



Benjamin Schmid
Technology Advisor

@bentolor

Moderne Android-Entwicklung mit Kotlin

2017-01-07

? for help

WAS IST KOTLIN?

WAS IST KOTLIN?

Russische Insel
im finnischen Golf, 32km vor Petersburg

Russische Insel

im finnischen Golf, 32km vor Petersburg



statisch typisierte
Allzweck-Sprache
für JVM und Browser



PRÄGNANZ
SICHERHEIT
LESBARKEIT
INTEROPERABILITÄT
TOOLING

DATENBLATT

Freie Software

APL 2.0 – IDE, compiler, libs, build tools

backed by JetBrains

20 Vollzeit & Community
„dogfooding“: 0,5 Mio. LOC

Stabil

Start: ~2010, v1.0 Feb. '16
Long-term backward compatibility

Integrationen

IntelliJ, Eclipse
Maven, Ant, Gradle

PRÄGNANZ & LESBARKEIT

Schneller an's Ziel!

PRÄGNANZ VS. LESBARKEIT

```
import java.util.*  
  
fun main(args: Array<String>) {  
    val name = if (args.size > 0) args[0] else "Publikum"  
    val zuschauer = Gast(name, title = Title.wertes)  
  
    println(zuschauer)  
    println("Hallo ${zuschauer.title} ${zuschauer.name}") ← String interpolation  
}  
  
data class Gast(val name: String,  
               var zeit: Date = Date(),  
               val title: Title?)  
enum class Title { Herr, Frau, wertes }
```

← Optional semicolons
← Top-level functions
← Type inference & named params
← String interpolation
← Data classes
← Default values
← **val** ≡ **public final**

TYPE INFERENCE

```
fun noTypeInfer(vals: List<Int>): Unit {  
    val distinct: Set<Int> = vals.toSet()  
    val first: Int = vals[0]  
    val negs: List<Int> = vals.filter(  
        fun(n: Int): Boolean {  
            return n < 0  
        }  
    )  
}
```



```
fun typeInfer(vals: List) {  
    val distinct = vals.toSet()  
    val first = vals[0]  
    val negs = vals.filter(  
        fun(n) = n < 0  
    )  
}
```

Typangaben üblicherweise **nur an Schnittstellen** erforderlich

DATA CLASSES

```
data class Gast(  
    val name: String,  
    var zeit: Date = Date(),  
    val title: Title?  
)
```

Werte-Container

- Um Methoden erweiterbar
- Vererbung ab Kotlin 1.1

Kotlin übernimmt

- Getter & Setter
- hashCode()
- equals()
- copy()
- toString()
- componentN()

... exakt dasselbe mit Java

```
public final class Gast {  
    @NotNull  
    private final String name;  
    @NotNull  
    private Date zeit;  
    @Nullable  
    private final Title title;  
  
    @NotNull  
    public final String getName() {  
        return this.name;  
    }  
  
    @NotNull  
    public final Date getZeit() {  
        return this.zeit;  
    }  
  
    public final void setZeit(@NotNull Date date) {  
        checkParameterIsNotNull(date, "<set-?>");  
        this.zeit = date;  
    }  
  
    @Nullable  
    public final Title getTitle() {  
        return this.title;  
    }  
  
    public Gast(@NotNull String name,  
               @NotNull Date zeit,  
               @Nullable Title title) {  
        checkParameterIsNotNull(name, "name");  
        checkParameterIsNotNull(zeit, "zeit");  
        this.name = name;  
        this.zeit = zeit;  
        this.title = title;  
    }>
```

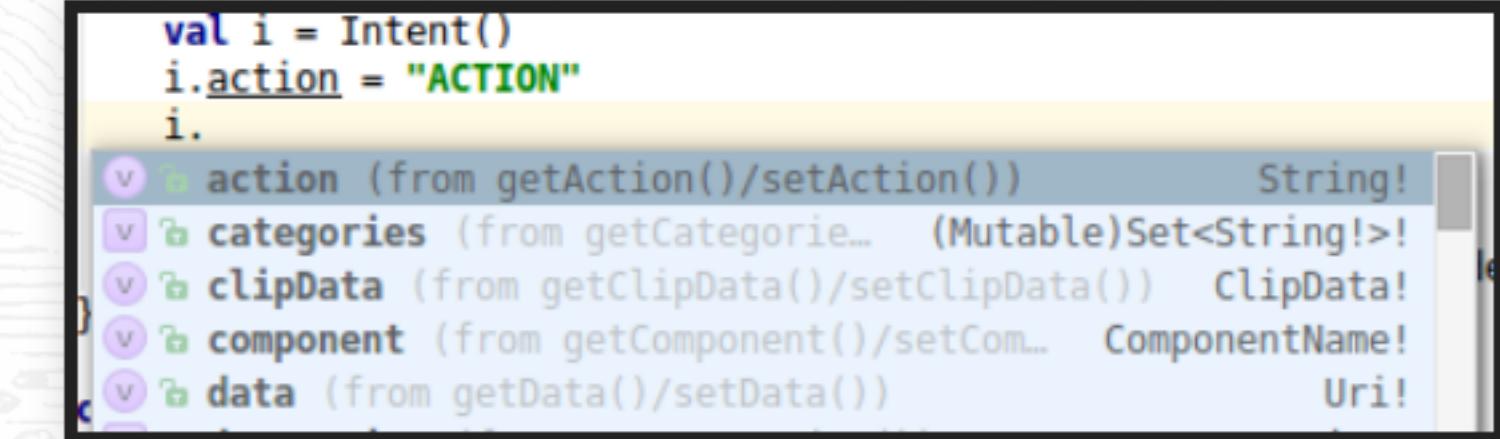
```
public Gast(String string, Date date,  
           Title title, int n) {  
    if ((n & 2) != 0) {  
        date = new Date();  
    }  
    this(string, date, title);  
}  
  
@NotNull  
public final String component1() {  
    return this.name;  
}  
  
@NotNull  
public final Date component2() {  
    return this.zeit;  
}  
  
@Nullable  
public final Title component3() {  
    return this.title;  
}  
  
@NotNull  
public final Gast copy(@NotNull String name,  
                      @NotNull Date zeit,  
                      @Nullable Title title) {  
    checkParameterIsNotNull(name, "name");  
    checkParameterIsNotNull(zeit, "zeit");  
    return new Gast(name, zeit, title);  
}  
  
public String toString() {  
    return "Gast(name=" + this.name +  
           ", zeit=" + this.zeit +  
           ", title=" + this.title + ")";  
}
```

```
public int hashCode() {  
    String s = this.name;  
    Date d = this.zeit;  
    Title t = this.title;  
    return ((s != null ? s.hashCode() : 0) * 31  
           + (d != null ? d.hashCode() : 0)) * 31  
           + (t != null ? t.hashCode() : 0);  
}  
  
public boolean equals(Object object) {  
    if (this == object) return true;  
  
    if (!(object instanceof Gast))  
        return false;  
  
    Gast gast = (Gast)object;  
    if (!areEqual(this.name, gast.name)  
        || !areEqual(this.zeit, gast.zeit)  
        || !areEqual(this.title, gast.title))  
        return false;  
  
    return true;  
}
```

... 3 Attribute!

PROPERTIES

- Klassen in Kotlin kennen **keine** Felder, nur Properties
- Compiler **generiert** Getter- & Setter sowie Backing Field
- **getX()**/**setX()**-Paare aus **Java** erscheinen als Property **x**
- Properties unterstützen **Delegation**



AUSDRÜCKE & KONVENTIONEN

Ausdrucksstärke

Mehr Ausdrücke (`if`, `?:`, `?.`, ...)

Pattern matching (`when`)

String Templates

Syntactic Sugar

`listOf(...)`, `repeat(3){...}`

`with(x){...}`, `x.apply{...}`, ...

`public` ist Standard

kein `new`; `;` optional

Operatoren & Dekomposition über Namen

Standard-Bibliothek

Konventionen

SICHERHEIT

БЕЗОПАСНОСТЬ

FIRST-CLASS IMMUTABLES

Nur-lesbare Variablen aufwandsfrei durch `val` statt `var`

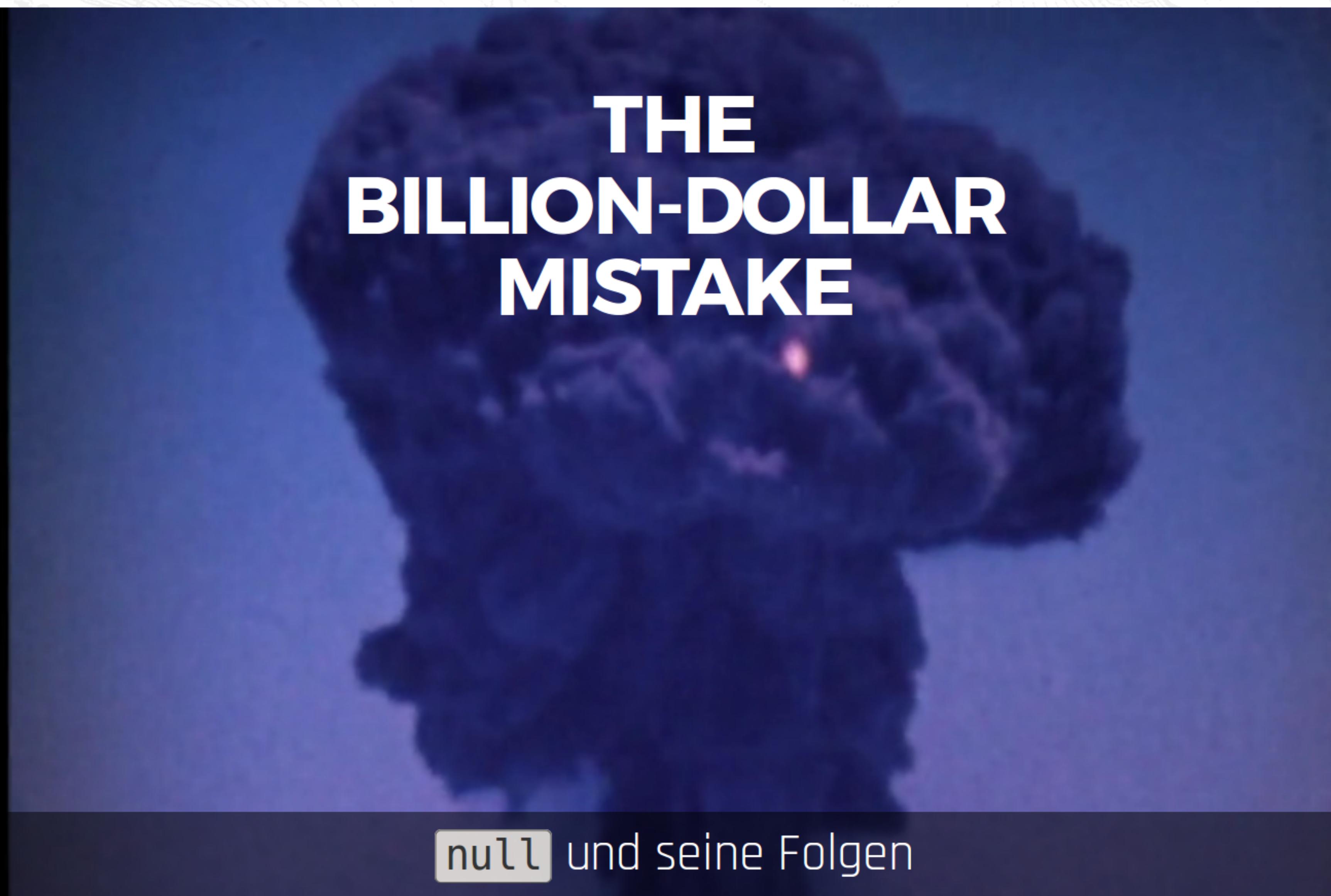
Standard-Collection Interfaces (`List`, `Set`, ...) sind read-only

Zum ändern muss man z.B. `MutableList` statt `List` nutzen

Klassen & Methoden sind `final` by Default

```
open class ExtendableClass {  
    open fun overridableMethod() { ... }  
}
```

Mutige Entscheidung!
... Framework-Entwickler
müssen Acht geben!



THE BILLION-DOLLAR MISTAKE

null und seine Folgen

NULLWERT-SICHERHEIT

Nur *explizit Nullwert-fähige Typen* akzeptieren `null`.
Ungeprüfte Zugriffe führen zu **Compile-Fehlern!**

```
var nullable: String? = null
var nonNullable: String = "Hi!"

// Ab hier compiliert nichts mehr ...
nonNullable = nullable
nonNullable = null
nullable.isEmpty()
```

Nullwerte sind Bestandteil des Typsystems!

An den Schnittstellen baut Kotlin Prüfungen & `@NotNull` Annotationen ein.

Effizienter Umgang mit nullwert-Typen

Safe navigation operator

```
nullable = nullable?.toUpperCase()
```

Elvis operator

```
nonNullable = nullable ?: ""
```

Kotlin extension für Nullable-Typen

```
// equals/orEmpty für String?  
nonNullable = nullable.orEmpty()  
val b = nullable.equals("foo")
```

Smartcast

```
if (nullable != null) {  
    nonNullable = nullable.trim()  
}
```

Ich-weiss-es-besser™

```
//NPE-Gefahr voraus!  
nullable!!!.trim(3)  
nonNullable = nullable!!
```

KOTLIN | & ANDROID |



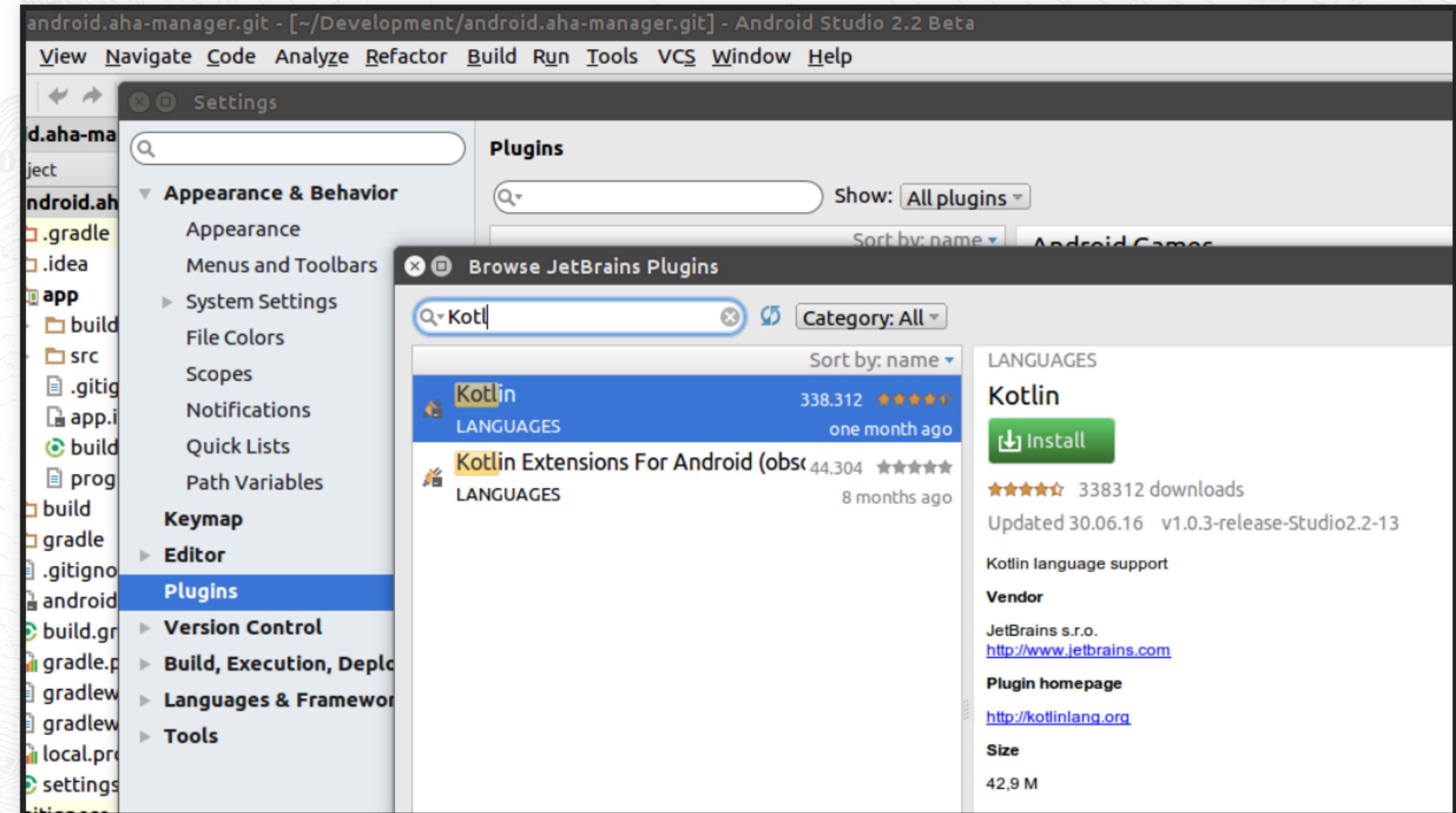
БРОУЗАТОРЫ



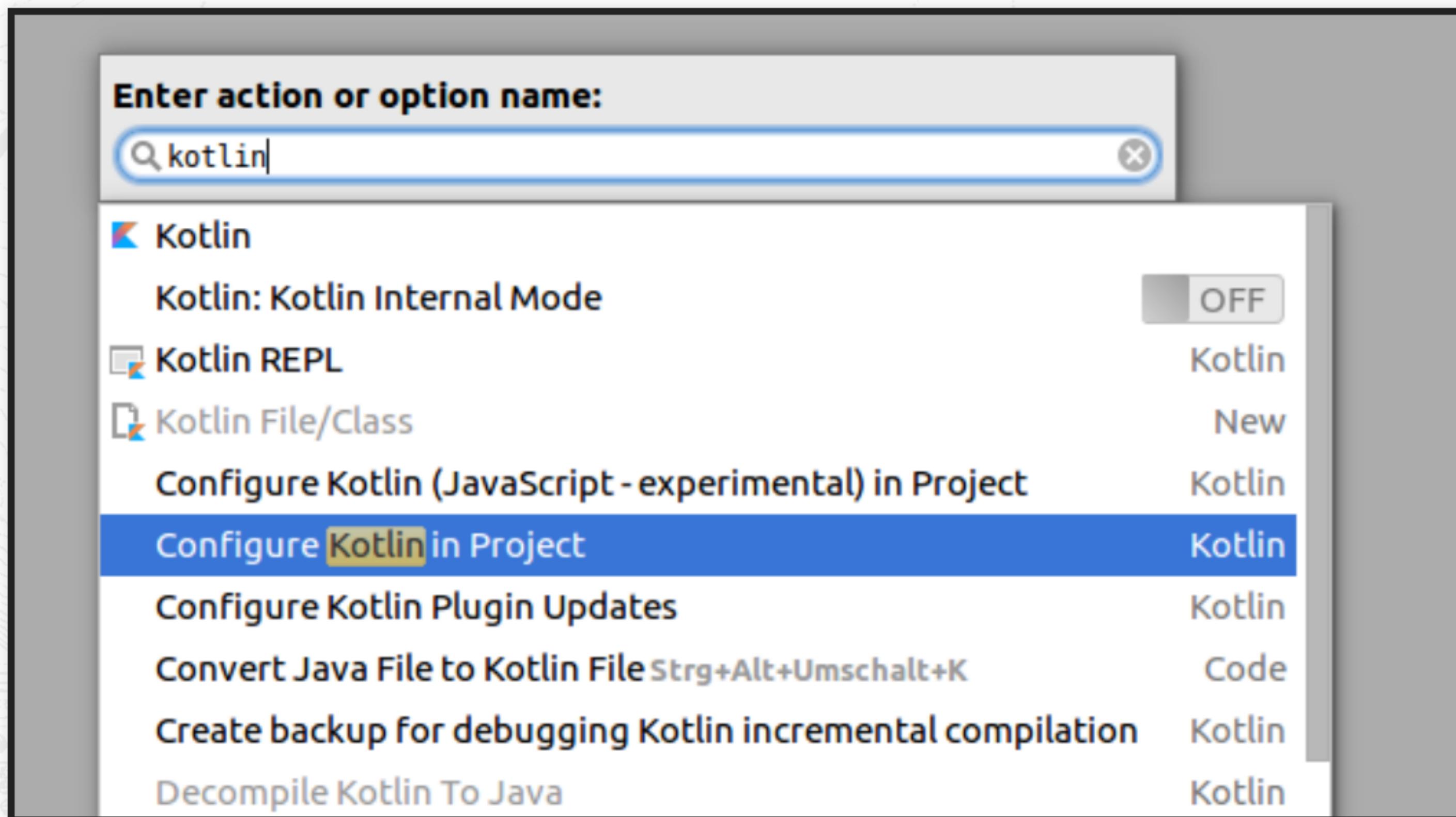
+ - EIN DREAM-TEAM?

- **effiziente** & **fehlerarme** Entwicklung – wie vielfältig illustriert
- konzipiert für den **industriellen Einsatz**
- erzeugt **Java 6 Bytecode** – und damit perfekt Android-kompatibel
- **100% Java Interoperabilität** – in beide Richtungen ermöglicht ...
- klassenweises **Mischen von Java & Kotlin**-Code
- **Tooling-Heimspiel** – Android Studio & Gradle 3 setzen auf Kotlin
- Geringe Runtimegröße & -overhead – ca. 736KB in 1.0.6

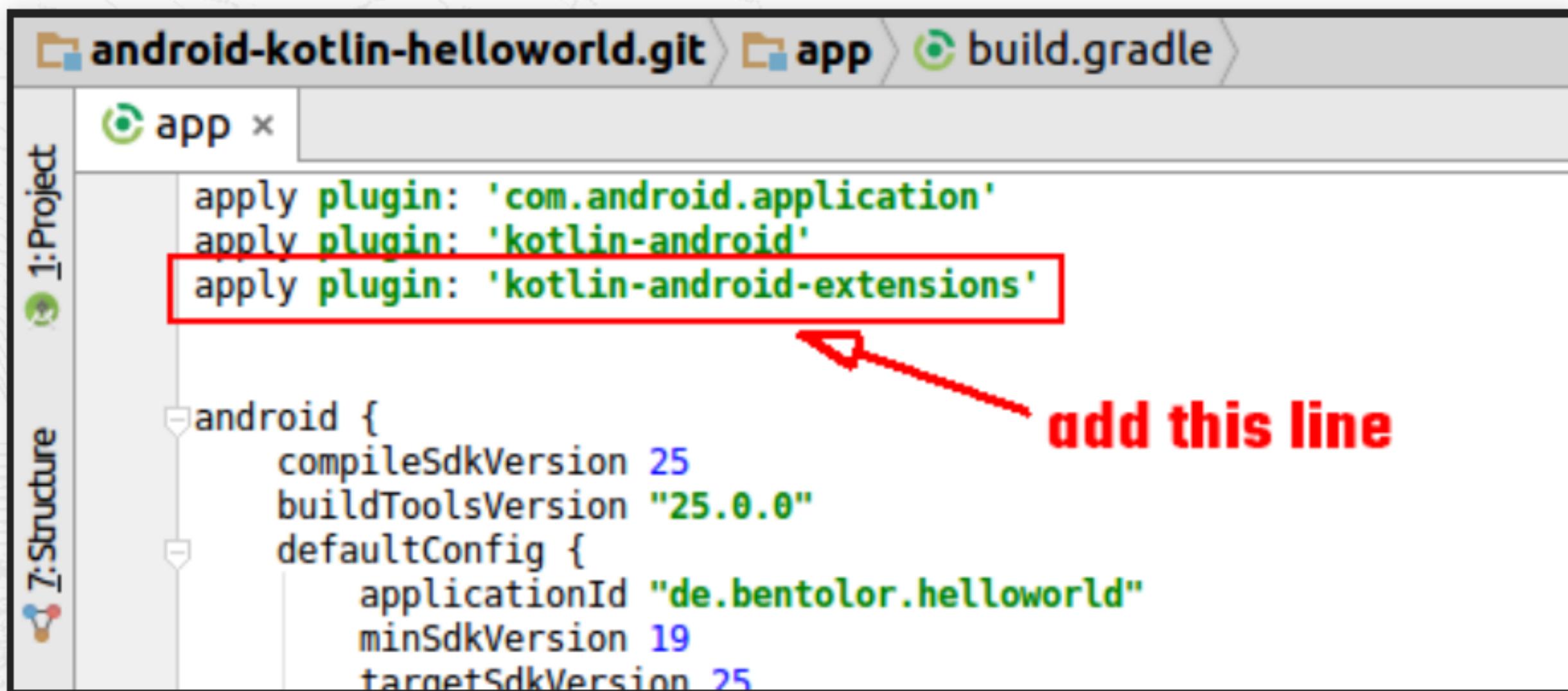
1. Android Studio Plugin installieren



2. Kotlin im Projekt aktivieren



3. Kotlin Extensions aktivieren (optional)

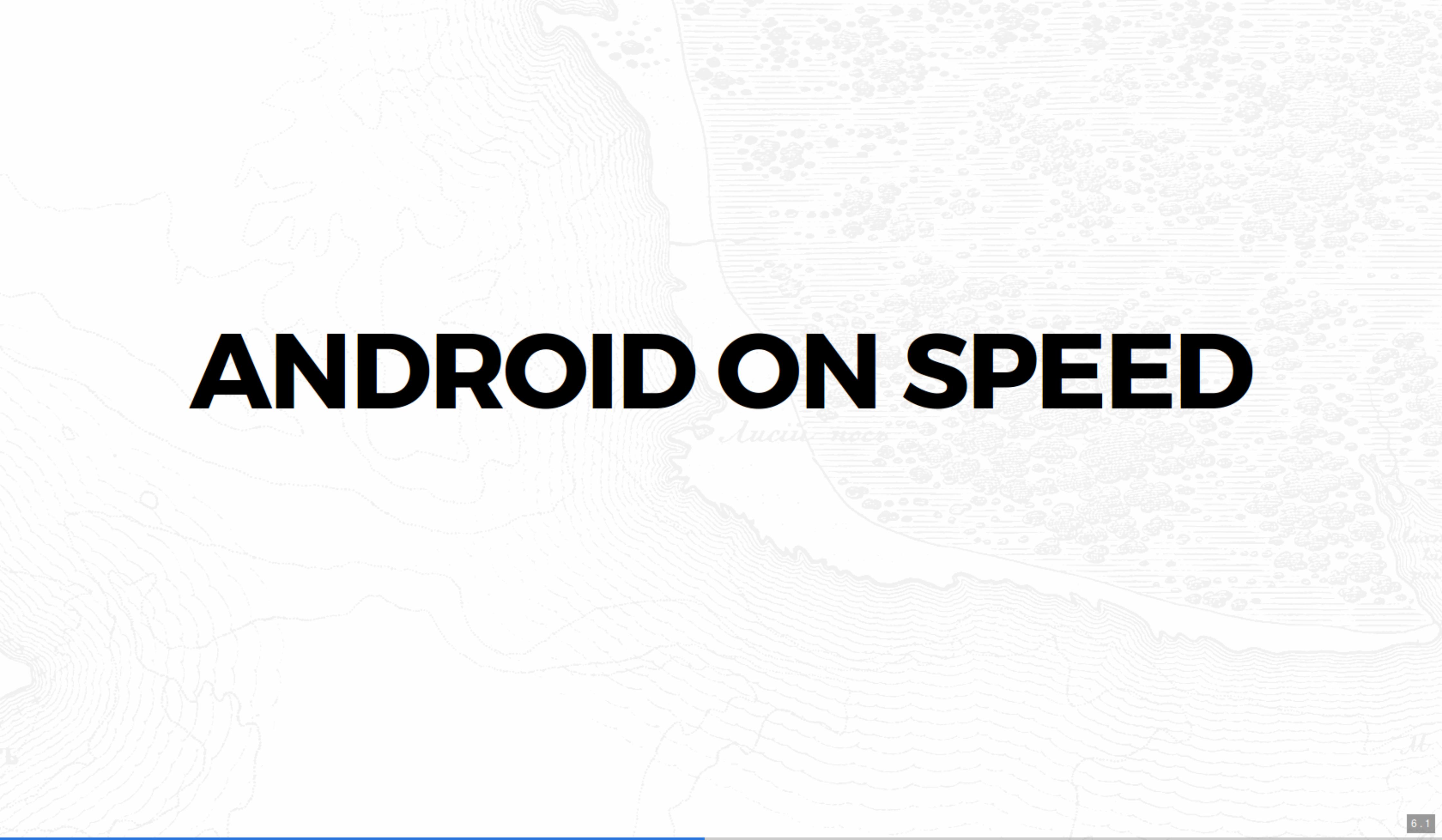


android-kotlin-helloworld.git app build.gradle

app

```
apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-android-extensions'  
  
android {  
    compileSdkVersion 25  
    buildToolsVersion "25.0.0"  
    defaultConfig {  
        applicationId "de.bentolor.helloworld"  
        minSdkVersion 19  
        targetSdkVersion 25
```

add this line



ANDROID ON SPEED

Typischer Android-Boilerplate ...

```
EditText editTitle = (EditText) v.findViewById(R.id.edit_title);
editTitle.setText(mItem.getTitle());

CheckBox enabledBox = (CheckBox) v.findViewById(R.id.enable_box);
enabledBox.setChecked(true);

Button createButton = (Button) v.findViewById(R.id.create_entry);
createButton.setOnClickListener(new OnClickListener() {
    @Override public void onClick(View button) {
        createElement();
    }
});
```

... der Zugriff auf die Views in den Activities oder
Fragments über die `findViewById()`-Methode:

VIRTUELLE PROPERTIES FÜR UI-ELEMENTE

Das **Kotlin Android Extensions** erzeugt virtuelle `mylayout.xml`-Pakete

```
import kotlinx.android.synthetic.main.mylayout.*
```

```
edit_title.setText(mItem.title)
enable_box.isChecked = true
create_entry.setOnClickListener { createElement() }
```

Ein einfacher Import erlaubt **typsicherer** Zugriff auf alle darin enthaltene View-Elemente direkt als simples Property.

Ohne zusätzlichen Code, Annotations oder Runtime!

LAMBDA-AUSDRÜCKE

```
view.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        doSomething(view);  
    }  
};
```



```
view.setOnClickListener({ view -> doSomething(view) })
```

EXTENSION METHODS / PROPERTIES

- ... erlauben das Erweitern fremder Klassen (Android, JRE)
- **this** zeigt auf das erweiterte Objekt

```
// Extension method für ein Android Fragment
fun Fragment.toast(message: String, duration: Int = Toast.LENGTH_SHORT) {
    Toast.makeText(this.getActivity(), message, duration).show()
}
```

```
// Extension property für das Android EditText Widget
var EditText.stringValue: String
    get() = text.toString()
    set(str) { setText(str) }
```

```
// Verwendung
toast("Hallo Zuschauer!")
nameField.stringValue = "Max Muster"
```

EXTENSIONS „FREI HAUS“

Zahlreiche Extensions sind in der Kotlin Standardbibliothek bereits enthalten.
Sie **erweitern Java Klassen** und sind **komfortabel über Code-Completion** zugänglich.

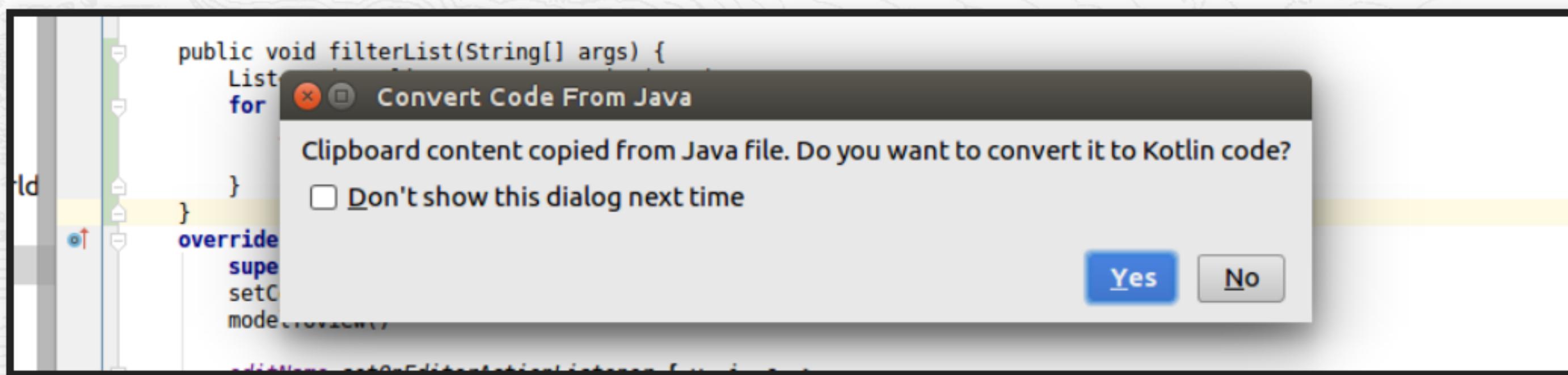
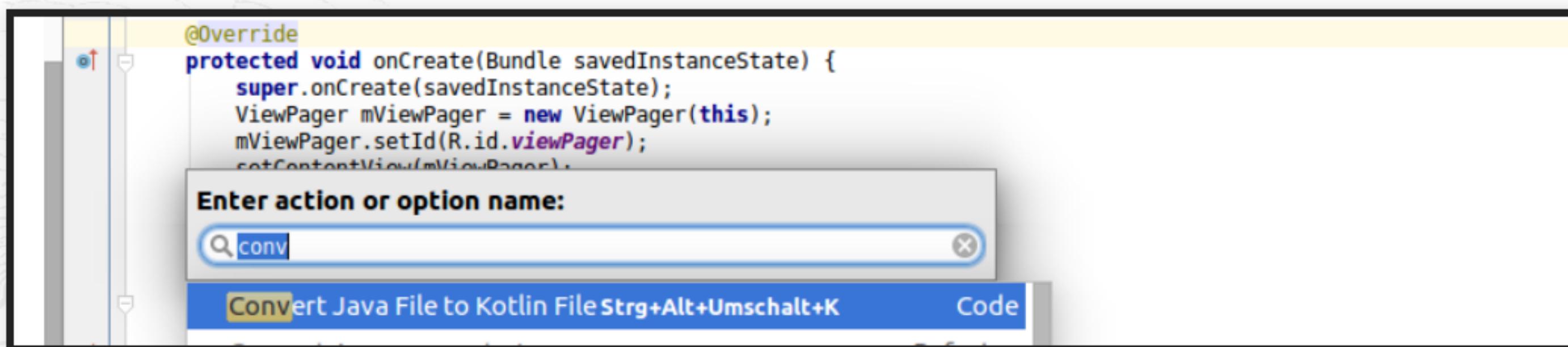
```
List<String> list = Arrays.asList("a", "b", "c");
for (int index = 0; index < list.size(); index++) {
    if (index % 2 == 0)
        System.out.println(index + " -> " + list.get(index));
}
```

Extensions helfen wesentlich beim **prägnanten**, dennoch **intuitiv lesbar** Code.

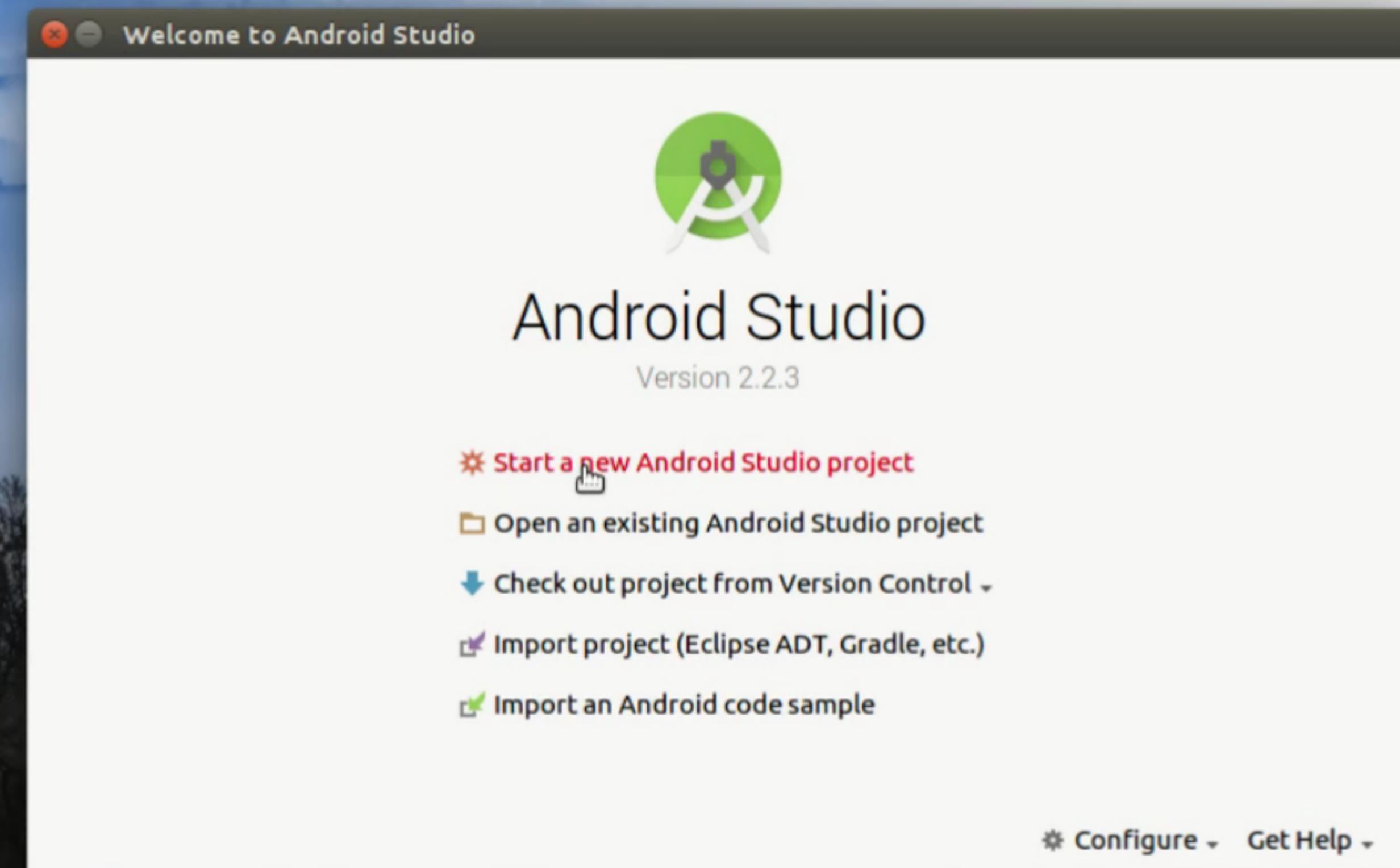
```
val list = listOf("a", "b", "c")
for (index in list.indices.filter { it % 2 == 0 }) {
    println("$index -> ${list[index]}")
```

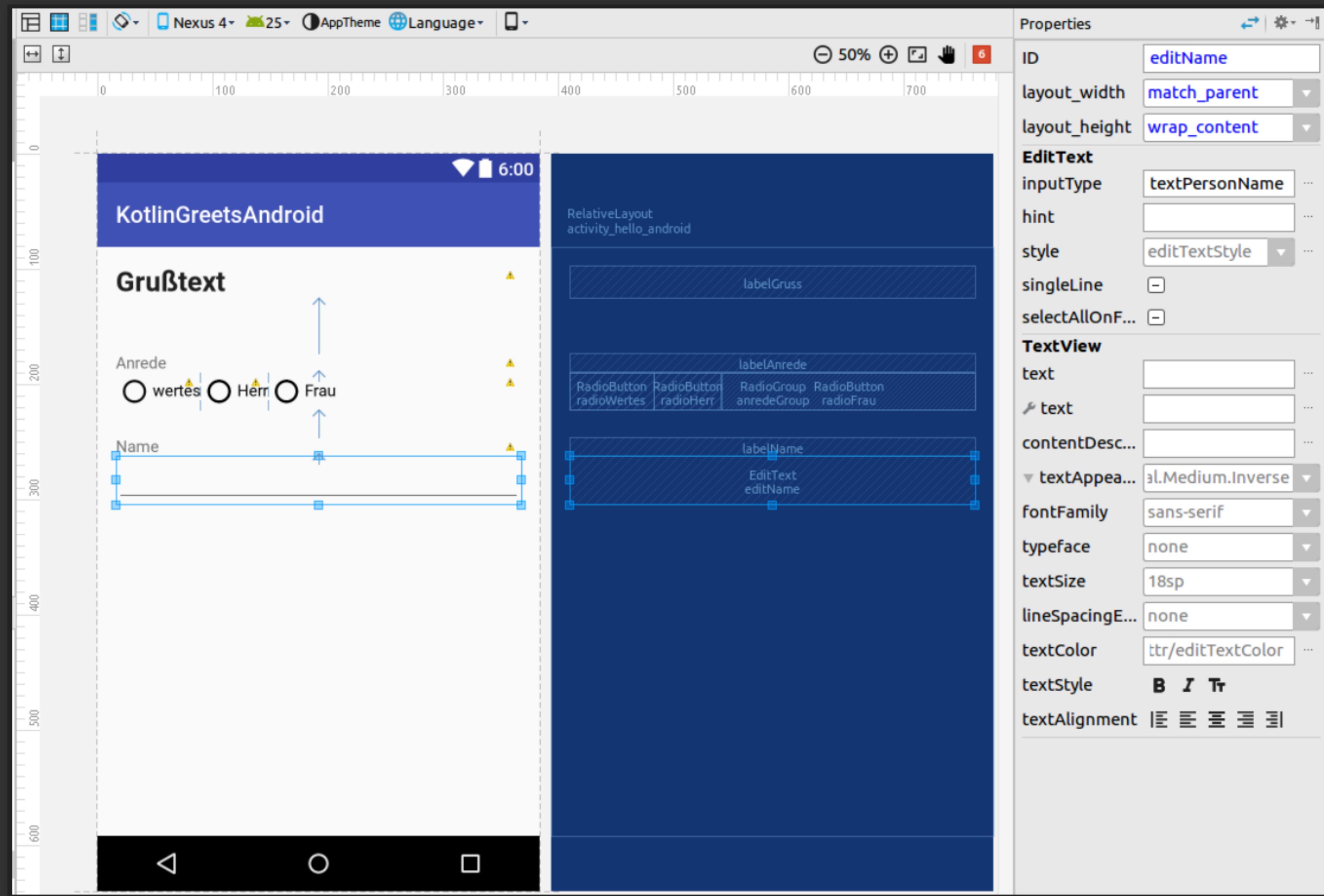
Im Beispiel: `listof`, `indices`, `filter`. Auch Bibliotheken für Android ([Anko](#))!

Automatische Konvertierung Java → Kotlin



DEMO





Hello Android: Code part #1

```
package de.bentolor.kotlingreetsandroid

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_hello_android.*
import java.util.*

class HelloAndroid : AppCompatActivity() {

    enum class Titel { wertes, Herr, Frau, Doktor }

    data class Gast(var titel: Titel = Titel.wertes,
                   var name: String,
                   val geburt: Date? = null) {
        val anrede: String
            get() = "Hallo $titel $name!"
    }

    private val gast = Gast(name = "Publikum")
```

Hello Android: Code part #2

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_hello_android)
    modelToView()

    editName.setOnEditorActionListener {...}
    anredeGroup.setOnCheckedChangeListener { group, id ->
        gast.titel = when(id) {
            radioHerr.id -> Titel.Herr
            radioFrau.id -> Titel.Frau
            else -> Titel.wertes
        }
        modelToView()
    }
}

fun modelToView() {
    labelGruss.text = gast.anrede
}
```



DRAWBACKS

Die (wenigen) Schattenseiten

64K Methoden Limit

- Dalvik Limit: 65.536 Methoden pro DEX file.
 - Jedes Kotlin Properties = +2 Methoden
 - Abhilfe: Multidex & `@JvmField`

Overhead (gering)

- IDE: Performance der Auto-completion (minimal)
- APK Größe: ~736KB & Checks
- Compilerlaufzeit im ersten Durchgang (~15%)

Auf Java abzielende Frameworks

- Schöne API umständlicher; besser Java nutzen

Android & **null** - Typische Konstellation

Werte für Initialisierung erst im Android-Callbacks bekannt ...

```
class EditPersonActivity : FragmentActivity() {  
  
    var person: Person  
    var anrede: String  
    var imm: InputMethodManager  
    ...  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        val personId = savedInstanceState?.getInt("value", 0)  
  
        person = PersonManager.load(id)  
        anrede = resources.getString(R.string.anrede)  
        imm = getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager  
    }  
}
```

So nicht möglich!

Lösungen: Verzögerte Initialisierung

lateinit Properties (empfohlen)

```
lateinit var person: Person
```

Wirft sprechende Exception,
falls ohne Initialisierung
gelesen wird!

Delegation des Properties

```
var anrede: String by Delegates.notNull()
```

Lazy-Initialisierung

```
val imm: InputMethodManager by lazy {  
    getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager  
}
```



-ADVANCED

FUNKTIONEN ALS FIRST-CLASS ELEMENTS

Funktionen sind Sprachelemente erster Klasse und können daher auch als **Variablen, Properties, Parameter** und **Rückgabewerte** genutzt werden.

Durch Verschachtelung lassen sich **Funktionen höherer Ordnung** (*Higher-order functions*) erschaffen

Higher-order functions – Beispiel

```
class FirstClassFunction(  
    val f1: (String) -> Int,  
    val f2: (Int) -> Boolean) {  
  
    fun strToBool(str: String): Boolean {  
        val f: (String) -> Boolean = higherOrderFun()  
        return f(str)  
    }  
  
    private fun higherOrderFun(): (String) -> Boolean {  
        return { x -> f2(f1(x)) }  
    }  
  
}  
  
fun main(args: Array<String>) {  
    val c = FirstClassFunction(  
        Integer::parseInt,  
        { it % 2 == 0 }  
    )  
    val strings = listOf("2", "7", "8")  
    println(strings.filter(c::strToBool)) // Kotlin 1.1+  
}
```

← Funktion als Parameter

← Funktion als Variable

← Funk. als Rückgabewert

← Kombination via Lambda

← Funk.-Referenz

← Lambda-Ausdruck

→ Was kommt raus?

Typsichere DSLs über „Empfänger“-Objekt

Es ist möglich, Funktionssignaturen zu definieren, die Lambda-Ausdrücke erwarten bei denen `this` auf Zielobjekte eines bestimmten Typs zeigen.

Darüber sind typsichere DSLs möglich:

```
class HTML {  
    fun body() { ... }  
}  
  
fun html(init: HTML.() -> Unit): HTML {  
    val html = HTML()  
    html.init()  
    return html  
}  
  
html {  
    body()  
}
```

- ← Erwartet Funktion mit `this` vom Typ `HTML`
- ← Erstellung des *receiver object*
- ← Führe Lambda auf *receiver object* aus

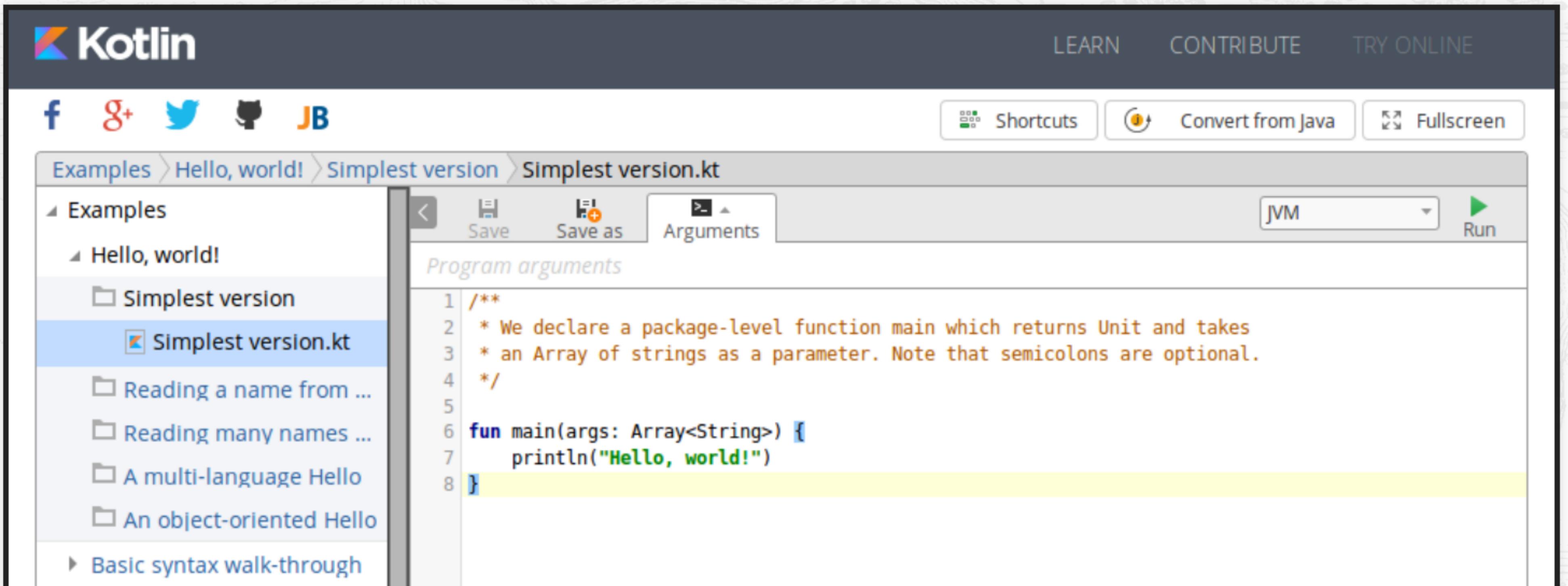
- ← Kurzform von `html({...})`
- ← **Lambda**-Ausdruck für einen `HTML`-„Empfänger“

Anko: Android-DSL für XML

```
relativeLayout {  
    layoutParams(width = matchParent, height = matchParent)  
    backgroundResource = R.drawable.background_activated  
  
    checkBox {  
        enabled = false  
        gravity = Gravity.CENTER  
        padding = dip(4)  
        layoutParams(width = wrapContent, height = wrapContent) {  
            alignParentRight()  
        }  
    }  
  
    textView {  
        textResource = R.string.aha_title  
        rightPadding = dip(4)  
        leftPadding = dip(4)  
        layoutParams(width = matchParent, height = wrapContent) {  
            leftOf(cb)  
        }  
    }  
}
```

FAZIT

IM BROWSER AUSPROBIEREN



The screenshot shows the Kotlin Try Online interface. At the top, there's a navigation bar with the Kotlin logo, social media links (Facebook, Google+, Twitter, GitHub, JetBrain), and buttons for LEARN, CONTRIBUTE, and TRY ONLINE. Below the navigation bar is a toolbar with Shortcuts, Convert from Java, and Fullscreen buttons. The main area displays a tree view of examples under 'Hello, world!', with 'Simplest version' selected. The code editor shows the 'Simplest version.kt' file:

```
1 /**
2 * We declare a package-level function main which returns Unit and takes
3 * an Array of strings as a parameter. Note that semicolons are optional.
4 */
5
6 fun main(args: Array<String>) {
7     println("Hello, world!")
8 }
```

→ <http://try.kotlin/>

QUELLEN & MATERIALIEN

Materialien für den Einstieg

- kotlinlang.org
- [Kotlin Koans](https://kotlinlang.org/docs/tutorials/kotlin-for-noobs/introduction.html)
auch online: try.kotlin.in in IntelliJ: Kotlin Edu Plugin
- Ein Überblick über Java-Alternativen für den industriellen Einsatz,
Benjamin Schmid

Referenzen

- Sprachreferenz

Verzeichnisse

- [Awesome Kotlin](#)
- [OSS Projects & Libraries](#)

Diverses

- [@kotlin](#) & [@bentolor](#)
- github.com/bentolor

Nennenswerte Frameworks

- [KAndroid](#)
- [Fuel](#) (HTTP f. Android)
- [RxKotlin](#) (Reactive)
- [spek](#) (Testing)
- [kovenant](#)
(Android Promises)
- [kotlinx.support](#)
(JDK7/8 features)
- [Klaxon](#) (JSON)

Bildnachweis: Kotlin Islands Maps, Atomic bomb explosion, Android Robot, Kotlin Logo, Hand-written arrows