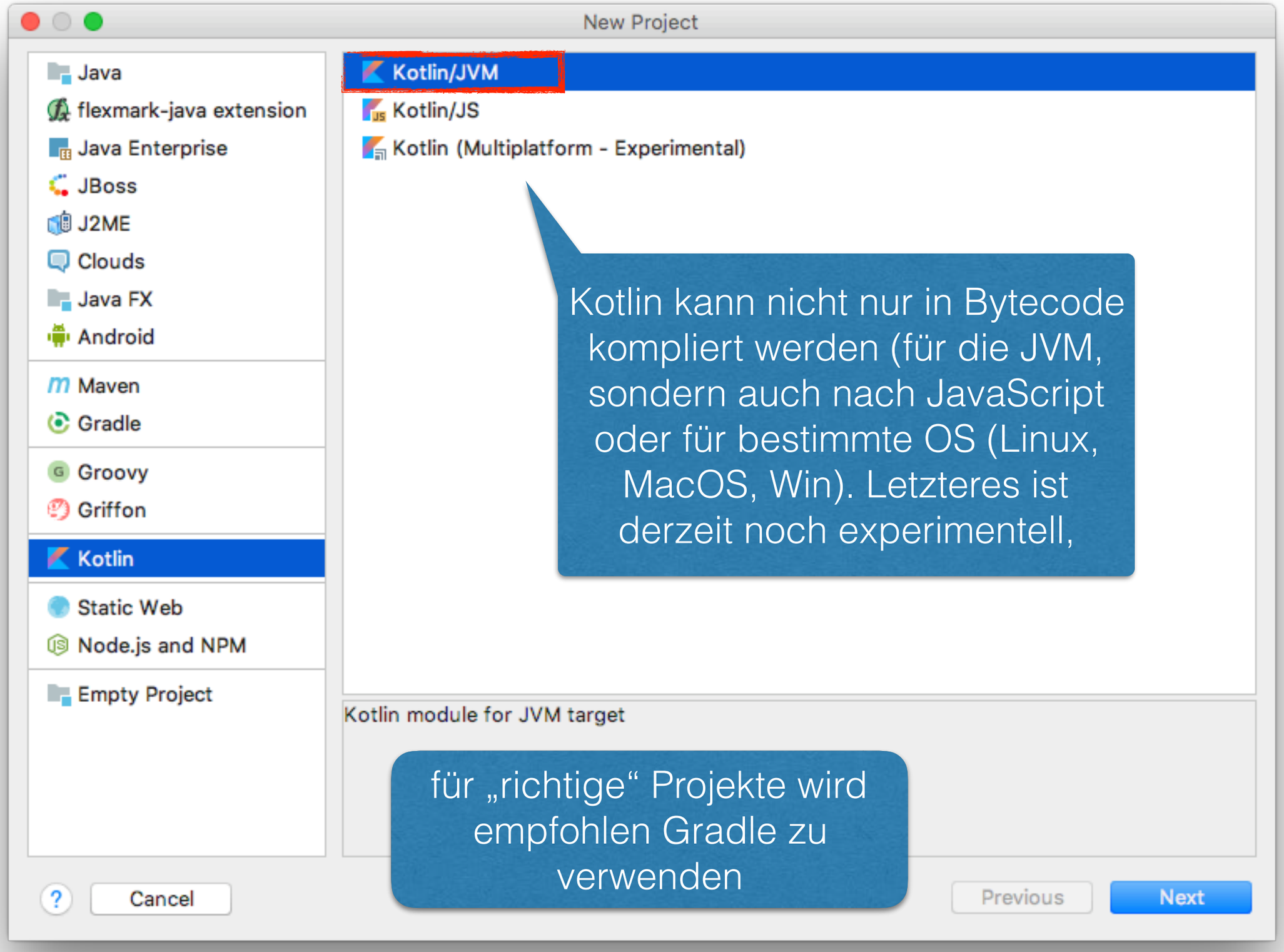
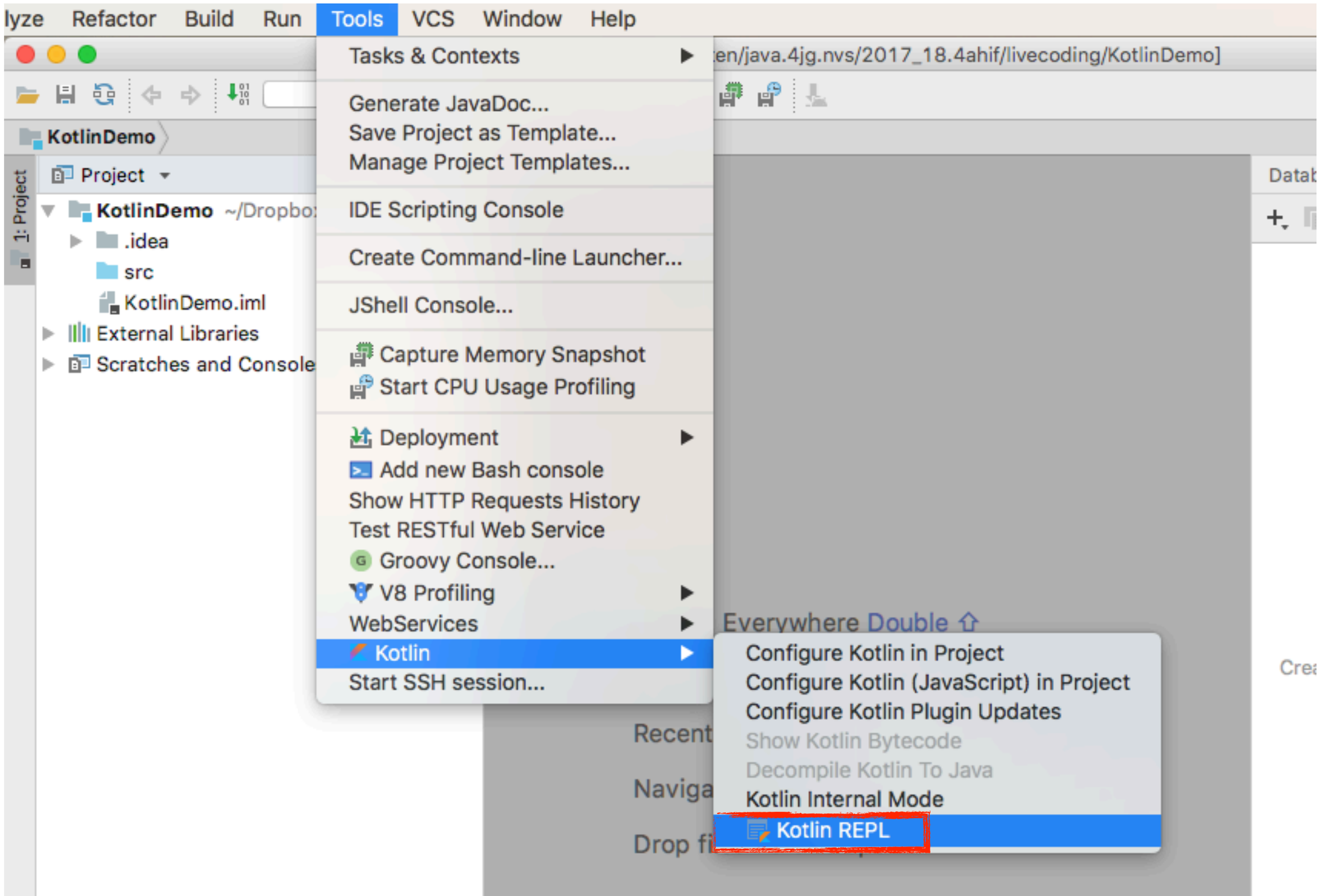
Programmieren auf Android - Introduction

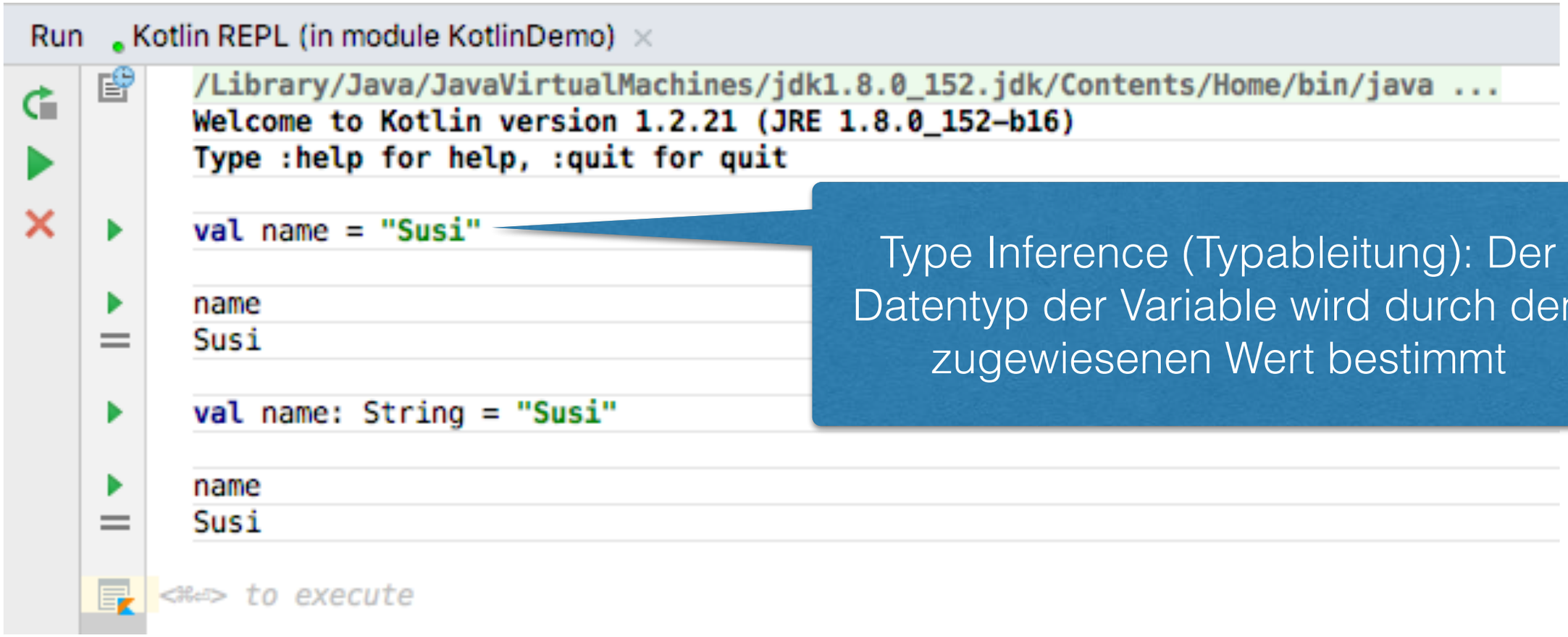
https://kotlinlang.org/


New Project
Java
flexmark-java extension
Java Enterprise
JBoss
J2ME
Clouds
Java FX
Android
Maven
Gradle
Groovy
Griffon
Kotlin
Static Web
Node.js and NPM
Empty Project
Kotlin/JVM
Kotlin/JS
Kotlin (Multiplatform - Experimental)
Kotlin kann nicht nur in Bytecode kompliert werden (für die JVM, sondern auch nach JavaScript oder für bestimmte OS (Linux, MacOS, Win). Letzteres ist derzeit noch experimentell,
Kotlin module for JVM target
für „richtige“ Projekte wird empfohlen Gradle zu verwenden
?
Cancel
Previous
Next

Read-Eval-Print-Loop



Type Inference



Expression ausführen mit ⌘-Enter bzw ^-Enter

Typinferenz

```
val number = 7

number
7

number::class.simpleName
Int

number::class.qualifiedName
kotlin.Int
```

explizites Typisieren

?

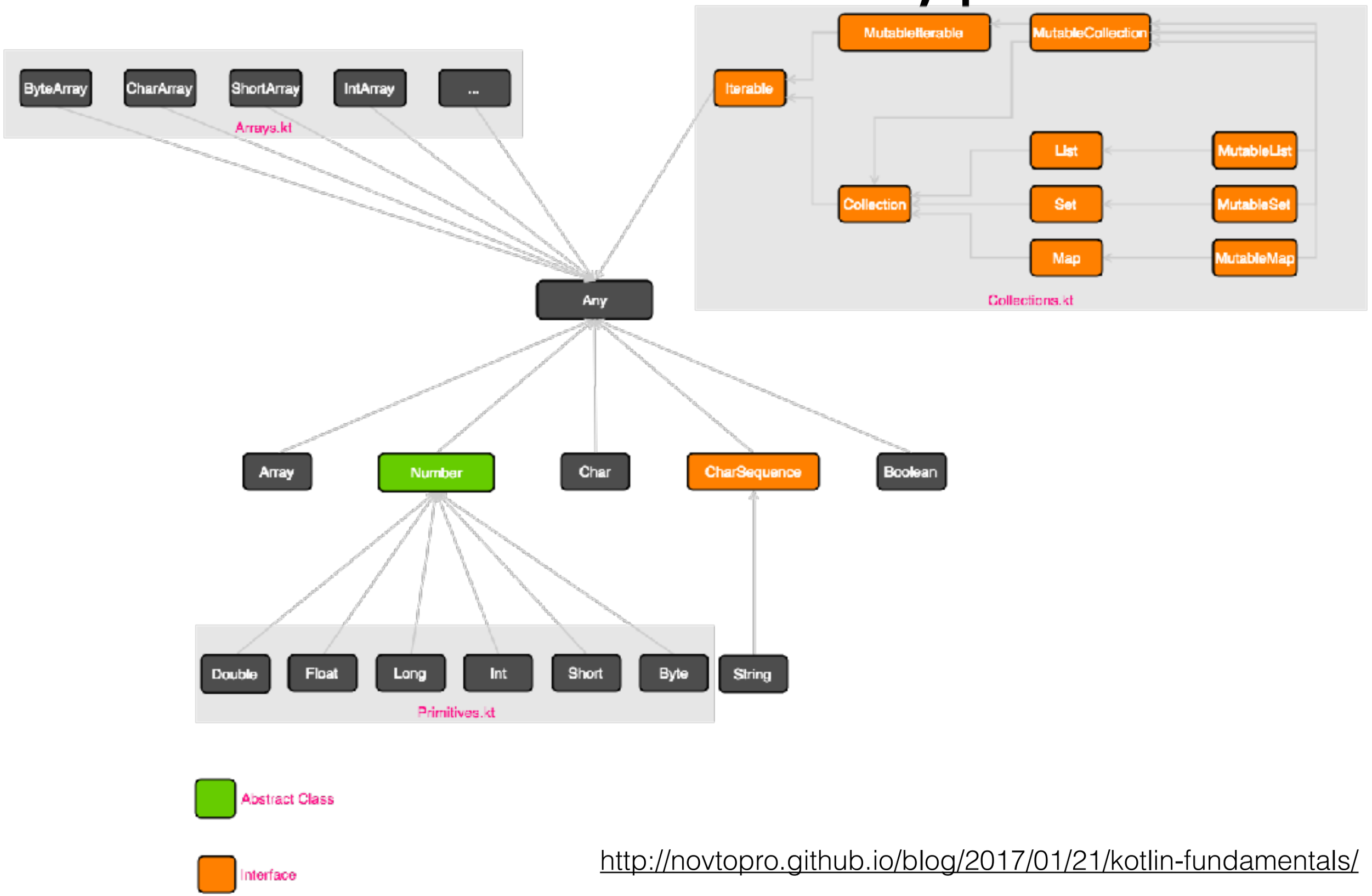
```
var name = "Susi"

name
Susi

name = "Mimi"

name
Mimi
```

Kotlin Basic Types



http://novtopro.github.io/blog/2017/01/21/kotlin-fundamentals/

var versus val

Run Kotlin REPL (in module KotlinDemo)

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
Welcome to Kotlin version 1.2.21 (JRE 1.8.0_152-b16)
Type :help for help, :quit for quit

var age = 17
age
17

var age: Int
age = 17
age
17

val name = "Susi"
name
Susi

val name: String
error: property must be initialized or be abstract
val name: String
^

<⌘⏎> to execute
```

bei einem var - Schlüsselwort kann man die Variable später initialisieren, da sie veränderlich (mutable) ist

bei Verwendung des Schlüsselwortes val wird eine read-only Variable erstellt (Konstante) —> immutable

Es wird empfohlen, immer val zu verwenden

Type inference und Null Safety

```
val str: String = null
error: null can not be a value of a non-null type String
val str: String = null
                  ^

val str: String? = null

val i: Int? = null
```

```
str.length
error: only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?
str.length
   ^

str?.length
null

val str: String? = "Susi"

str?.length
4
```

Die Datentypen sind vergleichbar mit java: Double, Float, Int, Byte, Boolean usw. Allerdings sind dies keine Primitivdatentypen, sondern Objekte

Elvis Operator

- wenn das Ergebnis NULL wäre, so gib (hier) -1 zurück

```
val str: String? = null
```

```
val strLength = str?.length ?: -1

strLength
-1
```

nullable String

```
val str: String? = "Kotlin"

if (str != null) {
    println(str.length)
}
6
```

Obwohl hier str nullable ist, kann in der if-Verzweigung str ohne Fragezeichen angegeben werden, da der Compiler erkennt, ….

if-expression 1

```
val i = 17

if (i > 15) {
    println("i is pretty small")
}
```

i is pretty small

```
if (i < 15) {
    println("i is pretty small")
} else {
    println("it's pretty large")
}
```

it's pretty large

```
if (i < 15) {
    println("i is pretty small")
} else if (i >= 15 && i <= 25) {
    println("it's okay")
} else {
    println("it's pretty large")
}
```

it's okay

alles noch wie in Java

if-expression 2

```
val x = if (i < 15) {
    println("i is pretty small")
} else if (i >= 15 && i <= 25) {
    println("it's okay")
} else {
    println("it's pretty large")
}
it's okay

x
kotlin.Unit
```

Erläutere den Unterschied zwischen **Expression** und **Statement**

Rückgabewert fehlt, daher hat x **keinen** Wert

Unit bedeutet (hier) soviel wie **void**

```
val x = if (i < 15) {
    println("i is pretty small")
    "small"
} else if (i >= 15 && i <= 25) {
    println("it's okay")
    "medium"
} else {
    println("it's pretty large")
    "large"
}
it's okay

x
medium
```

Der Datentyp des Rückgabewertes ist frei wählbar

Rückgabewert vorhanden, daher hat x **einen** Wert

when - expression 1

```
when (price) {
    0 -> println("For free today")
    !in 1..19 -> println("Not on sale")
    in 20..29 -> println("Normal price")
    10 + 20 -> println("Slightly overpriced")
    else -> println("Overpriced")
}
Not on sale
```

break ist nicht notwendig

wenn der Bedingungsteil fehlt, müssen in den Verzweigungen bool'sche Ausdrücke stehen

```
when {
    price <= 19 -> println("Sale")
    price <= 29 -> println("Normal price")
    else -> println("Overpriced")
}
Normal price
```

when - expression 2

```
val price = 39

price
39

val x = when {
    price <= 19 -> println("Sale")
    price <= 29 -> println("Normal price")
    else -> println("Overpriced")
}
Overpriced

x
kotlin.Unit
```

Variable x ist „void“, der Rückgabewert fehlt

```
val x = when {
    price <= 19 -> "Sale"
    price <= 29 -> "Normal price"
    else -> "Overpriced"
}

x
Overpriced
```

Korrektur: Rückgabewerte sind angegeben

Die Initialisierung der Konstante erfolgt

Collection

- Es wird streng unterschieden zwischen
  - **immutable Collections** (List<…>, Map<…>, Set<…>) und
  - **mutable Collections** (MutableList<…>, MutableMap<…>, MutableSet<…>)

Erstellen einer leeren Collection

```
val mutableIntList = mutableListOf<Int>()
```

```
mutableIntList.add(10)
mutableIntList.add(20)
mutableIntList.add(30)
true
```

```
mutableIntList
[10, 20, 30]
```

```
val mutableIntSet = mutableSetOf<Int>()
```

```
mutableIntSet.add(10)
mutableIntSet.add(20)
mutableIntSet.add(30)
mutableIntSet
[10, 20, 30]
```

```
val mutableIntStringMap = mutableMapOf<Int, String>()
```

```
mutableIntStringMap.put(10,"Zehn")
mutableIntStringMap.put(20,"Zwanzig")
mutableIntStringMap.put(30, "Dreißig")
mutableIntStringMap
{10=Zehn, 20=Zwanzig, 30=Dreißig}
```

Array 1

In Kotlin gibt es viele nützliche helper-Methoden

```
val array = arrayOf(2,3,5,7,11,13)

array
[Ljava.lang.Integer;@74d56f47
```

die toString()Methode ist nicht überschrieben

```
array.joinToString()
2, 3, 5, 7, 11, 13
```

daher behelfen wir uns mit einer helper-Methode

```
val array = intArrayOf(2,3,5,7,11,13)
```

das ergibt ein Array aus primitiven Integer-Datentypen (vergleichbar mit Java)

natürlich gibt es auch ein doubleArrayof(…) usw.

MutableList<…>

```
val list = listOf(1, 1, 2, 3, 5, 8, 13)

val mutableList = mutableListOf(1,1,1)

mutableList[0] = 99

mutableList
[99, 1, 1]
```

bei den Listen ist die toString()-Methode überschrieben

„Gegenprobe“

```
list[0] = 99
error: unresolved reference. None of the following candidates is applicable because of receiver type mismatch:
@InlineOnly public operator inline fun <K, V> MutableMap<Int, Int>.set(key: Int, value: Int): Unit defined in kotlin.collections
@InlineOnly public operator inline fun kotlin.text.StringBuilder /* = java.lang.StringBuilder */.set(index: Int, value: Char):
 Unit defined in kotlin.text
list[0] = 99
^
error: no set method providing array access
list[0] = 99
    ^
```

Set<…>

Set (im Deutschen „Menge“) darf jedes Element nur 1x enthalten.

```
val set = setOf(1, 1, 2, 3)

set
[1, 2, 3]

val mutable = mutableSetOf(1, 2, 2, 3)

mutable
[1, 2, 3]
```

Map<…>

```
val map = mapOf(Pair(1, "Kotlin"), Pair(2, "Android"))

map
{1=Kotlin, 2=Android}
```

Alternative:

```
val map: Map<Int, String> = mapOf(Pair(1, "Kotlin"), Pair(2, "Android"))
```

Die Typangabe ist allerdings nicht notwendig; die kann der Compiler ableiten

MutableMap<…>

```
val mutableMap = mutableMapOf(1 to "Kotlin", 2 to "Android")
```

Wir verwenden hier eine helper-Function to, die auch infix verwendet werden kann.

Alternative:

```
val mutableMap = mutableMapOf(1.to("Kotlin"), 2.to("Android"))

mutableMap
{1=Kotlin, 2=Android}
```

die **obere Variante** ist allerdings **besser lesbar** und wir verwenden diese daher

.toList() - Function

```
val set = setOf(1, 1, 2, 3)
```

```
set
[1, 2, 3]
```

```
val list = set.toList()
```

```
list
[1, 2, 3]
```

Aus einem Set kann ganz einfach eine Liste erstellt werden.

BEACHTE: Die erstellte Liste ist nur eine seichte Kopie (shallow copy)

Iterationen u. dgl.

for - Schleife

```
for (i in 1..10) {
    print("$i ")
}
```
1 2 3 4 5 6 7 8 9 10

```
for (c in "kotlin") {
    print("$c ")
}
```
k o t l i n

```
val languages = listOf("Kotlin", "Java", "Swift")
for (lang in languages) {
    println("$lang is great")
}
```
Kotlin is greatJava is greatSwift is great

```
for (i in 10 downTo 1) {
    print("$i ")
}
```
10 9 8 7 6 5 4 3 2 1

```
for (i in 10 downTo 1 step 2)
    print("$i ")
}
```
10 8 6 4 2

Alternative:

```
for (i in 10.downTo(1)) {
    print("$i ")
}
```
10 9 8 7 6 5 4 3 2 1

while - Loop

```
var i = 1

while (i <= 10) {
    print("$i ")
    i++
}
```

```
1 2 3 4 5 6 7 8 9 10
```

```
do {
    println(i)
    i++
} while (i <= 20)
```

```
11121314151617181920
```

Functions 1

Parameter

Rückgabewert

```
fun permitEntrance(age: Int): Boolean {
    return age >= 18
}

permitEntrance(7)
false

permitEntrance(44)
true
```

Alternative, falls eine Funktion nur eine expression im Function-Body enthält

```
fun permitEntrance(age: Int): Boolean = age >= 18

permitEntrance(7)
false

permitEntrance(44)
true
```

Functions 2: vararg

```
fun permitEntrance(vararg ages: Int): Boolean {
    return ages.any { age -> age >= 18 }
}

permitEntrance(11, 20, 7)
true

permitEntrance(11, 17, 7)
false
```

Challenge: Working will Nullables and If-Expressions

Now that you can create Kotlin files with a main() function, you're ready for some challenges to apply what you learned!

- Use the readLine() function to read an input from the command line
    - Notice the return type of readLine(), and use what you learned to work with it
- The user should input their name
- If the user enters an empty string, store a default value
- Use an if expression to define a different greeting message based on whether the user entered a name

Hints

- First, check readLine() returns null or is a blank string, and assign your default value if so

Lösung

KotlinAndroid › src › basics › main.kt

BEACHTE: im File main.kt gibt es KEINE Klassendeklaration, nur eine Function

```kotlin
package basics

fun main(args: Array<String>) {

    val anonymous = "Anonymous"

    print("Enter your name: ")
    val input = readLine()

    // Use anonymous name if none is given
    val name = if (input != null && input.isNotBlank()) {
        input
    } else {
        anonymous
    }

    // Another if-expression
    val message = if (name == anonymous) {
        "Conscious about your privacy, eh?"
    } else {
        "Welcome $name, the NSA has been informed of your name :)"
    }

    println(message)  // You could also directly use println() above
}
```

Challenge: Kotlin Collections and Loops

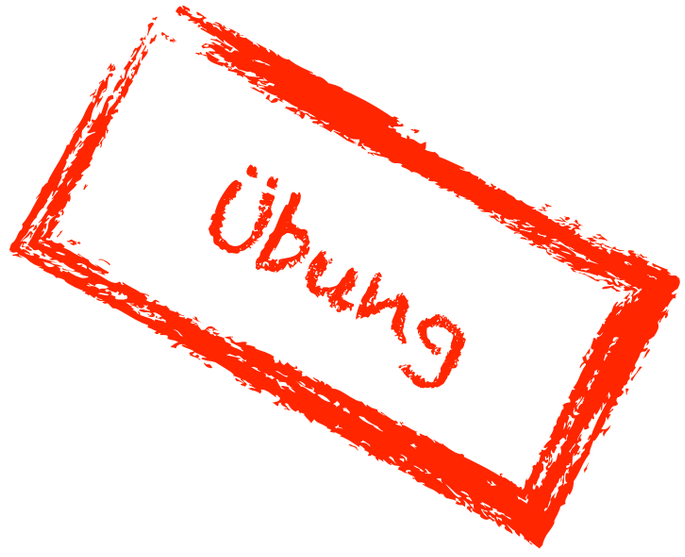
Apply what you learned about collections and the different kinds of loops in Kotlin!

- Create a collection of integers
- Use java.util.Random to fill the collection with 100 random numbers between 1 and 100.
- Go through the collection from start to end and print its elements up to the point where an element is less than or equal to 10
    - Do it without using “if” or “when”

Hints

- Make sure you import java.util.Random (same syntax as in Java)
- If you can’t add elements to your collection, think about the type of collection you’re using
- To avoid “if”, you may need to use a different type of loop

Lösung




```
package basics

import java.util.*

fun main(args: Array<String>) {

    var integers = mutableListOf<Int>();

    val rnd = Random()

    for (i in 1..100) {
        integers.add(rnd.nextInt(100) + 1)
    }

    var i = 0
    while (integers.get(i) > 10) {
        print("${integers.get(i)} ")
        i++
    }
}
```

Lösung 2


```
package basics

import java.util.*

/*
 * Challenge: Using collections and loops in Kotlin
 *
 * Kotlin clearly differentiates between mutable and immutable collections. Also, the "for" loop is
 * similar to the for-in (or for-each) loop in Java. "while" loops follow the same syntax as in
Java.
 * In this challenge, you'll apply what you learned about collections and loops.
 *
 * This challenge is part of the course "Kotlin for Android and Java Developers": [TODO: add link]
 *
 * @author Peter Sommerhoff
 */
fun main(args: Array<String>) {

    val randoms: MutableList<Int> = mutableListOf()  // you can also use mutableListOf<Int>()

    for (i in 1..100) {
        randoms.add(Random().nextInt(100) + 1)
    }

    var i = 0
    while(randoms[i] > 10) {
        println(randoms[i])
        i++
    }
}
```

Zugriff auf Liste auch mit [] möglich

Functions

Erstellen einer Function

KotlinAndroid › src › basics › parameters.kt

```
package basics

fun main(args: Array<String>) {

}

fun concat(texts: List<String>, separator: String = ", ") = texts.joinToString(separator)
```

Default-Wert für Parameter

Function-Body (da nur eine Expression)

Aufruf mit 1 Parameter

```kotlin
package basics

fun main(args: Array<String>) {
    val together = concat(listOf("Kotlin", "Java", "Scala"))
    println(together)
}

fun concat(texts: List<String>, separator: String = ", ") = texts.joinToString(separator)
```

```
Kotlin, Java, Scala
```

Hier wird der Default-Wert des Parameters verwendet

Aufruf mit 2 Parameter

```kotlin
package basics

fun main(args: Array<String>) {
    val together = concat(listOf("Kotlin", "Java", "Scala"), " : ")
    println(together)
}

fun concat(texts: List<String>, separator: String = ", ") = texts.joinToString(separator)
```

```
Kotlin : Java : Scala
```

Ändern der Parameter-Reihenfolge

```kotlin
package basics

import java.io.File.separator

fun main(args: Array<String>) {
    val together = concat(separator = " : ", texts = listOf("Kotlin", "Java", "Scala"))
    println(together)
}
```

```kotlin
fun concat(texts: List<String>, separator: String = ", ") = texts.joinToString(separator)
```

```
Kotlin : Java : Scala
```

Durch Verwendung benannter Parameter (named parameters) kann die Reihenfolge der Parameter vertauscht werden

Exception Handling

Aufruf mit Exception

KotlinAndroid › src › basics › exceptions.kt

```
package basics

import java.io.IOException

fun main(args: Array<String>) {
    val input = getExternalInput()
    println(input)
}

fun getExternalInput(): String {
    throw IOException("Could not read external input")
}
```

Run basics.ExceptionsKt

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
Exception in thread "main" java.io.IOException: Could not read external input
    at basics.ExceptionsKt.getExternalInput(exceptions.kt:11)
    at basics.ExceptionsKt.main(exceptions.kt:6)

Process finished with exit code 1
```

try-catch-Block (Versuch 1)

```
fun main(args: Array<String>) {
    try {
        val input = getExternalInput()
    } catch (e: Exception) {
        e.printStackTrace()
    } finally {
        println("Finished trying to read external input")
    }
    println(input)
}

fun getExternalInput(): String {
    throw IOException("Could not read external input")
}
```

Problem: der Scope von „input“

try-catch-Block (Versuch 2)

```
fun main(args: Array<String>) {

    var input: String? = null
    try {
        val input = getExternalInput()
    } catch (e: Exception) {
        e.printStackTrace()
    } finally {
        println("Finished trying to read external input")
    }
    println(input)
}

fun getExternalInput(): String {
    throw IOException("Could not read external input")
}
```

Probleme:

1. var —> man kann kein val mehr verwenden
2. String muss nullable sein —> String?

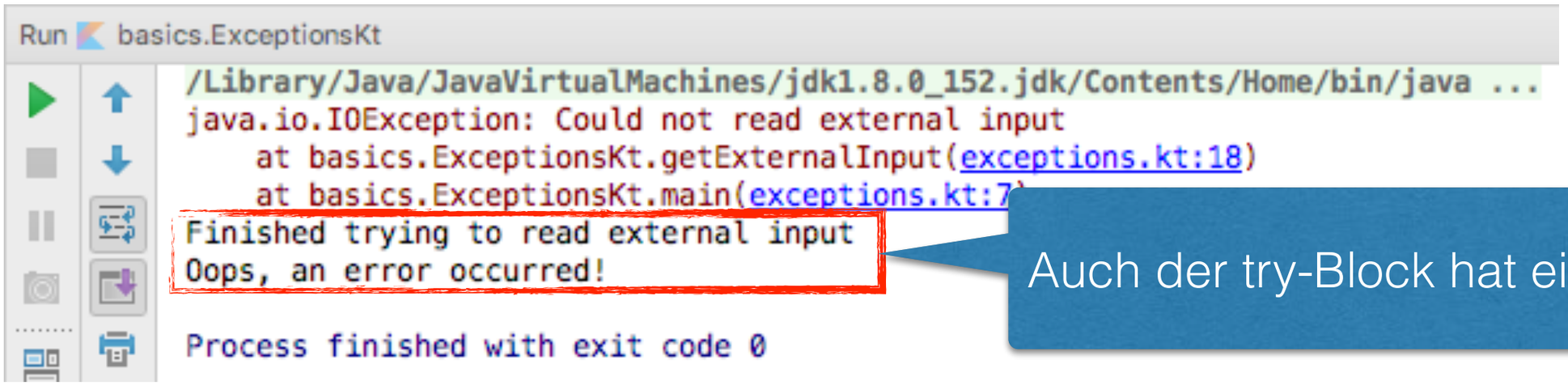
try-catch-Block (Versuch 3)

Der try-catch-Block wird „input“ zugewiesen —> expression

```
fun main(args: Array<String>) {
    val input = try {
        getExternalInput()
    } catch (e: Exception) {
        e.printStackTrace()
        "Oops, an error occurred!"
    } finally {
        println("Finished trying to read external input")
    }
    println(input)
}

fun getExternalInput(): String {
    throw IOException("Could not read external input")
}
```

Es gibt nun sowohl für den catch- als auch try-Block einen Rückgabewert



Auch der try-Block hat einen Rückgabewert

Checked Exceptions

```
package java.lang;

/**
 * The class {@code Exception} and its subclasses are a form of
 * {@code Throwable} that indicates conditions that a reasonable
 * application might want to catch.
 *
 * <p>The class {@code Exception} and any subclasses that are not also
 * subclasses of {@link RuntimeException} are <em>checked
 * exceptions</em>.  Checked exceptions need to be declared in a
 * method or constructor's {@code throws} clause if they can be thrown
 * by the execution of the method or constructor and propagate outside
 * the method or constructor boundary.
 *
 * @author  Frank Yellin
 * @see     java.lang.Error
 * @jls 11.2 Compile-Time Checking of Excep
 * @since   JDK1.0
 */
public class Exception extends Throwable {
```

Öffnet man die Klasse Exception.java, so sieht man, dass **Checked Exceptions** alle Klassen sind, die nicht von RuntimeException abgeleitet sind

Checked Exceptions

```
/**
* ...
* Checked exceptions need to be declared in a
* method or constructor's {@code throws} clause ...
```

```
package basics;

import java.io.IOException;

public class Main {

    public static void main(String[] args) {
        canThrowAnException();
    }

    static void canThrowAnException() {
        throw new IOException();
    }

}
```

Add exception to method signature
Surround with try/catch
Annotate class 'IOException' as @Deprecated

Checked Exceptions müssen in einer Methode behandelt werden (try-catch) oder in einer Methodensignatur propagiert werden.

Diese Propagation über viele Softwareschichten führt dazu, dass

1. die Traceability leidet und
2. die oberen Schichten über die Fehlermöglichkeiten der unteren Schichten Bescheid wissen müssen

~~Checked Exceptions~~

- Diese Gründe verursachen mehr Probleme als Nutzen

- Daher werden in vielen Sprachen, die von Java inspiriert bzw. abgeleitet wurden (C#, Ruby, …) keine Checked Exceptions mehr verwendet

- Auch Kotlin verwendet KEINE Checked Exceptions

Noch Fragen?