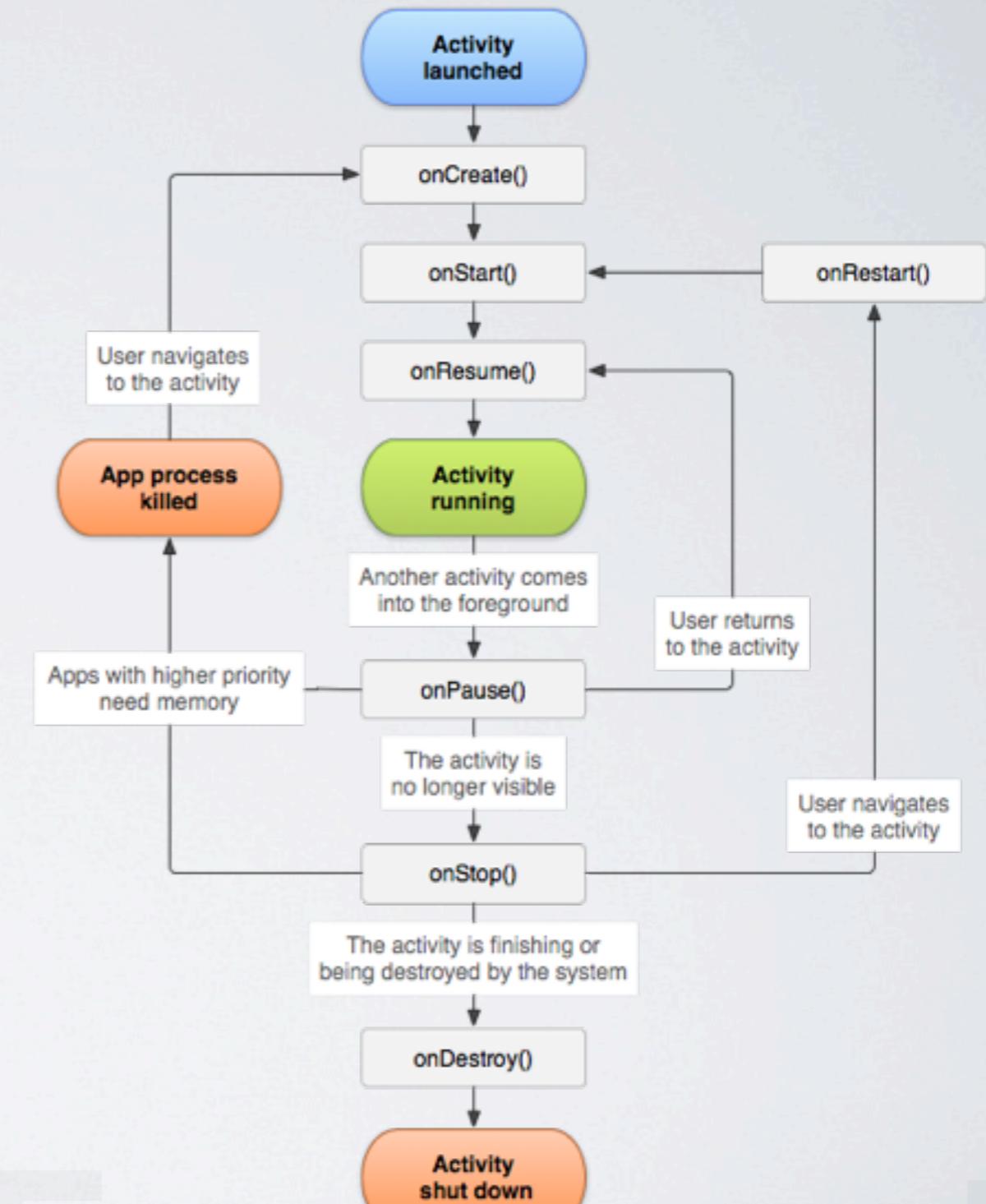
- 画面が消えたり



Activity & Fragment

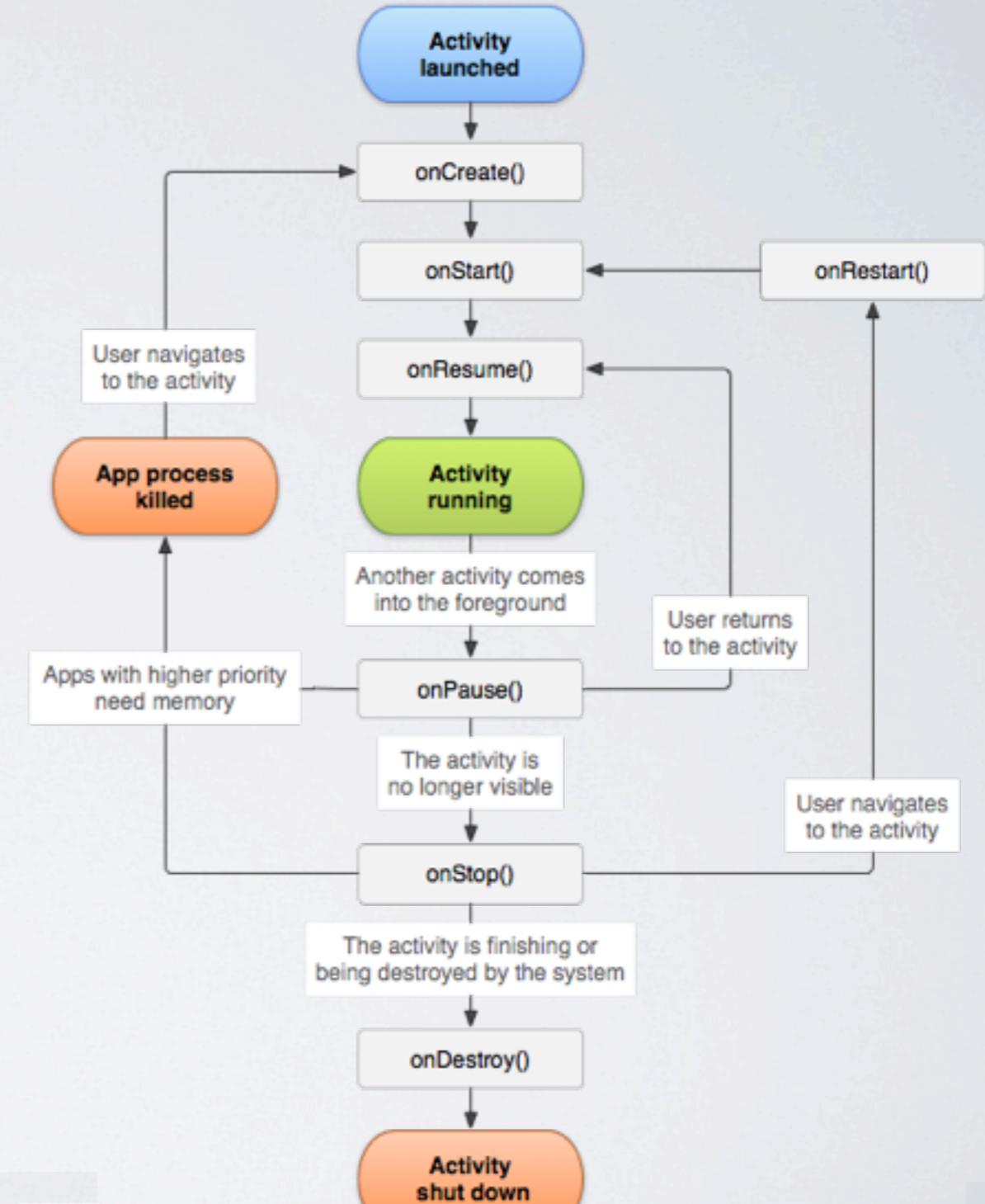
# Activity

- onCreate
    - Activityが生成される時
    - ViewをLayoutから読込む
    - 画面表示をする前準備
    - スレッドを開始するのもここ



# Activity

- onStart
  - 画面が見える時
    - リソースを割当てる
      - BroadcastReceiver登録
      - ContextMenu登録
      - Listener登録



Activity & Fragment

# Activity

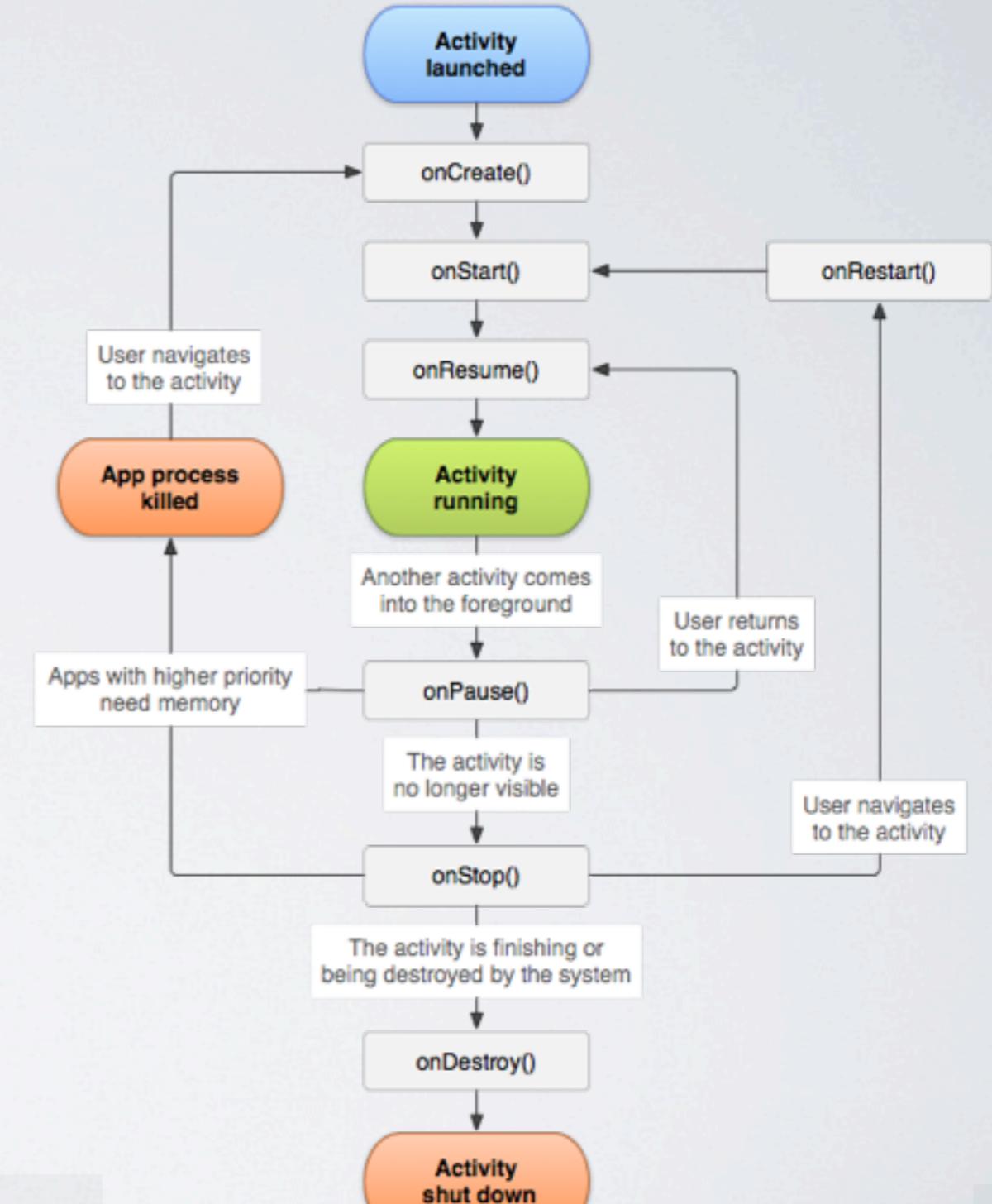
- onResume

- ユーザが操作可能な時

- 見た目の操作が始まる

- アニメーションとか

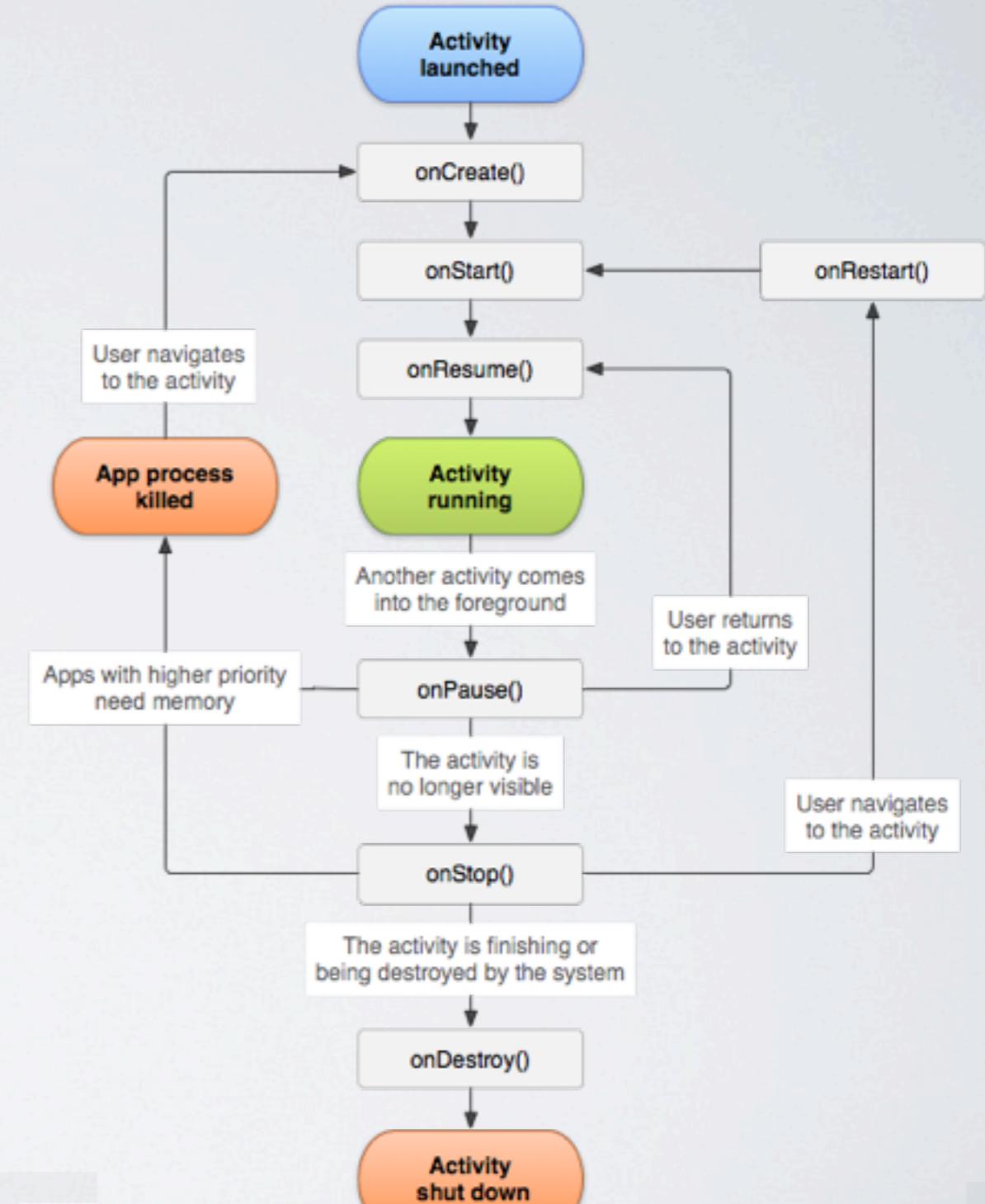
- カメラ起動とか



Activity & Fragment

# Activity

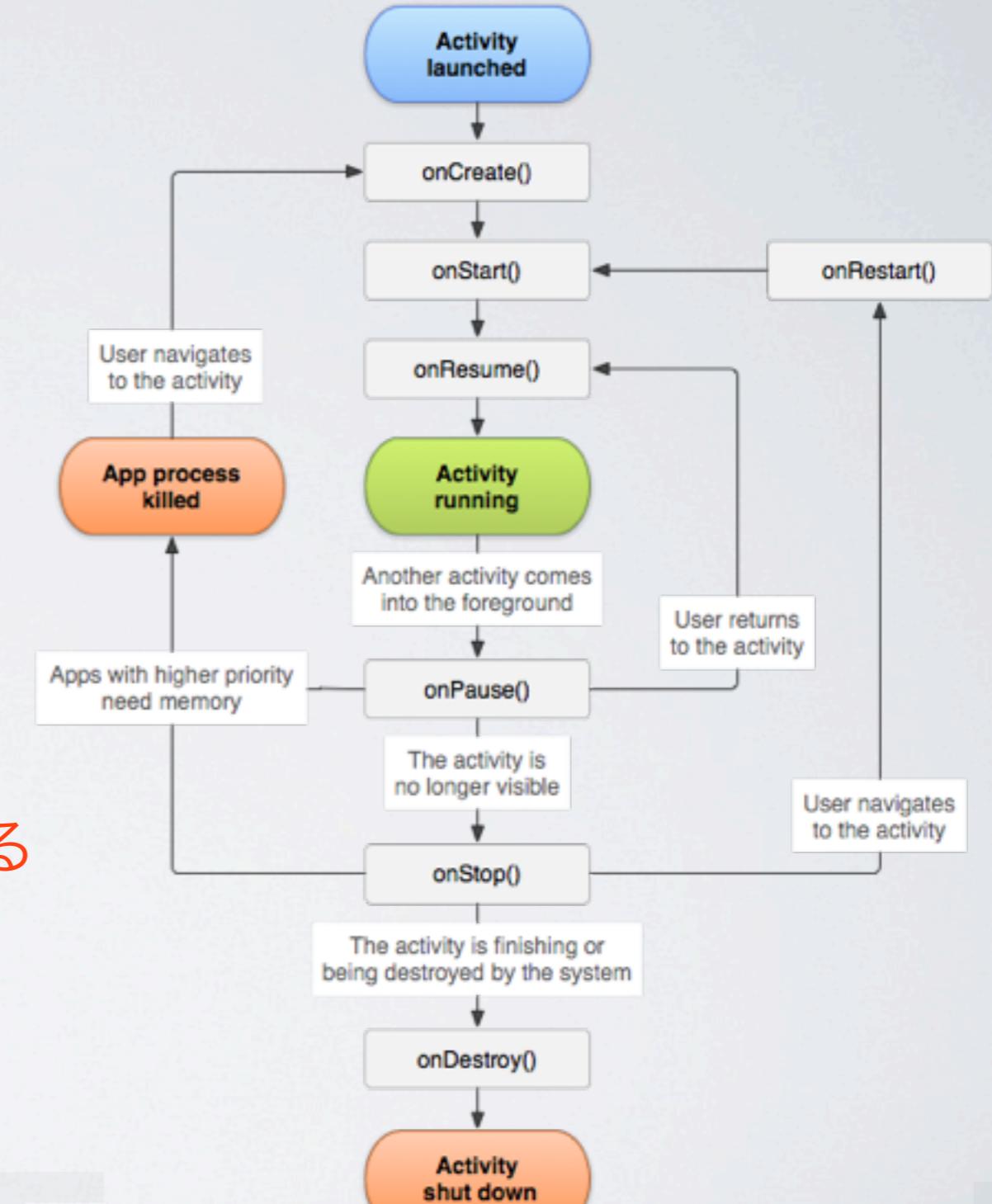
- onPause
  - ユーザが操作不可能な時
  - onResumeの逆
    - アニメーションを止めるとか



# Activity & Fragment

# Activity

- onStop
- 画面が見えなくなる時
  - onStartの逆
    - リソースの開放をする
    - 開放しないとメモリリークになる



# Activity & Fragment

# Activity

- `onDestroy`

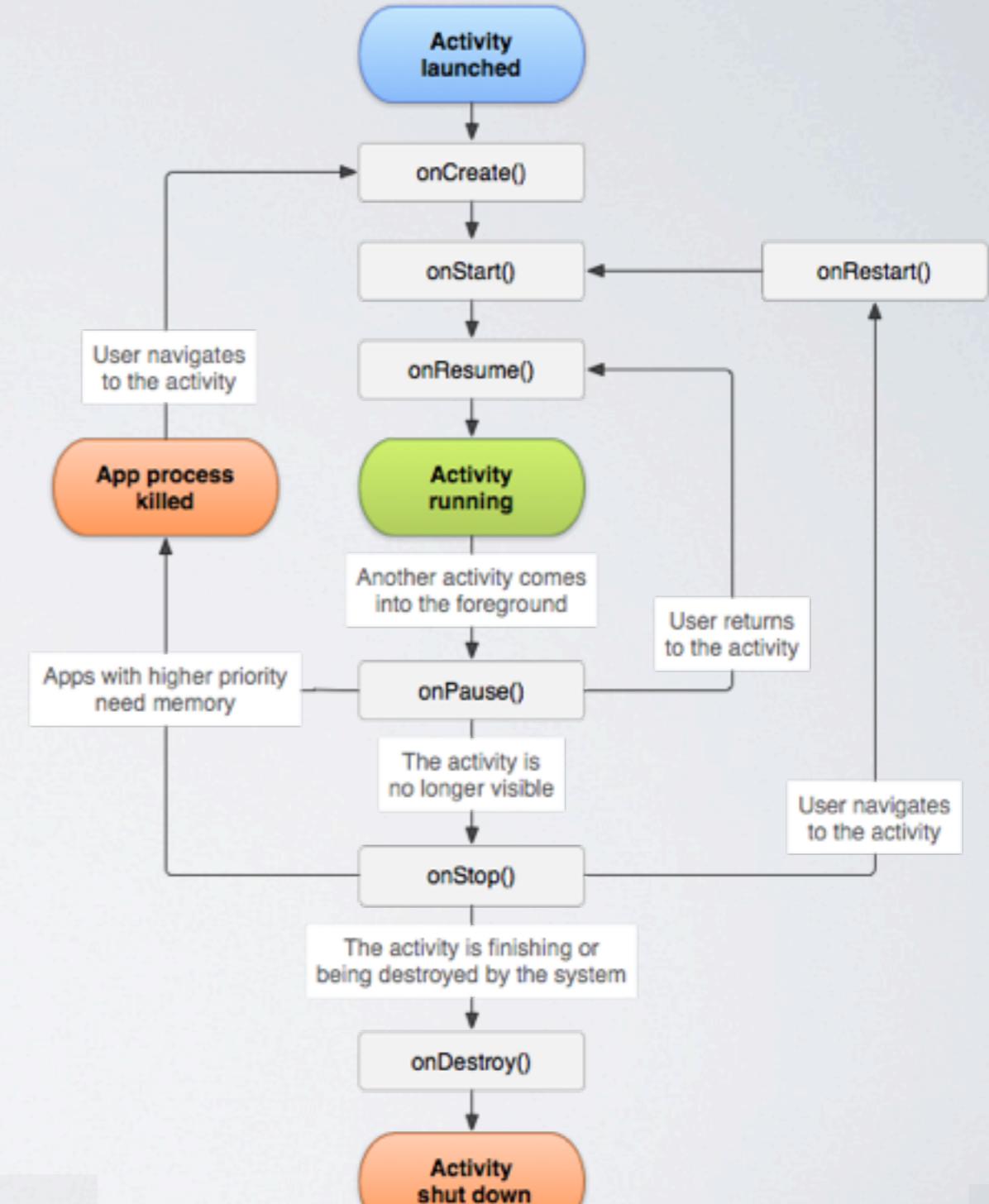
- Activityが死ぬ時

- `onCreate`の逆

- スレッドを止めるとかする

- 呼ばれない時もある

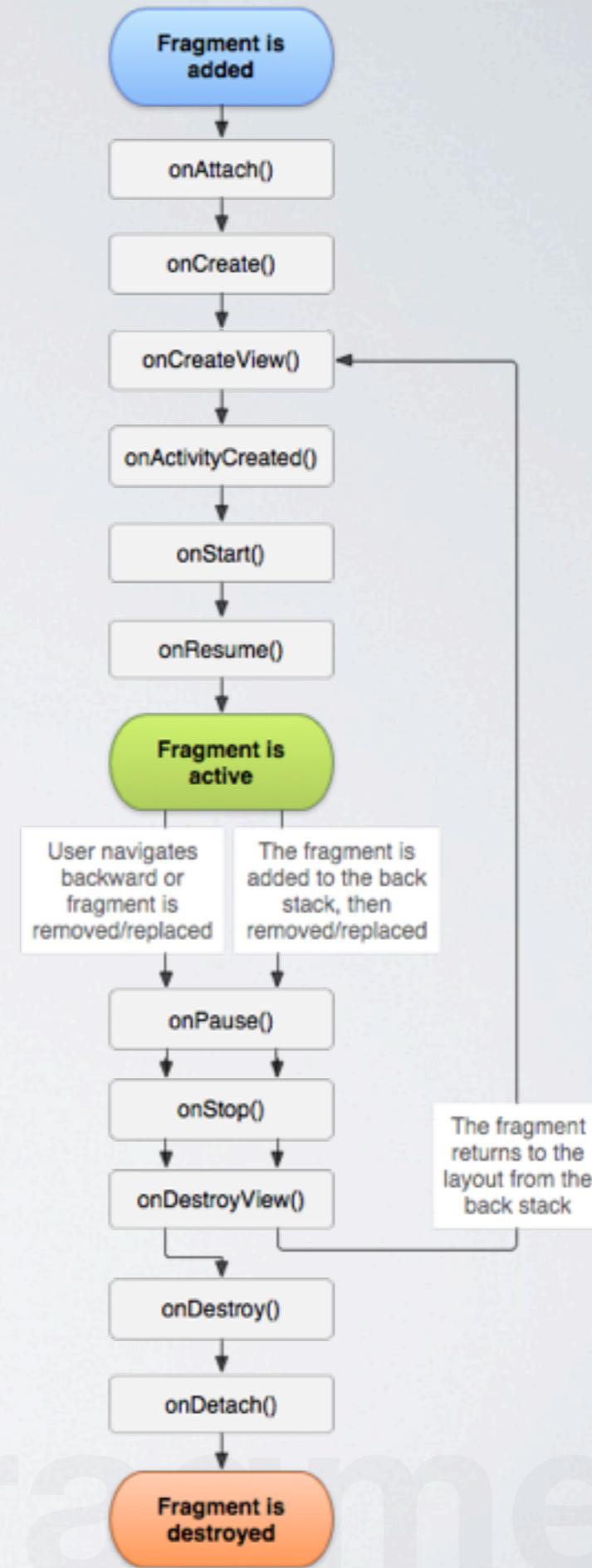
- プロセスが直接殺される時



Activity & Fragment

# Fragment

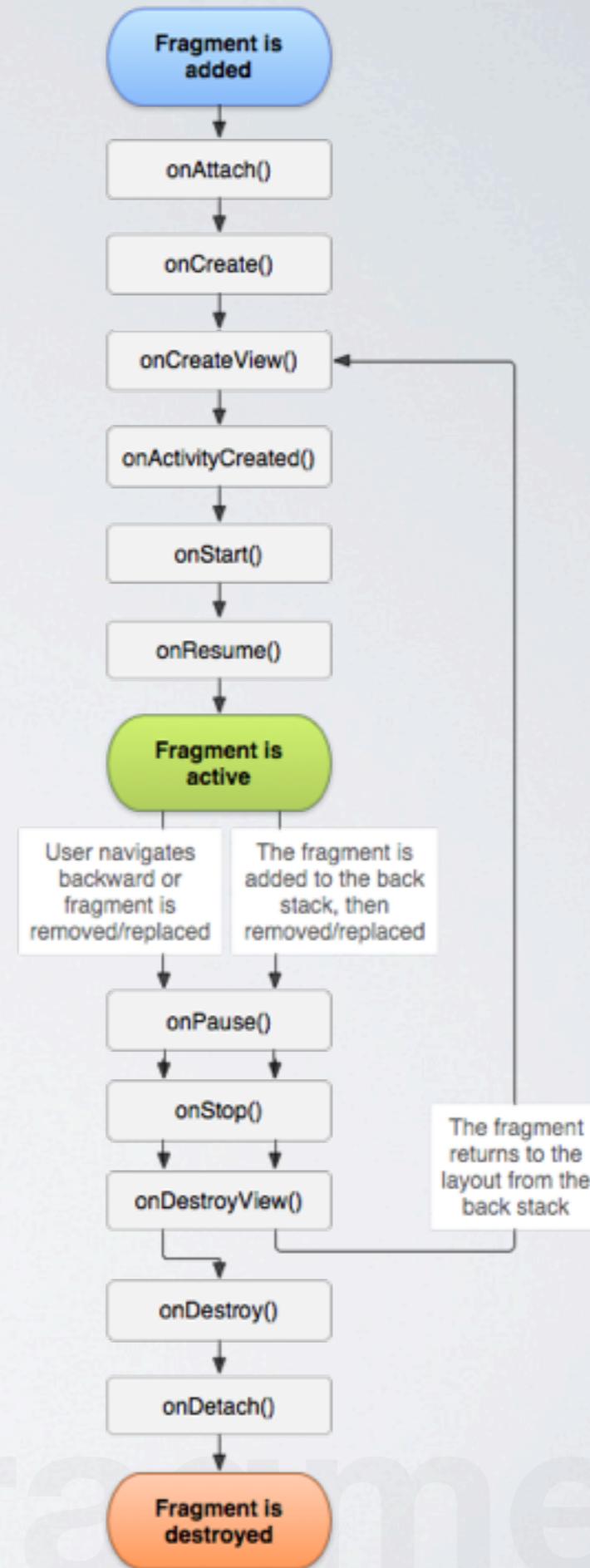
- Activityを構成する部品
- ライフサイクルがある
  - Activityが生成されたり
  - Activityが殺されたり
  - Activityから切り離されたり



Activity & Fragment

# Fragment

- onAttach
  - Activityに割当てられた時
    - ActivityにCallbackしたい時に  
Callback用オブジェクトを保持



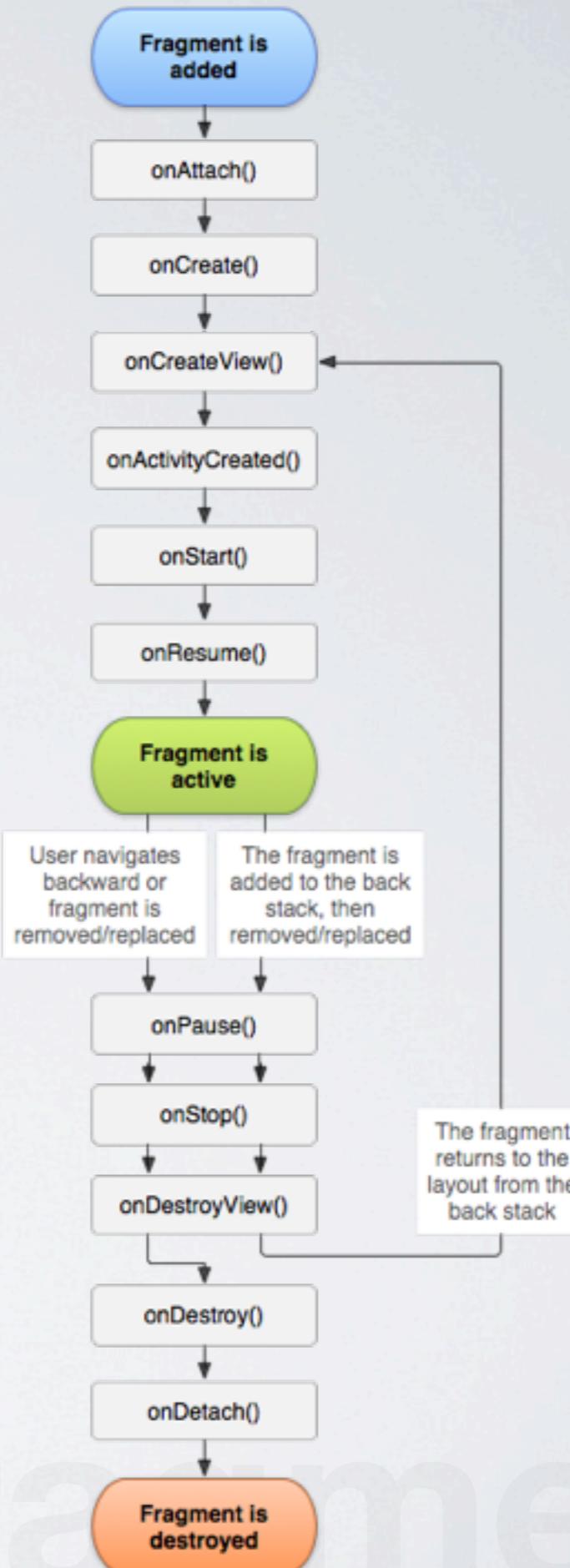
Activity & Fragment

# Fragment

- onCreate

- Fragmentの初期化

- ActivityのonCreateと同じフェーズ



Activity & Fragment

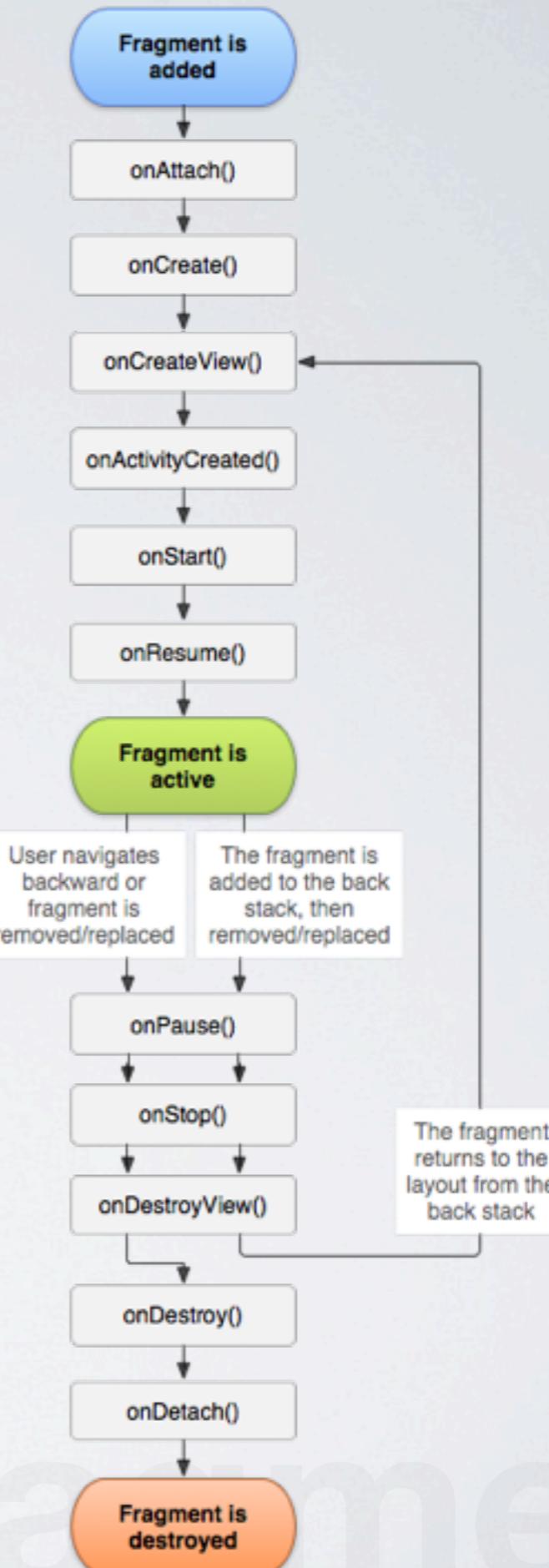
# Fragment

- onCreateView

- Fragmentのレイアウトを決める

- レイアウトを読み込む

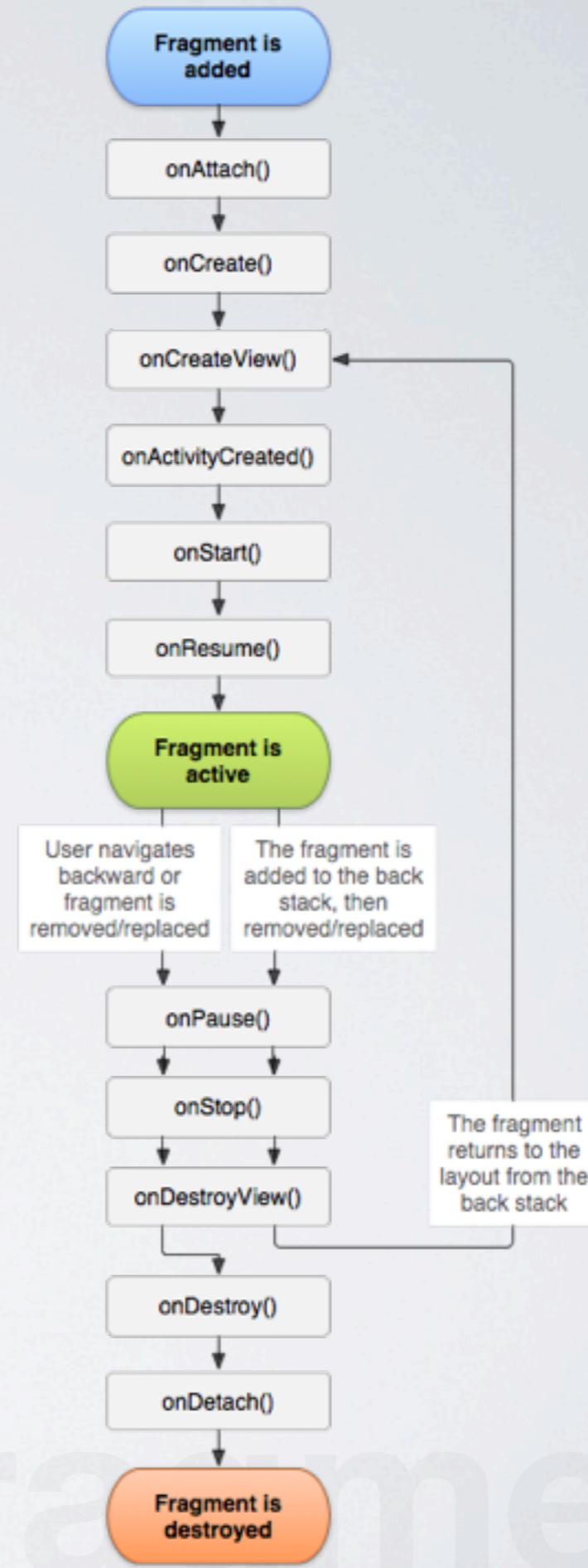
- 画面表示をする前準備



Activity & Fragment

# Fragment

- onActivityCreated
  - Activity#onCreateの後
    - あああ



Activity & Fragment

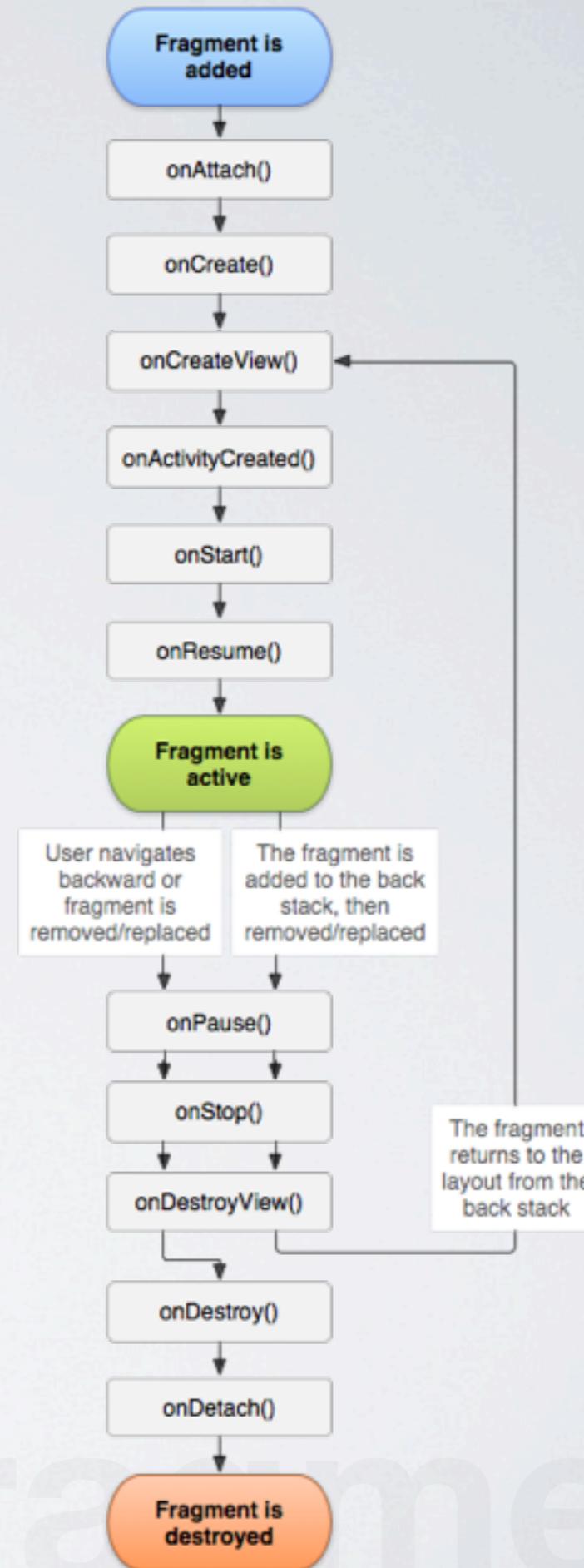
# Fragment

- onStart

- Activity#onStartの後

- Fragmentが見える状態になる

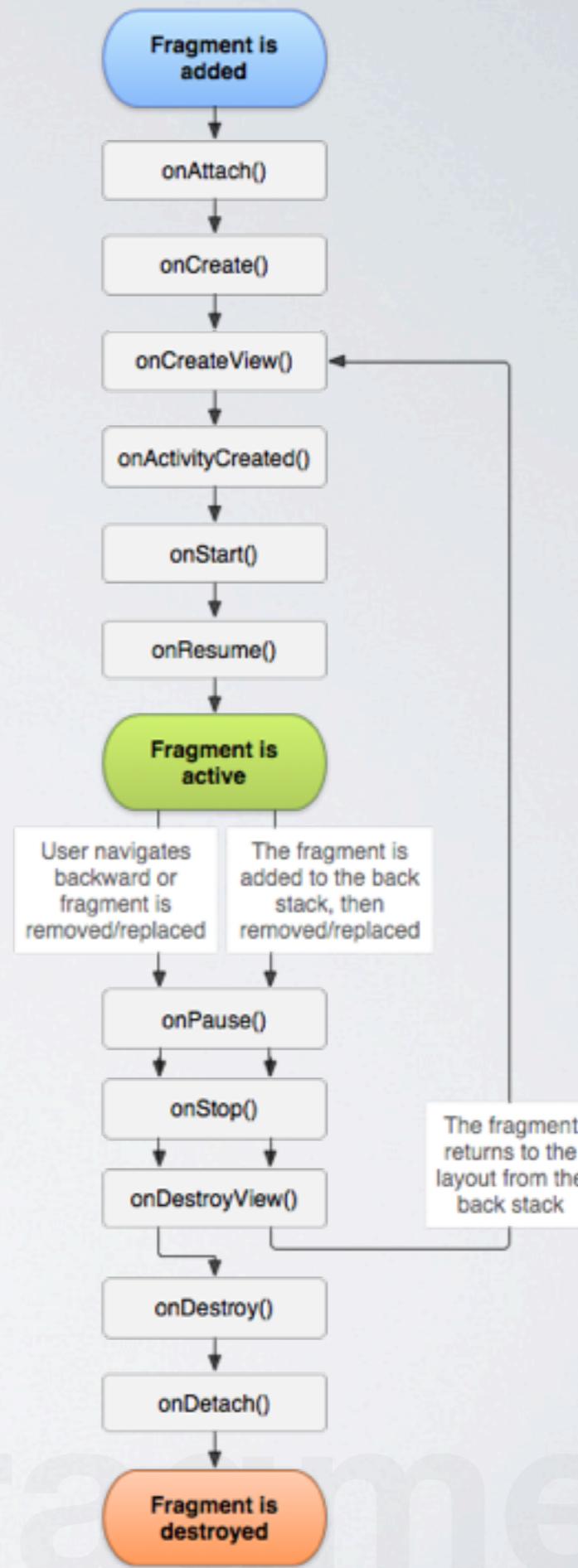
- インタラクションの前準備として  
リソースの割当てをする



Activity & Fragment

# Fragment

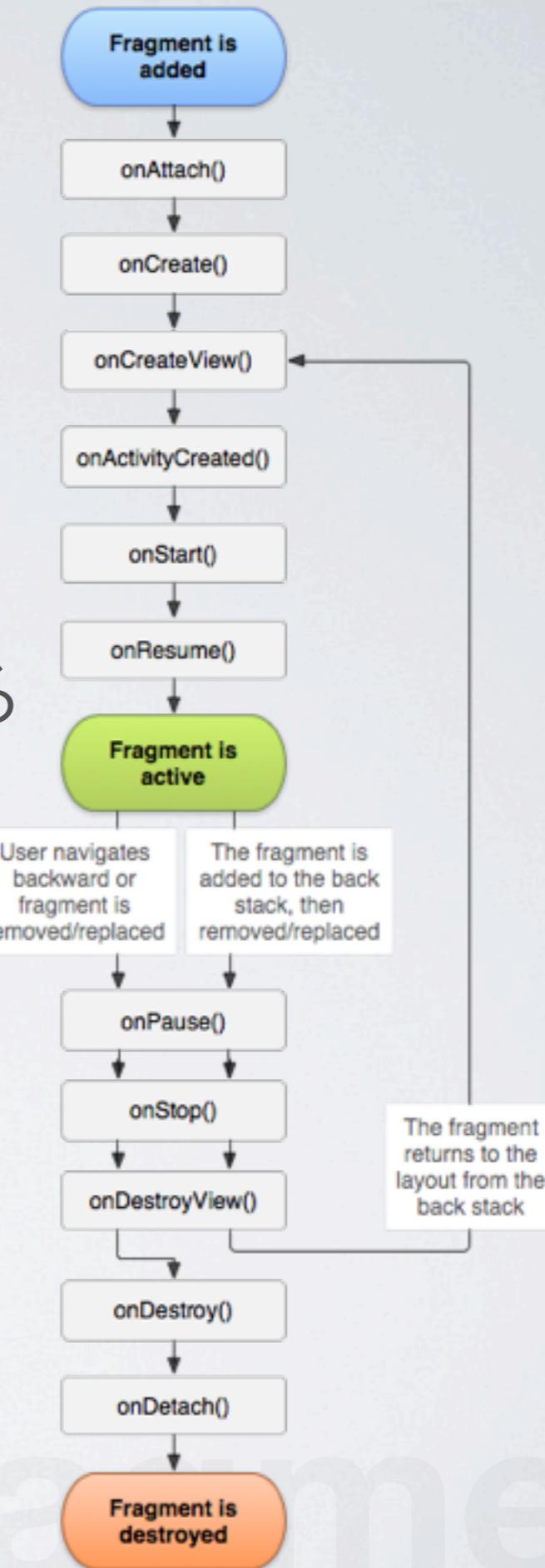
- onResume
  - Activity#onResumeの後
    - Fragmentが操作出来る状態になる



Activity & Fragment

# Fragment

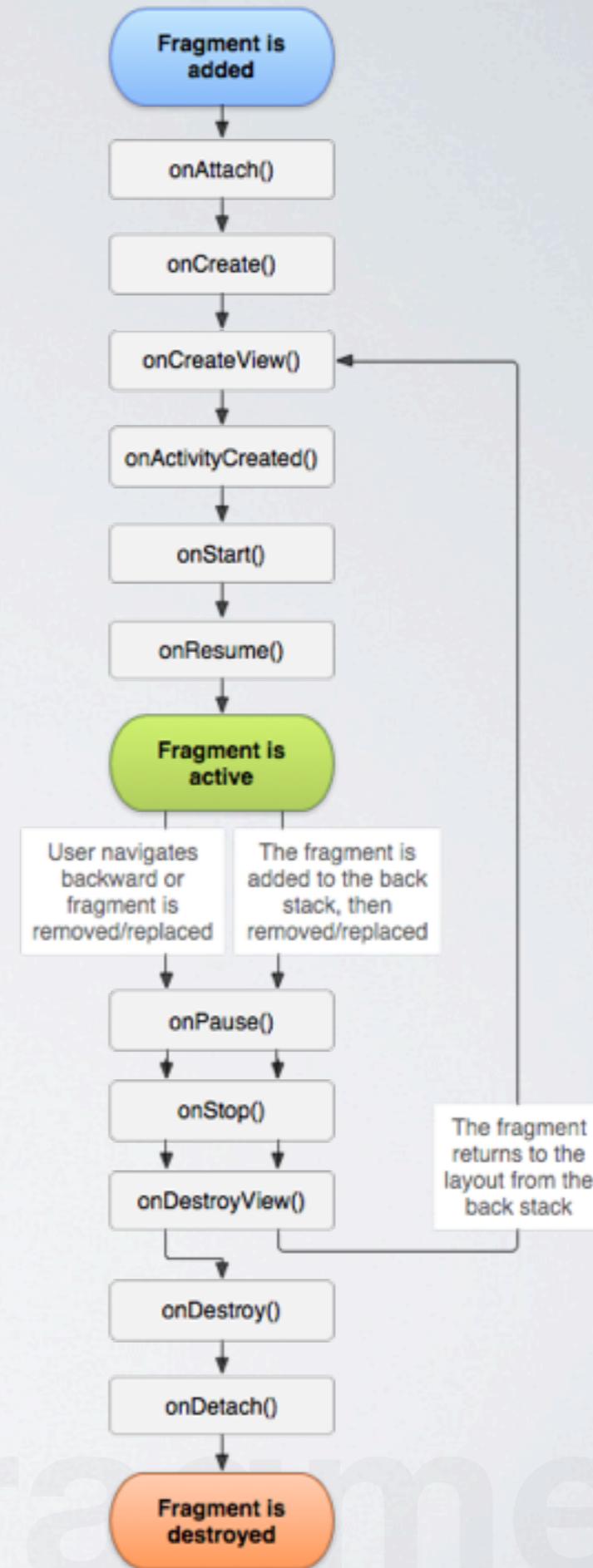
- onPause
  - Activity#onPauseの後
    - Fragmentが操作出来ない状態になる



Activity & Fragment

# Fragment

- onStop
  - Activity#onStopの後
    - Fragmentが見えない状態になる

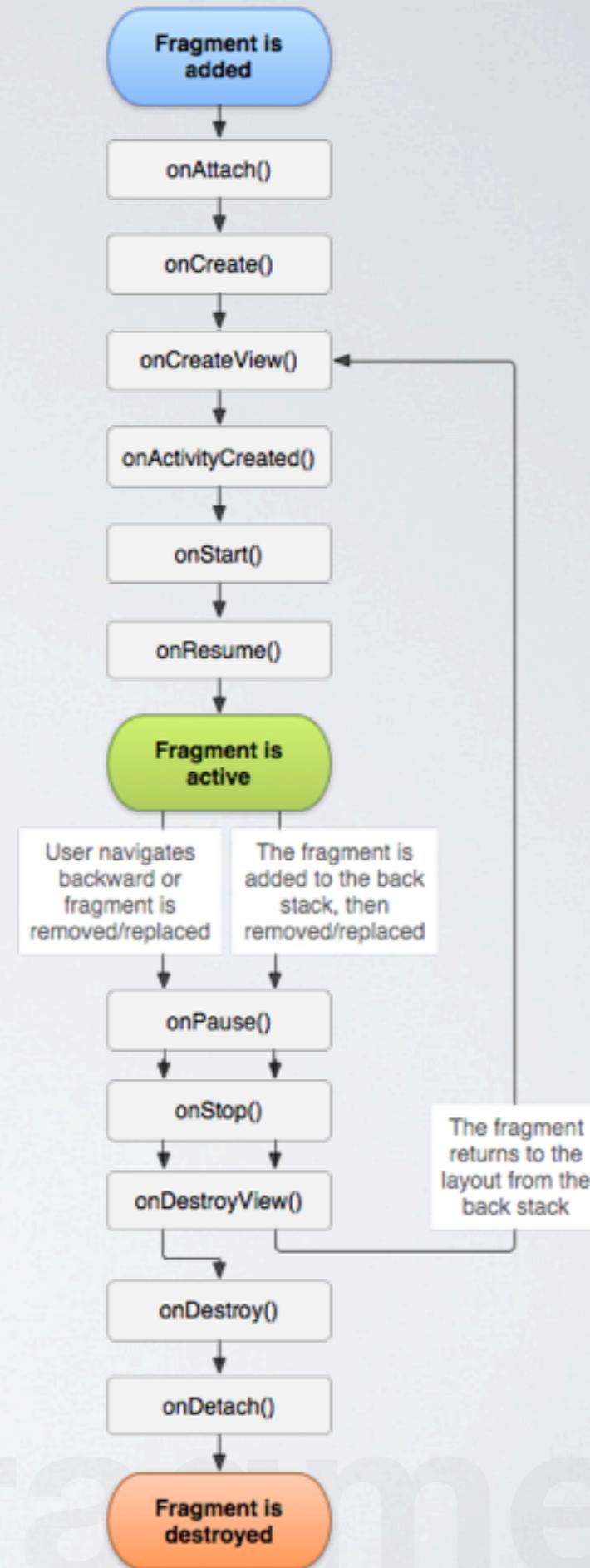


Activity & Fragment

# Fragment

- `onDestroyView`

- Viewのリソースを開放する
  - Fragmentが見えない状態になる
  - 開放してあげないとリークになる

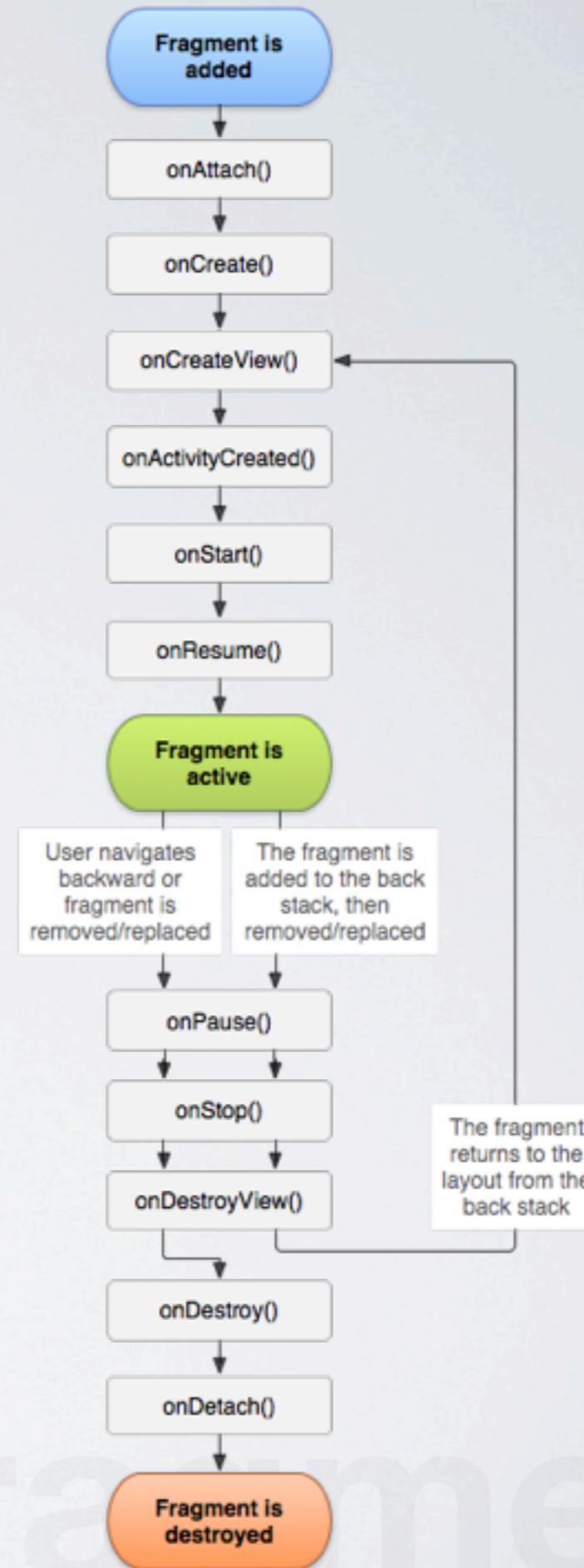


Activity & Fragment

# Fragment

- **onDestroy**

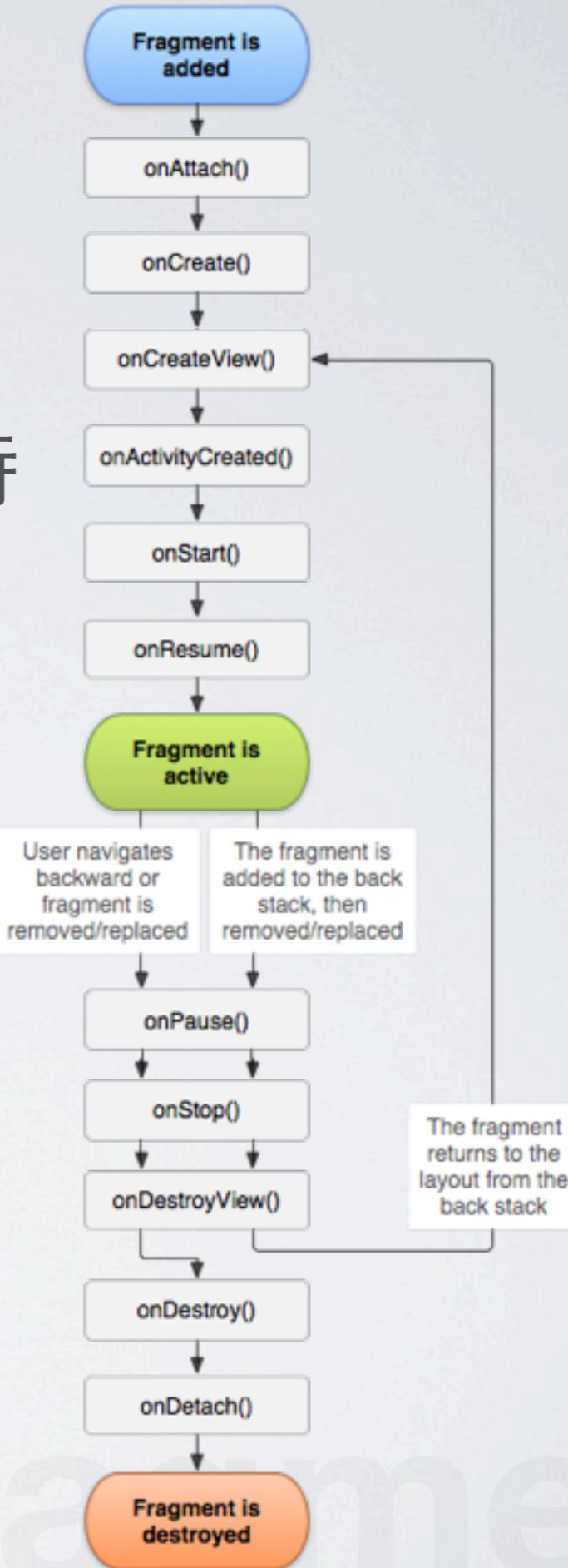
- Fragmentが死ぬ時
  - 破棄されてよい状態に
  - スレッドが動いていたら止める
- 呼ばれない時もある



Activity & Fragment

# Fragment

- onDetach
- FragmentがActivityから離れる時
  - もう要らない子になる(はず)なのでこのあとGCされる



Activity & Fragment