

# › APP-ENTWICKLUNG MIT ANDROID STUDIO

Teil 2: Aufbau einer Android App, Android Studio und Best Practices

Christian Deme / Fakultät Informatik / Angewandte Informatik / Prof. Dr.-Ing. Wolfgang Hess | SoSe 2024

13.11.2024



# › RECAP

# RECAP

---

## Entwurfskonzepte

- Design Thinking und User Requirements Engineering
- Projektmanagement

## Prototyping

- Erstellung von Mockups und Wireframes

## Projektmanagement

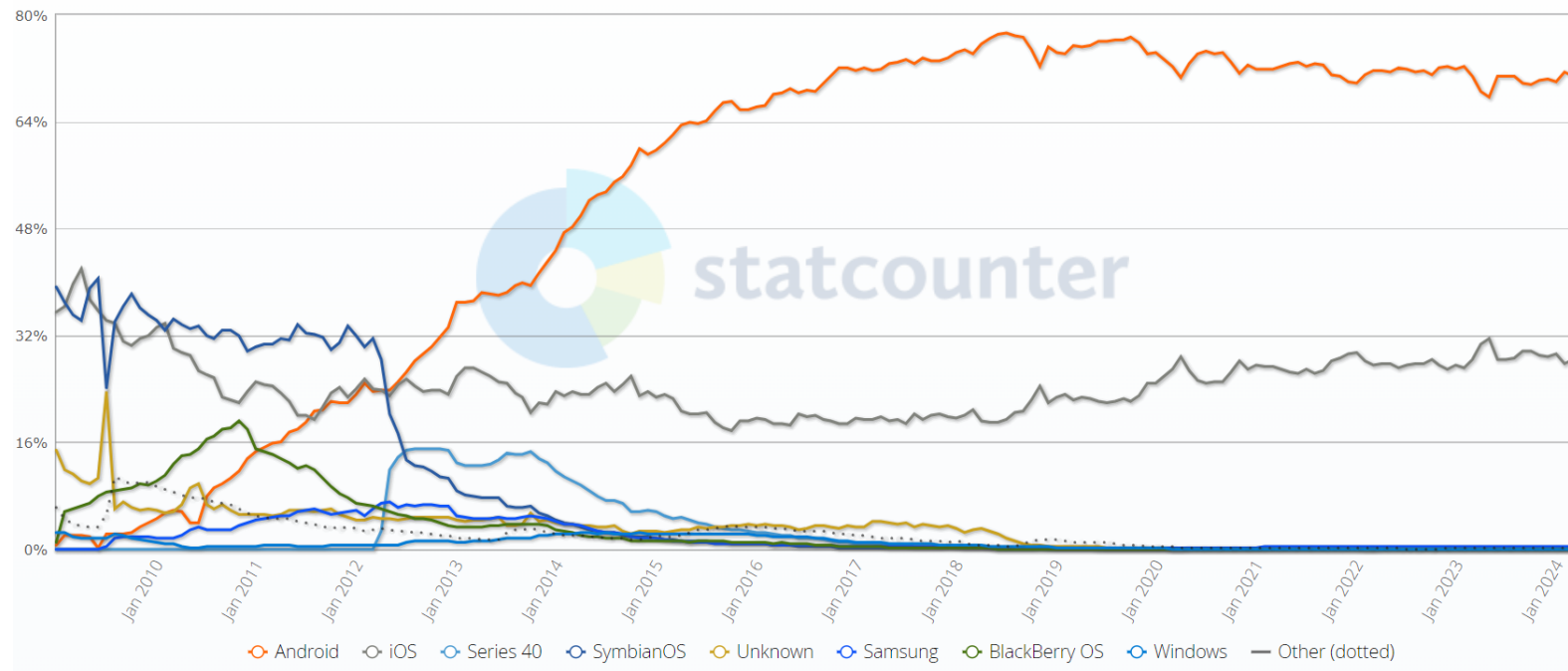




# › ANDROID APP

# WARUM ANDROID?

Mobile Operating System Market Share Worldwide  
Jan 2009 - May 2024



android  
studio



# ANDROID APP

---

Woraus besteht eine Android App?

## App-Komponenten

- Activities
- Services
- Broadcast Receivers
- Content Providers

# ANDROID APP

---

## Activities

- Einstiegspunkt für die Interaktion mit dem Nutzer
- repräsentiert eine einzelne **Bildschirmansicht**
- bilden eine zusammenhängende Nutzererfahrung, sind aber unabhängig voneinander
- andere Apps können auch andere Activities starten

# ANDROID APP

---

## Activities

Sie ermöglichen wichtige Interaktionen zwischen System und App:

- Verfolgen, was aktuell wichtig für den Nutzer ist, sodass das System diesen Prozess **aktiv** hält
- Behalten von gestoppten Activities, zu denen der Nutzer eventuell zurückkehrt, und Priorisieren dieser Prozesse
- Unterstützung beim Wiederherstellen des vorherigen Zustands nach einem Prozessabbruch
- Ermöglichen Nutzerabläufe zwischen Apps, wie z. B. das Teilen von Inhalten

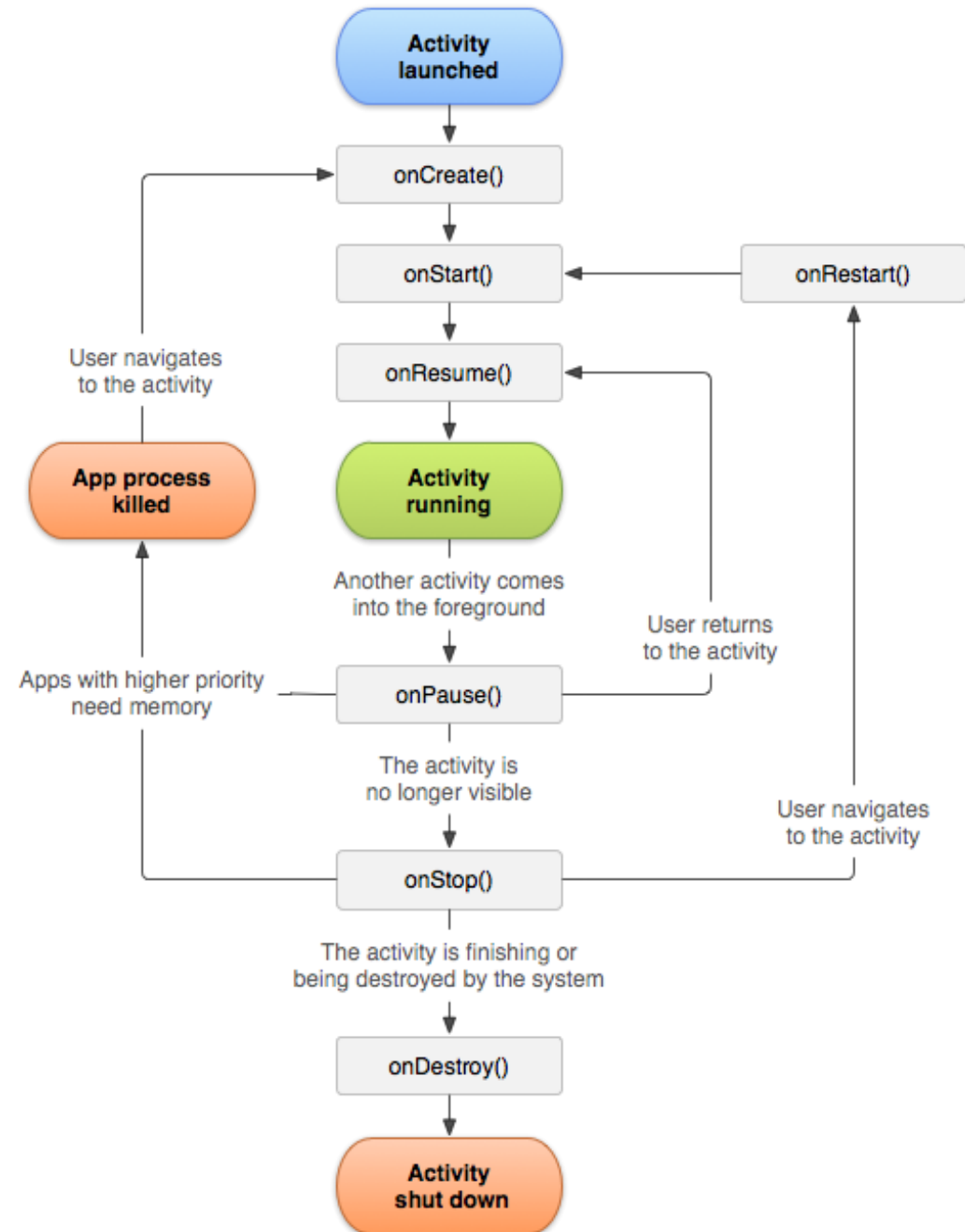


# ANDROID APP

---

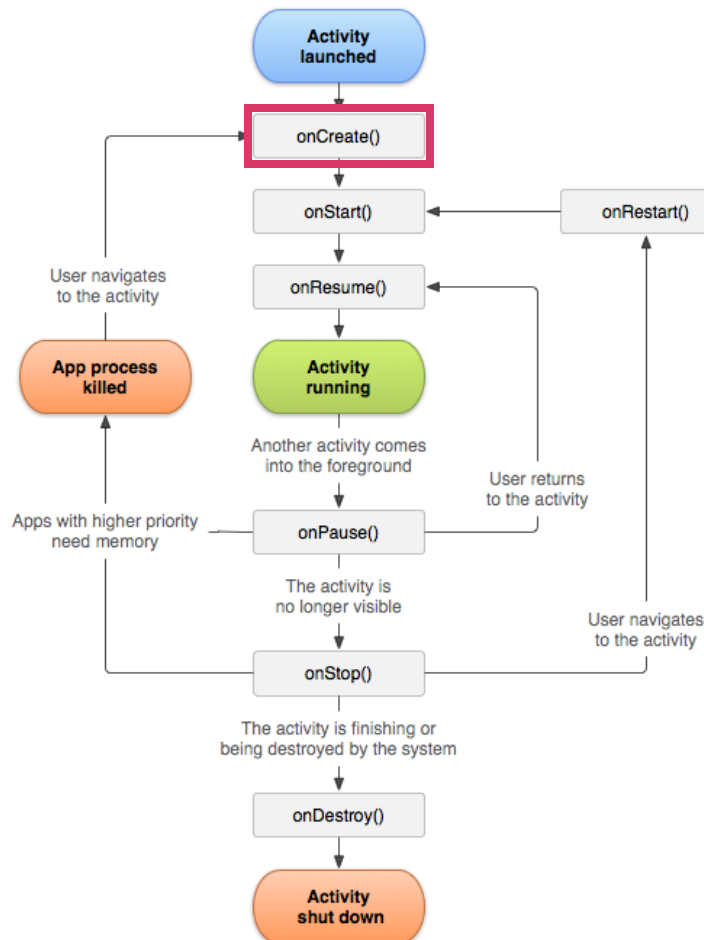
## Activities Lebenszyklus (Lifecycle)

- Wenn ein Nutzer durch die App navigiert, diese verlässt und wieder zurückkehrt, wechseln die Activity-Instanzen der App durch verschiedene Zustände in ihrem Lebenszyklus
- Die Activity-Klasse stellt dafür verschiedene Rückrufmethoden (Callbacks) bereit, die signalisieren, wenn eine Aktivität erstellt, angehalten, fortgesetzt oder zerstört wird
- Lebenszyklusmethoden ermöglichen es, die Aktivität entsprechend der jeweiligen Änderung im Zustand anzupassen



# ANDROID APP

onCreate()



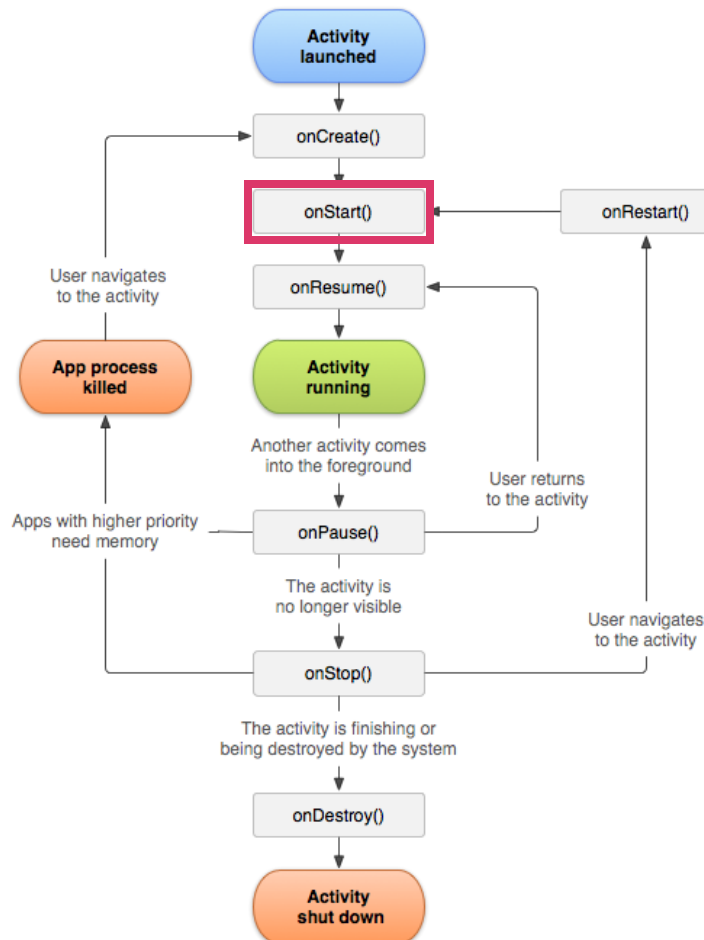
- wird aufgerufen, wenn das System die Aktivität zum ersten Mal erstellt
- es wird nur **grundlegende Logik** ausgeführt, die nur einmal während der gesamten Lebensdauer der Aktivität ablaufen soll

Anwendungsbeispiele:

Daten an Listen binden, Initialisierung des Layouts

# ANDROID APP

## onStart()

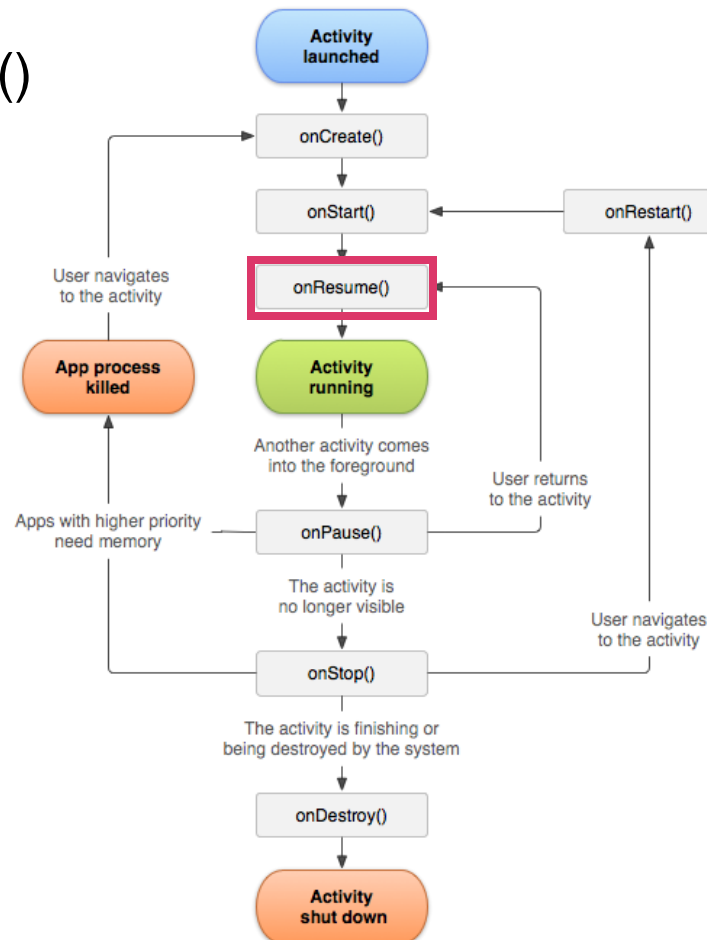


- sorgt dafür, dass die Aktivität für den Nutzer sichtbar wird, während die App die Aktivität auf den Übergang in den Vordergrund vorbereitet, wo sie interaktiv wird
- hier wird häufig der Code initialisiert, der die **Benutzeroberfläche** (UI) aufrechterhält
- nach Abschluss des Callbacks wechselt die App in den onResume() Status



# ANDROID APP

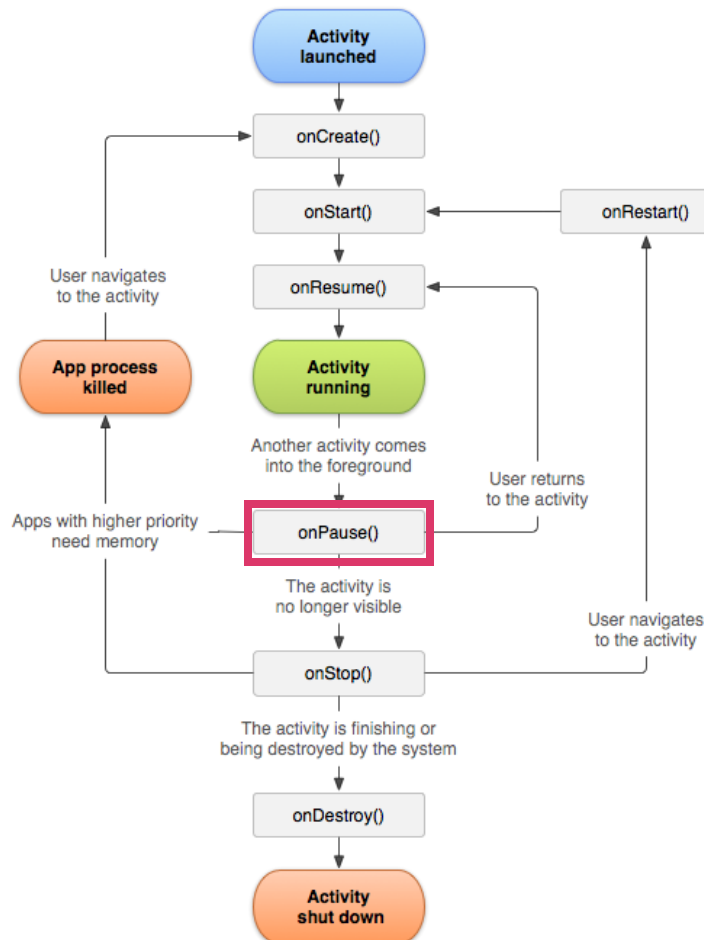
onResume()



- die App interagiert direkt mit dem Nutzer und bleibt darin, bis ein Ereignis die Fokussierung auf eine andere App oder Funktion umlenkt
- Lifecycle-Komponenten können alle Funktionen aktivieren, die laufen sollen, während die Komponente im Vordergrund sichtbar ist

# ANDROID APP

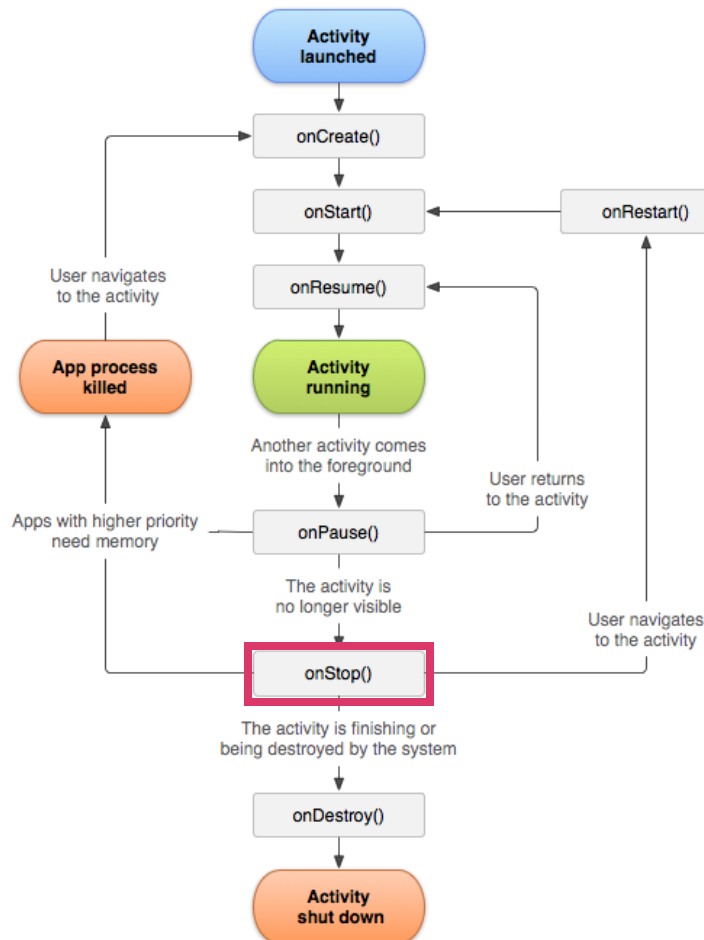
## onPause()



- wenn eine neu gestartete Aktivität den gesamten Bildschirm überlagert oder wenn die Aktivität endet und beendet wird
- es sollten **Ressourcen freigegeben** oder angepasst werden, die während der Unsichtbarkeit der App nicht benötigt werden

# ANDROID APP

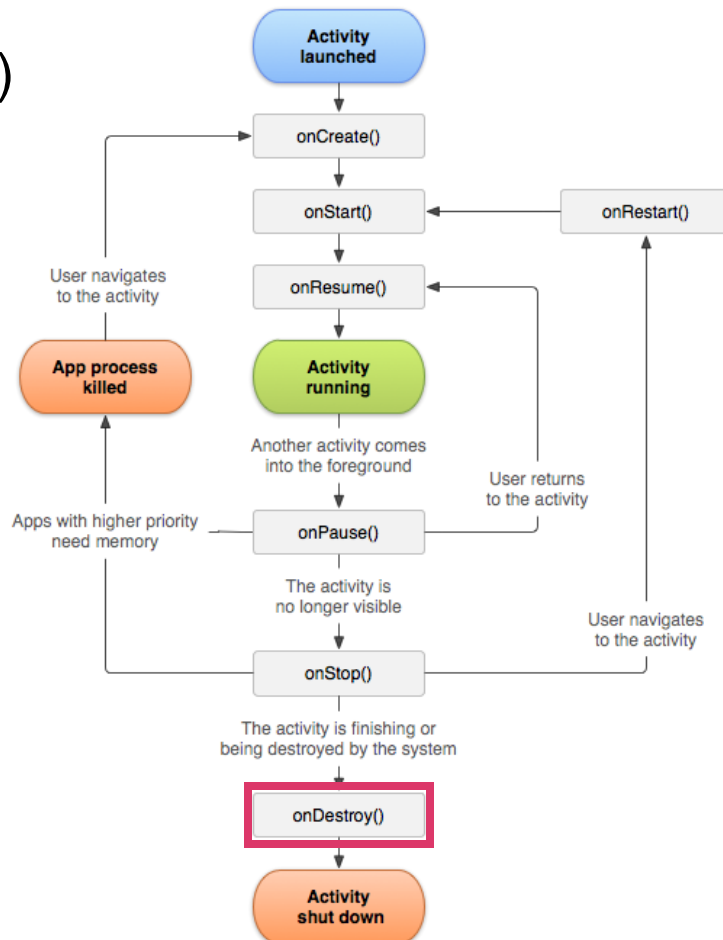
## onStop()



- sobald der Nutzer die Aktivität verlässt (nicht mehr im Vordergrund)
- typischerweise werden hier auch Ressourcen freigegeben, die **während der Pause den Akku belasten könnten**, beispielsweise GPS-Sensoren

## ANDROID APP

## onDestroy()



- wird aufgerufen, bevor die Aktivität endgültig zerstört wird (z.B. durch vollständiges Schließen der App)
- hier sollten alle Ressourcen freigegeben werden, die nicht bereits in vorherigen Rückrufen wie `onStop()` freigegeben wurden



# ANDROID APP

---

Vorteile gut implementierter Lebenszyklen

Die App kann **robust und performant** bleiben und dadurch folgende Probleme vermeiden:

- Abstürze, wenn der Nutzer einen Anruf erhält oder zu einer anderen App wechselt
- Unnötige Systemressourcen, die verbraucht werden, wenn die App nicht aktiv genutzt wird (z.B. Akku)
- Verlust des Fortschritts, wenn der Nutzer die App verlässt und später zurückkehrt
- Abstürze oder Verlust des Fortschritts beim Drehen des Bildschirms (zwischen Hoch- und Querformat)

# ANDROID APP

---

## Services

- Einstiegspunkt, um eine App im Hintergrund laufen zu lassen
- für langwierige Aufgaben oder Aufgaben für entfernte Prozesse
- Besitzen **keine Benutzeroberfläche**

## Zwei Arten von Services:

### Gestartete Services und Gebundene Services

# ANDROID APP

---

## Gestartete Services

- bleiben aktiv, bis ihre Aufgabe abgeschlossen ist, wie z. B. Datenabgleich im Hintergrund oder Musikwiedergabe
- für die Musikwiedergabe werden Services in den Vordergrund gesetzt, damit das System ihn priorisiert und der Nutzer eine Benachrichtigung erhält
- gewöhnliche Hintergrundservices haben diese Priorität nicht und können bei Bedarf gestoppt und später neu gestartet werden

# ANDROID APP

---

## Gebundene Services

- laufen, weil eine andere App oder das System sie nutzen
- bieten eine API für andere Prozesse, und das System erkennt die Abhängigkeit zwischen diesen Prozessen
- wenn Prozess A an einen Service in Prozess B gebunden ist, sorgt das System dafür, dass Prozess B aktiv bleibt



# ANDROID APP

---

Beispiele für Services

Zentrale Bausteine für viele Systemfunktionen:

- Live-Hintergründe
- Benachrichtigungs-Listener
- Bildschirmschoner
- Barrierefreiheitsdienste

# ANDROID APP

---

## Broadcast Receivers

- Komponenten, die es dem System ermöglichen, Ereignisse an die App zu übermitteln, auch wenn diese nicht aktiv laufen
- die App kann auf **systemweite Ankündigungen** reagieren, wie Benachrichtigungen über den Bildschirmstatus, niedrigen Akkustand oder das Aufnehmen eines Fotos
- haben keine Benutzeroberfläche, können jedoch Benachrichtigungen erstellen, um den Nutzer zu informieren
- Apps können aber auch eigene Broadcasts senden, um z. B. anderen Apps mitzuteilen, dass neue Daten verfügbar sind

# ANDROID APP

---

## Beispiel für Broadcast Receivers

### Wecker-App:

- kann einen Alarm planen, der eine Benachrichtigung über ein bevorstehendes Ereignis anzeigt
- da der Alarm an einen Broadcast Receiver gesendet wird, muss die App nicht durchgehend laufen, bis der Alarm ausgelöst wird

# ANDROID APP

---

## Content Providers

- verwalten eine gemeinsam genutzte Datenmenge, die in einem Dateisystem, einer SQLite-Datenbank, im Web oder anderen persistenten Speicherorten abgelegt sein können
- über den Content Provider können **andere Apps die Daten abfragen oder ändern**, sofern der Provider dies erlaubt
- eignen sich auch für das Verwalten von Daten, die nur für die eigene App privat bleiben

## Beispiel: Kontakte des Nutzers



# ANDROID APP

---

## Intents

- **asynchrones** Nachrichtensystem
- aktivieren Activities, Services und Broadcast Receiver
- verbinden Komponenten zur Laufzeit, egal, ob sie zur eigenen oder zu einer anderen App gehört

# ANDROID APP


---

## Beispiel für Intents

- ein Intent kann eine Activity anfordern, um ein Bild anzuzeigen oder eine Webseite zu öffnen
- Manche Activities geben ein Ergebnis zurück, wie beim Auswählen eines Kontakts: Das zurückgegebene Intent enthält dann eine URI zu diesem Kontakt

# ANDROID APP

## Manifest-Datei



```
<manifest ... >
  <application android:icon="@drawable/app_icon.png" ... >
    <activity android:name="com.example.project.ExampleActivity"
              android:label="@string/example_label" ... >
    </activity>
  </application>
</manifest>
```

- Bevor das Android-System eine App-Komponente starten kann, muss die Manifest-Datei die Komponente bekannt machen
- Diese Datei befindet sich im **Hauptverzeichnis** des App-Projekts und deklariert alle Komponenten der App

# ANDROID APP

---

## Manifest-Datei

Sie erfüllt **mehrere Aufgaben**:

- Gibt Benutzerberechtigungen an, wie z. B. Internetzugriff oder Leserechte für Kontakte
- Legt das minimale API-Level fest, das die App benötigt
- Gibt Hardware- und Softwarefunktionen an, wie Kamera oder Bluetooth
- Deklariert benötigte API-Bibliotheken wie Google Maps

# ANDROID APP

---

## Festlegen von App-Anforderungen

- Da Android auf **verschiedenen Gerätetypen** läuft, sollten in der Manifest-Datei die benötigten Funktionen und API-Level deklariert werden, um sicherzustellen, dass die App nur auf kompatiblen Geräten installiert werden kann
- Diese Anforderungen werden nicht vom System, sondern z. B. vom Google Play Store verwendet, um Geräte zu filtern



# ANDROID APP

---

## Ressourcen

- Elemente wie Bilder, Audiodateien und visuelle Elemente, die unabhängig vom Quellcode definiert sind
- App kann dadurch für unterschiedliche Gerätekonfigurationen, wie verschiedene Sprachen und Bildschirmgrößen, optimiert werden
- Jede Ressource hat eine **eindeutige ID**

Beispiel: Bild namens logo.png, das im Verzeichnis res/drawable/ gespeichert ist, eine ID R.drawable.logo, die als Ganzzahl-ID referenziert und in die Benutzeroberfläche eingebunden werden kann

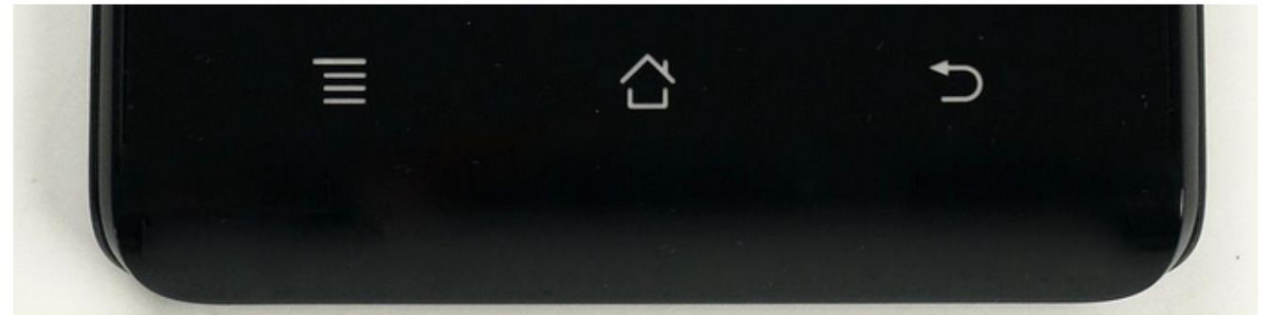
# NAVIGATION INNERHALB VON ANDROID

- Buttons
- Touchscreen und Gesten

1. Links: Zurück-Knopf. Mitte: Home-Knopf. Rechts: "Alle geöffneten Programme"-Taste.



2. Links: Menü-Knopf. Mitte: Home-Knopf. Rechts: Zurück-Knopf.



3. Links: Zurück-Knopf. Mitte: Home-Knopf. Rechts: "Alle geöffneten Programme"-Taste.





# › ANDROID STUDIO

# ANDROID STUDIO

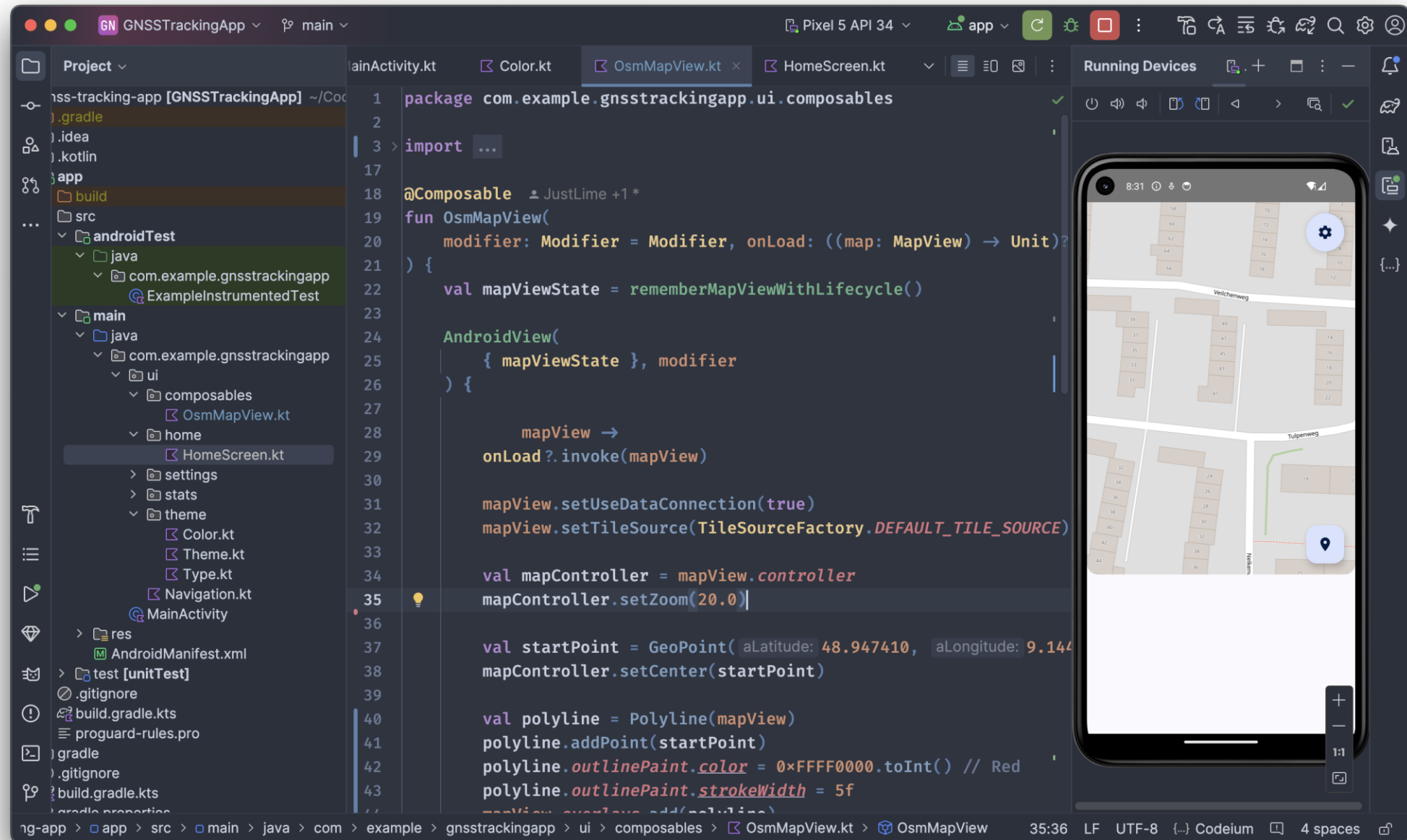
---

## Über Android Studio



- Veröffentlicht von Google und JetBrains
- Basiert auf der **IntelliJ IDEA Community Edition**
- Ermöglicht die Entwicklung von Android-Apps mit Hilfe von Kotlin





# KOTLIN

- **Plattformübergreifende** Programmiersprache
- Statisch typisiert
- Kann in Bytecode für die **JVM** (Java Virtual Machine) übersetzt werden
- Beherrscht Multithreading und asynchrone Prozesse (Coroutines)



# KOTLIN VS. JAVA

---

- Kotlin entfernt viel überflüssigen Boilerplate Code
- Kotlin ist syntaktisch nicht zu Java kompatibel, aber so gestaltet, dass **mit Java-Code interoperiert** werden kann



```
Person.java

public class Person {
    private String name;
    private int age;

    // Konstruktor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter und Setter für name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Getter und Setter für age
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    // Methode zum Begrüßen
    public String greet() {
        return "Hello, my name is " + name + " and I am " + age + " years old.";
    }

    // Main Methode zum Testen
    public static void main(String[] args) {
        Person person = new Person("Alice", 30);
        System.out.println(person.greet());
    }
}
```

```
Person.kt

class Person(var name: String, var age: Int) {

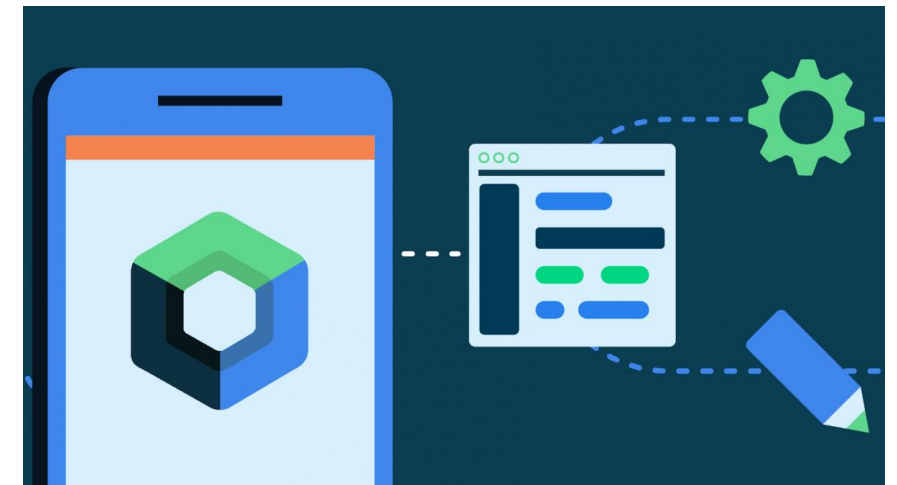
    // Methode zum Begrüßen
    fun greet() = "Hello, my name is $name and I am $age years old."
}

// Main Funktion zum Testen
fun main() {
    val person = Person("Alice", 30)
    println(person.greet())
}
```

# JETPACK COMPOSE

---

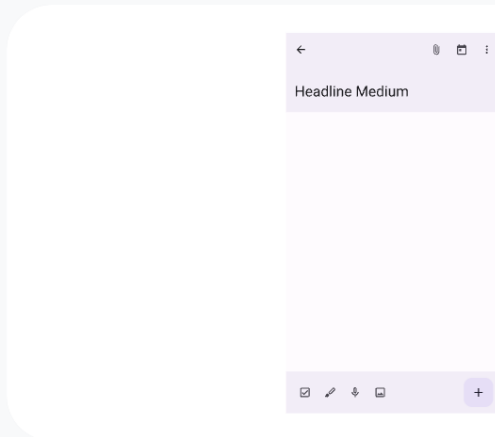
- Kotlin-basiertes, deklaratives **UI-Framework** für Android
- Entwickelt von Google
- Framework wurde production-ready im Juli 2021
- **Reaktive Programmierung**, ähnlich wie in modernen Web-Frameworks



# JETPACK COMPOSE

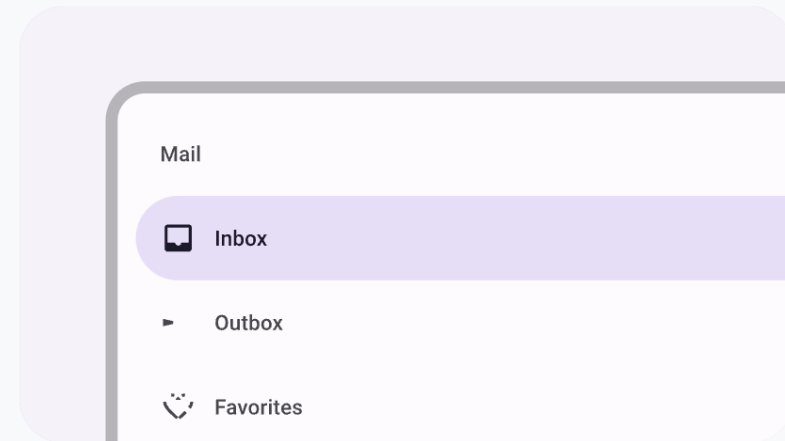
<https://developer.android.com/develop/ui/compose/components>

## Beispiele für Komponenten



### Scaffold

Use the `Scaffold` composable to provide structure for your screens.



### Drawers

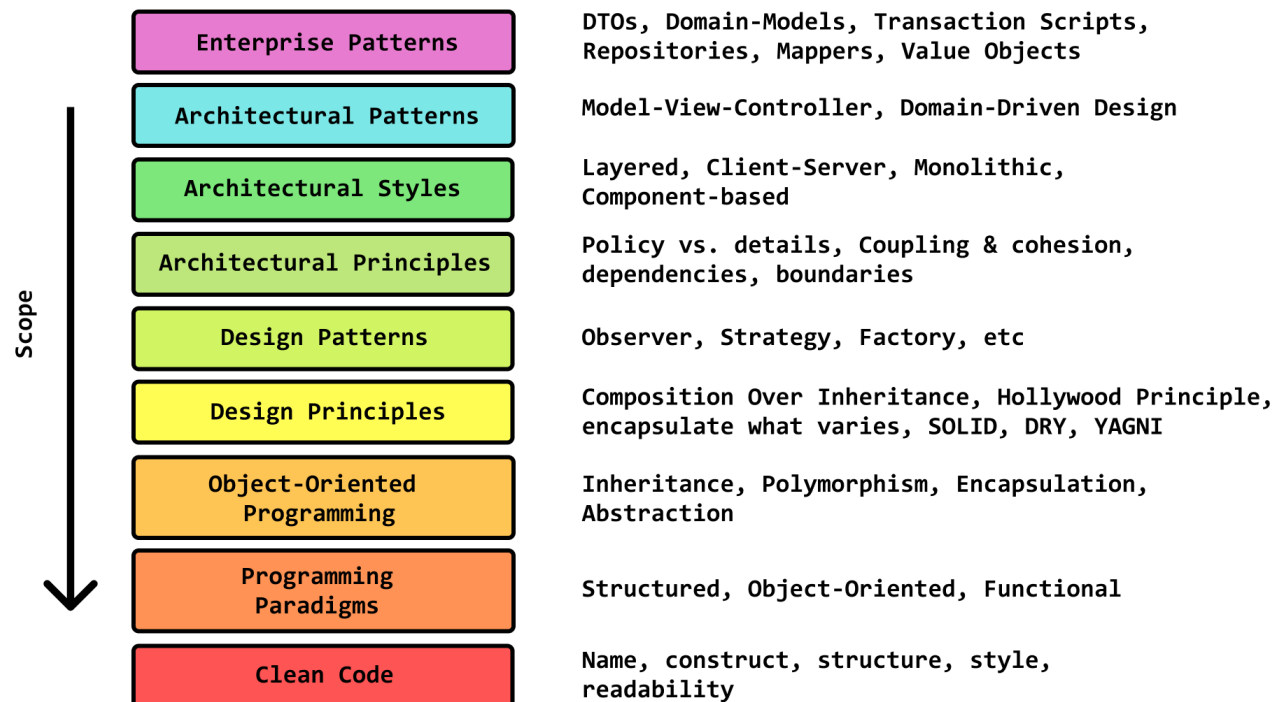
A drawer is a slide-out sidebar for navigation or additional content

# SOFTWARE DESIGN



## The Software Design & Architecture Stack

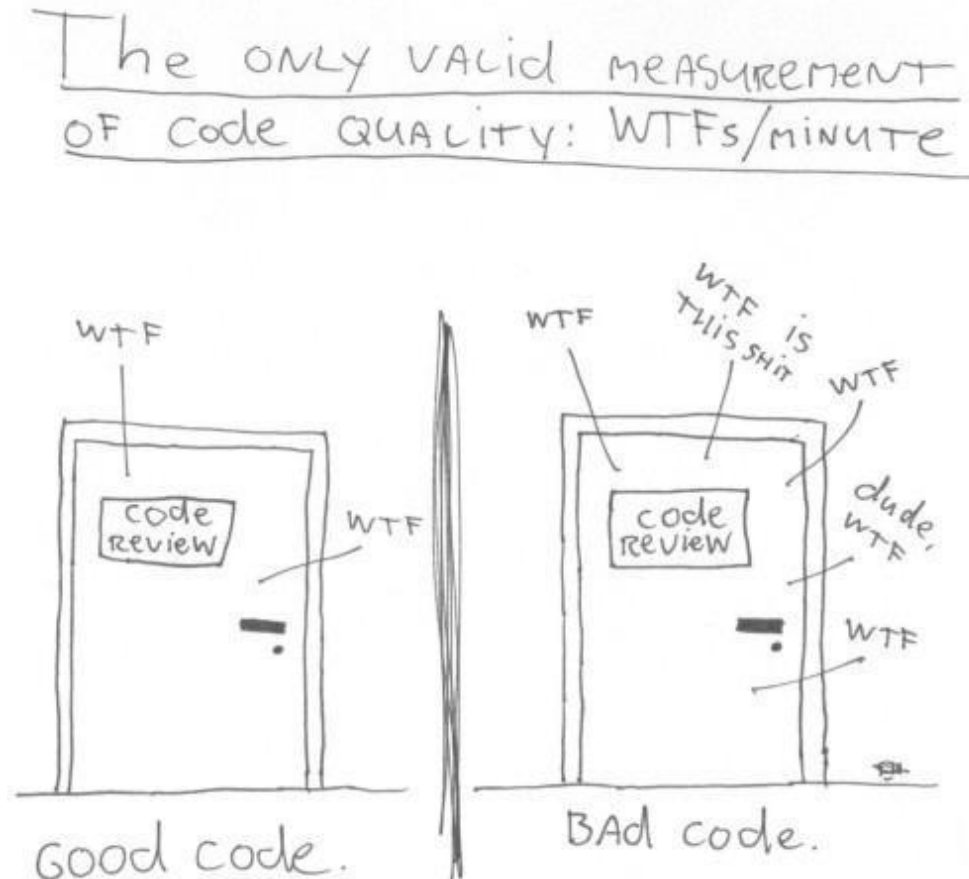
Khalil Stemmler  
@stemmlerjs



# CLEAN CODE

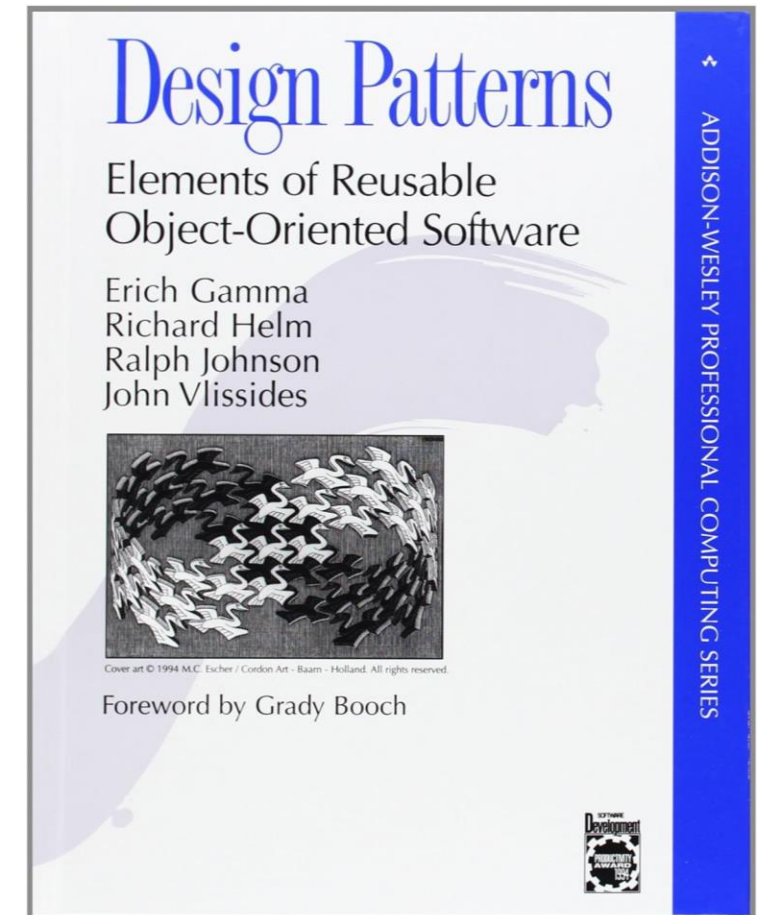
<https://refactoring.guru/>

- Refactoring
- Code Smells
- Design Patterns

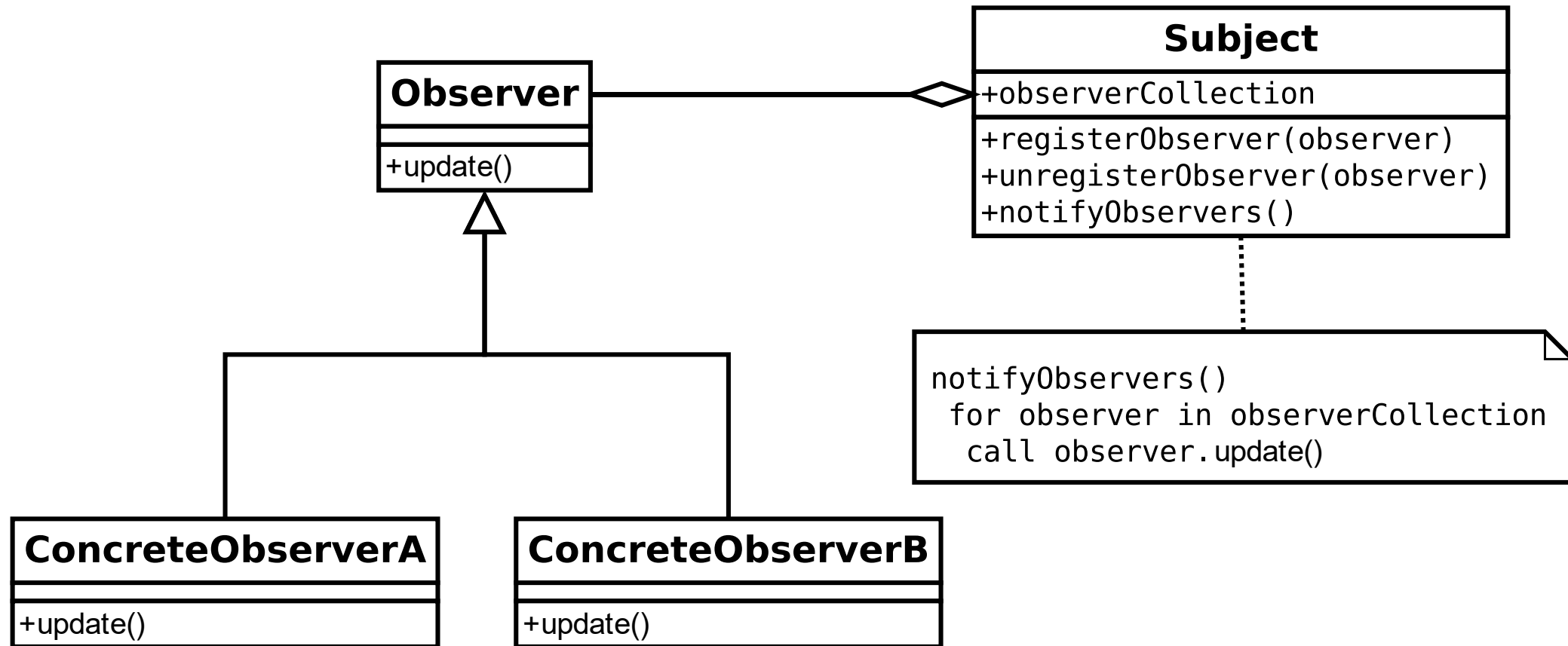


# DESIGN PATTERNS

- Eine **allgemeine, wiederverwendbare** Lösung für ein **häufig auftretendes** Problem innerhalb eines bestimmten Kontexts beim Softwaredesign
- Für **Skalierung** von Software unabdingbar



# OBSERVER PATTERN



# VIELEN DANK FÜR EURE AUFMERKSAMKEIT!

Bei weiteren Fragen kontaktiert mich bitte unter:



[cdeme@stud.hs-heilbronn.de](mailto:cdeme@stud.hs-heilbronn.de)