Von Boilerplate zu Minimalismus

Kotlin als zeitgemäße Alternative zu Java

David Grünberger
Master Interactive Technologies
Fachhochschule St. Pölten
it241515@fhstp.ac.at

5. Dezember 2024

Zusammenfassung

Kotlin hat sich in den letzten Jahren als eine beliebte Alternative zu Java etabliert, insbesondere in der mobilen Entwicklung aber auch darüber hinaus. Mit einem modernen Ansatz, der sich durch reduzierte Boilerplate-Syntax, Null-Sicherheit und fortschrittliche Sprachfeatures auszeichnet, konnte Kotlin eine beachtliche Akzeptanz erlangen. Der Artikel analysiert, wie Kotlin typische Schwächen von Java adressiert und beleuchtet Statistiken und Trends, die seine wachsende Verbreitung in der Softwareentwicklung unterstreichen.

1 Einleitung

Seit seiner Einführung im Jahr 1995 hat sich Java zu einer der dominantesten Programmiersprachen der Welt entwickelt. Mit seiner weitreichenden Anwendung in Bereichen wie Unternehmenssoftware, Webentwicklung und insbesondere der Android-Entwicklung bleibt Java ein entscheidender Bestandteil der Softwareentwicklung. Doch trotz seiner Beliebtheit wird Java häufig für seine hohe Komplexität und den umfangreichen Boilerplate-Code kritisiert, der die Entwicklung unnötig aufbläht und die Lesbarkeit des Codes beeinträchtigt. [1]

Die Programmiersprache Kotlin, die 2011 von JetBrains entwickelt wurde, ist als Antwort auf diese und andere Herausforderungen entstanden. Kotlin verfolgt das Ziel, eine moderne, prägnante und sichere Programmiersprache zu schaffen, die gleichzeitig mit bestehendem Java-Code vollständig interoperabel ist. Im Vergleich zu Java bietet Kotlin zahlreiche Verbesserungen, die den Entwicklungsprozess effizienter und fehlerfreier gestalten. [2]

Dieser Artikel untersucht, wie Kotlin als zeitgemäße Alternative zu Java genutzt werden kann, beleuchtet konkrete Verbesserungen durch die Sprache und zeigt, warum sich immer mehr Unternehmen und Entwickler/innen für Kotlin entscheiden.

2 Die wachsende Popularität von Kotlin

2.1 Statistische Verbreitung

Seit der offiziellen Unterstützung durch Google im Jahr 2017 hat Kotlin eine bemerkenswerte Erfolgsgeschichte geschrieben. Die Programmiersprache wird mittlerweile weltweit von Millionen Entwickler/innen genutzt und wächst kontinuierlich in ihrer Verbreitung. [3]

- Eine Studie von AppBrain ergab, dass über 80 % aller Android-Apps Kotlin verwenden, darunter prominente Anwendungen wie Pinterest, Netflix und Trello. [3]
- Im GitHub-Report State of the Octoverse 2024 wird Kotlin als eine der am schnellsten wachsenden Sprachen aufgeführt. Die Zahl der Repositories, die Kotlin nutzen, ist in den letzten fünf Jahren stark angestiegen. [4]
- Laut der Stack Overflow Developer Survey 2024 zählt Kotlin zu den Top-20-Programmiersprachen mit einer besonders hohen Zufriedenheitsrate unter Entwicklern: Mehr als 80 % der Befragten gaben an, gerne mit Kotlin zu arbeiten. [1]

Diese Statistiken zeigen, dass Kotlin sich nicht nur in der mobilen Entwicklung etabliert hat, sondern auch in anderen Bereichen wie der Backend-Entwicklung, Data Science und Multiplattform-Anwendungsentwicklung zunehmend genutzt wird.

2.2 Unternehmensadoption

Die wachsende Popularität von Kotlin lässt sich nicht nur in der breiten Entwicklergemeinschaft erkennen, sondern auch in der steigenden Nutzung durch namhafte Unternehmen in diversen Branchen.

Unternehmen wie Netflix, Slack, Trello und Square haben Kotlin in ihren Entwicklungsprozessen integriert, vor allem aufgrund seiner Vorteile hinsichtlich Produktivität und Code-Qualität. Netflix nutzt beispielsweise Kotlin Multiplatform für die Entwicklung der App Prodicle, die bei der Produktionsplanung für

TV-Shows und Filme hilft. Trello, eine beliebte Projektmanagement-Plattform, setzt Kotlin ein, um die Wartbarkeit und Weiterentwicklung ihrer mobilen Anwendungen zu erleichtern. Dank der ausdrucksstarken Syntax von Kotlin können Funktionen effizienter implementiert und aktualisiert werden. Square, ein Unternehmen für mobile Zahlungslösungen, nutzt Kotlin für seine Android-Anwendungen, wobei besonders die Sicherheits- und Lesbarkeitsvorteile geschätzt werden. [5] [6]

3 Boilerplate in Java: Ein Hindernis für Effizienz

Ein zentrales Problem von Java, das viele Entwickler/innen bei der Arbeit mit der Sprache frustriert, ist der hohe Anteil an Boilerplate-Code. Boilerplate-Code bezeichnet repetitiven und unnötig ausführlichen Code, der keine logische Funktionalität bietet, sondern lediglich für die Struktur oder den Ablauf der Anwendung notwendig ist. Ein einfaches Beispiel hierfür sind die Java-Datenklassen. Wenn man eine Klasse erstellen möchte, die als Datencontainer dient, muss man in Java typischerweise Konstruktoren, Getter- und Setter-Methoden sowie die *toString()*, *equals()* und *hashCode()* Methoden manuell implementieren. Dieser zusätzliche Aufwand führt zu einer erhöhten Codegröße, erschwert die Wartbarkeit des Codes und erhöht die Wahrscheinlichkeit von Fehlern.

Code-Beispiel 1: Eine einfache Datenklasse in Java

```java
public class User {
    private String name;
    private int age;

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() { return age; }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
         return
        "User{id="+id+",name='"+name+"'}";
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
        o.getClass()) return false;
        User user = (User) o;
        return age == user.age &&
        Objects.equals(name, user.name);
    }
    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

Das *Code-Beispiel 1* zeigt, wie viel Boilerplate-Code erforderlich ist, um selbst so eine einfache Datenklasse zu erstellen, die nur die Merkmale eines Users halten soll. Dies erschwert die Wartbarkeit und erhöht die Wahrscheinlichkeit von Fehlern. [7]

Kotlin adressiert dieses Problem im Vergleich durch eine prägnantere Syntax und durch Features, die es Entwickler/innen ermöglichen, den Code deutlich zu vereinfachen. So werden zum Beispiel in Kotlin's data classes die Methoden *equals()*, *hashCode()* und *toString()* automatisch generiert, ebenso wie Getter und Setter für Properties, die direkt in der Konstruktor-Annotation hinzugefügt werden können. [8]

Code-Beispiel 2: Eine einfache Datenklasse in Kotlin

```kotlin
data class User(val name: String, val age: Int)
```

4 Minimalistische Syntax in Kotlin

Ein zentrales Merkmal von Kotlin ist seine minimalistische und ausdrucksstarke Syntax, die darauf abzielt, den Code so kurz und prägnant wie möglich zu gestalten, ohne auf Lesbarkeit zu verzichten. Im Vergleich zu Java hat Kotlin eine weitaus kompaktere Syntax, die es Entwickler/innen ermöglicht, schneller und mit weniger Aufwand zu arbeiten.

Ein gutes Beispiel für die minimalistische Syntax in Kotlin ist die Verwendung von Variablen. In Java muss für jede Variable explizit der Typ angegeben werden und in vielen Fällen ist auch eine initiale Zuweisung erforderlich. In Kotlin hingegen wird der Typ automatisch aus dem Kontext abgeleitet, der geschriebene Code ist dadurch weitaus kürzer.

Code-Beispiel 3: Automatische Ableitung des Typs in Kotlin

```kotlin
var name = "Kotlin" // Typ String
val age = 5 // Typ Int
```

Im *Code-Beispiel 3* wird der Typ von name als *String* und der Typ von age als *Int* automatisch vom Compiler abgeleitet, basierend auf den zugewiesenen Werten. Diese Art der Variablendeklaration reduziert den Aufwand für Entwickler/innen und trägt zur Leserlichkeit des Codes bei. [9]

Ebenfalls wird dargestellt, dass es in Kotlin die Möglichkeit gibt, Properties als *immutable* zu kennzeichnen – durch die Angabe vom Keyword *val* wird eine erneute Zuweisung eines Wertes zu einem späteren Zeitpunkt verhindert. Die Werte von Properties, welche mit dem Keyword *var* instanziert wurden, können hingegen jederzeit wieder neu zugewiesen werden. [10]

5 Moderne Sprachfeatures in Kotlin

Neben der Reduktion von Boilerplate-Syntax bietet Kotlin auch moderne Sprachfeatures, die in Java gänzlich fehlen oder umständlich implementiert werden müssen.

5.1 Null-Sicherheit

Java ist bekannt für die häufig auftretende *NullPointerException* (NPE), da jedem beliebigen Typ zur Laufzeit grundsätzlich auch der Wert null zugewiesen werden kann. Dies kann zu unvorhergesehenen Fehlern führen und schwer zu debuggen sein. [11]

Kotlin löst dieses Problem durch eine strikte Trennung von nullable und non-nullable Typen. Dem Typ String kann in Kotlin beispielsweise standardmäßig nicht der Wert *null* zugewiesen werden, während ein Objekt vom Typ *String?* (ausgewiesen mit dem Fragezeichen am Ende des Typs) nullable ist. Jede potenzielle Null-Zuweisung wird vom Compiler erkannt, was sicherstellt, dass Entwickler das Problem bereits während der Entwicklung behandeln müssen. [12]

Code-Beispiel 4: Nullable-Typen in Kotlin

```
var name: String = "Kotlin"  // Non-nullable
var nickname: String? = null  // Nullable

fun printNickname(nickname: String?) {
    println(nickname?.toUpperCase() ?: "Unknown")
}
```

Im *Code-Beispiel 4* wird in der Funktion der Elvis-Operator (?:) verwendet, um auf einen potentiellen null-Wert vom übergebenen Parameter zu reagieren und in Folge einen Default-Wert zurückzugeben. [12]

5.2 Erweiterungsfunktionen (Extension Functions)

Extension Functions in Kotlin ermöglichen es, bestehende Klassen um neue Funktionen zu erweitern, ohne deren Code zu ändern. Dies ist besonders nützlich, wenn mit Bibliotheken gearbeitet wird, deren Quellcode nicht verändert werden kann oder soll. Erweiterungsfunktionen bieten eine elegante Möglichkeit, Funktionalitäten hinzuzufügen, ohne die Prinzipien der Objektorientierung zu verletzen. [4]

Code-Beispiel 5: Extension Function in Kotlin

```
fun String.capitalizeFirst(): String {
  return this.replaceFirstChar { it.uppercase() }
}
```

Im *Code-Beispiel 5* wird der String-Klasse eine neue Funktion hinzugefügt, das erste Zeichen zu einem Großbuchstaben umwandelt. Diese Funktion kann in Folge wie eine reguläre Methode auf jedem String-Objekt aufgerufen werden, was den Code viel sauberer macht, als eine separate Hilfsklasse dafür erstellen zu müssen. [4]

5.3 Coroutines : Moderne asynchrone Programmierung

Eine der herausragendsten Funktionen von Kotlin ist die Unterstützung von Coroutines, die eine moderne und elegante Lösung für asynchrone Programmierung darstellen. In Java sind Entwickler oft auf vergleichsweise komplexe Tools wie *Threads*, *CompletableFuture* oder *Executors* angewiesen, um parallele oder asynchrone Abläufe zu implementieren. Diese Ansätze können jedoch schnell zu unübersichtlichem und schwer wartbarem Code führen, insbesondere wenn mehrere Ebenen von Callbacks oder Abhängigkeiten ins Spiel kommen. Kotlin bietet mit Coroutines eine Lösung, die sowohl lesbarer als auch effizienter ist.

5.3.1. Funktionsweise von Coroutines

Coroutines ermöglichen die Implementierung von asynchronen Prozessen, ohne die lineare Lesbarkeit von Code zu beeinträchtigen. Statt neue Threads zu erzeugen, die ressourcenintensiv sind, nutzen Coroutines sogenannte Lightweight Threads, die sich auf einem einzigen Hintergrundthread mehrfach schachteln lassen. Dadurch wird die Effizienz massiv gesteigert, insbesondere in Umgebungen mit einer großen Anzahl von gleichzeitigen Aufgaben.

Eine Coroutine kann mit der Funktion launch gestartet werden, die im Kontext von Kotlin's Coroutines-Bibliothek definiert ist. Dabei kann der Ausführungskontext (z. B. Main-Thread oder Hintergrund-Thread) präzise gesteuert werden. [13]

Code-Beispiel 6 : Starten einer Coroutine in Kotlin

```
import kotlinx.coroutines.*

fun main() = runBlocking {
    launch {
        delay(1000L)
        // Simuliert eine Hintergrundaufgabe

        println("Hello from Coroutine!")
    }
    println("Main Thread Execution")
}
```

Im *Code-Beispiel 6* startet launch eine neue Coroutine, die asynchron arbeitet. Die Funktion *delay()* pausiert die Coroutine für die angegebene Zeit, ohne den Main-Thread zu blockieren. Im Gegensatz zu *Thread.sleep()* wird dadurch keine unnötige Blockierung erzeugt.

5.3.2. Schlüsselkonzepte in Coroutines

Suspend Functions

Eine suspend-Funktion ist eine spezielle Funktion, die eine Coroutine pausieren und später wieder fortsetzen kann. Diese Funktionen arbeiten nahtlos mit anderen Coroutines zusammen und ermöglichen eine strukturierte Handhabung asynchroner Abläufe. [14]

Code-Beispiel 7: Suspending Function in Kotlin

```
suspend fun fetchData(): String {
    delay(2000L)
    // Simuliert einen Netzwerkaufruf

    return "Data fetched"
}
```

Scopes und Dispatcher

Coroutines werden immer in einem Scope gestartet. Der Scope definiert, wie lange eine Coroutine aktiv bleibt, und kann verwendet werden, um Coroutines bei Bedarf abzubrechen. Dispatcher legen fest, welcher Thread oder Thread-Pool für die Coroutine genutzt wird, z. B. Dispatchers.IO für I/O-Operationen. [15]

Code-Beispiel 8 : Coroutine Scope und Dispatcher

```
val scope = CoroutineScope(Dispatchers.IO)
scope.launch {
    val result = fetchData()
    println(result)
}
```

6 Fazit

Kotlin hat sich seit seiner Einführung als ernstzunehmende Alternative zu Java etabliert. Durch die Kombination aus minimalistischer Syntax, innovativen Sprachfeatures und einer starken Fokussierung auf Entwickler/innenfreundlichkeit adressiert Kotlin viele der Schwächen, die in Java über Jahre hinweg als hinderlich empfunden wurden. Insbesondere in Bereichen wie der Null-Sicherheit, der funktionalen Programmierung und der Unterstützung für asynchrone Programmierung mit Coroutines hebt sich Kotlin deutlich ab.

Die Reduktion von Boilerplate-Code steigert die Produktivität und Lesbarkeit, was nicht nur Zeit spart, sondern auch die Wahrscheinlichkeit von Fehlern reduziert. Die Einführung von modernen Konzepten wie Extension Functions und Coroutines zeigt, dass Kotlin nicht nur darauf abzielt, bestehende Probleme zu lösen, sondern auch zukünftige Anforderungen der Softwareentwicklung proaktiv zu adressieren.

Die steigende Popularität der Sprache – insbesondere in der Android-Entwicklung, aber auch darüber hinaus – unterstreicht ihren Erfolg. Unternehmen, die Kotlin einführen, berichten von gesteigerter Zufriedenheit ihrer Entwicklerteams sowie von einer signifikanten Reduktion des Wartungsaufwands, da der Code schlanker und robuster wird.

Kotlin zeigt, dass moderne Programmiersprachen nicht nur effizient und funktional, sondern auch intuitiv und zugänglich sein können. Obwohl Java weiterhin ein bedeutender Akteur in der Welt der Programmierung bleibt, positioniert sich Kotlin als dessen natürliche Weiterentwicklung, die sowohl die Gegenwart als auch die Zukunft der Softwareentwicklung prägen könnte.

Literatur

[1] „2024 Stack Overflow Developer Survey“. Zugegriffen: 2. Dezember 2024. [Online]. Verfügbar unter: https://survey.stackoverflow.co/2024/

[2] „FAQ | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/faq.html

[3] „Kotlin - Android SDK statistics“, AppBrain. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://www.appbrain.com/stats/libraries/details/kotlin/kotlin

[4] G. Staff, „Github Octoverse 2024“, The GitHub Blog. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://github.blog/news-insights/octoverse/octoverse-2024/

[5] „Top Apps Built with Kotlin Multiplatform [2023 Update]“. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://www.netguru.com/blog/top-apps-built-with-kotlin-multiplatform

[6] „Who Uses Kotlin? Exploring the Popularity and Adoption of Kotlin in the Development Community“. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://www.techstackk.com/programming/kotlin/who-uses-kotlin?pid=118

[7] „What Is a Pojo Class? | Baeldung“. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://www.baeldung.com/java-pojo-class

[8] „Data classes | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/data-classes.html

[9] „Type inference - Kotlin language specification“. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/spec/type-inference.html

[10] „Properties | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/properties.html

[11] „Null Pointer Exception In Java“, GeeksforGeeks. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://www.geeksforgeeks.org/null-pointer-exception-in-java/

[12] „Null safety | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/null-safety.html

[13] „Coroutines | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/coroutines-overview.html

[14] „Composing suspending functions | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/composing-suspending-functions.html

[15] „Coroutine context and dispatchers | Kotlin“, Kotlin Help. Zugegriffen: 5. Dezember 2024. [Online]. Verfügbar unter: https://kotlinlang.org/docs/coroutine-context-and-dispatchers.html