Fachbereich Automatisierung und Informatik	▲ Hochschule Harz Hochschule für angewandte Wissenschaften Wernigerode

Handbuch

im Fach

„Mobile Programmierung mit Android“

Kotlin

SoSe 2023

Stand 07.03.2023

Inhaltsverzeichnis

2 Abbildungsverzeichnis

3 Android

Dieses Kaptels zeigt die beiden Installationsprozeduren und die Prozedure des „Nachladens“. Android wird während des Arbeitens immer mal wieder neue Dateien laden.

Hinweis:

- **Der Rechner sollte ein 64-Bit-System sein und mindestens 8 GB Hauptspeicher haben.**

3.1 Installation

Die Installation verläuft in **fünf** Schritten:

- Download der Exe-Datei bzw. dgm-Datei
  - Aktuelle Version 2022.1.10
- Erste Installation
- Aufruf und laden weiterer Dateien
- Runterladen des SDK
- Runterladen der virtuellen Maschine

3.1.1 Erste Installation

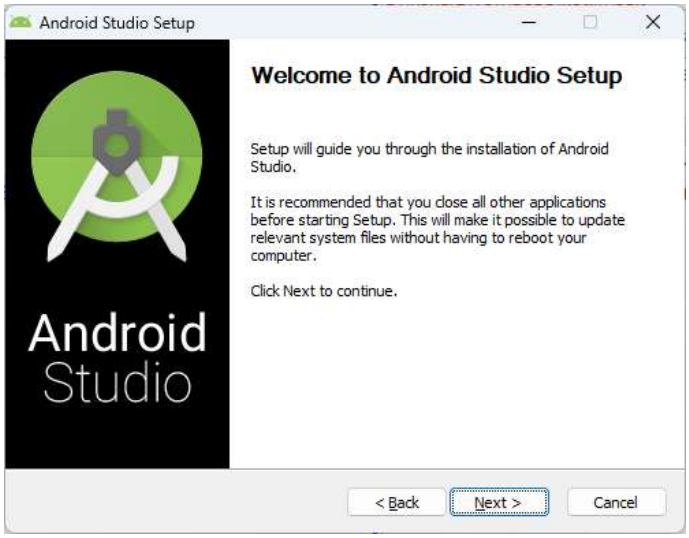


Abbildung 1 Begrüßungsdialog

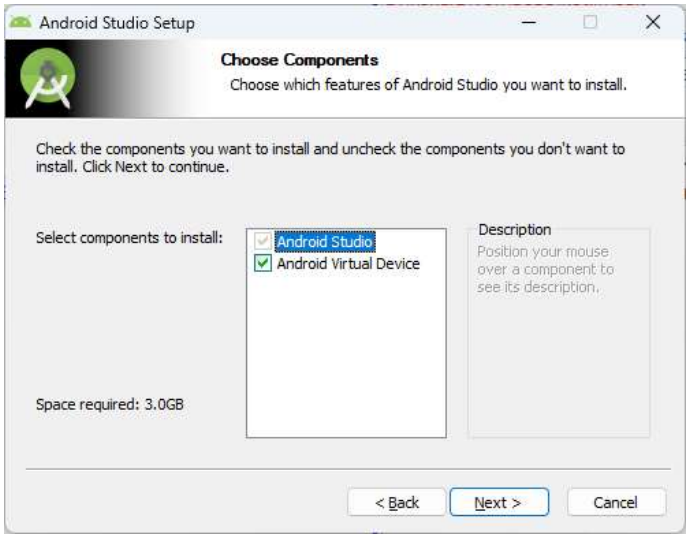


Abbildung 2 Bitte mit der Virtualisierung

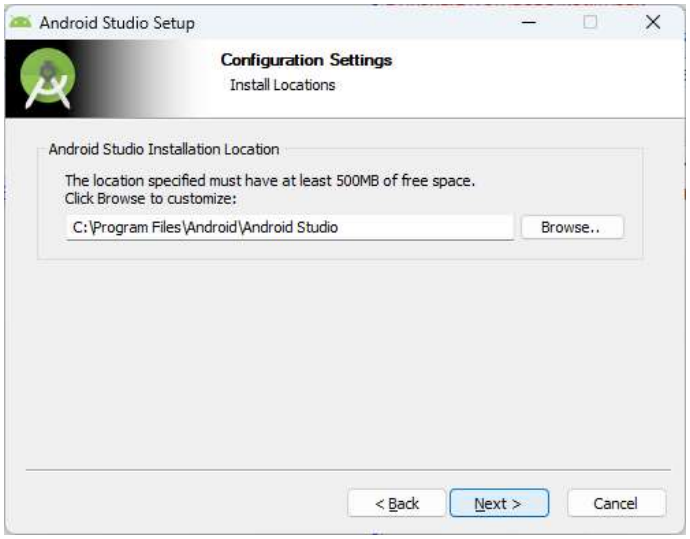


Abbildung 3 Installationsordner (Meine Wahl: c:\Android Studio)

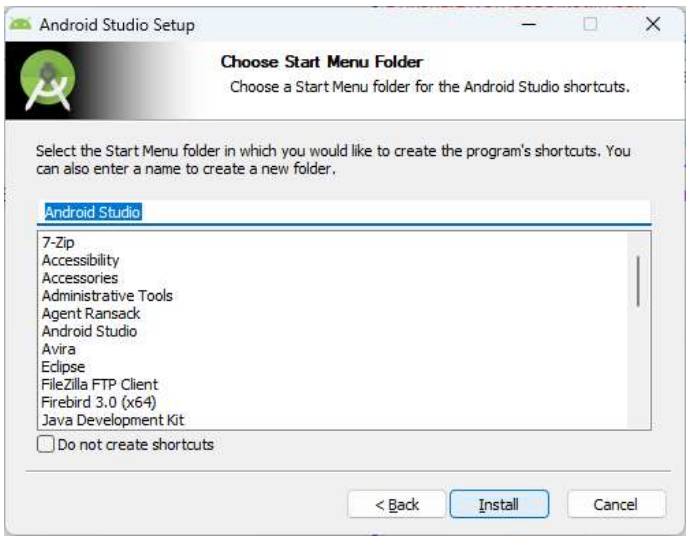


Abbildung 4 Eintrag im Startmenü

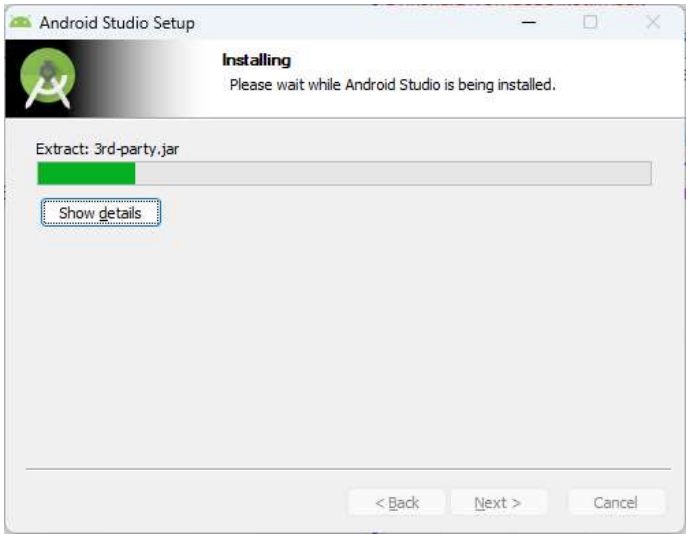


Abbildung 5 Entzippen des Installationsprogramms

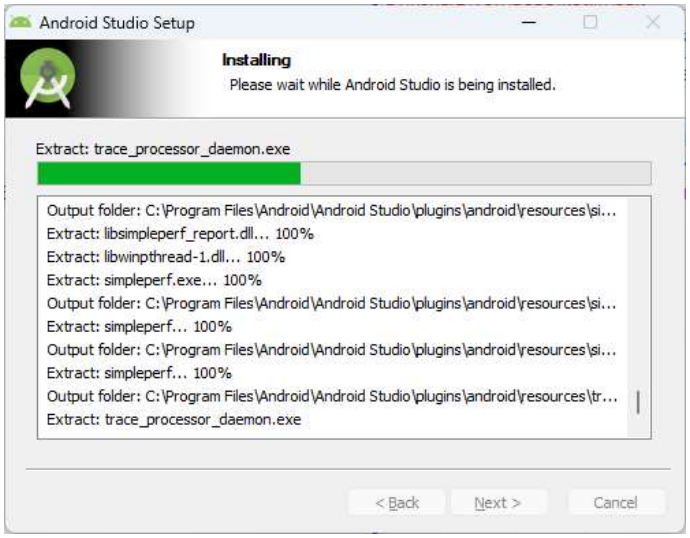


Abbildung 6 Installationsanzeige

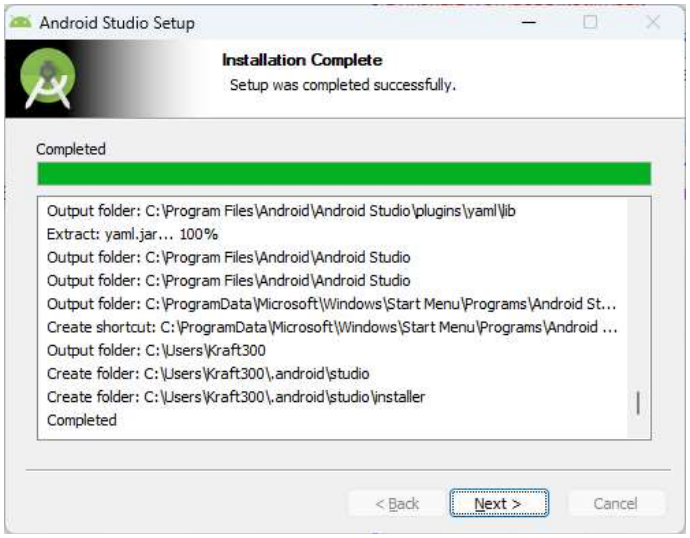


Abbildung 7 Die Installation wurde beendet

Abbildung 8 Der erste Teile, der schnellere, ist beendet.

3.1.2 Zweite Installation

Nun starten des Android Studios:

Im zweiten Teile werden die SDK, Emulatoren etc. geladen.

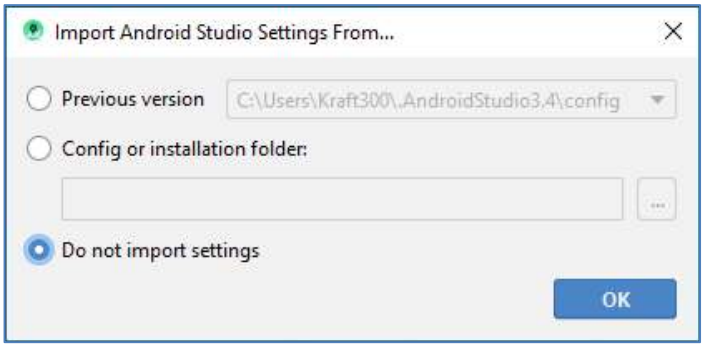
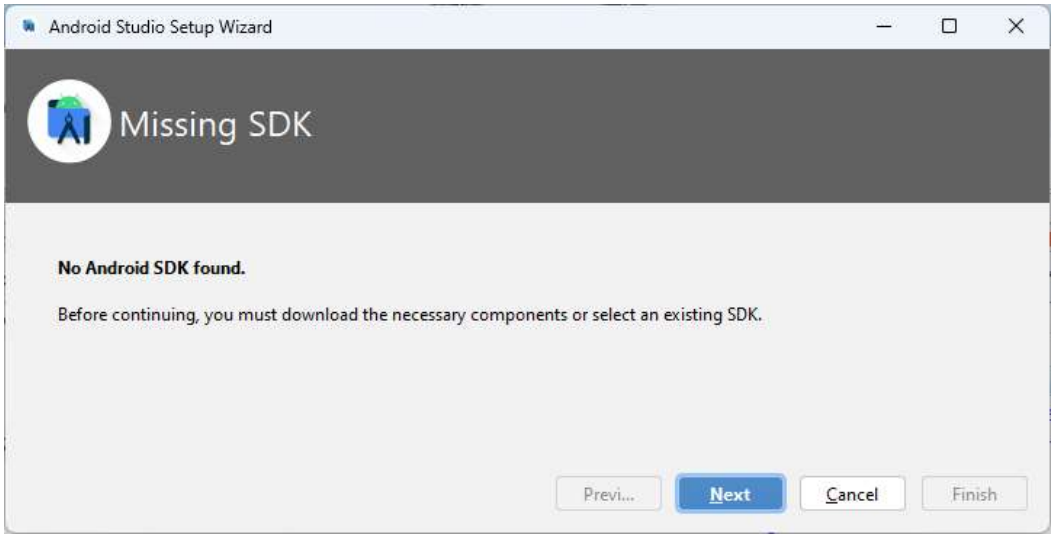
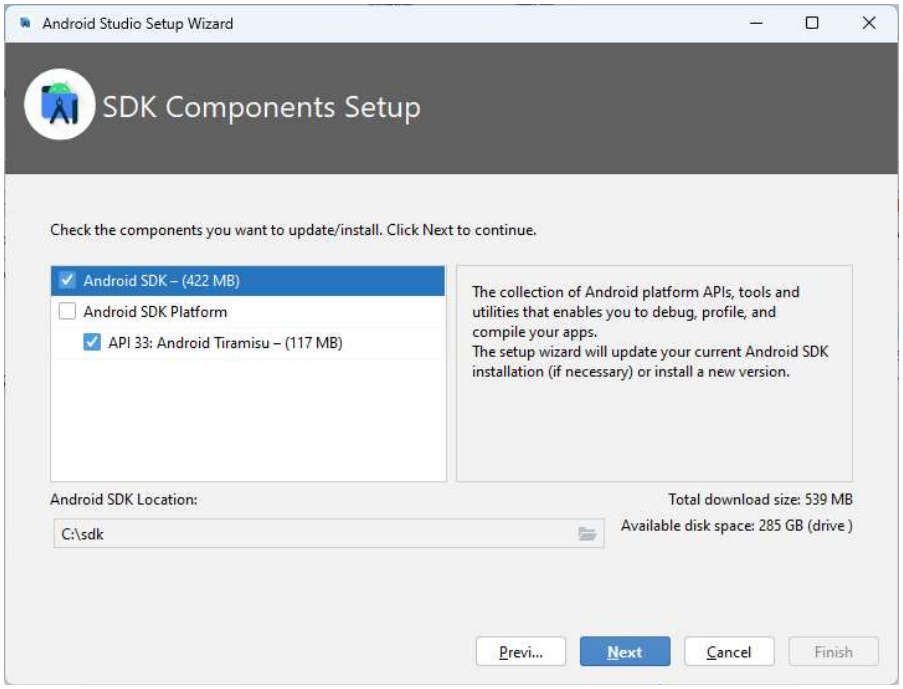


Abbildung 9 Import von vorherigen Versionen)optional)

Odern für SDK: c:\sdk





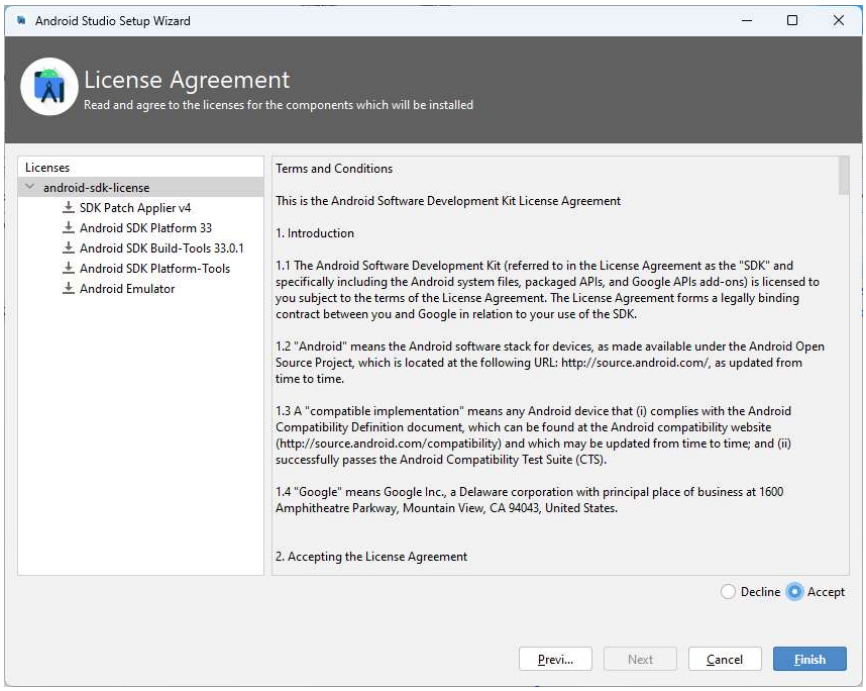


Abbildung 10 SDK installieren

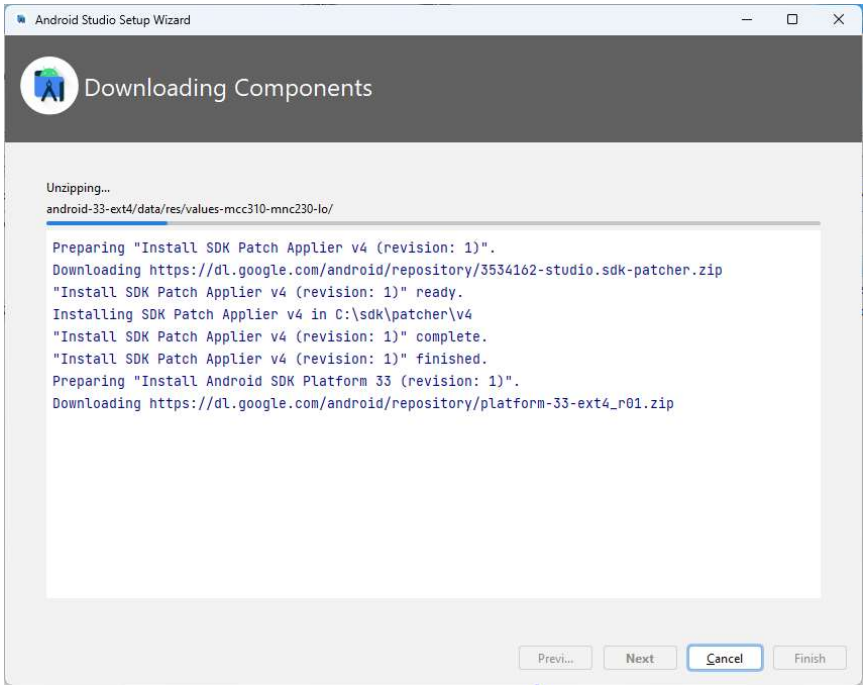
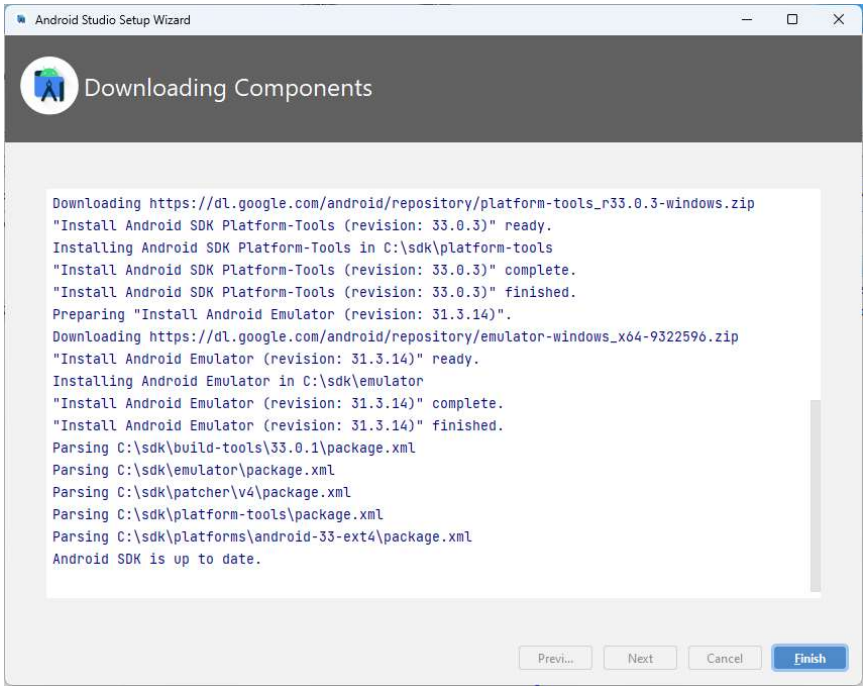


Abbildung 11 Download des SDK



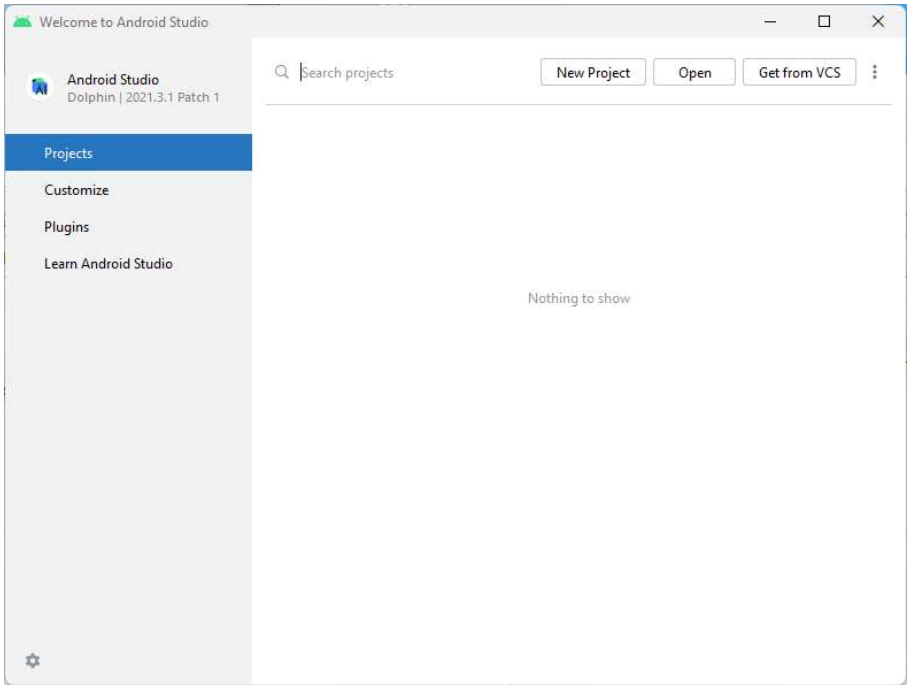


Abbildung 12 Anzeige des Startdialogs von Android Studio

Schalter „New Project“

Nun ein neues Projekt anlegen. Danach muss das SDK und mindestens eine virtuelle Maschine angelegt werden.

3.2 Erstellen eines neuen Projektes

a) Aufruf des Androis Studios

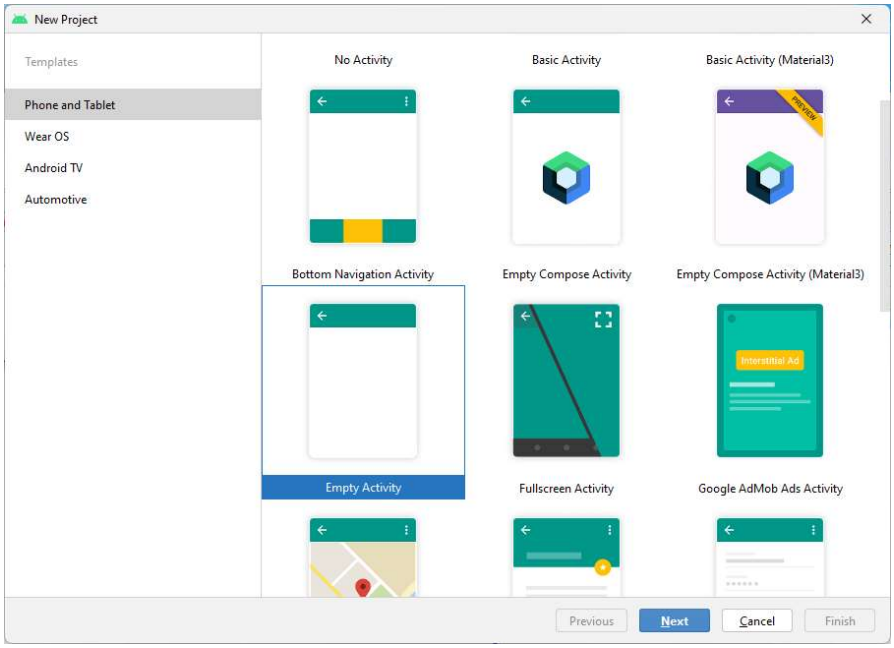


Abbildung 13 Android Startfenster

b) Schalter „Start anew Android Studio project“ anklicken

c) Auswahl des Projektes „Empty Acitivity“

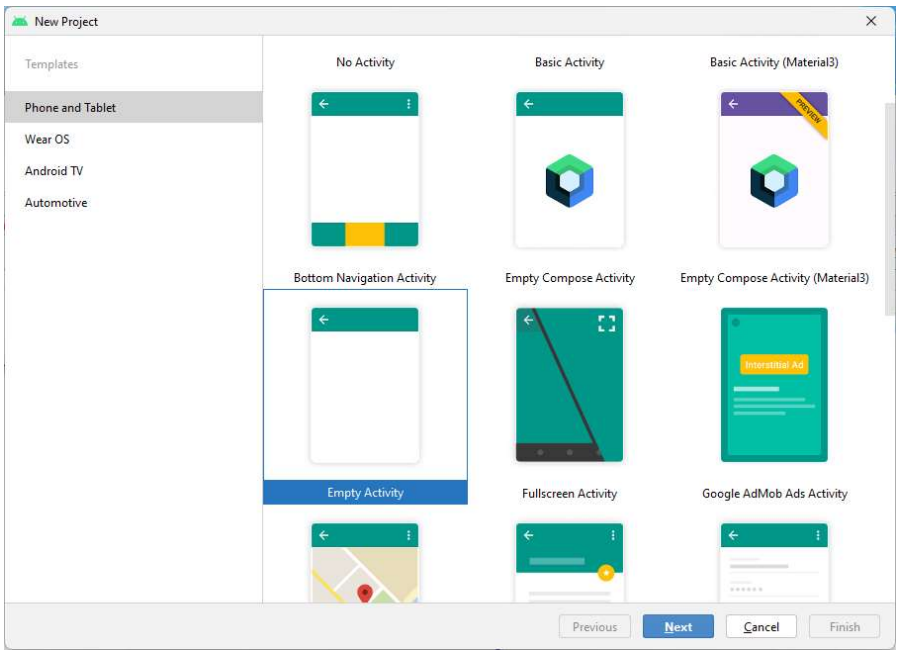


Abbildung 14 Auswahl "Empty Activity"

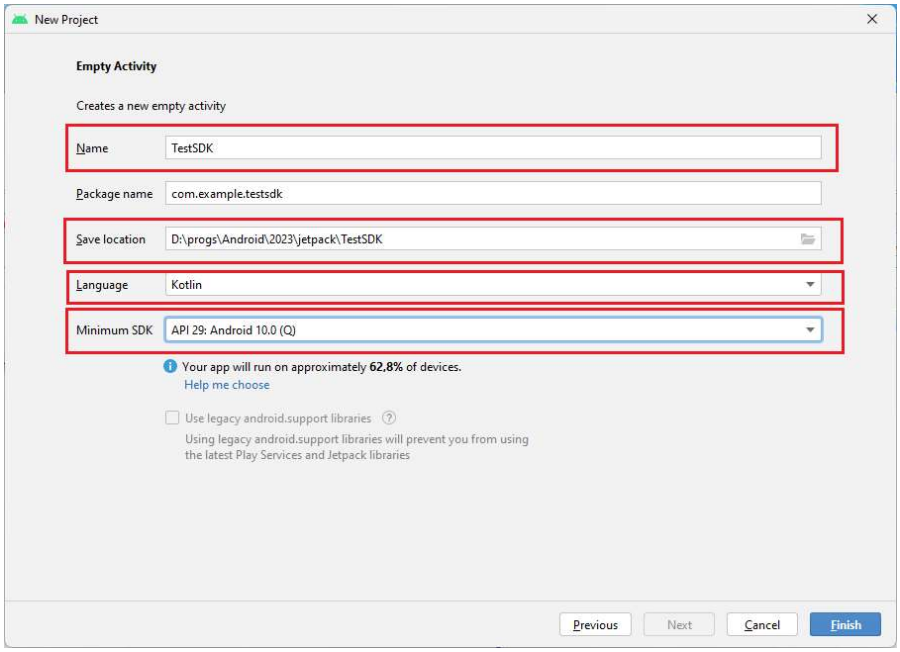


Abbildung 15 Eigenschaften des neuen Projektes

Einträge:

- Name TestSDK
- Save location D:\progs\Android\2023\jetpack\TestSDK
  - **Bitte erstellen einen Ordner für Ihre Projekte**
- Language **Kotlin**
- Minimum SDK API 29: Android 10.0 (Q)

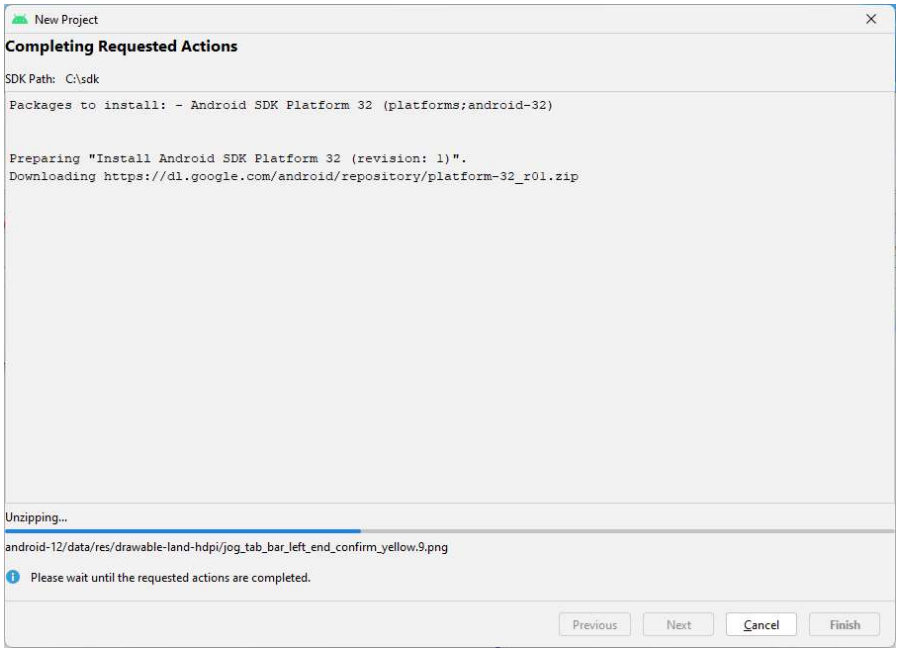


Abbildung 16 Installation des SDK 32

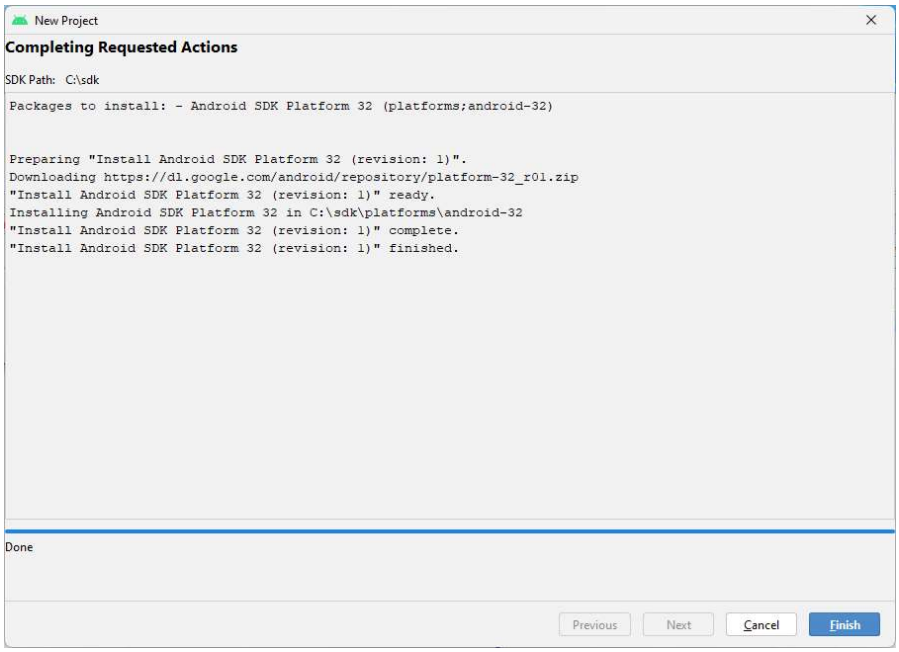


Abbildung 17 SDK mit der Version 32 installiert

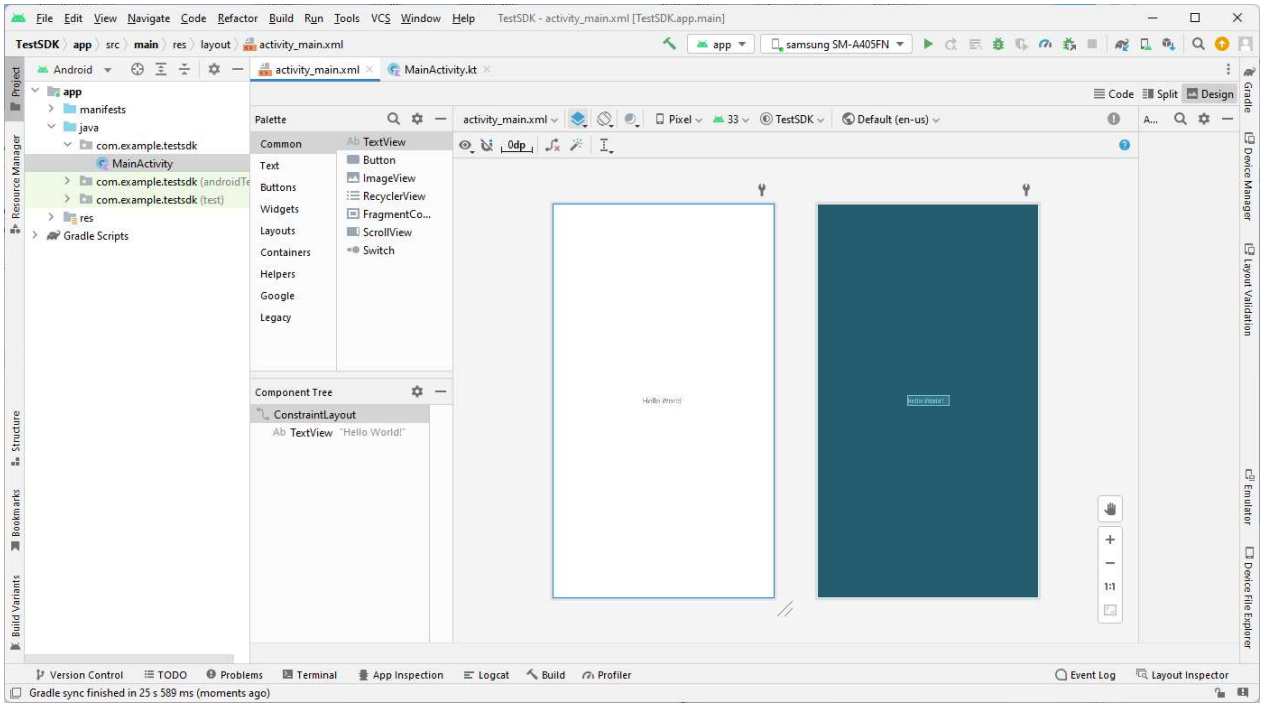


Abbildung 18 Anzeige des neuen Projekt im Android Studio

3.3 SDK-Installation

Menü Tools
Eintrag SDK Manager

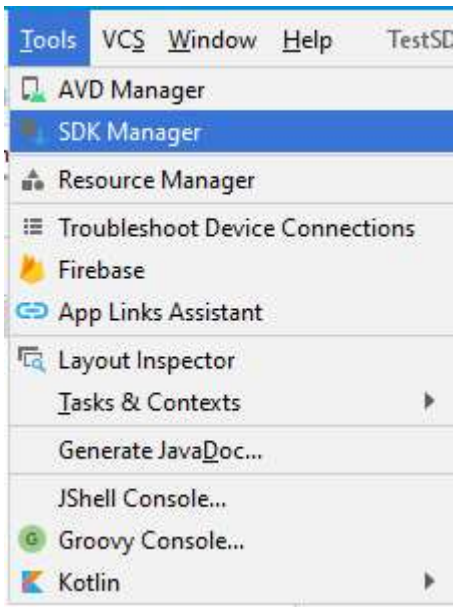


Abbildung 19 Aufruf SDK Manager

Nun Anklicken des Android SDK Tiramisu (33).

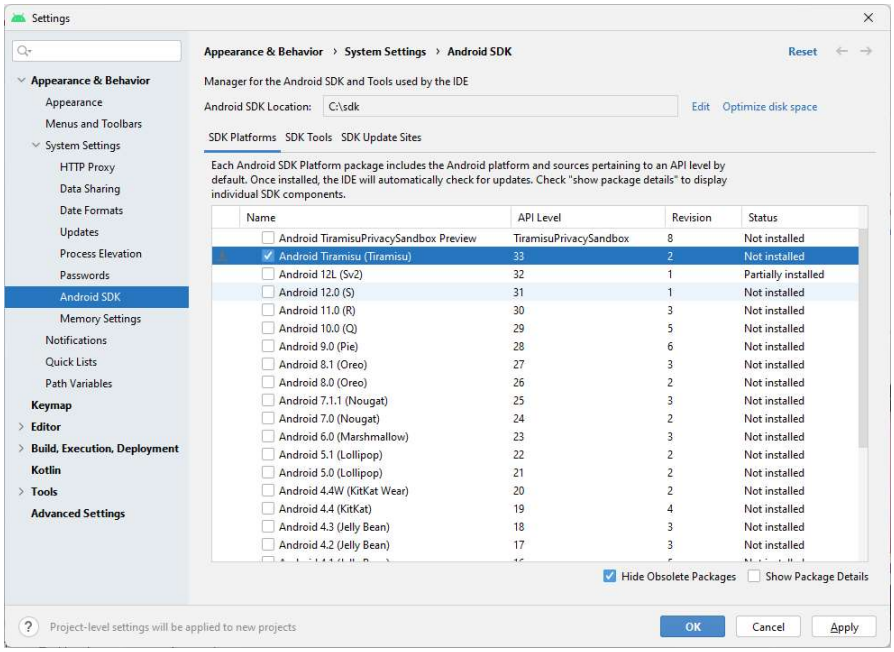


Abbildung 20 SDK erstellen (Auswahl der Komponenten)

Nun den Schalter „OK“ für die Auswahl des SDK bestätigen.

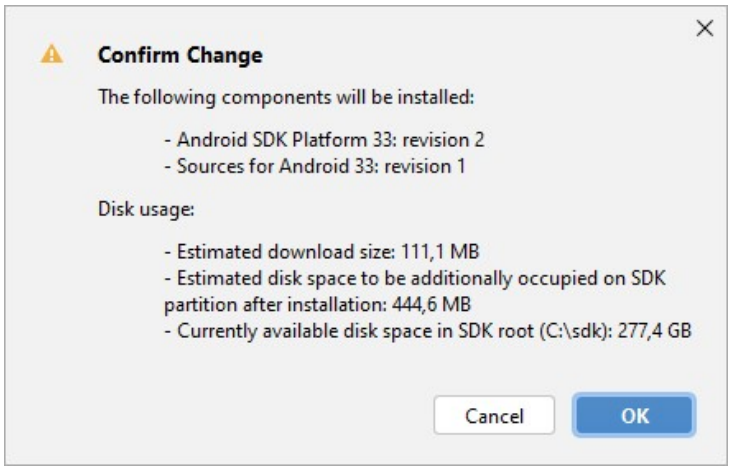


Abbildung 21 SDK Auswahl bestätigen

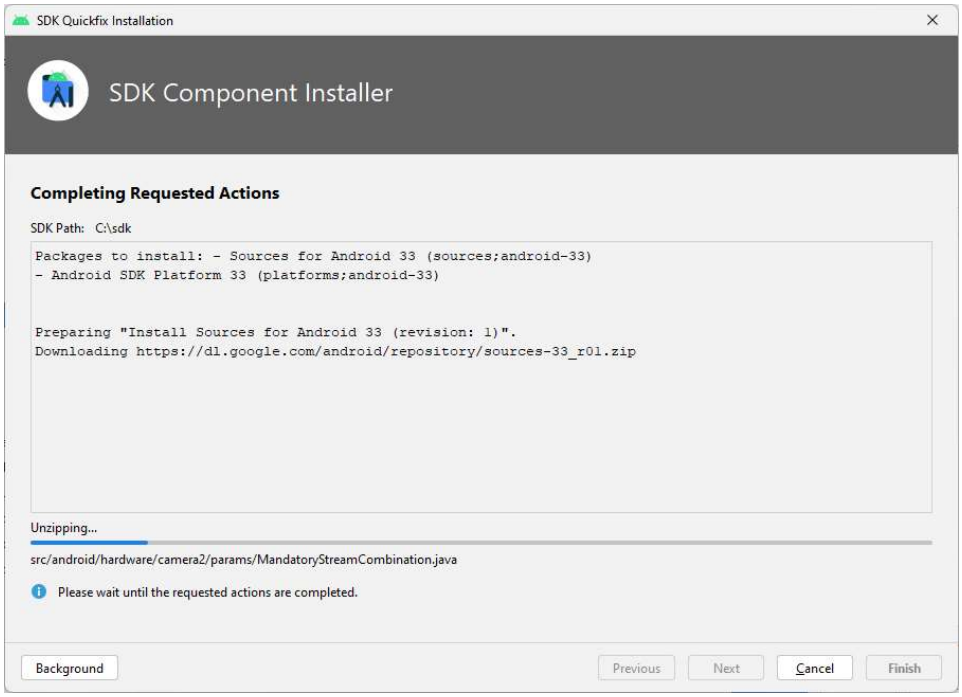


Abbildung 22 Runterladen der Komponenten für das SDK 33

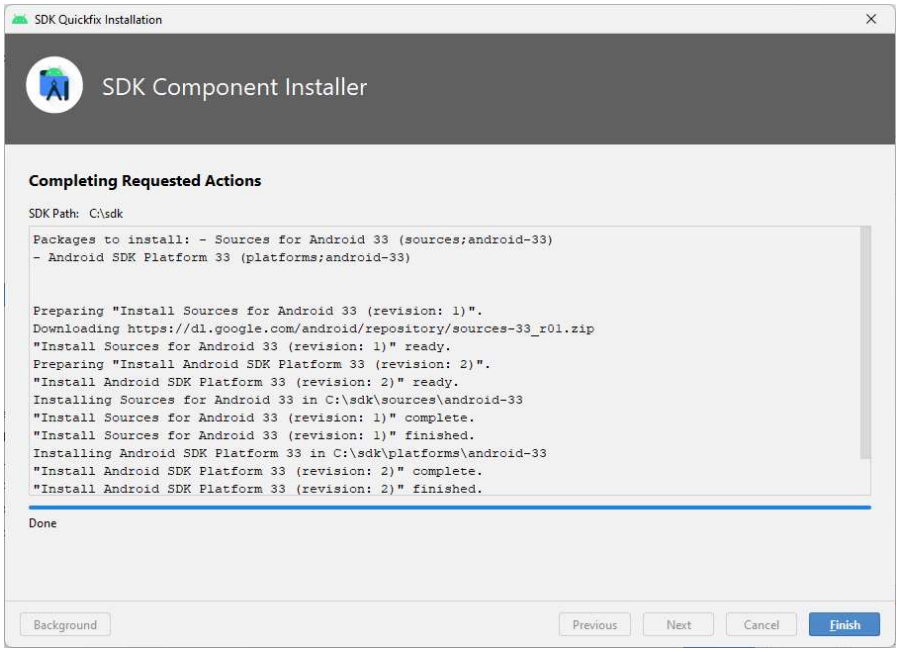


Abbildung 23 Installation des SDK Version 29

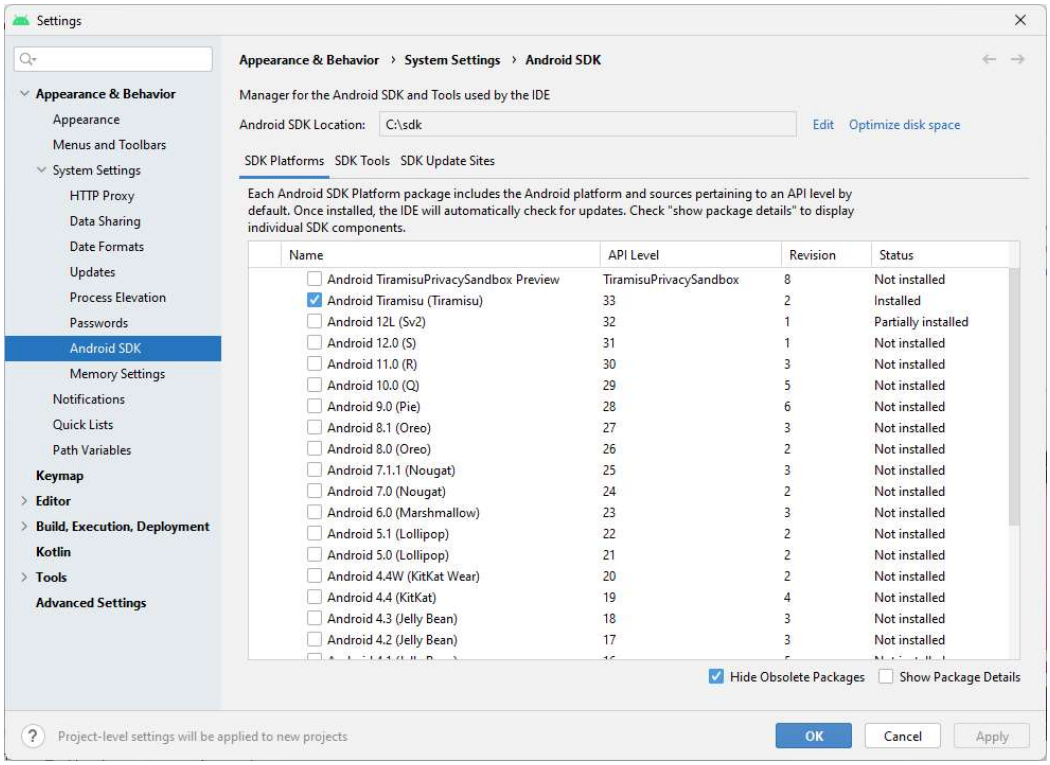


Abbildung 24 SDK 33 ist nun installiert

Mit dem Schalter „ok“ den Dialog beenden.

3.4 Virtuelle Maschine (AVD-Manager)

Menü **Tools**,
Eintrag **Device-Manager**

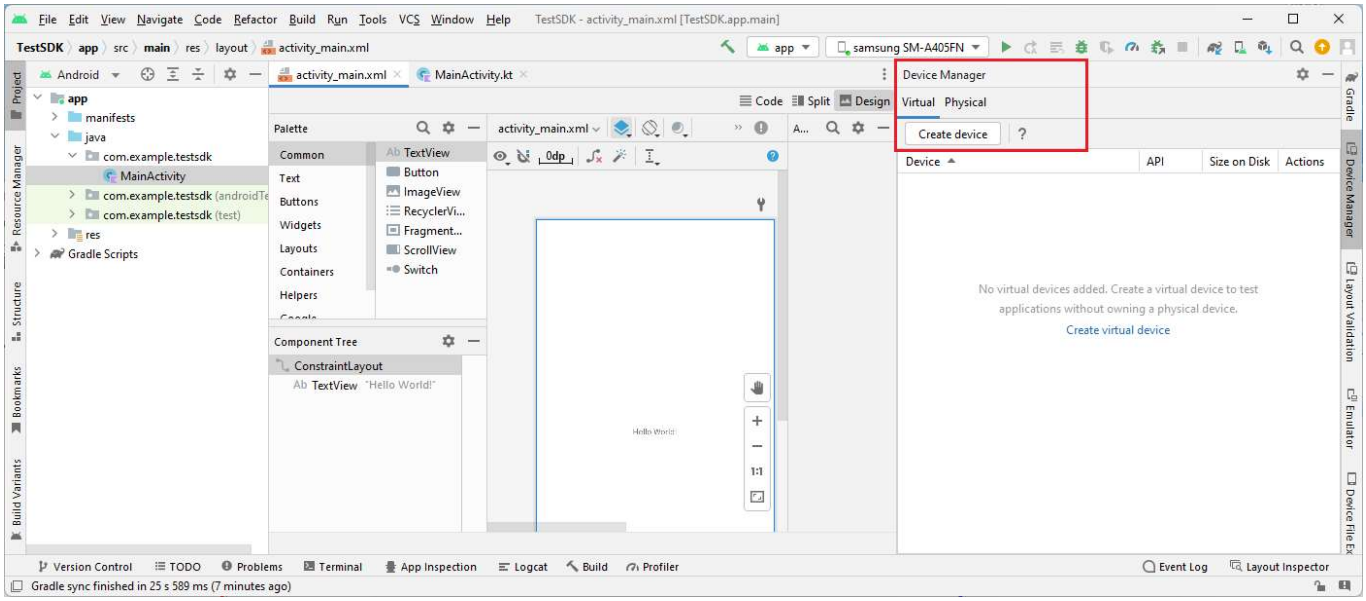


Abbildung 25 Erstellen einer virtuellen Maschine

Nun den Schalter „Create device“ anklicken

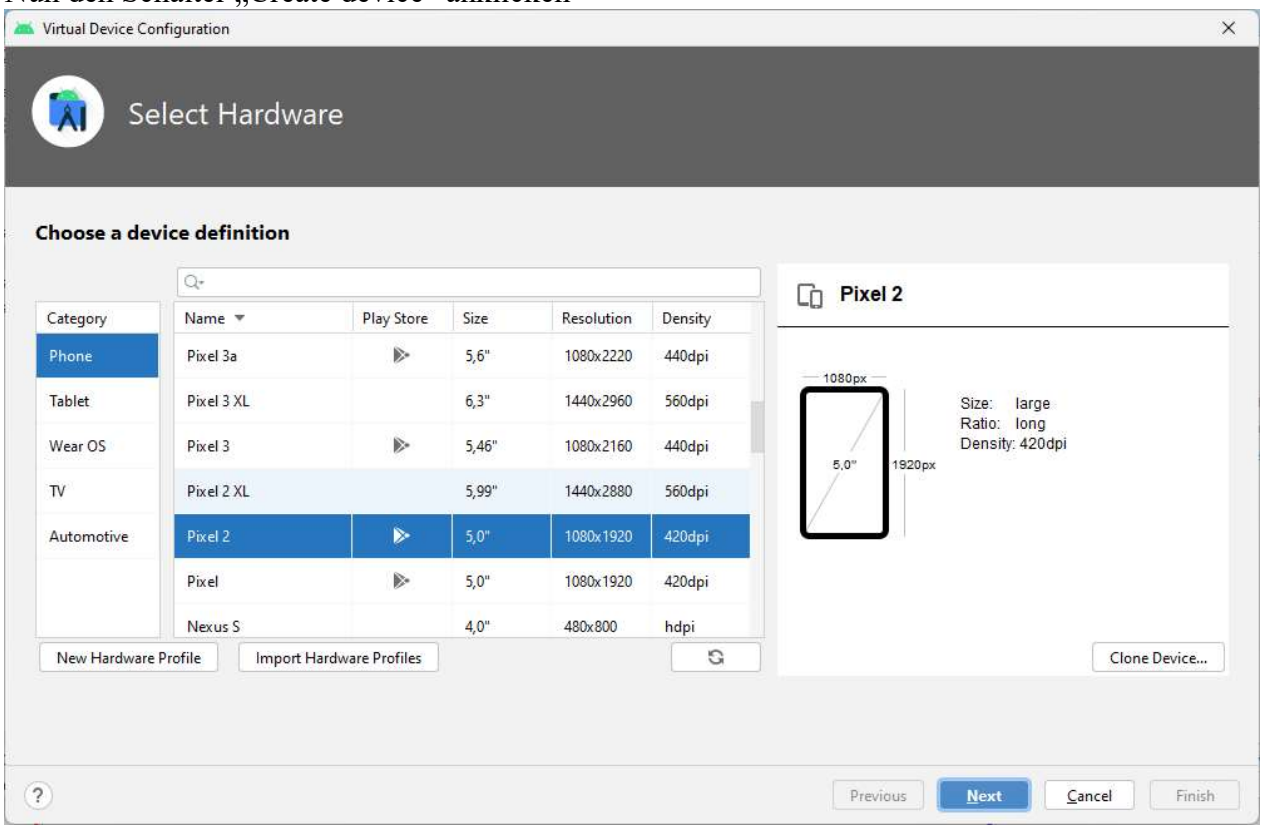


Abbildung 26 Auswahl des Smartphones

Bitte wählen Sie nicht ein Smartphone aus, welches eine zu hohe Auflösung hat. Die Anzeige des „Smartphone“ muss ja auch auf Ihrem Bildschirm passen.

Nun den Schalter „Next“ anklicken.

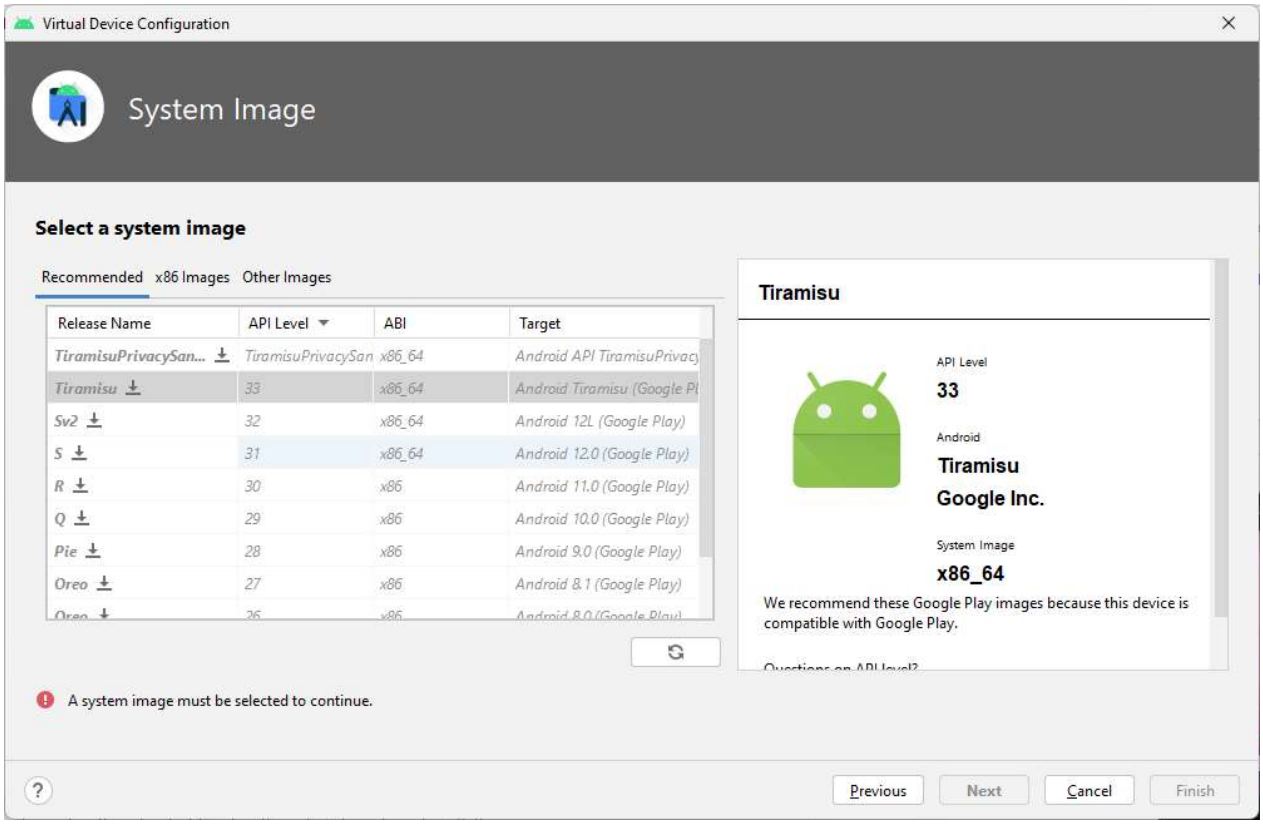


Abbildung 27 Auswahl des System-Images für das virtuelle Device

Das Symbol ⬇ zeigt an, dass man dieses System-Images runterladen muss.
Nun den Eintrag „R“ anklicken. Dann wird das System-Image runtergeladen.

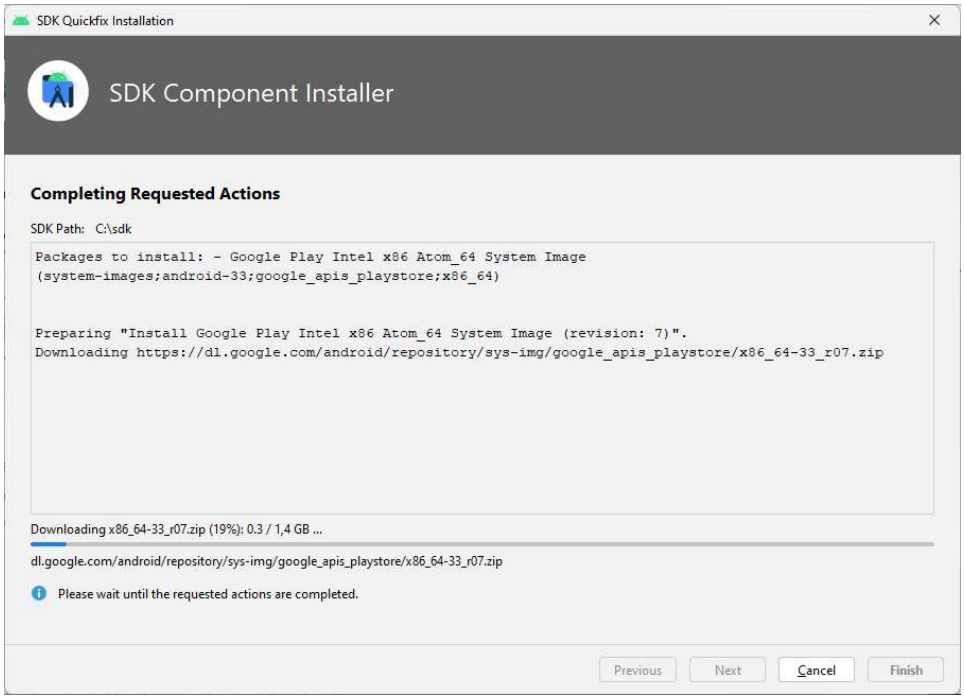


Abbildung 28 Laden des System-Images 33

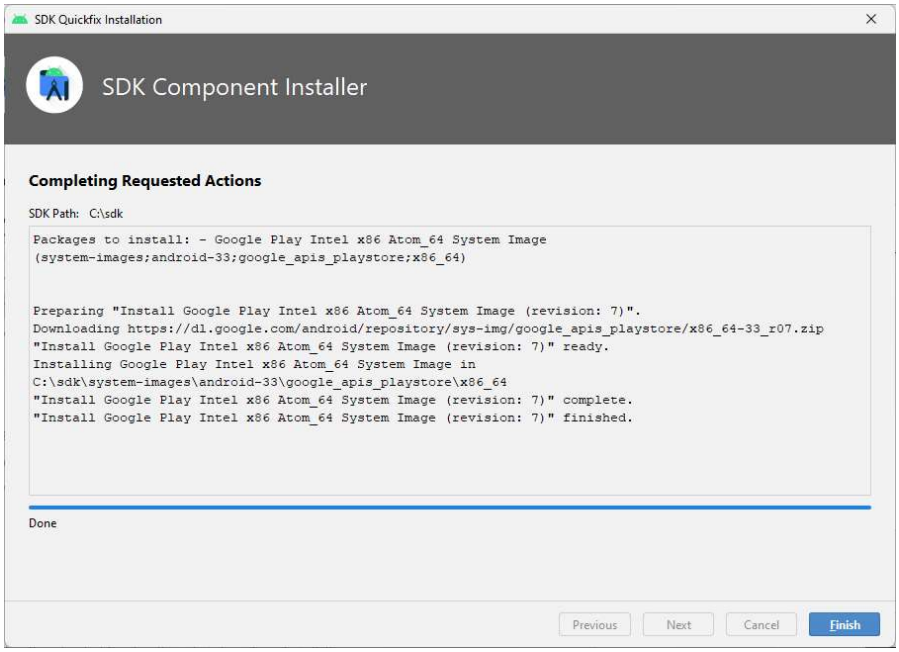


Abbildung 29 Nun ist ein weiterer Download fertig

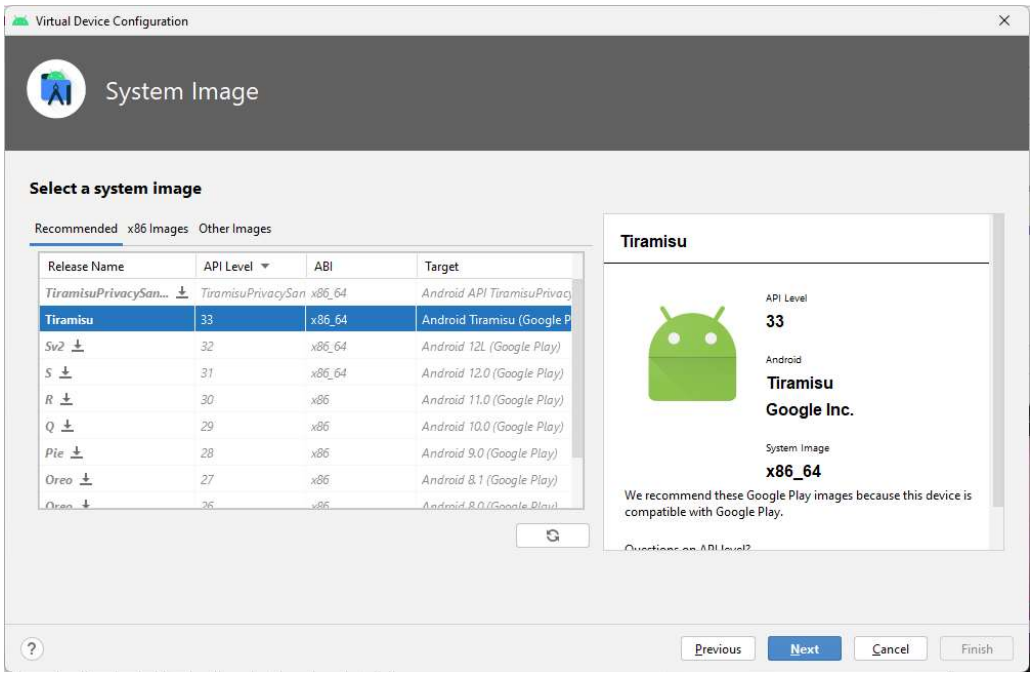


Abbildung 30 Anzeige, dass System-Image 30 installiert wurde

Schalter „Next“ bestätigen

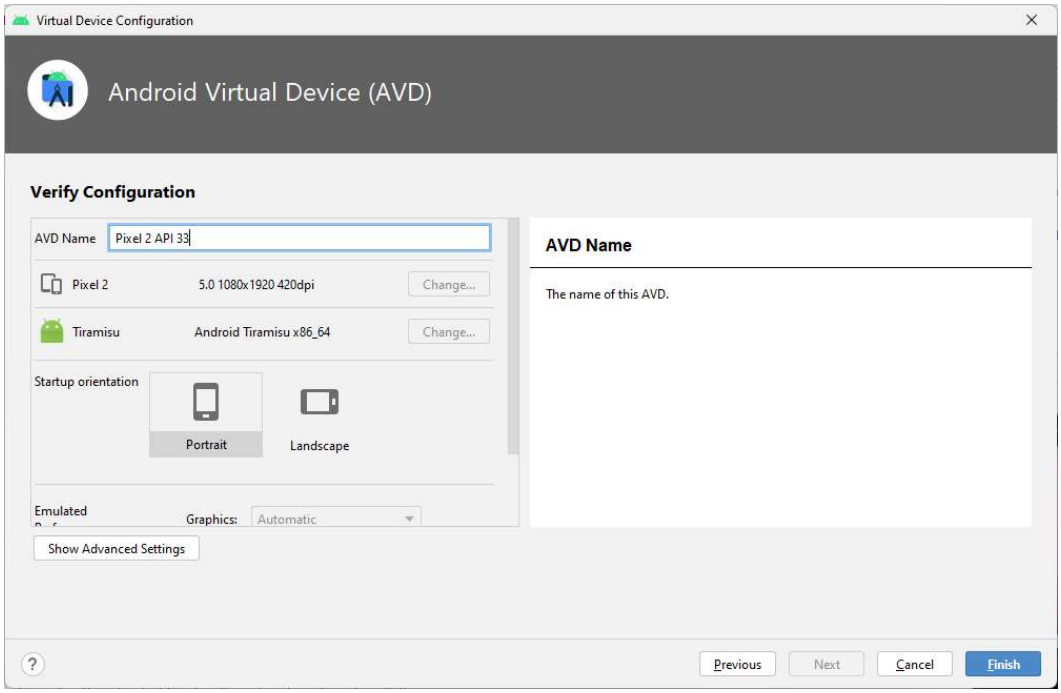


Abbildung 31 Hier kann man weitere Optionen setzen

Anzeige der Erweiterten Optionen:

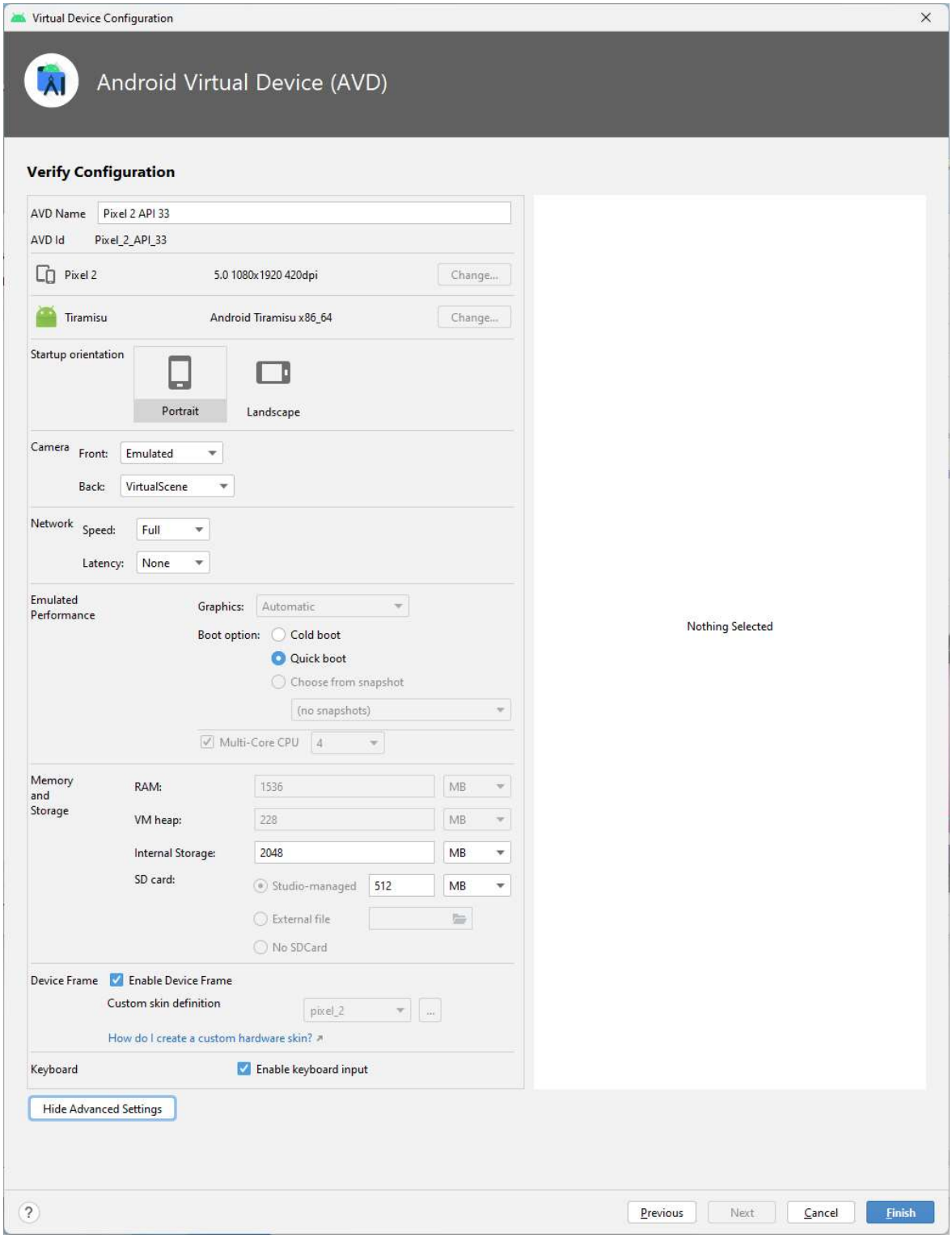


Abbildung 32 Anzeige der Erweiterten Optionen

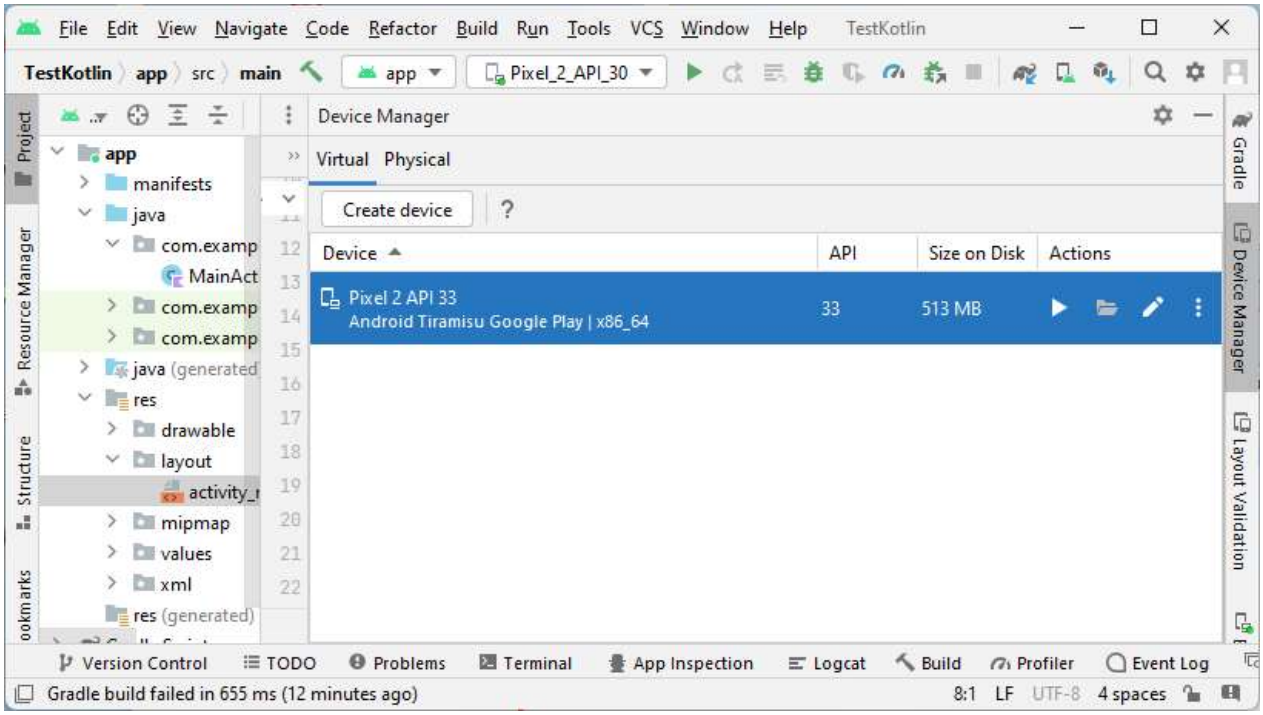


Abbildung 33 „Abschlussdialog“

Nun starten des Projektes mit

- Menu Run, Eintrag Run 'App'
- oder
- Shift F10

Ergebnis:

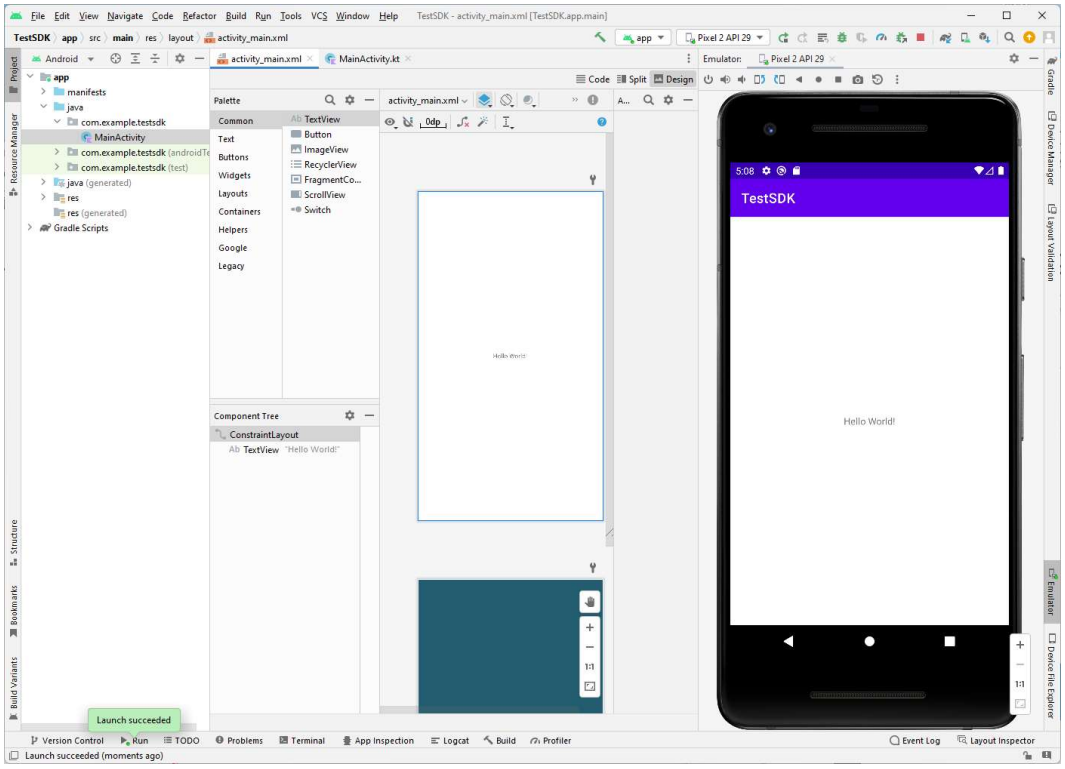


Abbildung 34 Anzeige des Emulators in Android Studio

Sie können aber auch das Emulator-Fenster separat anzeigen lassen.

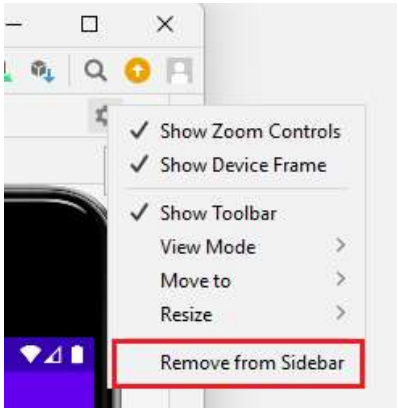


Abbildung 35 das Emulator-Fenster separat anzeigen lassen

4 Kotlin

4.1 Kotlin Advantages

Following are some of the advantages of using Kotlin for your application development.

1. Easy Language − Kotlin supports object-oriented and functional constructs and very easy to learn. The syntax is pretty much similar to Java, hence for any Java programmer it is very easy to remember any Kotlin Syntax.

2. Very Concise − Kotlin is based on Java Virtual Machine (JVM) and it is a functional language. Thus, it reduce lots of boiler plate code used in other programming languages.

3. Runtime and Performance − Kotlin gives a better performance and small runtime for any application.

4. Interoperability − Kotlin is mature enough to build an interoperable application in a less complex manner.

5. Brand New − Kotlin is a brand new language that gives developers a fresh start. It is not a replacement of Java, though it is developed over JVM. Kotlin has been accepted as the first official language of Android Application Development. Kotlin can also be defined as - **Kotlin = Java + Extra updated new features**.

4.2 main

```
fun main() {
   var string: String  = "Hello, World!"
   println("$string")
}
```

```
fun main(args: Array<String>){
    println("Hello, world!")
}
```

4.3 Semicolon (;) in Kotlin

Kotlin code statements do not require a semicolon (;) to end the statement like many other programming languages, such as Java, C++, C#, etc. do need it.

4.4 Kommentare

```
// This is a comment
/* This is a multi-line comment and it can span
 * as many lines as you like
 */
```

4.5 Keywords

4.5.1 (a) Kotlin Hard Keywords

as	as?	break	class
continue	do	else	false
for	fun	if	in
!in	interface	is	!is
null	object	package	return
super	this	throw	true
try	typealias	typeof	val
var	when	while	

4.5.2 (b) Kotlin Soft Keywords

Following is the list of keywords (soft) in the context when they are applicable and can be used as identifiers in other contexts:

by	catch	constructor	delegate
dynamic	field	file	finally
get	import	init	param
property	receiver	set	setparam
value	where		

4.5.3 (c) Kotlin Modifier Keywords

Following is the list of tokens which act as keywords in modifier lists of declarations and can be used as identifiers in other contexts:

actual	abstract	annotation	companion
const	crossinline	data	enum
expect	external	final	infix
inline	inner	internal	lateinit
noinline	open	operator	out
override	private	protected	public
reified	sealed	suspend	tailrec
vararg			

4.6 Namensgebung

A-ZÜÄÖß #%

4.7 Variablen

```
val  alter = 12    Konstante
var alter  = 12    Variable, Mutable
val MAX = 1000   Konstante
val  name = "Anton"  Konstante
println("Name = $name")
println("Alter = $alter")
var s:String="abc"
var s="abc"                    // automatisch ein String
val MAX:Int=100
val MAX=100
var n=100                      // kann verändert werden
```

4.8 Datentypen

Data Type	Size (bits)	Data Range
Byte	8 bit	-128 to 127
Short	16 bit	-32768 to 32767
Int	32 bit	-2,147,483,648 to 2,147,483,647
Long	64 bit	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bit	1.4012984632481707e-45 to 3.40282346638528860e+38
Double	64 bit	4.94065645841246544e-324 to 1.79769313486231570e+308
Boolean		true / false
Char		'a'
String		"abc"
Escape		\r, \n, \t, \b
Raw String		var rawString :String = """This is going to be a multi-line string and will not have any escape sequence""";

4.9 String

String: index von 0 bis n-1
val s="abc"
s.length integer 3
s.count() "3"
s.lastindex 2
s.uppercase()
s.lowercase()
t="def"
u=s.plus(t) abcdef
u.drop(2) cdef
u.dropLaste(2) abcd
u.indexOf("cd") 2 -1 = nicht vorhanden
u.indexOf("cd", startindex, ignoreCase) 2
u.compareTo(s) 0=equals 1=ungleich

4.10 Null-Werte

Ein normaler Datentyp hat KEINE null-Werte!

```
val boolNull: Boolean? = null
var alterNull : Int? = 12
var alterNull : Int? = null
```

4.11 Konvertierung

- toByte()
- toShort()
- toInt()
- toLong()
- toFloat()
- toDouble()
- toChar()

```
val x: Int = 100
val y: Long = x  // Not valid assignment
val y: Long = x.toLong()
```

4.12 Arithmetische Operatoren

- \+ +=
- \- -=
- * *=
- / /=
- % %=
- ++x
- x++
- --x
- x--
- ! invert Boolean

4.13 Relationale Operatoren

- >
- <
- >=
- <=
- ==
- !=

4.14 Logische Operatoren

- &&
- ||
- !

4.15 Bitwise Operations

Kotlin does not have **any bitwise operators** but Kotlin provides a list of helper functions to perform bitwise operations.

Following is the list of Kotlin Bitwise Functions:

Function	Description	Example
shl (bits)	signed shift left	x.shl(y)
shr (bits)	signed shift right	x.shr(y)
ushr (bits)	unsigned shift right	x.ushr(y)
and (bits)	bitwise and	x.and(y)
or (bits)	bitwise or	x.or(y)
xor (bits)	bitwise xor	x.xor(y)
inv()	bitwise inverse	x.inv()

```
var x:Int = 60  // 60 = 0011 1100
x.shl(2) = 240
x.shr(2) = 15
x.and(y)  = 12
x.or(y)  = 61
x.xor(y)  = 49
x.inv()  = -61
```

4.16 logische Operatoren (and / or)

Kotlin provides and() and or() functions to perform logical AND and logical OR operations between two boolean operands.

These functions are different from && and || operators because these functions do not perform short-circuit evaluation but they always evaluate **both** the operands.

```
val x: Boolean = true
val y: Boolean = false
val z: Boolean = true
x.and(y) = false
```

```
x.or(y) = true
x.and(z) = true
```

4.17 Arrays

```
val fruits = arrayOf("Apple", "Mango", "Banana", "Orange")
val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")
val num = intArrayOf(1, 2, 3, 4)
Parameter in einer Funktion:
   Feld:Array<Float>
```

- byteArrayOf()
- shortArrayOf()
- intArrayOf
- longArrayOf()
- charArrayOf()

4.17.1 Zugriff

s= fruits[3]
fruits[3]="Kirschen"
s= fruits.get(3)
fruits.set(3, "Kirschen")

4.17.2 Länge:

Länge des Feldes: size

4.17.3 Schleifen

```
val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")
for( item in fruits ){
  println( item )
}
if ("Mango" in fruits){
   println( "Mango exists in fruits" )
}else{
   println( "Mango does not exist in fruits" )
}
```

4.17.4 Parameter

```
// Deklarieren
val templiste = arrayOf<String>("Celsius", "Kelvin","Fahrenheit")

// Funktionsaufruf
functionString(templiste)

// Funktionsdeklaration
fun functionString(liste:Array<String>) {  }
```

4.17.5 distinct

```
val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange", "Apple")
val distinct = fruits.distinct()
for( item in distinct ){
   println( item )
}
```

4.17.6 empty

```
fruits.isEmpty()
```

4.18 range, funktioniert anders als in Python

1..10 // Range of integers starting from 1 to 10
a..z // Range of characters starting from a to z
A..Z // Range of capital characters starting from A to Z

```
for ( num in 1.rangeTo(4) ) {
   println(num)          1,2,3,4
}
for ( num in 1..4 ) {
   println(num)          1,2,3,4
}
for ( num in 1.. until 4 ) {        <
   println(num)          1,2,3
}

for ( num in 4 downTo 1 ) {
   println(num)          4,3,2,1
}
for ( num in 1..10 step 2 ) {
   println(num)          1,3,5,7,9
}
for ( ch in 'a'..'e' ) {
   println(ch)         a,b,c,d,e
}
for ( ch in ('a'..'e').reversed ) {
   println(ch)         e,d,c,b,a
}
```

}

4.18.1 Filter Ranges

```
val a = 1..10
val f = a.filter { T -> T % 2 == 0 }     [2,4,6,8,10]
```

4.18.2 Range-Utility Funktionen

```
val a = 1..10
println(a.min())         1
println(a.max())         10
println(a.sum())         55
println(a.average())       5.5
println(a.count())       10
```

4.19 If bedingungen (if else, when, for, while)

```
if (condition) {
   // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

4.19.1 Kotlin if...else Expression

```
val result = if (condition) {
   // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

Beispiel:

```
val result = if (age > 18) {
     "Adult"
  } else {
     "Minor"
  }
```

Alternative:

```
val result = if (age > 18) "Adult" else  "Minor"
val max = if (a > b) a else b
```

4.20 When Expression (switch)

```
val result = when (day) {
   1 -> "Monday"
   2 -> "Tuesday"
   3 -> "Wednesday"
   4 -> "Thursday"
   5 -> "Friday"
   6 -> "Saturday"
   7 -> "Sunday"
   else -> "Invalid day."
 }
```

Oder

```
val day = 2
when (day) {
  1 -> println("Monday")
  2 -> println("Tuesday")
  3 -> println("Wednesday")
  4 -> println("Thursday")
  5 -> println("Friday")
  6 -> println("Saturday")
  7 -> println("Sunday")
  else -> println("Invalid day.")
}
```

Kombinierte Abfragen:

```
val day = 2
when (day) {
   1, 2, 3, 4, 5 -> {
     println("Weekday")
     println("Wann ist Wochenende?")
   }
   else -> println("Weekend")
}
```

4.20.1 Range in when Conditions

```
val day = 2
when (day) {
   in 1..5 -> println("Weekday")
   else -> println("Weekend")
}
```

4.20.2 Expression in when Conditions

```
val x = 20
val y = 10
val z = 10
when (x) {
   (y+z) -> print("y + z = x = $x")
   else -> print("Condition is not satisfied")
}
```

Abfrage nach einem Type

```
var num: Any = "GeeksforGeeks"
when(num){
   is Int -> println("It is an Integer")
   is String -> println("It is a String")
   is Double -> println("It is a Double")
}
```

Funktion benutzen

```
var num = 8
when{
   isOdd(num) ->println("Odd")
   isEven(num) -> println("Even")
   else -> println("Neither even nor odd")
}

when  {                              // ohne Bedingung
   isOdd(num) ->println("Odd")
   !isOdd(num) -> println("Even")
   else -> println("Neither even nor odd")
}
```

4.21 For-Schleife

https://www.tutorialspoint.com/kotlin/kotlin_for_loop.htm

```
val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")
for( item in fruits ){
  println( item )
}

for (item in 1..5) {          // 1,2,3,4,5
   println(item)
}
```

1..10 // Range of integers starting from 1 to 10
a..z // Range of characters starting from a to z
A..Z // Range of capital characters starting from A to Z

```
for ( num in 1.rangeTo(4) ) {
   println(num)          1,2,3,4
}
for ( num in 1..4 ) {
   println(num)          1,2,3,4
}
for ( num in 1.. until 4 ) {        <
   println(num)          1,2,3
}

for ( num in 4 downTo 1 ) {
   println(num)          4,3,2,1
}
for ( num in 1..10 step 2 ) {
   println(num)          1,3,5,7,9
}
for ( ch in 'a'..'e' ) {
   println(ch)         a,b,c,d,e
}
for ( ch in ('a'..'e').reversed ) {
   println(ch)         e,d,c,b,a
```

}

4.21.1 Filter Ranges

```
val a = 1..10
val f = a.filter { T -> T % 2 == 0 }     [2,4,6,8,10]
```

Optional: break und continue

```
outerLoop@ for (i in 1..3) {
   innerLoop@ for (j in 1..3) {
       println("i = $i and j = $j")
       if (i == 2){
           break@outerLoop
       }
   }
}
```

4.22 While-Schleife

```
var i = 5;
while (i > 0) {
   println(i)
   i--
}
```

```
var i = 5;
do{
    println(i)
    i--
}while(i > 0)
```

Optional: break und continue

```
var i=1
outerLoop@ while (i<4) {
   var j=1
   innerLoop@ while (j<4) {
       println("i = $i and j = $j")
       if (i == 2){
           break@outerLoop
       }
   }
}
```

4.23 Funktionen / Methoden

```
fun myfunc() {
}
fun myfunc() {
}
fun printSum(a:Int, b:Int){
   println(a + b)
}
fun add(a:Int, b:Int):Int{
   sum = a + b
   return sum
}
```

4.23.1 void Funktion

```
fun printSum(a:Int, b:Int):Unit{
   println(a + b)
}
fun printSum(a:Int, b:Int){    Unit wird automatisch gesetzt
   println(a + b)
}
```

4.23.2 Rekursive Funktionen

```
fun factorial(x:Int):Int{
   val result:Int
   if( x <= 1){
      result = x
   }else{
      result = x*factorial(x-1)
   }
   return result
}
```

4.24 Kotlin Tail Recursion

A recursive function is eligible for tail recursion if the function call to itself is the last operation it performs.
Following is a Kotlin program to calculate the factorial of number 10 using tail recursion. Here we need to ensure that the multiplication is done before the recursive call, not after.

```
fun main(args: Array<String>) {
   val a = 4
   val result = factorial(a)
   println( result )
}

fun factorial(a: Int, accum: Int = 1): Int {
    val result = a * accum
    return if (a <= 1) {
        result
    } else {
        factorial(a - 1, result)
    }
}
```

4.25 FunktionsPointer oder Higher-Order Functions

```
fun main(args: Array<String>) {
   val result = calculate(4, 5, ::sum)
```

```
    println( result )                9
    val result = calculate(4, 5, ::mult)      20
    println( result )
}
fun sum(a: Int, b: Int) = a + b            Kurzschreibweise
fun mult(a: Int, b: Int) = a * b           Kurzschreibweise

fun calculate(a: Int, b: Int, operation:(Int, Int) -> Int): Int {
    return operation(a, b)
}
```

4.26 Lambda Funktionen

Kotlin lambda is a function which has no name and defined with a curly braces {} which takes zero or more parameters and body of function.

The body of function is written after variable (if any) followed by -> operator.
Syntax: `{variable with type -> body of the function}`

Example:

```
fun main(args: Array<String>) {
    val upperCase = { str: String -> str.toUpperCase() }
    println( upperCase("hello, world!") )
}
```

Parameter: **str: String**
Funktionsbody: **str.toUpperCase()**

4.27 inline Funktionen

An **inline** function is declared with **inline** keyword. The use of inline function enhances the performance of higher order function. The inline function tells the compiler to copy parameters and functions to the call site.

Example:

```
fun main(args: Array<String>) {
    myFunction(  {println("Inline function parameter")}  )  // λ ohne Parameter
}
// {println("Inline function parameter")}   ist ein Funktionsparameter !
inline fun myFunction(function:()-> Unit){
    println("I am inline function - A")
    function()
    println("I am inline function - B")
}
```

4.28 Immutable Listen (nicht veränderbar)

- listOf()
- listOf<T>()

Beispiele:

```
val numbers = listOf("one", "two", "three", "four")
println(numbers.toString())
```

oder

```
val itr = theList.listIterator()
    while (itr.hasNext()) {
        println(itr.next())
    }
```

oder

```
val theList = listOf("one", "two", "three", "four")
for (i in theList.indices) {
   println(theList[i])
}
```

oder

```
val theList = listOf("one", "two", "three", "four")
theList.forEach { println(it) }
```

Methoden von Listen:

- size()
- if (a in theList) {}
- if (theList.contain(a)) {}
- isEmpty()
- indexOf(a) // von 0 bis n-1
- get(index)
- list3 = list1+list2 // Operator-Overloading
- list3 = list1-list2 // Operator-Overloading
- list3 = list1.slice(2..4) // 2,3,4
- list3 = list1.filterNotNull() // kopy
- list3 = list1.filter(a>10) // kopy
- list1.drop(1) // erste Elemente gelöscht
- list1.drop(3) // die ersten DREI Elemente werden gelöscht
- listArray = list1.groupBy(a %3) // listArray enthält DREI TeilListen
- listeKeyMap = list1.map(a/10) // list enthält die Keys für eine HashMap
- list3 = list1.chunked(3) // list3 enthält TeilListen mit jeweils DREI Elemente (beim Letzten?)
- list3 = list1.windowed(size) // list3 enthält TeilListen mit jeweils DREI Elemente (beim Letzten?)
    - val list1 = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    - val list3 = list1.windowed(33)
    - [[10, 12, 30], [12, 30, 31], [30, 31, 40], [31, 40, 9], [40, 9, -3], [9, -3, 0]]
- list3 = list1.windowed(size,step) // list3 enthält TeilListen mit jeweils DREI Elemente (beim Letzten?)
    - val list1 = listOf(10, 12, 30, 31, 40, 9, -3, 0)
    - val list3 = list1.windowed(3,3)
    - [[10, 12, 30], [31, 40, 9]]
- add(obj)
- add(index,obj)
- remove(obj)

https://www.tutorialspoint.com/kotlin/kotlin_sets.htm

4.29 Mutable Listen (veränderbare Listen)

- ArrayList<T>()
- arrayListOf()
- mutableListOf()
- **Parameter:**
    - vorlesung:MutableList<Student>

Beispiel:

```
val numbers = mutableListOf("one", "two", "three", "four")
numbers.add("five")
```

4.30 Immutable Set

```
val theSet = setOf("one", "two", "three", "four")    // nicht änderbar
```

Methoden:

- toString()
- asIterable().iterator()
- indices
- size()
- in // true / false
- contains(…)
- isEmpty
- indexOf(…) // von 0
- elementAt(i)
- set3 = set1 + set2
- set3 = set1 - set2
- filterNotNull() // löscht null-Werte
- sortedDescending()
- sortedAscending()
- drop(n) // löscht die ersten n-Elemente
- groupBy{ it % 3}
- map{ it / 3 }
- chunked(3) // Liste mit jeweils 3 Elementen
- windowed(3) // Liste mit 3 Elementen, jeweils um eins verschoben

```
val itr = theSet.asIterable().iterator()
while (itr.hasNext()) {
    println(itr.next())
}
for (i in theSet.indices) {
    println(theSet.elementAt(i))
}
theSet.forEach { println(it) }
```

4.31 Mutable Sets

```
val theMutableSet = mutableSetOf("one", "two", "three", "four")
```

Methoden: siehe Immutable Set

4.32 Immutable Map

Methoden:

- entries() Anzeige key/value
- keys()
- values()
- toString()
- size identisch mit count()
- count()
- containsKey("two")
- containsValue("two")
- isEmpty()
- get("two") identisch mit ["two"], wenn nicht vorhanden?
- ["two"]
- val map3 = map1 + map2
- val map3 = map1 - map2
- remove("two") identisch mit theMap -= listOf("two")
- theMap -= listOf("two")
- toSortedMap()
- filterKeys{ it == "two"}
- filter{ it.key == "two" || it.value == 4}

```
val theMap = mapOf("one" to 1, "two" to 2, "three" to 3, "four" to 4)
println(theMap)
```

Klassisch:

```
val theMap = HashMap<String, Int>()
theMap["one"] = 1
theMap["two"] = 2
theMap["three"] = 3
theMap["four"] = 4
```

oder

```
val theMap = mapOf(Pair("one", 1), Pair("two", 2), Pair("three", 3))
```

Iterator:

```
val itr = theMap.keys.iterator()
while (itr.hasNext()) {
    val key = itr.next()
    val value = theMap[key]
    println("${key}=$value")
}
```

For:

```
for ((k, v) in theMap) {
      println("$k = $v")
}
theMap.forEach {
      k, v -> println("Key = $k, Value = $v")
}
```

4.33 Mutable Map

Methoden:

- entries() Anzeige key/value
- keys()
- values()
- toString()
- size identisch mit count()
- count()
- containsKey("two")
- containsValue("two")
- isEmpty()
- get("two") identisch mit ["two"], wenn nicht vorhanden?
- ["two"]
- val map3 = map1 + map2
- val map3 = map1 - map2
- remove("two") identisch mit theMap -= listOf("two")
- theMap -= listOf("two")
- toSortedMap()
- filterKeys{ it == "two"}
- filter{ it.key == "two" || it.value == 4}

- HashMap
- hashMapOf()
- mutableMapOf()

```
val theMutableMap = mutableMapOf("one" to 1, "two" to 2, "three" to 3, "four" to
4)
println(theMutableMap)
```

Hinzufügen:

```
theMap.put("four", 4)
theMap["five"] = 5
theMap.remove("two")
```

4.34 enum

```
enum class Direction {
    NORTH, SOUTH, WEST, EAST
}
```

```
enum class Color(val rgb: Int) {
    RED(0xFF0000),
    GREEN(0x00FF00),
    BLUE(0x0000FF)
}
```

4.35 Generic, keine Wildcards Generic (?)

```
class Box<T>(t: T) {
    var value = t
}
val box: Box<Int> = Box<Int>(1)
```

oder

```
val box = Box(1)      // 1 has type Int, ➔ it is Box<Int>
```

4.36 Klassen

```
class ClassName { // Class Header
   // private, protected, public, internal (innerhalb des Moduls)
    var a:Int;
   // Variables or data members
   // Member functions or Methods
   constructor (a:Int) {
    this.a=a
   }
   constructor ():this(42)              // !!
}
```

Constructor wird per Parameter erzeugt!, ohne Zuweisung

```
class Person constructor(val firstName: String, val age: Int=20) {
   init {
      this.firstName = firstName
      this.age = _age
   }
}

class Student ( var lastname:String="???", var mnr:Int=123)
  val std = Student2("Annemarie",42)
  val s:String="std: lastname: "+std2a.lastname+ "  mnr: "+std2a.mnr

// durch data nun mit equals, hash und toString
data class Student ( var lastname:String="???", var mnr:Int=123)
  val std = Student3()
  val s:String="std: "+std3b.toString() // Student(lastname="???", mnr=123)
  val std2 = std.copy()
  val s:String="std=std2 "+std.equals(std2)       // true
```

```
abstract class Tutorial {
        abstract var year: Int
        abstract var name: String
    }
    open class ParentClass {
        var var1 = false
        var var2: String? = null
    }

    data class ChildClass(
        var var3: Long
    ) : ParentClass()

    data class Book (
        override var year: Int = 2022,
        override var name: String = "Kotlin Tutorials",
        var isbn: String
    ) : Tutorial()
```

Keywords:

- private, protected, public, internal
- open class
  - damit ableitbar, aber ohne toString()
- open variable
- data class
  - data class können nicht abgeleitet werden
  - können aber eine Oberklasse haben
  - können erweitert werden durch Interface

- sealed class
- override
- inner class
  - Zugriff auf die Variablen, Methoden der Hauptklasse à la Java
- Nested class, innerClass
  - Ist static (default)
  - **Kein** Zugriff auf die Variablen, Methoden der Hauptklasse à la Java
- Properties à la C#
  - class Person () {
  - var lastname: String=""
  - get() {
  - return **field**
  - }
  - set(value) {
  - if (!value.isEmpty()) {
  - **field**=value
  - }
  - }
  - constructor(lastname:String):this() {
  - this.lastname = lastname
  - }
  - }
- : Superclass
  - : statt extends
- Ohne **new**
- Default Parameter
- super à Java
- interface kann echte Funktionen haben
- „Multivererbung“

Beispiele:

```
class OuterClass{
   // Members of Outer Class
   class NestedClass{
      // Members of Nested Class
   }
}
```

```
val obj = Outer.Nested()

   print(obj.foo())
```

```
class OuterClass{
   // Members of Outer Class
   class inner InnerClass{
      // Members of Inner Class
   }
}
```

```
val outerObj = OuterClass()
val innerObj = outerObj.InnerClass()
```

```
open class Person1 (var lastname:String, var firstname:String="-")

class StudentPerson1 (lastname:String, firstname:String="-", var
```

```
mnr:Int):Person1(lastname,firstname)

fun bn9_Click(view: View) {
    val p1 = Person1("Bates","Kim")
    var s:String = "p1: lastname: "+p1.lastname

val std1 = StudentPerson1("Bates","Kim",123)
    s=s+"\n"+ "std1: lastname: "+std1.lastname+ " firstname: "+std1.firstname+"
mnr: "+std1.mnr
    show("std1: "+s)

}
```

```
open class Person2 () {
    var lastname: String=""
    get() {
        return field
    }
    set(value) {
        if (!value.isEmpty()) {
            field=value
        }
    }
    var firstname: String=""
        get() {
            return field
        }
        set(value) {
            if (!value.isEmpty()) {
                field=value
            }
        }

    constructor(lastname:String, firstname:String):this() {
        this.lastname = lastname
        this.firstname = firstname
    }
}
```

4.36.1 Type Aliases

```
fun userInfo():Triple<String, String, Int>{
   return Triple("Zara","Ali",21)
}

typealias User = Triple<String, String, Int>
```

Neu:

```
fun userInfo():User{
   return Triple("Zara","Ali",21)
}
```

Beispiel:

```
open class Outer {
    private val a = 1
    protected open val b = 2
    internal open val c = 3
    val d = 4  // public by default
```

```
    protected class Nested {
        public val e: Int = 5
    }
}

class Subclass : Outer() {
    // a is not visible
    // b, c and d are visible
    // Nested and e are visible

    override val b = 5   // 'b' is protected
    override val c = 7   // 'c' is internal
}

class Unrelated(o: Outer) {
    // o.a, o.b are not visible
    // o.c and o.d are visible (same module)
    // Outer.Nested is not visible, and Nested::e is not visible either
}
```

4.36.2 „Multivererbung“

```
open class Rectangle {
    open fun draw() { /* ... */ }
}

interface Polygon {
    fun draw() { /* ... */ } // interface members are 'open' by default
}

class Square() : Rectangle(), Polygon {
    // The compiler requires draw() to be overridden:
    override fun draw() {
        super<Rectangle>.draw() // call to Rectangle.draw()
        super<Polygon>.draw() // call to Polygon.draw()
    }
}
```

4.37 Abstrakte Klassen

Abstrakte Klassen sind immer „open“

```
abstract class Person(_name: String) {
   var name: String
   abstract var age: Int

   // Initializer Block
   init {
      this.name = _name
   }

   abstract fun setPersonAge(_age:Int)
   abstract fun getPersonAge():Int

   fun getPersonName(){
       println("Name = $name")
   }
}
class Employee(_name: String): Person(_name) {
    override var age: Int = 0

    override fun setPersonAge(_age: Int) {
       age = _age
    }
    override fun getPersonAge():Int {
       return age
    }
}

fun main(args: Array<String>) {
   val employee = Employee("Zara")
   var age : Int

   employee.setPersonAge(20)

   age = employee.getPersonAge()

   employee.getPersonName()
   println("Age = $age")

}
```

4.38 Interface

```
interface IPrint  {
   var myVar: Int            // abstract property
   fun print():String    // abstract method

   fun hello() {
      println("Hello there, $myVar")
   }
}
```

1. Variante:
Es folgt die Klasse Person mit einem Default-Konstruktor, einem Contructor und setter und getter.

```
class Person (): IPrint {
    override var myVar: Int=0           // wegen Interface
    var lastname: String=""
    get() {
        return field
    }
    set(value) {
       if (!value.isEmpty()) {
        field=value
       }
    }
    var firstname: String=""
    get() {
      return field
    }
    set(value) {
       if (!value.isEmpty()) field=value
            }

    constructor(lastname:String, firstname:String):this() {
       this.lastname = lastname
       this.firstname = firstname
    }

    override fun toString():String {
       return "lastname: "+lastname+"   firstname: "+firstname
    }

    override fun print():String {       // Interfqace
        return toString()
    }
}
```

2. Variante:

// hier hat die Klasse Person einen ersten Konstruktor, per Parameter.
// aber um mit setter und getter arbeiten zu können, muss man die Parameter zuweisen:
// **var lastname: String=lastname**

```
class Person (lastname:String,  firstname:String): IPrint {
   override var myVar: Int=0           // wegen Interface

   var lastname: String=lastname
   get() {
      return field
   }
   set(value) {
     if (!value.isEmpty()) {
        field=value
     }
   }
   var firstname: String=firstname
   get() {
     return field
   }
   set(value) {
     if (!value.isEmpty()) field=value
   }

   override fun toString():String {
     return "lastname: "+lastname+"   firstname: "+firstname
   }

   override fun print():String {       // Interface
     return toString()
   }
}
```

Es folgt der Aufruf:

```
val p1:Person = Person("Smith","Jason")
p1.firstname="Anja"
p1.firstname=""
var s:String = "Person3: "+p1.print()
p1.myVar = 42
p1.hello()

val p2:Person4 = Person4("Smith","Jason")
p2.firstname="Anja"
p2.firstname=""

s+= "\nPerson4: "+p1.print()
show(s)
```

4.39 Delegation

https://www.tutorialspoint.com/kotlin/kotlin_delegation.htm

Kotlin supports “delegation” design pattern by introducing a new keyword “by”. Using this keyword or delegation methodology, Kotlin allows the derived class to access all the implemented public methods of an interface through a specific object. The following example demonstrates how this happens in Kotlin.
Man kann also auf Methoden

4.40 Extension functions

Diese Eigenschaft ist sehr rar. Man kann Klassen durch einen einfachen Mechanismus, Extension Methods bzw. Properties, erweitern. Kotlin macht davon rege Gebrauch. Die Klasse Collection hat über 165 Erweiterungen.

4.40.1 Beispiele:

Es gibt natürlich viele Funktion in dem mathematischen Package (kotlin.math).
Man kann über Extension alle Funktion in die Klassen Int, Float oder Double einfügen. Die Funktionen sind aber keine statischen Methoden, sondern werden über die aktuelle Variable aufgerufen!

Die Erweiterungen müssen in der Klasse, in der sie aufgerufen werden sollen, definiert werden. Es sollte aber ein Import möglich sein.

Extension Quadrat (eher unsinng):

```
fun Int.sqr(x: Int): Int {
   return x * x
}
```

Extension Quadrat:

```
fun Int.sqr(): Int {
    return this * this
}
```

Extension Wurzel:

```
fun Int.sqrt(): Double {
    return kotlin.math.sqrt(this.toDouble())
}
```

Extension Quadrat:

```
fun Double.sqr(): Double {
    return this * this
}
```

Extension Wurzel:

```
fun Double.sqrt(): Double {
     return kotlin.math.sqrt(this)
}
```

Wie man sieht, kann man mit „this“ auf den aktuellen Wert der Variablen zu greifen.

Aufruf der oberen Beispiele:

```
var i1 = 3
val i2 = i1.sqr(i1)
val i3 = i1.sqr()
val i4 = i1.sqrt()

var d1: Double = 2.0;
val d2 = d1.sqr()
val d3 = d1.sqrt()

val s1 = "i1: " + i1 + "    i2: " + i2 + "    i3: " + i3 + "    i4: " + i4
val s2 = "\nd1: " + d1 + "    d2: " + d2 + "    d3: " + d3
show(s1 + s2)
```

Ergebnis:

```
i1: 3      i2: 9      i3: 9    i4: 1.7320508075688772
d1: 2.0    d2: 4.0             d3: 1.4142135623730951
```

Das nächste zeigt die Erweiterung einer Liste:

```
fun Collection<Int>.mittelwert(): Double {
    return (this.sum().toDouble() / this.size)
}
```

Aufruf:

```
val numbers = mutableListOf<Int>(11, 44, 22, 77)
numbers.add(42)
numbers.add(12)
val summe = numbers.sum()
val anzahl = numbers.size
val xm = numbers.mittelwert()
val s1 =  "summe: " + summe + "    anzahl: " + anzahl + "   xm1: " +
(summe.toDouble() / anzahl)
val s2 = "\nMittelwert: " + xm
```

Ausgabe:

```
summe: 208    anzahl: 6   xm1: 34.666666666666664
Mittelwert: 34.666666666666664
```

4.41 Links

Kotlin:

- https://www.tutorialspoint.com/kotlin/index.htm
- https://kotlinlang.org/docs
- https://www.w3schools.com/kotlin/index.php

Android Studio JetPack Compose:

- https://developer.android.com/jetpack/compose/documentation
- https://developer.android.com/jetpack/compose/tutorial
- https://www.jetpackcompose.net
- https://www.geeksforgeeks.org/listview-in-android-using-jetpack-compose/

5 Stichwortverzeichnis