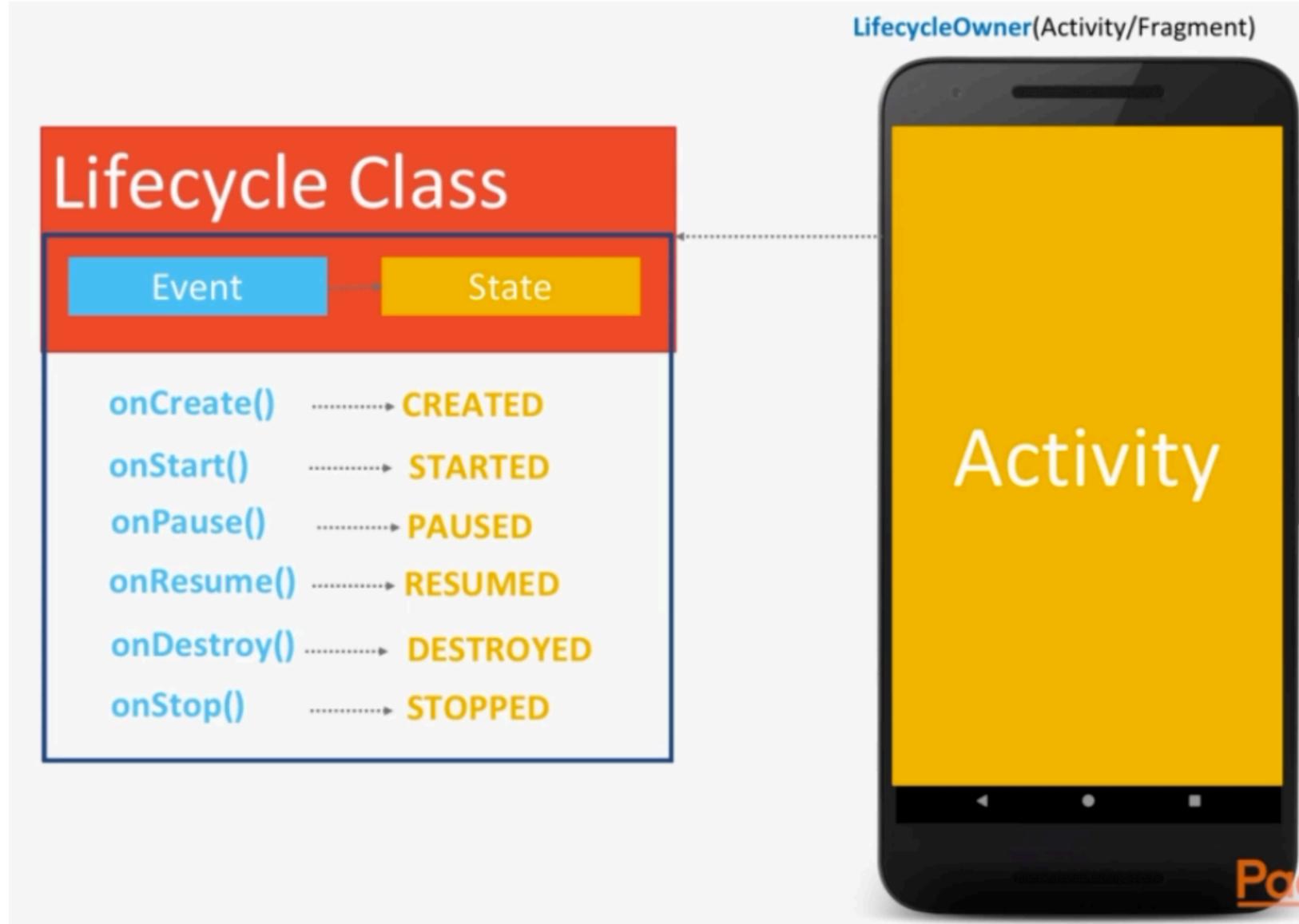
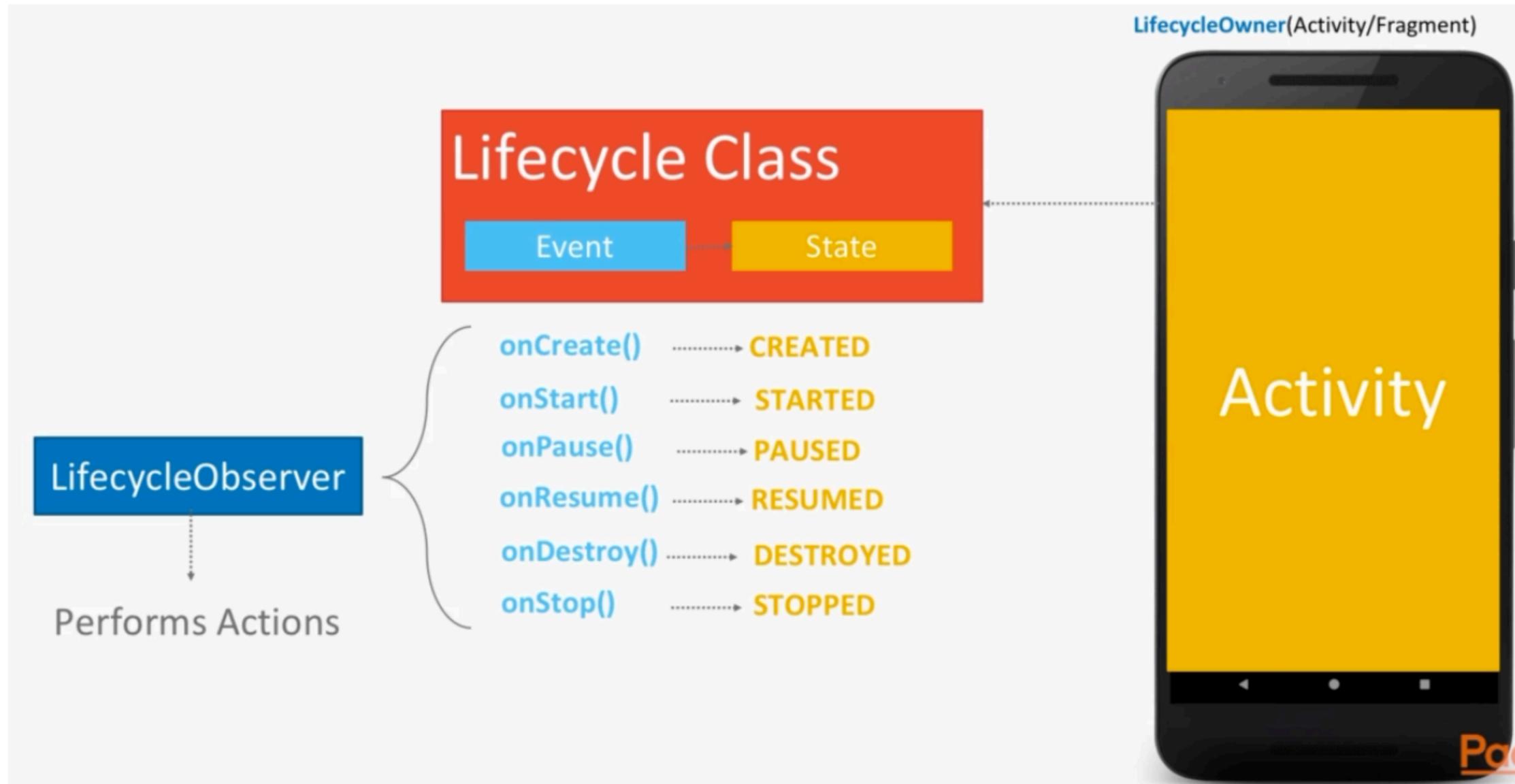


# Lifecycle-Aware Components, ViewModel, LiveData

Jetpack Architecture





# Documentation

- [OVERVIEW](#)
- [GUIDES](#)
- [REFERENCE](#)
- [SAMPLES](#)
- [DESIGN & QUALITY](#)

## App Basics

- Introduction
- ▶ Build your first app
- App fundamentals
- ▶ App resources
- ▶ App manifest file
- ▶ App permissions

## Devices

- ▶ Device compatibility
- ▶ Wear
- ▶ Android TV
- ▶ Android Auto
- ▶ Android Things
- ▶ Chrome OS devices

## Core topics

- ▶ Activities
- ▼ Architecture Components

# Handling Lifecycles with Lifecycle-Aware Components



Part of [Android Jetpack](#).

Lifecycle-aware components perform actions in response to a change in the lifecycle status of another component, such as activities and fragments. These components help you produce better-organized, and often lighter-weight code, that is easier to maintain.

A common pattern is to implement the actions of the dependent components in the lifecycle methods of activities and fragments. However, this pattern leads to a poor organization of the code and to the proliferation of errors. By using lifecycle-aware components, you can move the code of dependent components out of the lifecycle methods and into the components themselves.

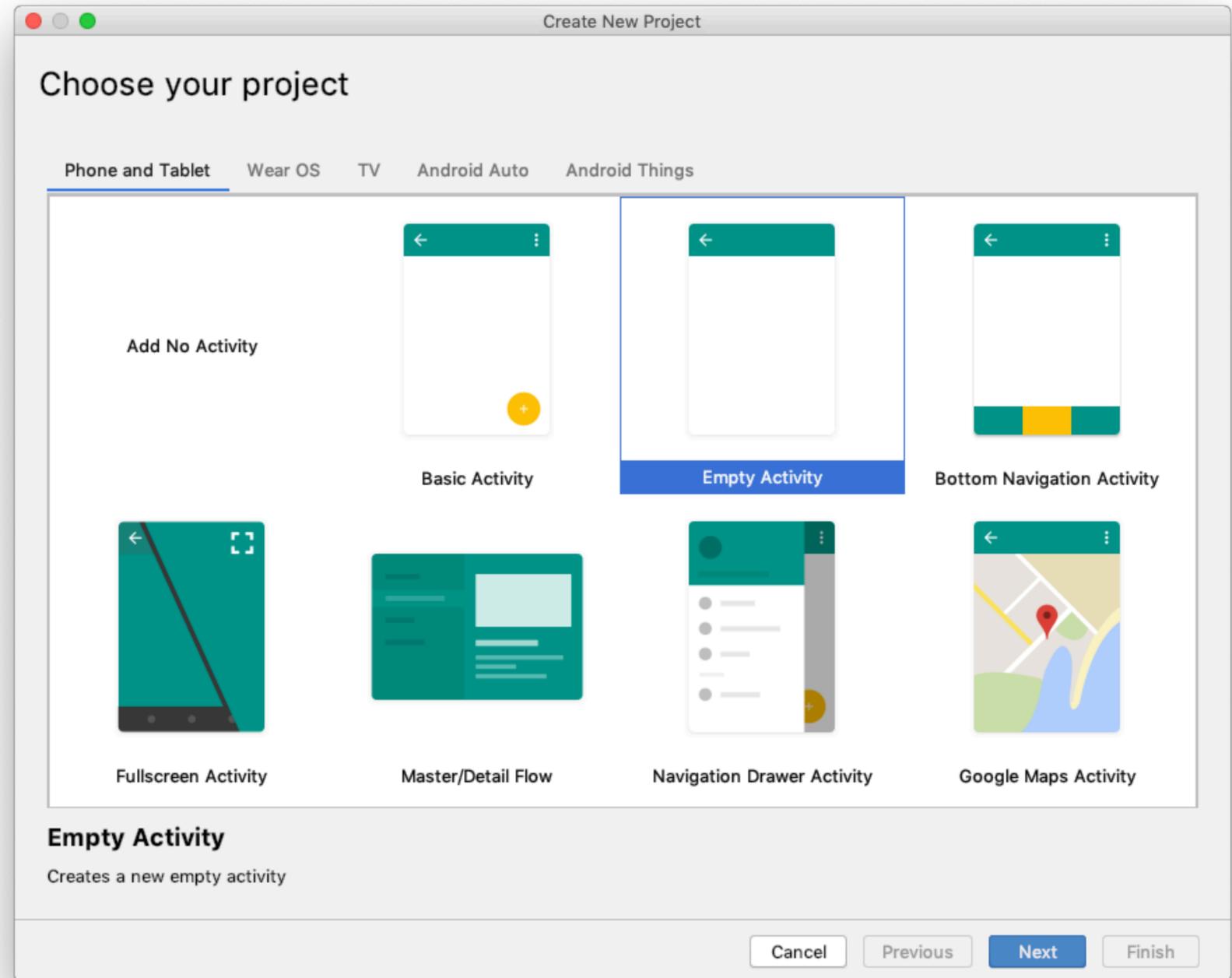
The `android.arch.lifecycle` package provides classes and interfaces that let you build *lifecycle-aware* components —which are components that can automatically adjust their behavior based on the current lifecycle state of an activity or fragment.

★ **Note:** To import `android.arch.lifecycle` into your Android project, see the instructions for declaring dependencies in the [Lifecycle release notes](#).

## Contents

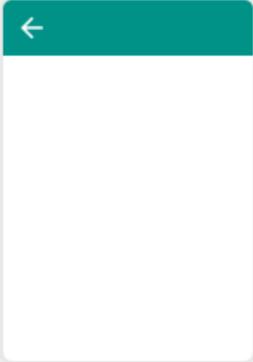
- Lifecycle
- LifecycleOwner
  - Implementing a custom LifecycleOwner
- Best practices for lifecycle-aware components
- Use cases for lifecycle-aware components
- Handling on stop events
- Additional resources

# Lifecycle-Aware Components



Create New Project

## Configure your project



Empty Activity

Creates a new empty activity

Name  
LifecycleAware

Package name  
at.htl.lifecycleaware

Save location  
/Users/stuetz/work/jetpack.packt/projects/LifecycleAware

Language  
Kotlin

Minimum API level  
API 23: Android 6.0 (Marshmallow)

**i** Your app will run on approximately **62.6%** of devices.  
[Help me choose](#)

This project will support instant apps

Use AndroidX artifacts

Cancel Previous Next **Finish**

The screenshot shows an IDE window for an Android project named "LifecycleAware". The main editor displays the Kotlin code for MainActivity.kt:

```
1 package at.htl.lifecycleaware
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5
6 class MainActivity : AppCompatActivity() {
7
8     override fun onCreate(savedInstanceState: Bundle?) {
9         super.onCreate(savedInstanceState)
10        setContentView(R.layout.activity_main)
11    }
12 }
13
```

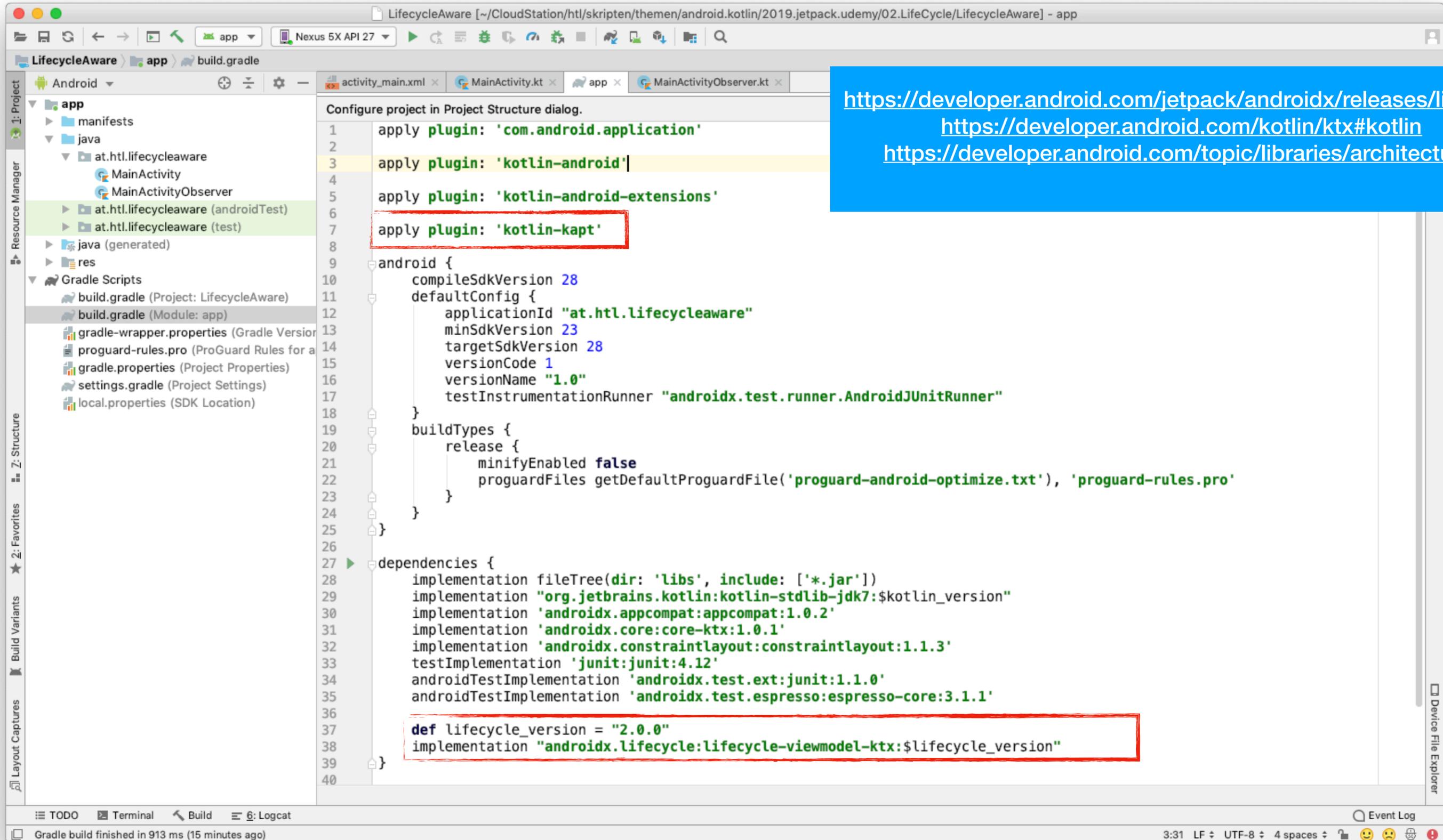
The left sidebar shows the Project Manager with the following structure:

- app
  - manifests
  - java
    - at.htl.lifecycleaware
      - MainActivity
      - at.htl.lifecycleaware (androidTest)
      - at.htl.lifecycleaware (test)
    - java (generated)
    - res
    - Gradle Scripts

The bottom panel shows the Build Output window with the following log:

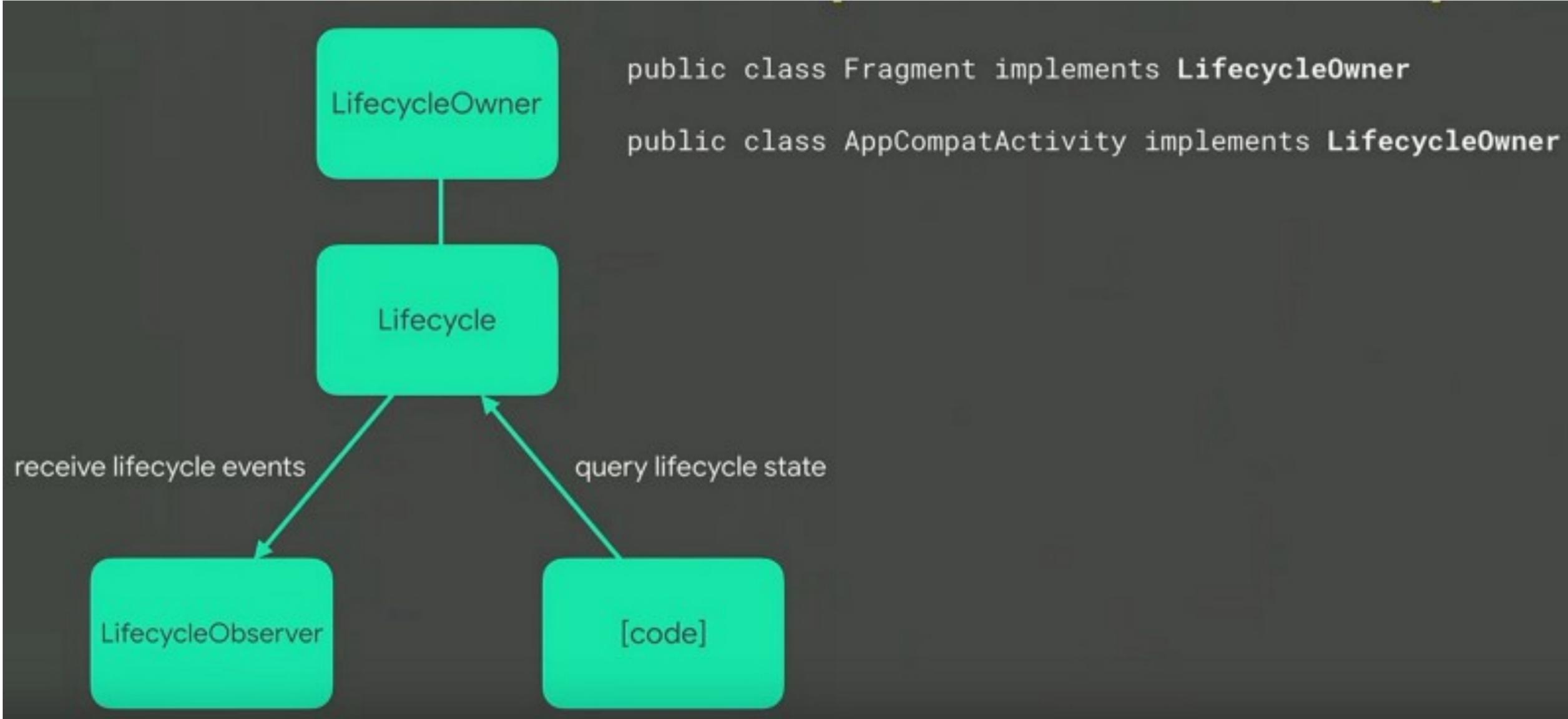
```
Build: Build Output x Sync x
Build: completed successfully at 2019-02-17 20:33 841 ms
  Run build /Users/stuetz/CloudStation/htl/skripten/themen/android.kotlin/2019.jetpack.udemy/02.LifeCycle/LifecycleAware 529 ms
    Load build 5 ms
    Configure build 293 ms
    Calculate task graph 61 ms
    Run tasks 169 ms
```

The status bar at the bottom indicates "Gradle build finished in 856 ms (moments ago)" and "13:1 LF UTF-8 4 spaces".

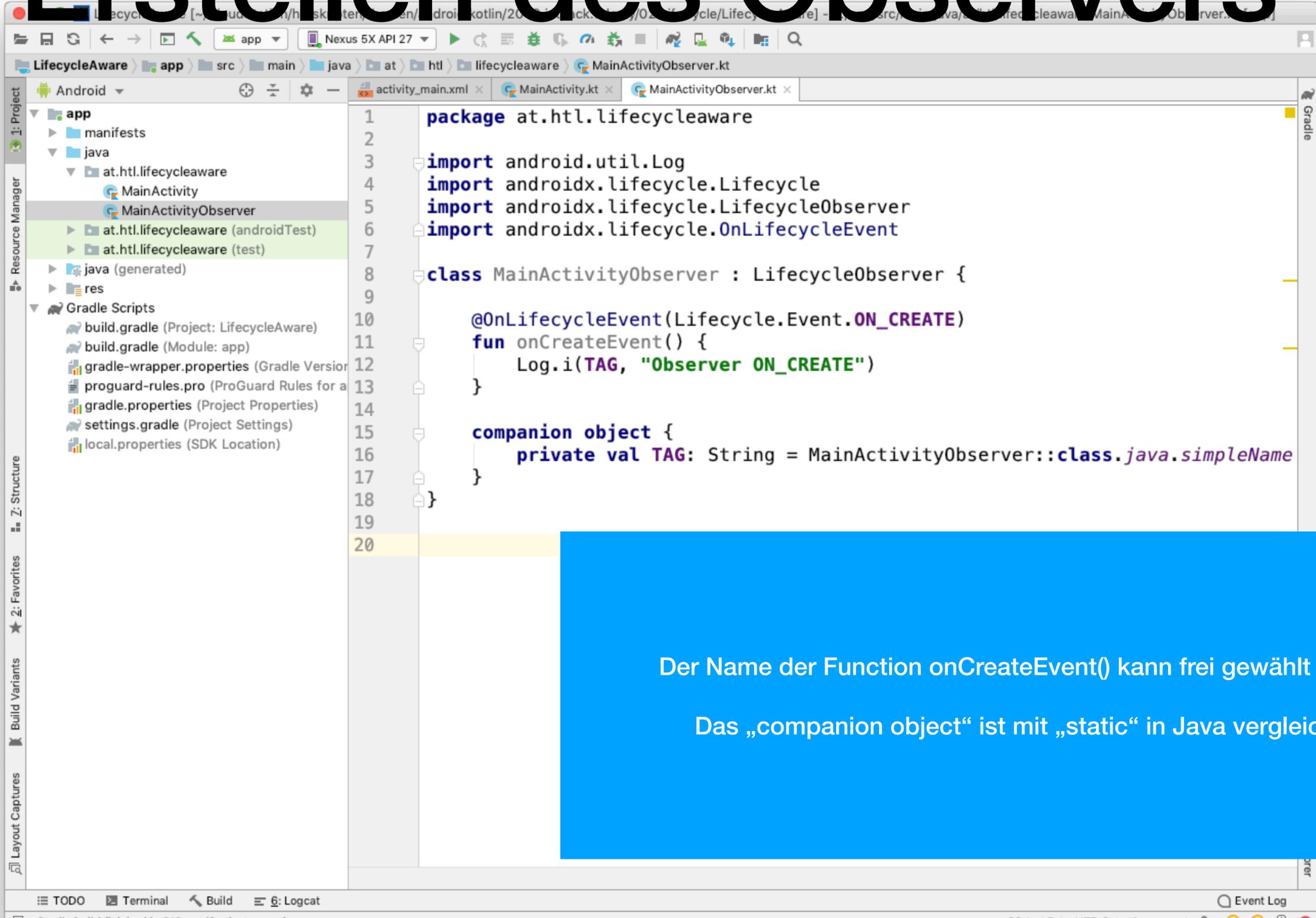


<https://developer.android.com/jetpack/androidx/releases/lifecycle>  
<https://developer.android.com/kotlin/ktx#kotlin>  
<https://developer.android.com/topic/libraries/architecture/>

<https://developer.android.com/jetpack/androidx/releases/lifecycle>



# Erstellen des Observers



```
1 package at.htl.lifecycleaware
2
3 import android.util.Log
4 import androidx.lifecycle.Lifecycle
5 import androidx.lifecycle.LifecycleObserver
6 import androidx.lifecycle.OnLifecycleEvent
7
8 class MainActivityObserver : LifecycleObserver {
9
10     @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
11     fun onCreateEvent() {
12         Log.i(TAG, "Observer ON_CREATE")
13     }
14
15     companion object {
16         private val TAG: String = MainActivityObserver::class.java.simpleName
17     }
18 }
19
20
```

Der Name der Function onCreateEvent() kann frei gewählt werden.

Das „companion object“ ist mit „static“ in Java vergleichbar

# Registrieren des Observers beim Owner

```
1 package at.htl.lifecycleaware
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import android.util.Log
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13         Log.i(TAG, "attached Observer to Owner - onCreate")
14         lifecycle.addObserver(MainActivityObserver())
15     }
16
17     companion object {
18         private val TAG: String = MainActivity::class.java.simpleName
19     }
20 }
21
```

Die Activity ist der Lifecycle-Owner

Log.i(TAG, "attached Observer to Owner - onCreate")  
lifecycle.addObserver(MainActivityObserver())

companion object {  
private val TAG: String = MainActivity::class.java.simpleName  
}

The screenshot shows an IDE window for a project named "LifecycleAware". The main editor displays the following Kotlin code for MainActivity.kt:

```
1 import ...
2
3
4
5
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13         Log.i(TAG, "attached Observer to Owner - onCreate")
14         lifecycle.addObserver(MainActivityObserver())
15     }
16
17     companion object {
18         private val TAG: String = MainActivity::class.java.simpleName
19     }
20 }
```

The Logcat window at the bottom shows the following output, with a search filter "Main" applied:

```
2019-02-17 21:14:17.612 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.612 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.614 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.615 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.616 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.618 3968-3968/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:14:17.688 3968-3968/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:14:17.707 3968-3968/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
```

Annotations in the image:

- A blue box with the text "Filtern Sie die Ausgaben mit „Main“" points to the search filter "Main" in the Logcat window.
- Another blue box with the text "Hier sieht man das Ergebnis" points to the log output line "2019-02-17 21:14:17.688 3968-3968/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate".

On the right, a smartphone displays the app's interface with a green header "LifecycleAware" and the text "Hello World!" on a white background.

The screenshot shows an IDE window with the following content:

- Project Structure:** A tree view on the left showing the project hierarchy: `app` (containing `manifests`, `java`, `res`, and `Gradle Scripts`), `at.htl.lifecycleaware` (containing `MainActivity` and `MainActivityObserver`), and test variants.
- Main Activity Code:** The main editor displays the following Kotlin code for `MainActivity`:

```
1 package at.htl.lifecycleaware
2
3 import ...
4
5
6
7 class MainActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {...}
10
11
12
13
14
15
16
17     override fun onStart() {
18         super.onStart()
19         Log.i(TAG, "Owner onStart")
20     }
21
22     override fun onPause() {
23         super.onPause()
24         Log.i(TAG, "Owner onPause")
25     }
26
27     override fun onResume() {
28         super.onResume()
29         Log.i(TAG, "Owner onResume")
30     }
31
32     override fun onDestroy() {
33         super.onDestroy()
34         Log.i(TAG, "Owner onDestroy")
35     }
36
37     override fun onStop() {
38         super.onStop()
39         Log.i(TAG, "Owner onStop")
40     }
41
42     companion object {
43         private val TAG: String = MainActivity::class.java.simpleName
44     }
45 }
```
- Override Members Dialog:** A dialog box titled "Override Members" is open, showing a search for `onStart`. The list of methods includes `onStart(): Unit`, which is highlighted. Other methods include `setSupportProgressBarIndeterminate`, `getDelegate`, `onContentChanged`, `onWindowStartingSupportActionMode`, `startSupportActionMode`, `onTitleChanged`, `onSaveInstanceState`, and `openOptionsMenu`. The dialog has buttons for "Copy JavaDoc", "Select None", "Cancel", and "OK".
- Annotations:** A green box highlights the text "Override Methods via ^O (Ctrl+O for Win/Linux)". A blue box contains the text: "Mit Strg-O kann man das „Override Members“-Fenster öffnen. Nun gibt man zB „onStart“ ein. Anschließend navigiert man mit der Pfeiltaste ↓ bis man die korrekte Methode findet."

The screenshot shows an IDE window for a project named "LifecycleAware". The main editor displays the Kotlin file "MainActivityObserver.kt". The code defines a class "MainActivityObserver" that implements the "LifecycleObserver" interface. The class contains five lifecycle event methods, each logging a message when triggered:

```
7 /
8 class MainActivityObserver : LifecycleObserver {
9
10     @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
11     fun onCreateEvent() {
12         Log.i(TAG, "Observer ON_CREATE")
13     }
14
15     @OnLifecycleEvent(Lifecycle.Event.ON_START)
16     fun onStartEvent() {
17         Log.i(TAG, "Observer ON_START")
18     }
19
20     @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
21     fun onResumeEvent() {
22         Log.i(TAG, "Observer ON_RESUME")
23     }
24
25     @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
26     fun onPauseEvent() {
27         Log.i(TAG, "Observer ON_PAUSE")
28     }
29
30     @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
31     fun onStopEvent() {
32         Log.i(TAG, "Observer ON_STOP")
33     }
34
35     @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
36     fun onDestroyEvent() {
37         Log.i(TAG, "Observer ON_DESTROY")
38     }
39
40     companion object {
```

The methods from line 15 to 38 are enclosed in a red rectangular box. The IDE interface includes a Project Explorer on the left showing the file structure, a toolbar at the top, and a status bar at the bottom with various tool icons and a system tray.

LifecycleAware [~/CloudStation/htl/skripten/themen/android.kotlin/2019.jetpack.udemy/02.LifeCycle/LifecycleAware] - .../app/src/main/java/at/htl/lifecycleaware/MainActivityObserver.kt [app]

activity\_main.xml x MainActivity.kt x MainActivityObserver.kt x

38 } MainActivityObserver

Logcat

Emulator Nexus\_5X\_API\_27 And at.htl.lifecycleaware (4664) Verbose Main Regex Show only selected application

```
2019-02-17 21:44:53.652 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.654 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.654 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.702 4664-4664/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:44:53.706 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
2019-02-17 21:44:53.710 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStart
2019-02-17 21:44:53.710 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_START
2019-02-17 21:44:53.713 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onResume
2019-02-17 21:44:53.713 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_RESUME
```

Die App wird initialisiert. Die Callback-Methoden werden das erste Mal aufgerufen

Install successful

Install successful (moments ago) 39:1 LF UTF-8 4 spaces



The screenshot shows the Android Studio interface with the Logcat window open. The Logcat window displays a series of log messages for the application `at.htl.lifecycleaware` (PID 4664). A red box highlights a sequence of log messages indicating the activity's destruction and recreation:

```
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_PAUSE
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onPause
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_STOP
2019-02-17 21:45:32.181 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStop
2019-02-17 21:45:32.182 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_DESTROY
2019-02-17 21:45:32.182 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onDestroy
2019-02-17 21:45:32.231 4664-4664/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:45:32.231 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
2019-02-17 21:45:32.232 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStart
2019-02-17 21:45:32.232 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_START
2019-02-17 21:45:32.236 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onResume
2019-02-17 21:45:32.236 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_RESUME
```

A blue callout box contains the text: **Beim Drehen des Smartphones wird die Activity zerstört und neu erstellt**

The bottom of the IDE shows a green message: **Install successful**. The virtual device at the bottom right displays the application's UI with the text **Hello World!**

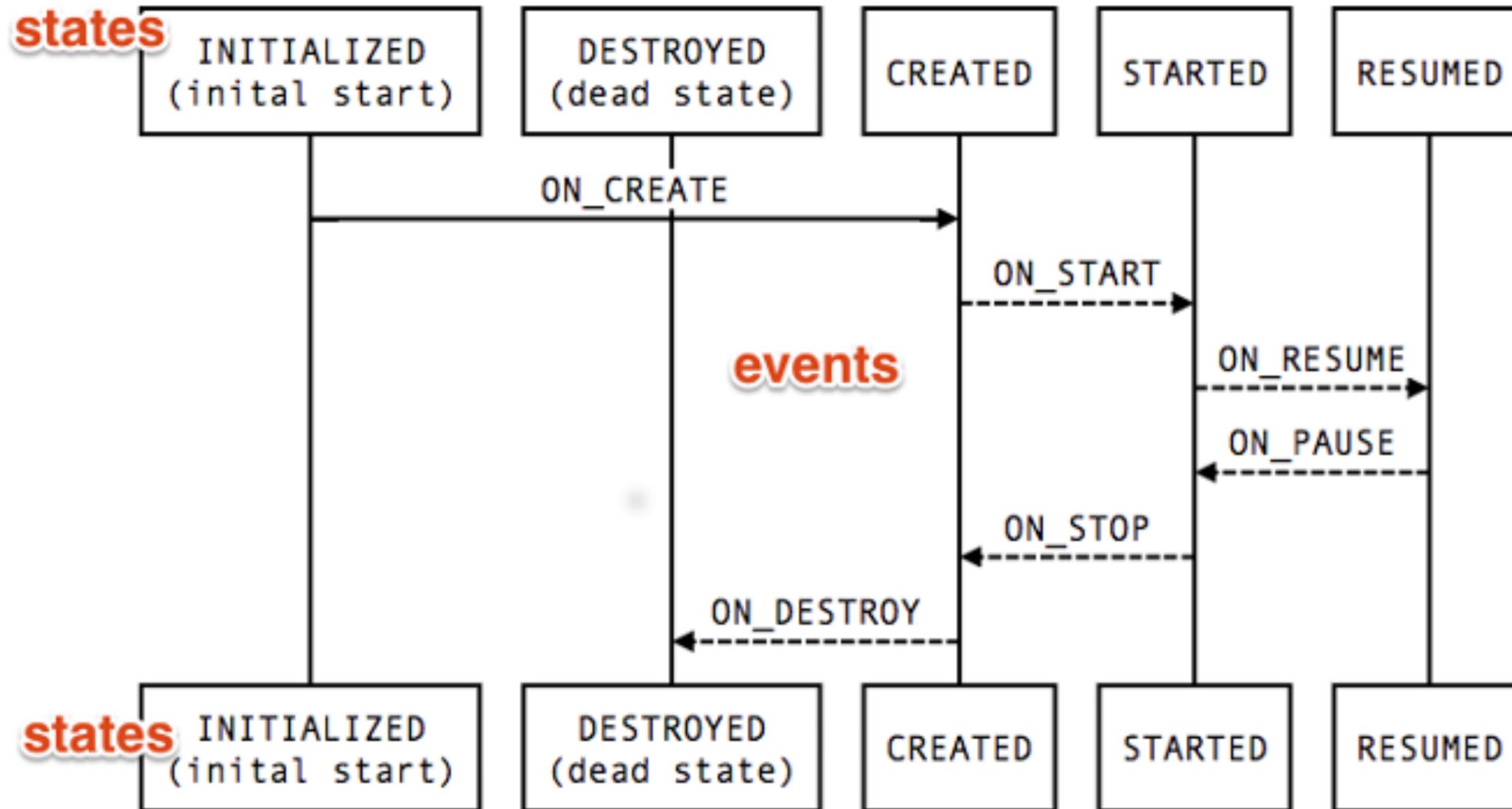
The screenshot shows the Android Studio IDE with the Logcat window open. The Logcat window displays a series of log messages for the application `at.htl.lifecycleaware (4664)`. A red box highlights a sequence of log messages starting from `2019-02-17 21:46:06.020` to `2019-02-17 21:46:06.079`. A blue callout box with the text `Funktioniert auch beim zweiten Mal` is positioned over the Logcat window. To the right of the IDE, a smartphone emulator is shown displaying the application `LifecycleAware` with the text `Hello World!` on the screen. The IDE interface includes a toolbar at the top, a project view on the left, and a status bar at the bottom.

```
2019-02-17 21:44:53.652 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.653 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.654 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.654 4664-4664/at.htl.lifecycleaware I/zygote: at void at.htl.lifecycleaware.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:11)
2019-02-17 21:44:53.702 4664-4664/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:44:53.706 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
2019-02-17 21:44:53.710 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStart
2019-02-17 21:44:53.710 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_START
2019-02-17 21:44:53.713 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onResume
2019-02-17 21:44:53.713 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_RESUME
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_PAUSE
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onPause
2019-02-17 21:45:32.178 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_STOP
2019-02-17 21:45:32.181 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStop
2019-02-17 21:45:32.182 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_DESTROY
2019-02-17 21:45:32.182 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onDestroy
2019-02-17 21:45:32.231 4664-4664/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:45:32.231 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
2019-02-17 21:45:32.232 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStart
2019-02-17 21:45:32.232 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_START
2019-02-17 21:45:32.236 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onResume
2019-02-17 21:45:32.236 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_RESUME
2019-02-17 21:46:06.020 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_PAUSE
2019-02-17 21:46:06.020 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onPause
2019-02-17 21:46:06.020 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_STOP
2019-02-17 21:46:06.029 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStop
2019-02-17 21:46:06.029 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_DESTROY
2019-02-17 21:46:06.030 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onDestroy
2019-02-17 21:46:06.073 4664-4664/at.htl.lifecycleaware I/MainActivity: attached Observer to Owner - onCreate
2019-02-17 21:46:06.074 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_CREATE
2019-02-17 21:46:06.076 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onStart
2019-02-17 21:46:06.076 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_START
2019-02-17 21:46:06.079 4664-4664/at.htl.lifecycleaware I/MainActivity: Owner onResume
2019-02-17 21:46:06.079 4664-4664/at.htl.lifecycleaware I/MainActivityObserver: Observer ON_RESUME
```

# Zusammenfassung

- Lifecycle-Aware Components reagieren auf Änderungen des Zustands anderer UI-Komponenten
  - Eintragen der Dependency und des kapt-Compilers
  - Erstellen des Observer mit seinen @OnLifecycleEvent-Methoden
  - Registrieren des Observer beim Owner (der Activity oder dem Fragment)
  - Überschreiben der On-Methoden beim Owner

# States and Events



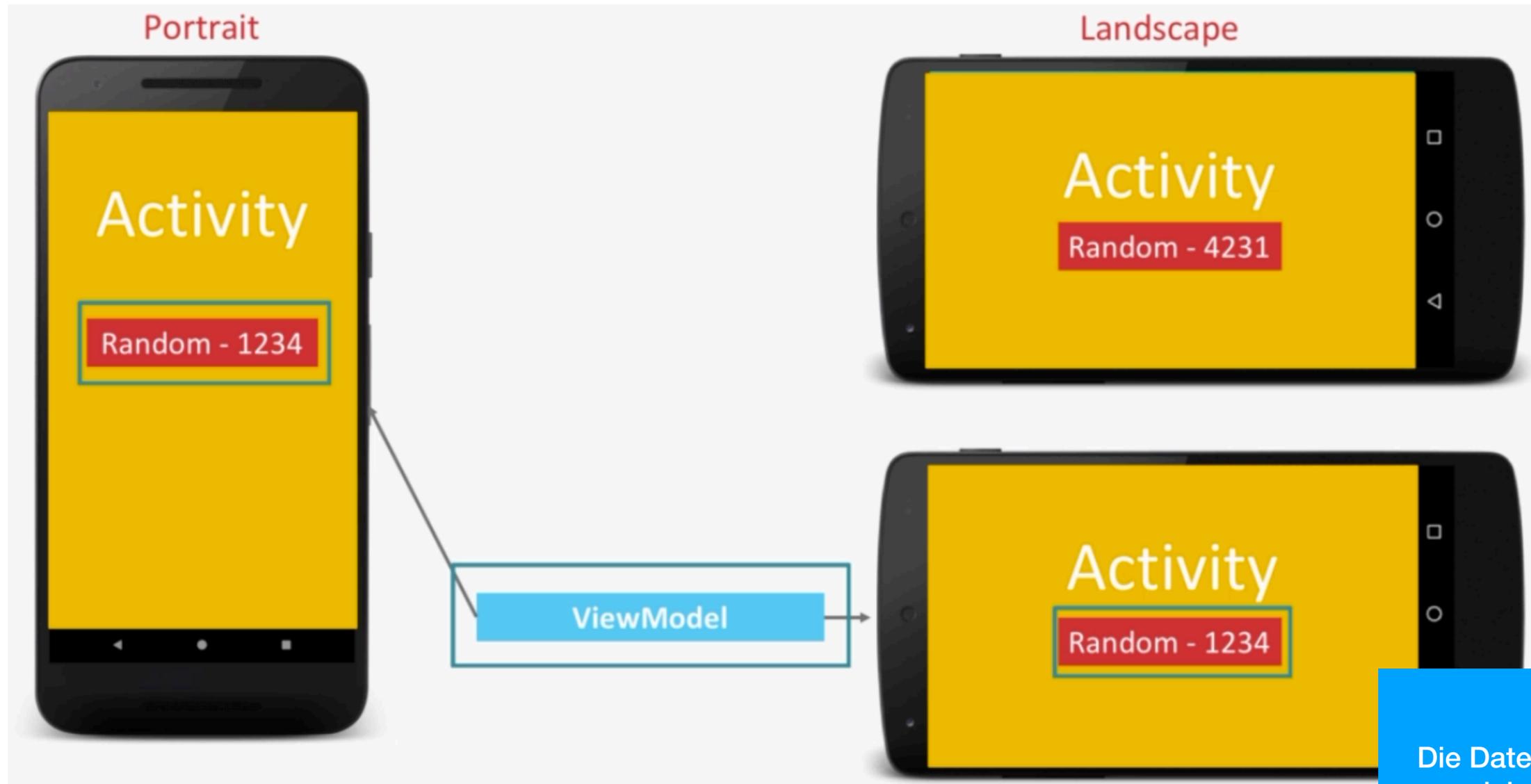
<https://developer.android.com/topic/libraries/architecture/lifecycle>

# ViewModel

# Problem

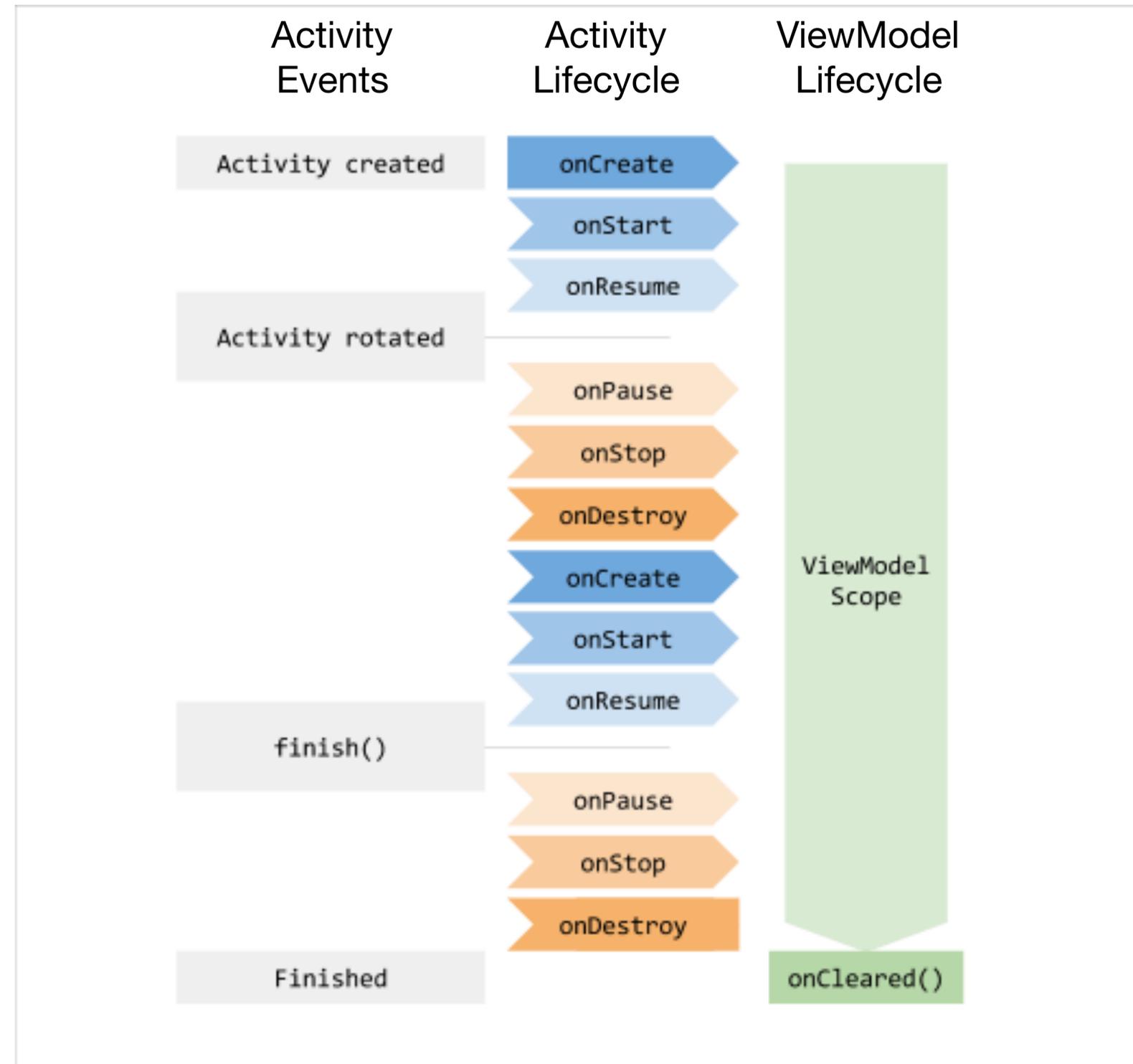


# Lösung - ViewModel

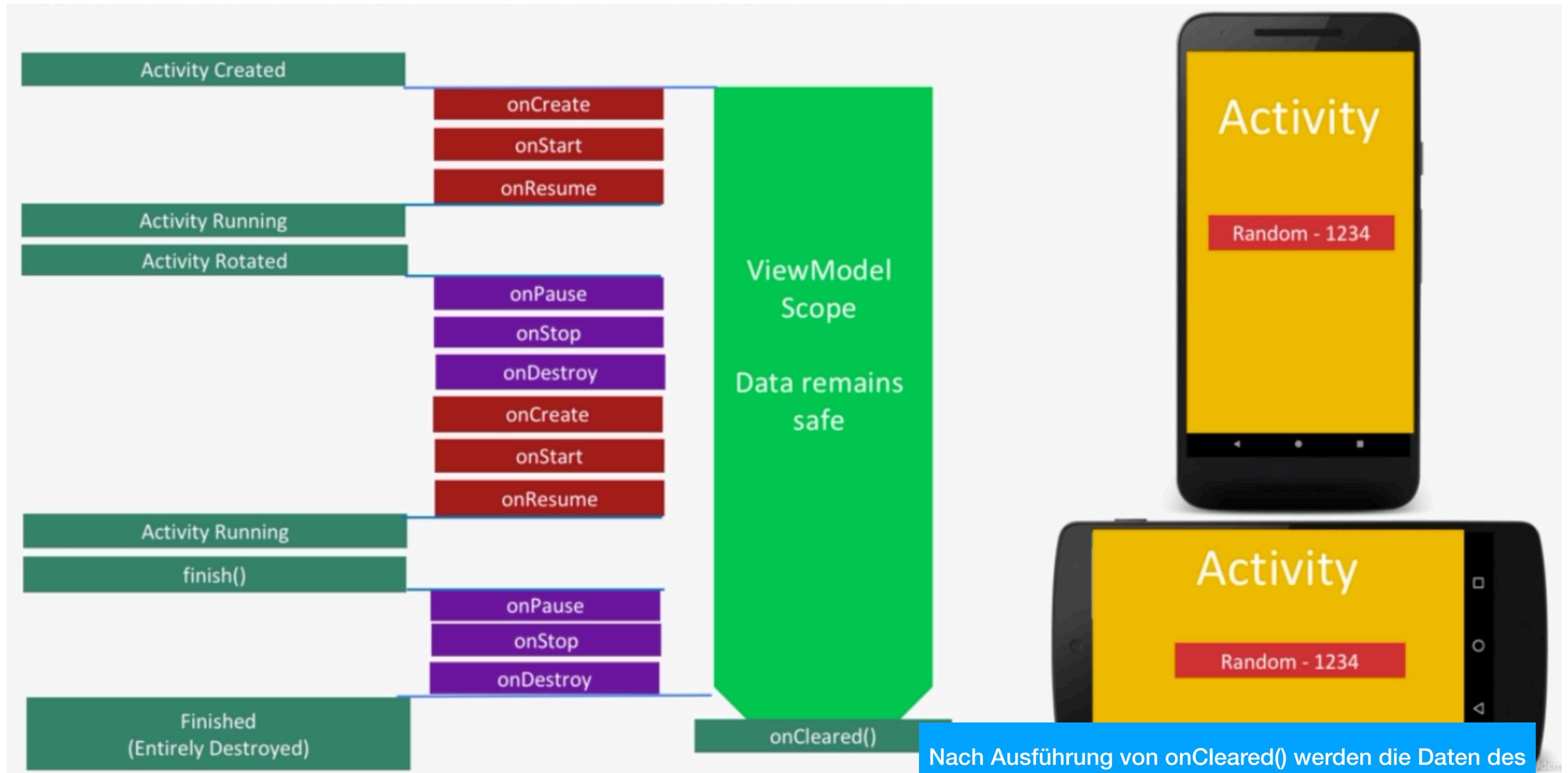


Die Daten werden im ViewModel gespeichert. Wird die neue Activity erstellt, so können diese gespeicherten Daten wiederverwendet werden.

# Lebenszyklus

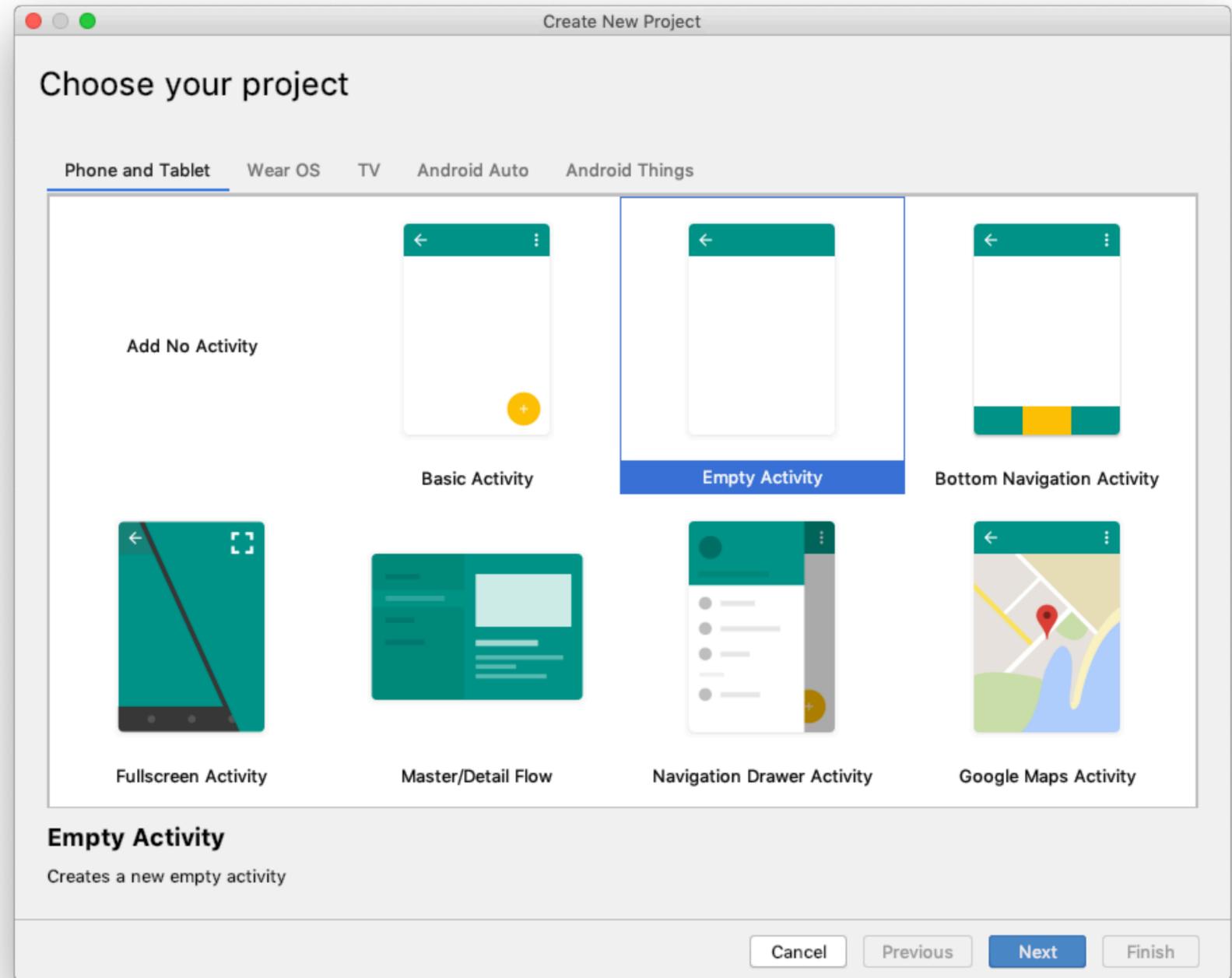


# Beispiel



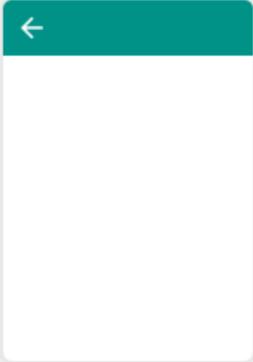
Nach Ausführung von onCleared() werden die Daten des ViewModels gelöscht

# **Variante 1: Ohne ViewModel**



Create New Project

## Configure your project



Empty Activity

Creates a new empty activity

Name  
ViewModelLiveData

Package name  
at.htl.viewmodellivedata

Save location  
ndroid.kotlin/2019.jetpack.udemy/02.LifeCycle/ViewModelLiveData

Language  
Kotlin

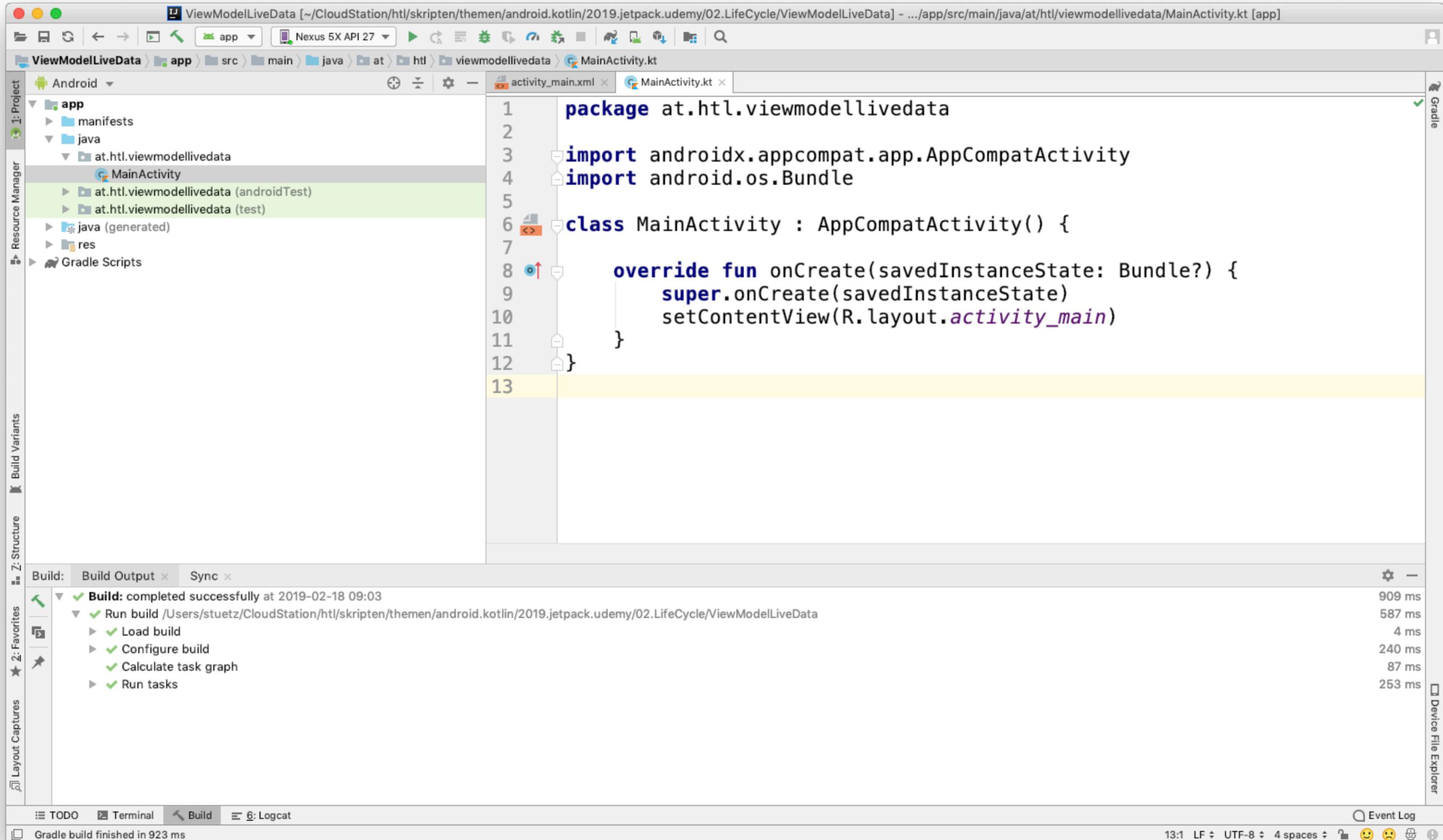
Minimum API level  
API 23: Android 6.0 (Marshmallow)

**i** Your app will run on approximately **62.6%** of devices.  
[Help me choose](#)

This project will support instant apps

Use AndroidX artifacts

Cancel Previous Next Finish



The screenshot shows the Android Studio IDE with the following components:

- Project View (Left):** Shows the project structure for 'app', including 'manifests', 'java', and 'res' folders. The 'at.htl.viewmodellivedata' package is highlighted.
- Palette (Top Left):** Lists common widgets like Button, ImageView, RecyclerView, ScrollView, and Switch.
- Design View (Center):** Displays a visual representation of the layout with a blue background and a white TextView containing 'Hello World!'. Blue dashed lines indicate constraints.
- Attributes Panel (Right):** Shows the properties for the selected TextView. The 'id' attribute is highlighted with a red box and set to 'tv\_number'. Other attributes include 'text' (Hello World!), 'layout\_width' (wrap\_content), and 'layout\_height' (wrap\_content).
- Component Tree (Bottom Left):** Shows the hierarchy: ConstraintLayout > Ab tv\_number - "Hello World!".
- Bottom Bar:** Contains tabs for Design, Text, and Logcat. A status bar at the bottom indicates 'Gradle build finished in 923 ms (54 minutes ago)'.

**Wir vergeben eine ID für die TextView**

Declared Attributes	
layout_width	wrap_content
layout_height	wrap_content
layout_constraintBottom	parent
layout_constraintLeft	parent
layout_constraintRight	parent
layout_constraintTop	parent
id	tv_number
text	Hello World!

Common Attributes	
text	Hello World!

# MainActivityDataGenerator.kt

```
1 package at.htl.viewmodellivedata
2
3 import android.util.Log
4 import kotlin.random.Random
5
6 class MainActivityDataGenerator {
7
8     private lateinit var myRandomNumber: String
9
10    fun getNumber(): String {
11        Log.i(TAG, "Get number")
12
13        if (!::myRandomNumber.isInitialized) {
14            this.createNumber()
15        }
16        return myRandomNumber
17    }
18
19    // kotlin.random.Random
20    private fun createNumber() {
21        Log.i(TAG, "Create new number")
22        myRandomNumber = "Number: " + (Random.nextInt(10 - 1) + 1)
23    }
24
25    companion object {
26        private val TAG = MainActivityDataGenerator::class.java.simpleName
27    }
28
29 }
```

New Kotlin File/Class

Name: MainActivityDataGenerator

Kind: Class

Cancel OK

`::myRandomNumber` entspricht `this::myRandomNumber`. Der `::`-Operator dient zum Zugriff auf Klassenattribute (property reference)  
<https://kotlinlang.org/docs/reference/reflection.html#property-references>  
Der `::`-Operator konvertiert auch eine Methode in einen Lambda-Ausdruck ()  
<https://stackoverflow.com/a/52463295>  
<https://kotlinlang.org/docs/reference/reflection.html#function-references>

%

```
// java.util.Random
private fun createNumber() {
    Log.i(TAG, "Create new number")
    val random = java.util.Random()
    myRandomNumber = "Number: " + (random.nextInt(10 - 1) + 1)
}
```

# :: - Operator

## Function References

When we have a named function declared like this:

```
fun isOdd(x: Int) = x % 2 != 0
```

We can easily call it directly ( `isOdd(5)` ), but we can also use it as a function type value, e.g. pass it to another function. To do this, we use the `::` operator:

```
val numbers = listOf(1, 2, 3)
println(numbers.filter(::isOdd))
```

Target platform: JVM Running on kotlin v. 1.3.21

Here `::isOdd` is a value of function type `(Int) -> Boolean`.

<https://kotlinlang.org/docs/reference/reflection.html#function-references>

## Property References

To access properties as first-class objects in Kotlin, we can also use the `::` operator:

```
val x = 1

fun main() {
    println(::x.get())
    println(::x.name)
}
```

Target platform: JVM Running on kotlin v. 1.3.21

The expression `::x` evaluates to a property object of type `KProperty<Int>`, which allows us to read its value using `get()` or retrieve the property name using the `name` property. For more information, please refer to the [docs on the KProperty class](#).

<https://kotlinlang.org/docs/reference/reflection.html#property-references>

# MainActivity.kt

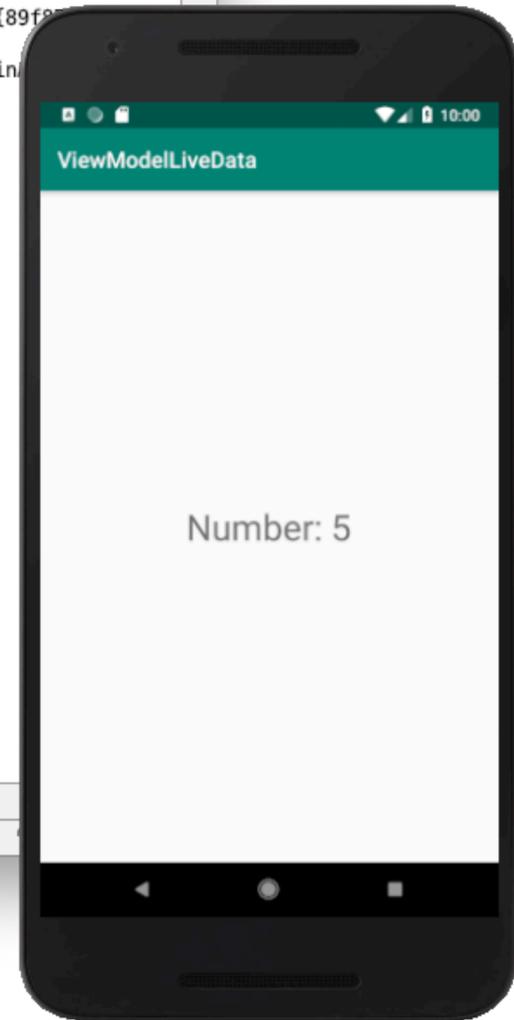
```
2
3  import androidx.appcompat.app.AppCompatActivity
4  import android.os.Bundle
5  import android.util.Log
6  import kotlinx.android.synthetic.main.activity_main.*
7
8  class MainActivity : AppCompatActivity() {
9
10     lateinit var data: MainActivityDataGenerator
11
12     override fun onCreate(savedInstanceState: Bundle?) {
13         super.onCreate(savedInstanceState)
14         setContentView(R.layout.activity_main)
15
16         data = MainActivityDataGenerator()
17         val myRandomNumber = data.getNumber()
18         tv_number.text = myRandomNumber
19
20         Log.i(TAG, "Random Number set")
21     }
22
23     companion object {
24         private val TAG = MainActivity::class.java.simpleName
25     }
26 }
27
```

The screenshot shows an IDE window for a project named 'ViewModelLiveData'. The code editor displays the following Kotlin code for `MainActivityDataGenerator`:

```
class MainActivityDataGenerator {  
    private lateinit var myRandomNumber: String  
  
    fun getNumber(): String {
```

The Logcat window below shows the following log messages, with a red box highlighting the last three lines:

```
2019-02-18 10:00:04.392 9851-9851/at.htl.viewmodellivedata I/MainActivityDataGenerator: Get number  
2019-02-18 10:00:04.392 9851-9851/at.htl.viewmodellivedata I/MainActivityDataGenerator: Create new number  
2019-02-18 10:00:04.393 9851-9851/at.htl.viewmodellivedata I/MainActivity: Random Number set
```



The screenshot shows an IDE with the following components:

- Code Editor:** Displays the Kotlin code for `MainActivityDataGenerator`. The class has a `private lateinit var myRandomNumber: String` property and a `fun getNumber(): String` function.
- Logcat:** Shows a series of log messages. A red box highlights the message: `2019-02-18 10:00:59.678 9851-9851/at.htl.viewmodellivedata I/MainActivityDataGenerator: Get number`. Other messages show the app starting and the number being set.
- Smartphone Emulator:** Displays the text "Number: 2" on a white background with a green header bar.

Man sieht: Die „Daten“ gehen verloren und werden neu generiert

# **Variante 2: mit ViewModel**

ViewModelLiveData [~/CloudStation/htl/skripten/themen/android.kotlin/2019.jetpack.udemy/02.LifeCycle/ViewModelLiveData] - app

Android

Project: ViewModelLiveData

app

Configure project in Project Structure dialog.

```
1  apply plugin: 'com.android.application'
2
3  apply plugin: 'kotlin-android'
4
5  apply plugin: 'kotlin-android-extensions'
6
7  apply plugin: 'kotlin-kapt'
8
9  android {
10     compileSdkVersion 28
11     defaultConfig {
12         applicationId "at.htl.viewmodellivedata"
13         minSdkVersion 23
14         targetSdkVersion 28
15         versionCode 1
16         versionName "1.0"
17         testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
18     }
19     buildTypes {
20         release {
21             minifyEnabled false
22             proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
23         }
24     }
25 }
26
27 dependencies {
28     implementation fileTree(dir: 'libs', include: ['*.jar'])
29     implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
30     implementation 'androidx.appcompat:appcompat:1.0.2'
31     implementation 'androidx.core:core-ktx:1.0.1'
32     implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
33     testImplementation 'junit:junit:4.12'
34     androidTestImplementation 'androidx.test.ext:junit:1.1.0'
35     androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
36
37     def lifecycle_version = "2.0.0"
38     // ViewModel and LiveData
39     implementation "androidx.lifecycle:lifecycle-extensions:$lifecycle_version"
40
41 }
```

Gradle build finished in 575 ms (moments ago)

42:1 LF UTF-8 4 spaces

<https://developer.android.com/jetpack/androidx/releases/lifecycle>

# Erstellen eines ViewModels

```
class MainActivityDataGenerator: ViewModel() {  
  
    private lateinit var myRandomNumber: String  
  
    fun getNumber(): String {  
        Log.i(TAG, "Get number")  
  
        if (!::myRandomNumber.isInitialized) {  
            this.createNumber()  
        }  
        return myRandomNumber  
    }  
  
    // kotlin.random.Random  
    private fun createNumber() {  
        Log.i(TAG, "Create new number")  
        myRandomNumber = "Number: " + (Random.nextInt(10 - 1) + 1)  
    }  
  
    companion object {  
        private val TAG = MainActivityDataGenerator::class.java.simpleName  
    }  
}
```

Wir verwenden den bereits bestehenden MainActivityDataGenerator als ViewModel

# ViewModel in MainActivity verwenden

```
class MainActivity : AppCompatActivity() {  
  
    // lateinit var data: MainActivityDataGenerator  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val model = ViewModelProviders  
            .of(this)  
            .get(MainActivityDataGenerator::class.java)  
        val myRandomNumber = model.getNumber()  
  
        //data = MainActivityDataGenerator()  
        //val myRandomNumber = data.getNumber()  
        tv_number.text = myRandomNumber  
  
        Log.i(TAG, "Random Number set")  
    }  
  
    companion object {  
        private val TAG = MainActivity::class.java.simpleName  
    }  
}
```

Beachte: Verwende ViewModelProviders

ViewModelProviders (androidx.life  
ViewModelProvider (androidx.li...  
Did you know that Quick Documentation View (F1) works in completion lookups as well? >>

The screenshot shows the Android Studio IDE with the following components:

- Code Editor:** Displays Kotlin code for MainActivity.kt, showing lines 18 and 19: `.of(this)` and `.get(MainActivityDataGenerator::class.java)`.
- Logcat:** Shows a log entry for MainActivityDataGenerator: `I/MainActivityDataGenerator: Create new number`, which is highlighted with a red box.
- Text Box:** A blue box with the text "Eine neue Zufallszahl wird generiert" (A new random number is generated).
- Emulator:** Displays the app's UI with the text "Number: 6".
- Notifications:** A green "Install successful" notification and a grey "Gradle build finished in 1 m 47 s 313 ms" notification are visible.

The screenshot shows the Android Studio IDE with the following components:

- Project Explorer:** Shows the project structure for 'app' with folders for 'manifests', 'java', and 'at.htl.viewmodellivedata'.
- Code Editor:** Displays the Kotlin file 'MainActivity.kt' with lines 18 and 19 visible. Line 18 contains `.of(this)` and line 19 contains `.get(MainActivityDataGenerator::class.java)`.
- Logcat:** Shows log messages for the application. The following messages are highlighted with a red box:

```
2019-02-18 10:20:29.843 10616-10616/at.htl.viewmodellivedata I/MainActivity: Random Number set
2019-02-18 10:21:04.279 10616-10616/at.htl.viewmodellivedata I/MainActivityDataGenerator: Get number
2019-02-18 10:21:04.279 10616-10616/at.htl.viewmodellivedata I/MainActivity: Random Number set
```
- Device File Explorer:** Shows the virtual device's file system.
- Bottom Bar:** Includes tabs for 'TODO', 'Terminal', 'Build', 'Logcat', 'Profiler', and 'Run'. The status bar shows 'Install successful (a minute ago)' and 'Event Log'.

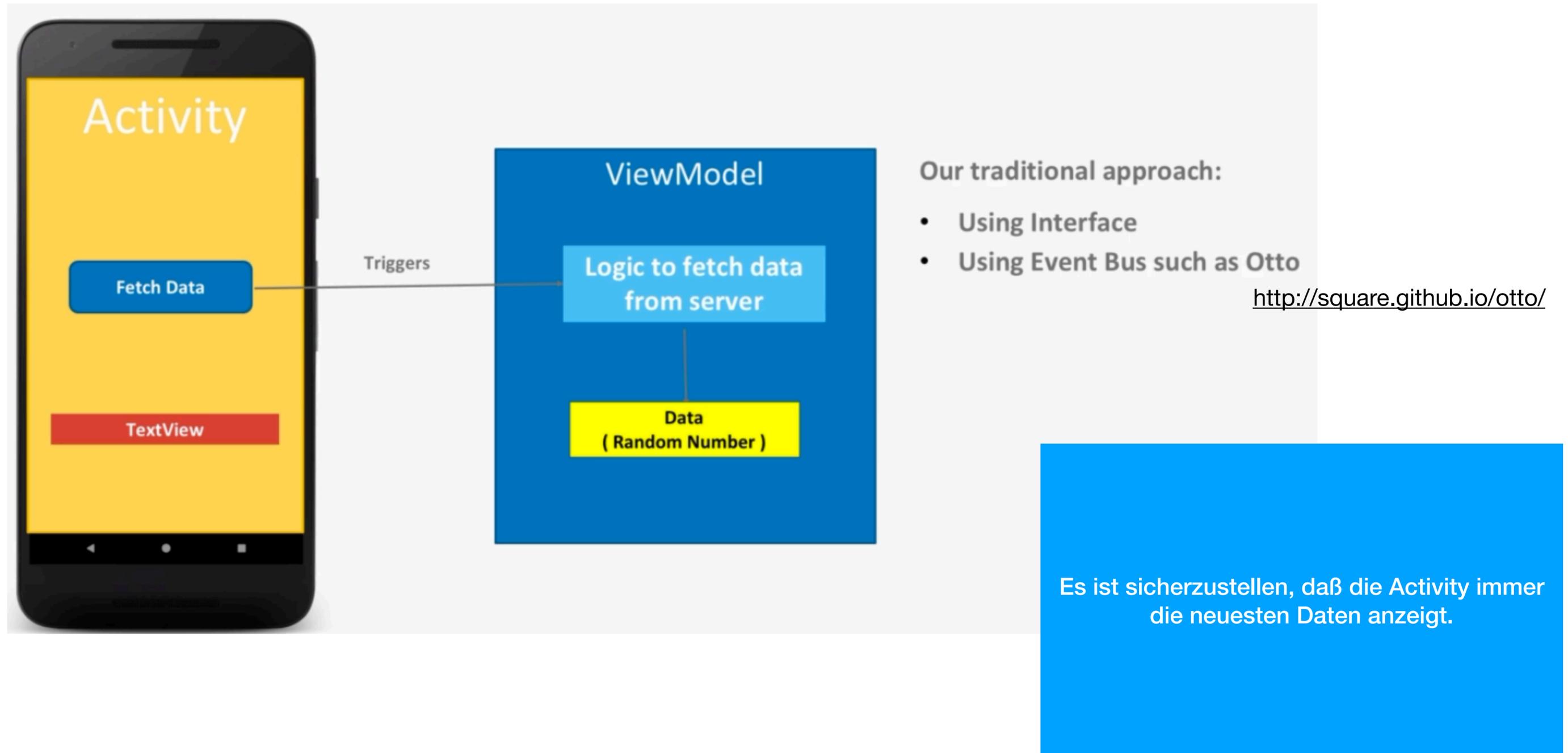
**Blue Text Box:**

Die bestehende Zufallszahl wird wiederverwendet.  
Die Daten gehen nicht verloren.

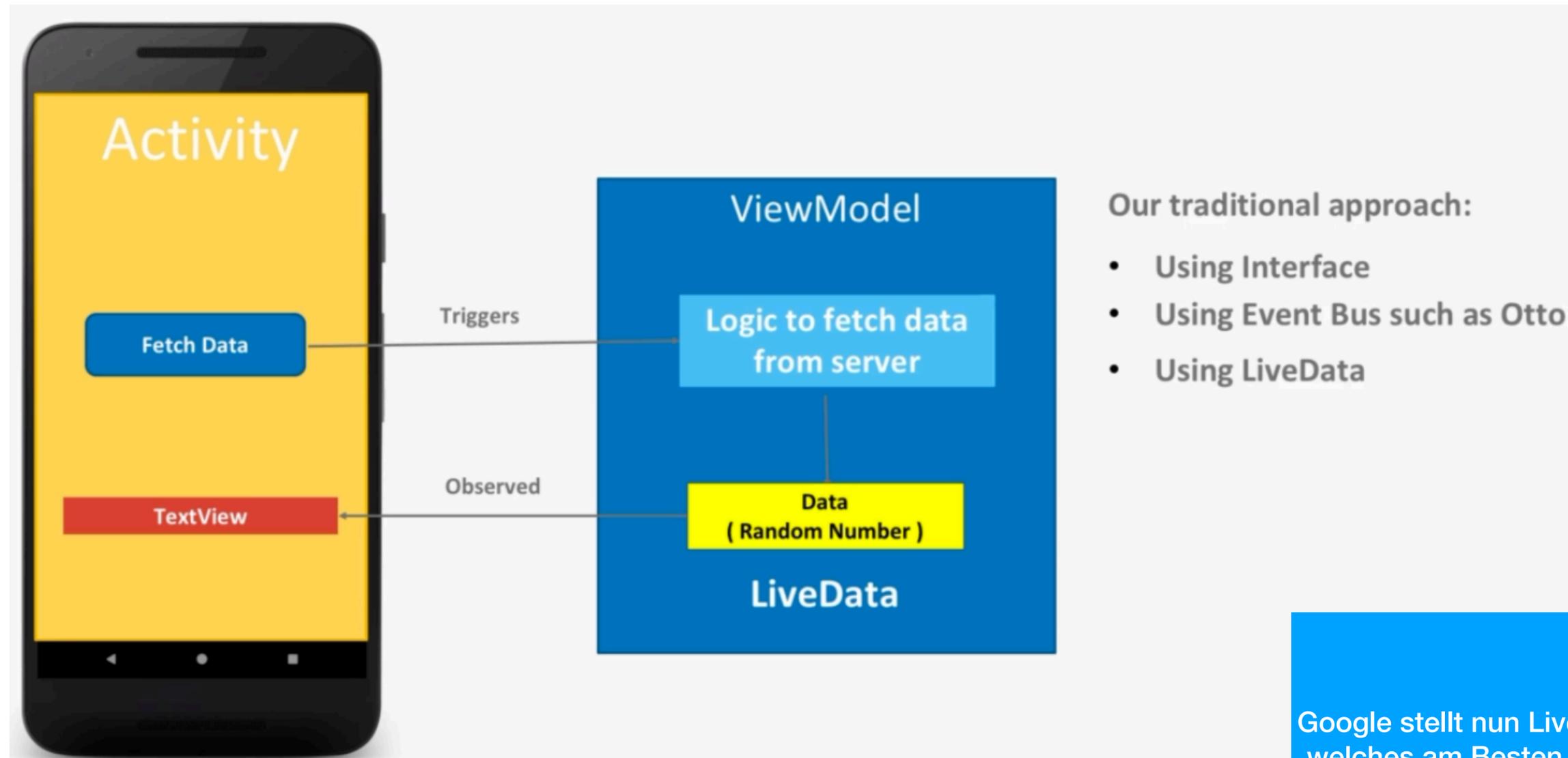
**Virtual Device:** The virtual device screen shows the application 'ViewModelLiveData' with the text 'Number: 6' displayed.

# LiveData

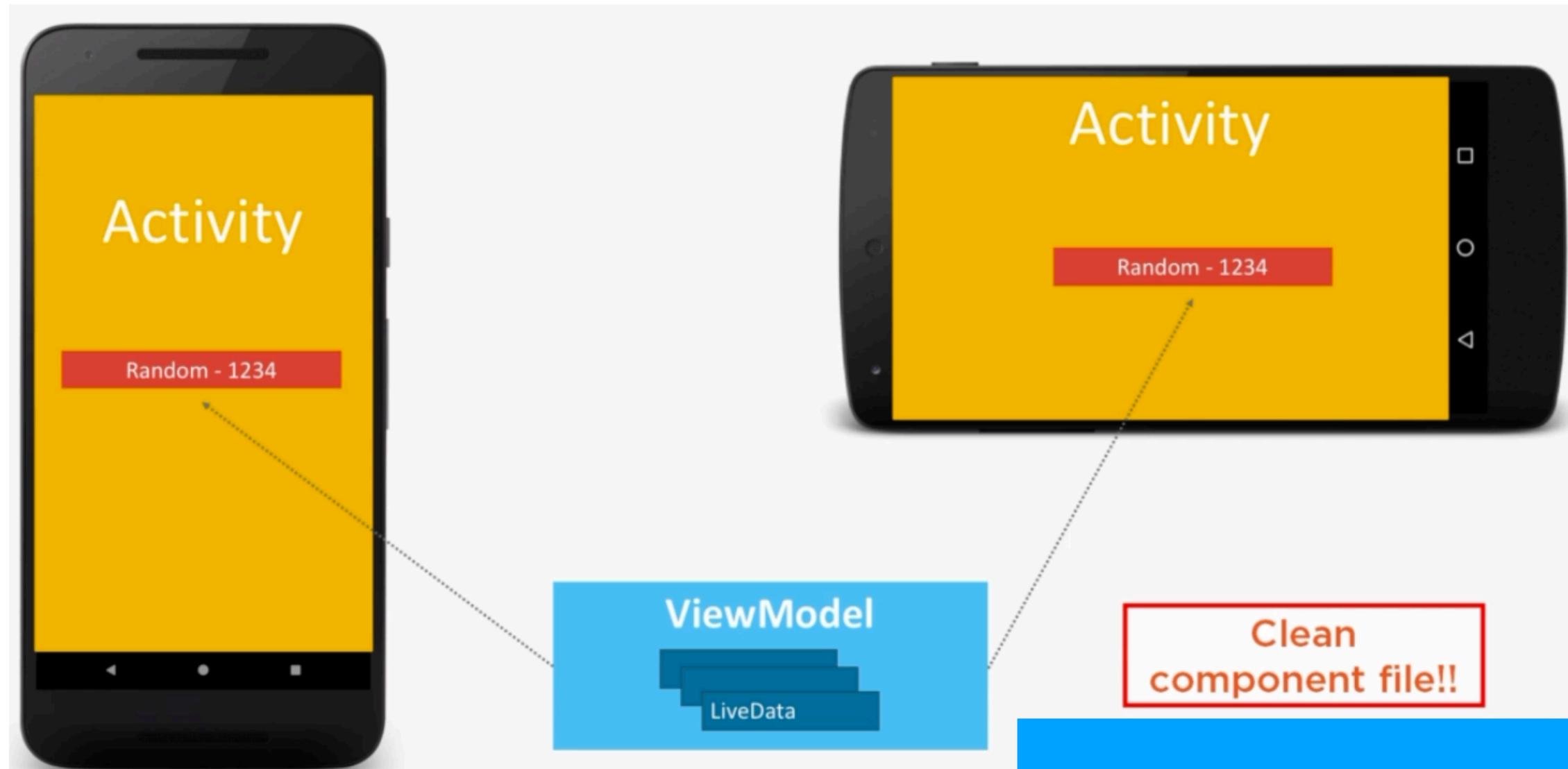
# Problem



# Lösung



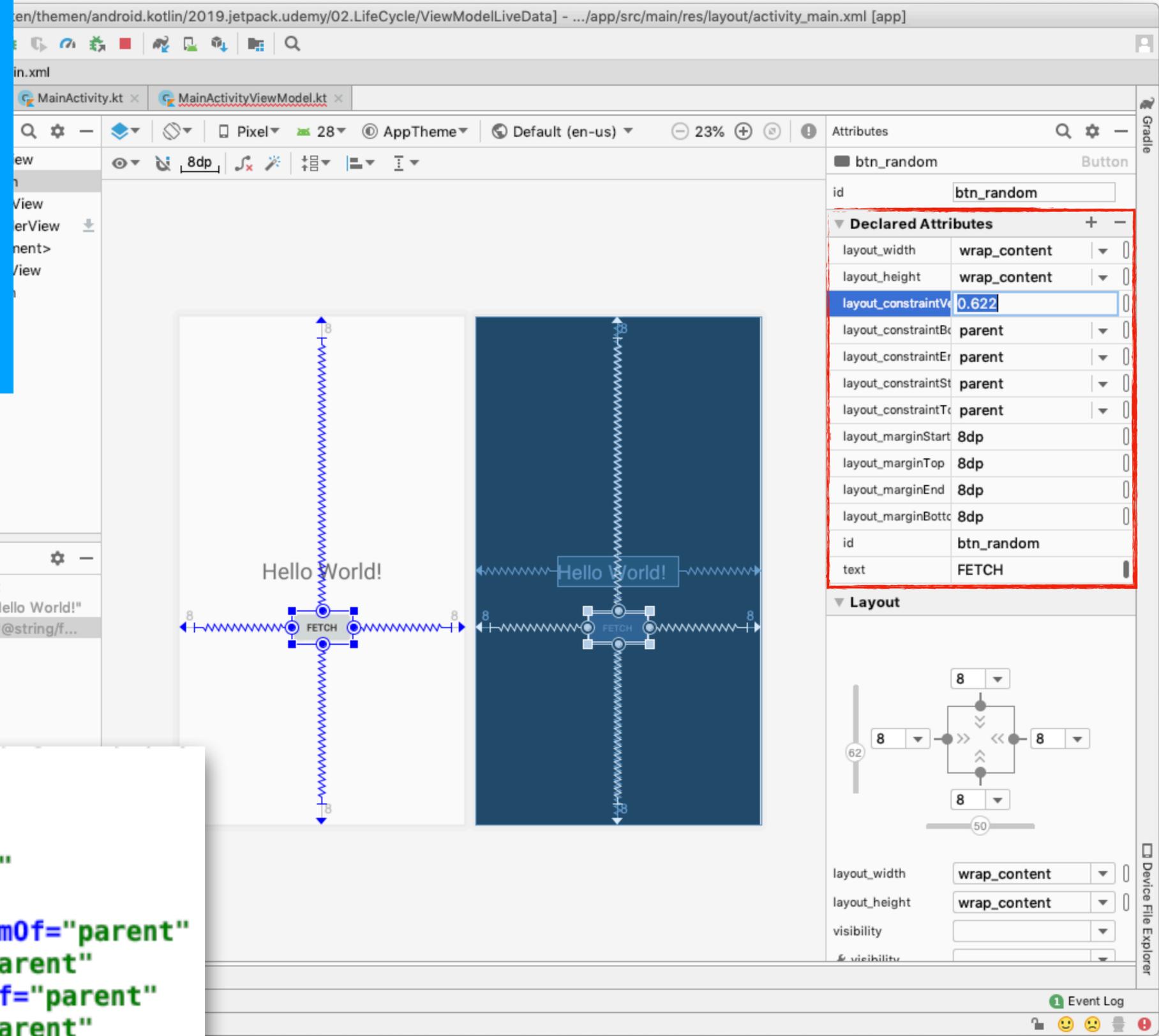
Google stellt nun LiveData zur Verfügung, welches am Besten mit dem ViewModel kombiniert wird.



Zunächst führen wir ein paar Änderungen an unserem Beispiel durch:

Zunächst erstellen wir einen neuen Button.

Dabei wird eine String-Resource erstellt.



```
<Button
    android:id="@+id/btn_random"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/fetch"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.622"/>
```

# MainActivityViewModel.kt

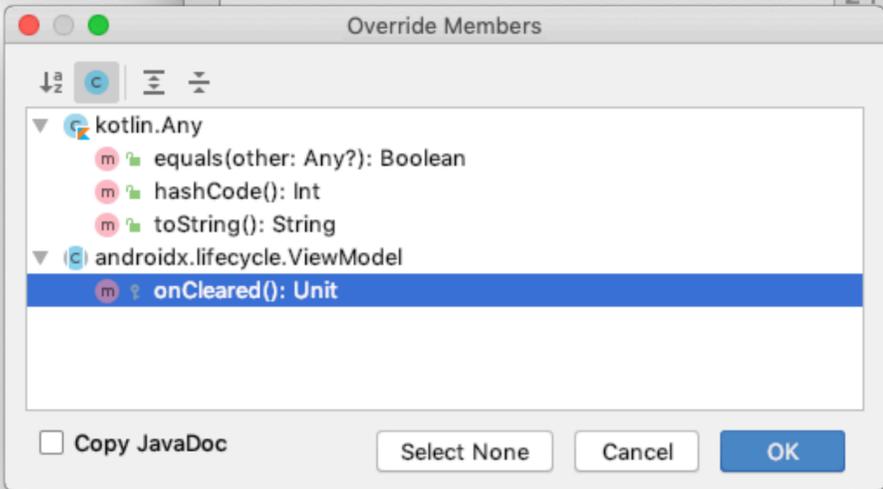
Wir verwenden MutableLiveData, da wir keine Zugriff auf Methoden freigeben, die die Daten ändern könnten. MutableLiveData wird von LiveData abgeleitet

```
1 package at.htl.viewmodellivedata
2
3 import ...
4
5 class MainActivityViewModel: ViewModel() {
6
7     private lateinit var myRandomNumber: MutableLiveData<String>
8     //private lateinit var myRandomNumber: String
9
10    fun getNumber(): MutableLiveData<String> {
11        Log.i(TAG, "Get number")
12
13        if (!::myRandomNumber.isInitialized) {
14            myRandomNumber = MutableLiveData()
15            this.createNumber()
16        }
17        return myRandomNumber
18    }
19
20    fun createNumber() {
21        Log.i(TAG, "Create new number")
22        myRandomNumber.value = "Number: " + (Random.nextInt(10 - 1) + 1)
23
24        // myRandomNumber = "Number: " + (Random.nextInt(10 - 1) + 1)
25    }
26
27    override fun onCleared() {
28        super.onCleared()
29        Log.i(TAG, "ViewModel destroyed")
30    }
31
32    companion object {
33        private val TAG = MainActivityViewModel::class.java.simpleName
34    }
35 }
```

```
package androidx.lifecycle;
/**
 * {@link LiveData} which publicly exposes {@link #setValue(T)}
 *
 * @param <T> The type of data hold by this instance
 */
@WeakerAccess
public class MutableLiveData<T> extends LiveData<T> {
    @Override
    public void postValue(T value) { super.postValue(value); }

    @Override
    public void setValue(T value) { super.setValue(value); }
}
```

Override Methods via ^O (Ctrl+O for Win/Linux)



# MainActivity.kt

```
package at.htl.viewmodellivedata

import ...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val model = ViewModelProviders
            .of(this)
            .get(MainActivityViewModel::class.java)
        val myRandomNumber = model.getNumber()

        myRandomNumber.observe(this, Observer<String> { number ->
            tv_number.text = number
            Log.i(TAG, "Random number set")
        })

        btn_random.setOnClickListener {
            model.createNumber()
        }
    }

    companion object {
        private val TAG = MainActivity::class.java.simpleName
    }
}
```

die MainActivity beobachtet das  
ViewModel und aktualisiert die  
View

onClickListener für den Button

The screenshot shows an IDE window for a Kotlin project. The main editor displays the following code for MainActivityViewModel:

```
1 package at.htl.viewmodellivedata
2
3 import ...
7
8 class MainActivityViewModel: ViewModel() {
9
10     private lateinit var myRandomNumber: MutableLiveData<String>
11     //private lateinit var myRandomNumber: String
12
13     fun getNumber(): MutableLiveData<String> {
14         Log.i(TAG, "Get number")
15
16         if (!myRandomNumber.isInitialized) {
```

The Logcat window below shows the following log entries:

```
2019-02-18 11:21:20.510 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.510 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.511 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.511 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.512 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.513 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.611 11532-11532/at.htl.viewmodellivedata I/MainActivityViewModel: Get number
2019-02-18 11:21:20.611 11532-11532/at.htl.viewmodellivedata I/MainActivityViewModel: Create new number
2019-02-18 11:21:20.619 11532-11532/at.htl.viewmodellivedata I/MainActivity: Random number set
```

A blue box highlights the last three lines of the Logcat output. To the right, a smartphone displays the app's UI with the text "Number: 8" and a "FETCH" button.

Nach dem Starten wird die Zahl neu generiert

The screenshot shows an IDE window for a project named 'ViewModelLiveData'. The main editor displays the Kotlin code for 'MainActivityViewModel.kt':

```
1 package at.htl.viewmodellivedata
2
3 import ...
7
8 class MainActivityViewModel: ViewModel() {
9
10     private lateinit var myRandomNumber: MutableLiveData<String>
11     //private lateinit var myRandomNumber: String
12
13     fun getNumber(): MutableLiveData<String> {
14         Log.i(TAG, "Get number")
15
16         if (!myRandomNumber.isInitialized) {
```

The Logcat window below shows the following log entries:

```
2019-02-18 11:21:20.510 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.510 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.511 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.511 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.512 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.513 11532-11532/at.htl.viewmodellivedata I/zygote: at void at.htl.viewmodellivedata.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt:14)
2019-02-18 11:21:20.611 11532-11532/at.htl.viewmodellivedata I/MainActivityViewModel: Get number
2019-02-18 11:21:20.611 11532-11532/at.htl.viewmodellivedata I/MainActivityViewModel: Create new number
2019-02-18 11:21:20.619 11532-11532/at.htl.viewmodellivedata I/MainActivity: Random number set
2019-02-18 11:23:12.684 11532-11532/at.htl.viewmodellivedata I/MainActivityViewModel: Create new number
2019-02-18 11:23:12.684 11532-11532/at.htl.viewmodellivedata I/MainActivity: Random number set
```

A blue callout box contains the text: "Durch Klicken des Button wird eine neue Zahl generiert und automatisch in der Activity dargestellt."

On the right, a smartphone emulator displays the app's UI with the title 'ViewModelLiveData' and the text 'Number: 4' above a 'FETCH' button.

# Wie gehts weiter?

- <https://codelabs.developers.google.com/codelabs/android-room-with-a-view/#0>
-



Noch  
Fragen?

# Source

<https://www.udemy.com/android-jetpack-architecture-components/>

The screenshot shows the UdeMy website interface. At the top, there's a navigation bar with the UdeMy logo, a search bar, and links for 'UdeMy for Business', 'Become an instructor', 'Log In', and 'Sign Up'. Below the navigation bar, the breadcrumb trail reads 'Development > Mobile Apps > Android Game Development'. The main content area features the course title 'Android Jetpack Architecture Components' in large white text on a dark background. Below the title is a subtitle: 'Utilize Android Jetpack Architecture components to make your Android application development flexible and maintainable'. There are also indicators for 'NEW', a star rating of 0.0 (0 ratings), and '4 students enrolled'. The course is created by Packt Publishing and was last updated in 2/2019. On the right side of the course card, there are buttons for 'Gift This Course' and 'Wishlist'. A video preview player is visible with a play button and the text 'Preview this course'.



## What you'll learn

- ✓ Get introduced to Android architecture components
- ✓ Provide stability in your app by handling life cycles, view models, and live data
- ✓ Load data gradually and gracefully in RecyclerView by using the Paging library
- ✓ Explore how to perform CRUD operations in the Room database
- ✓ Use the Data Binding library to bind data to the UI
- ✓ Implement effective in-app navigation by using the Navigation architecture component
- ✓ Implement a local database to store structured data by using the Room database
- ✓ Schedule tasks asynchronously by using Work Manager

**€10.99** ~~€124.99~~ 91% off

🕒 5 hours left at this price!

Add to cart

Buy now

30-Day Money-Back Guarantee

### Includes

- 📺 3 hours on-demand video
- 📄 1 downloadable resource
- 🌐 Full lifetime access
- 📱 Access on mobile and TV
- 🏆 Certificate of Completion