Inhaltsverzeichnis

1 Design & Konzept

In diesem Kapitel ...

1.1 Grundlegende Designentscheidungen

Bevor auf Entscheidungen eingegangen wird ...

1.1.1 Android als Plattform

Das Framework richtet sich ausschließlich an Entwickler die Applikationen für die Plattform Android entwickeln. Es ist damit nicht kompatibel zu iOS, dem Web oder Serverseitigen Anwendungen. Die Spezialisierung lässt es jedoch zu, besser auf mögliche Eigenheiten der Plattform einzugehen. Ein weiterer Grund für diese Entscheidung stellt die Tatsache dar, dass MVI seinen Anfang in der Entwickelung von Webseiten fand und es sich im Fall Android um einen Nachzügler handelt.

1.1.2 Framework

Warum Framwork? Besonderheiten...

1.1.3 Kotlin als Programmiersprache

Die Applikationen in Android und das Android-SDK selbst sind bis vor wenigen Jahren fast ausschließlich in der Sprache Java entwickelt wurden. Seit der Google I/0 2017 gehört jedoch eine weitere Sprache zu den offiziell unterstützen: Kotlin. Sie wird von dem Unternehmen Jetbrains entwickelt, die untere anderem die Entwicklungsumgebung Intellij für Java produzieren. Dieses bildet auch die Grundlage für Android Studio. Kotlin hat in den letzten Jahren an Bodenhaftung gewonnen und findet auch intern bei Google Verwendung.
Die Sprache wird als statisch typisierte, objektorientierte Programmiersprache bezeichnet und verfügt über eine hohe Interoperabilität zu Java. Dies bedeutet, dass innerhalb eines in Java geschriebenen Programms ohne viel Aufwand Kotlin genutzt werden kann. Dies ist ein wichtiger Faktor für die immer weiter ansteigende Beliebtheit, da es eine einfache Integration und bisherige Projekte gestattet. Kotlin bringt ein verbesserte Syntax mit und macht beispielsweise die Verwendung von null”explizit. Zu den Verbesserungen gehören dabei auch:

- Ableitung von Typen
- Alles ist eine Expression
- Funktionen sind First-Class-Funktionen”und bilden eine Funktionale Grundlage
- Datenklassen machen den Umgang mit unveränderliche Datenstrukturen einfach

- Erweiterungsfunktionen
- Kovarianz und Kontravarianz werden explizit anwendet
- Standardwerte für Parameter

Listing 1: Kotlin Beispiel

```kotlin
data class Example(
  privat val defaultMessage: String = "Hello World"
  privat val maybeNull: String? = null
){

  // Expression
  fun isHelloWorld() = when(message){
    "Hello World" -> true
    else -> false
  }

  // "?" findet bei null verwendung
  fun printIfNotNull() {
    maybeNull?.run {
     print(this)
    }
  }
}

// Erweiterungsfunktion und Funktion als Parameter
fun HelloWorld.extensionFunction(function: () -> String) {
 val message = function()
 println(message)
}

// kein new Schlüsselwort nötig
// keine Semikolon nötig
val example = Example()

// copy wird automatisch generiert bei einer "data" Klasse
val newExample = example.copy(defaultMessage = "New Message")

// ist der letzte Parameter eine Funktion, so kann auf Klammern
// verzichtet werden
newExample.extensionFunction { "Hello" }
```

Insgesamt is anhand Listing 1 zu erkennen, das Kotlin eine deutlich prägnantere und schlankere Syntax besitzt. Sie vermeidet damit einen großen Teil des mit Java verbundenen ”Boilerplate-Codes”und kann für eine höheren Grad an Produktivität sorgen. Besonders der Umgang von Null als Teil des Typsystems kann vor der berühmten NNullpointer-Exceptionretten. Des weiteren besteht ein größerer Fokus auf dem Einsatz von Konzepten aus der funktionalen Programmierung, welche durch Erweiterungsfunktionen für bswp. Listen zum Einsatz kommen.

1.2 Intent

Jeder Intention geht ein Ereignis voraus, das entweder vom Nutzer oder der Anwendung selbst initiiert wurde. Es stellt dabei den Einstieg in den von MVI definierten Kreislauf (aus Abbildung) dar.
Klickt der Nutzer in einer Anwendung auf einen ”Zurück”Knopf, so ist seine Intention zum vorherigen Bildschirm zurückzukehren oder die Anwendung zu beenden. Dieses Ereignis kann ohne weitere Informationen stattfinden. Anders ist es, wenn seitens des Nutzers innerhalb eine Liste ein Item ausgewählt wird und dessen Details gelistet werden sollen. Hierfür muss zusätzlich zu der eigentlichen Intention das ausgewählte Item (oder seine ID) übermittelt werden.
Daraus ergeben sich zwei Arten von ”Intents”: Eines ohne und eines mit zusätzlichen Nutzdaten (Englisch payload). Dies bedeutet, dass eine Struktur existieren muss, die entweder Daten beinhaltet oder nur eine semantische Bedeutung hat.

Listing 2: Intent Klasse

```
1 class Intent<T> (val payload: T)
```

Für diesen Fall eignet sich eine Klasse mit einem generischen Typ als Attribut, wie Listing 2 zeigt.Das «T>”in der Klassen Definition dient dabei als Platzhalter für den eigentlich Typ, z.B. für ein ”Item”aus dem obigen Beispiel.

Abbildungsverzeichnis