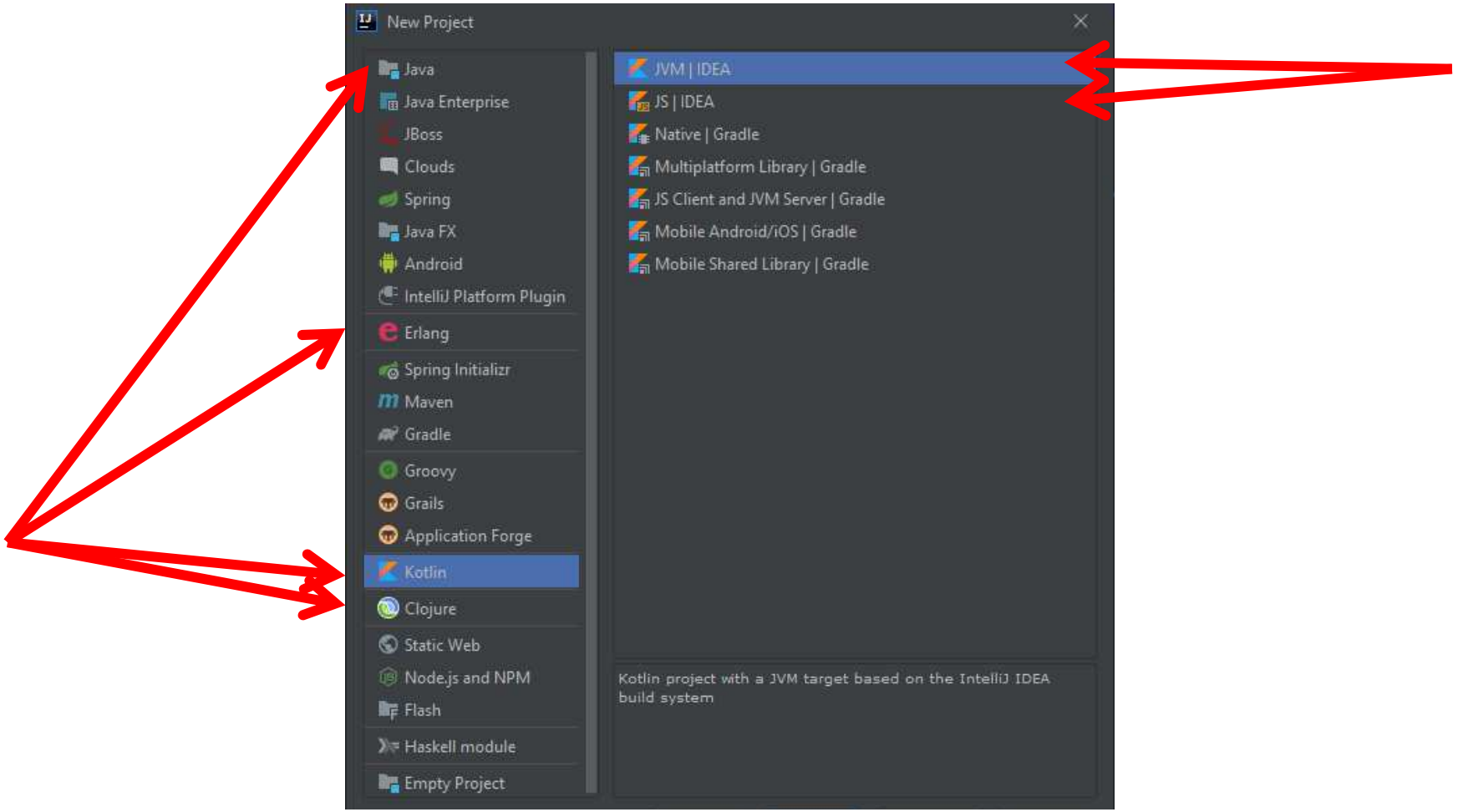
Kotlin

Peter Borovanský, KAI, I-18, borovan(a)ii.fmph.uniba.sk



Kotlin

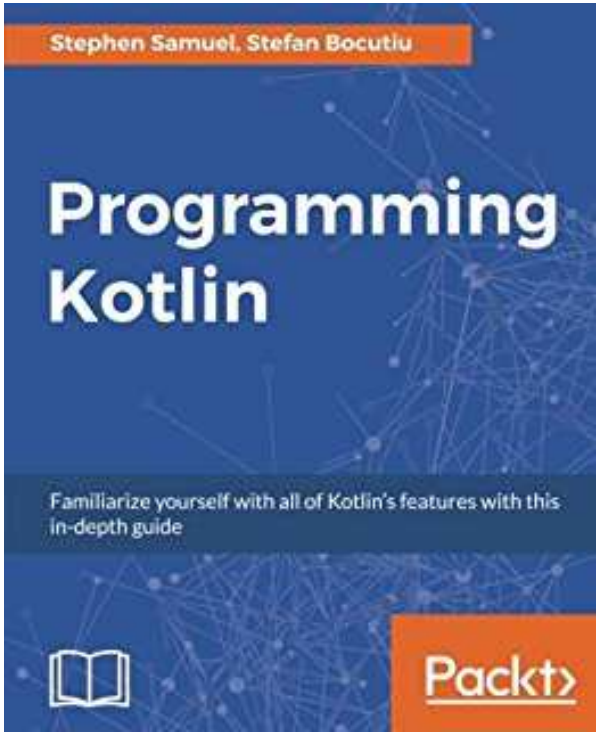
Modern Android development with Kotlin (September 2017) Part 1

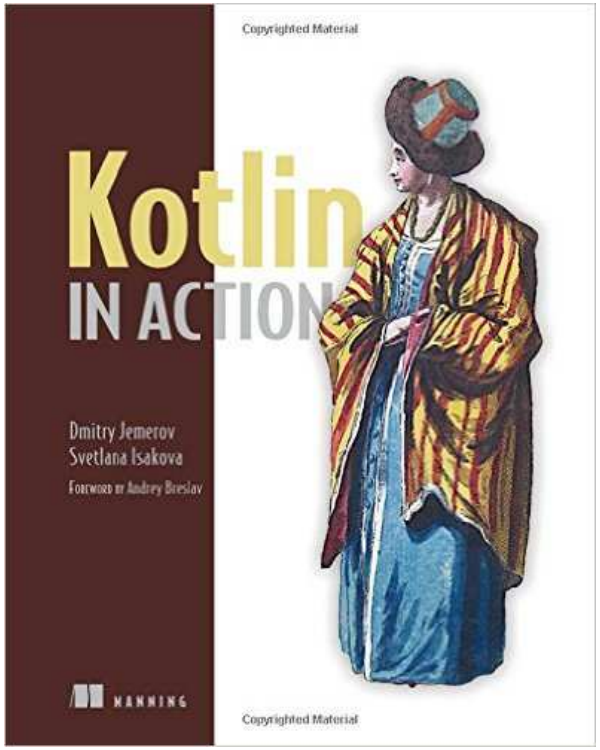
It is really hard to find one project that covers all the things that are new in Android Development, so I decided to write one. In this article we will use the following:

https://proandroiddev.com/modern-android-development-with-kotlin-september-2017-part-1-f976483f7bd6

Literatúra

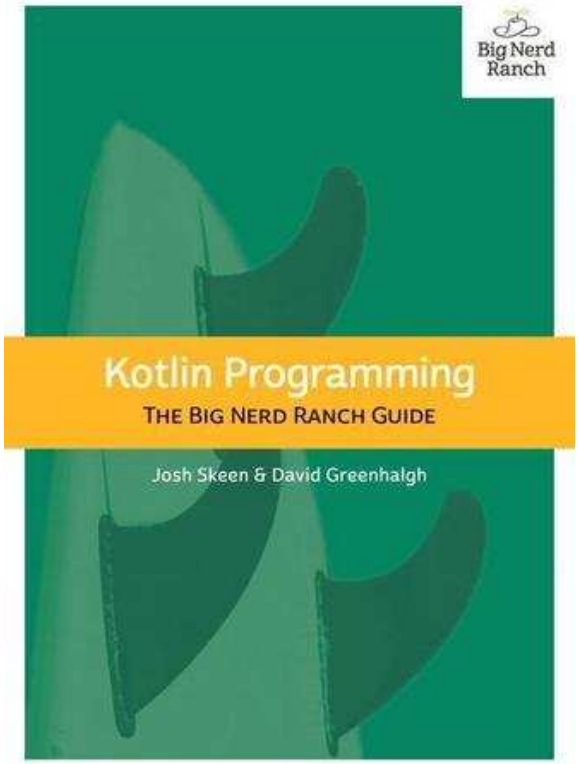
- Kotlin in Action
  https://github.com/panxl6/Kotlin-in-action/blob/master/ebook/Kotlin_in_Action_v12_MEAP.pdf
- Programming in Kotlin
  https://www.packtpub.com/application-development/programming-kotlin

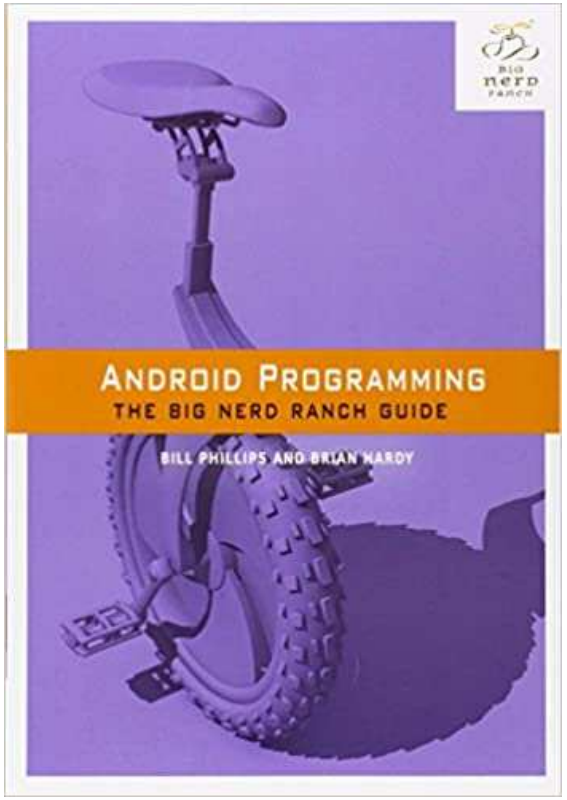




Literatúra – for nerds

- Kotlin Programming – The Big Nerd Ranch Guide
  https://www.megaknihy.sk/programovanie/20375234-kotlin-programming.html
- Android Programming: The Big Nerd Ranch Guide (4th Edition)
  https://www.bignerdranch.com/books/android-programming-the-big-nerd-ranch-guide-4th/





Literatúra

- https://kotlinlang.org/ Kotlin Playground (https://play.kotlinlang.org/)
- Swift is like Kotlin (http://nilhcem.com/swift-is-like-kotlin/)

prekladový slovník pre iOSákov

Swift

```
print("Hello, world!")
```

Kotlin

```
println("Hello, world!")
```

Constants

Swift

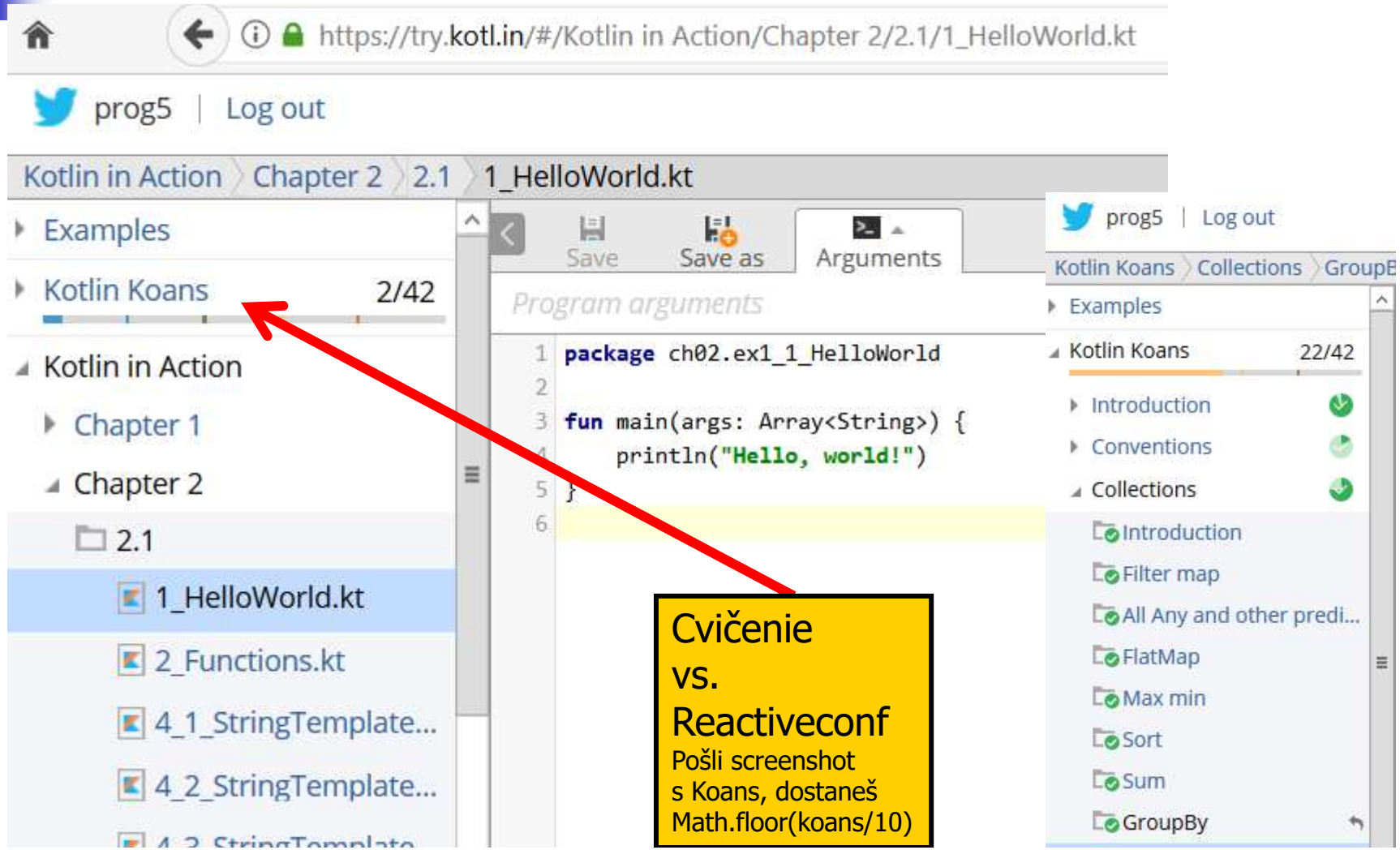
```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

Kotlin

```
var myVariable = 42
myVariable = 50
val myConstant = 42
```

Kotlin Playground

https://play.kotlinlang.org/



Čo sa na try.kotlin naučíte

Kotlin

Progress:30%

- Introduction
  - ✓ Hello, world!
  - ✓ Named arguments
  - ✓ Default arguments
  - ✓ Lambdas
  - ✓ Strings
  - ✓ Data classes
  - ✓ Nullable types
  - ✓ Smart casts
  - ✓ Extension functions
  - ✓ Object expressions
  - ✓ SAM conversions
  - ✓ Extensions on collecti

Kotlin

Progress:48%

- ▸ Introduction
- ▾ Conventions
  - ✓ Comparison
  - ✓ In range
  - ✓ Range to
  - ✓ For loop
  - ✓ Operators overloading
  - ✓ Destructuring declarat
  - ✓ Invoke

Kotlin

Progress:78%

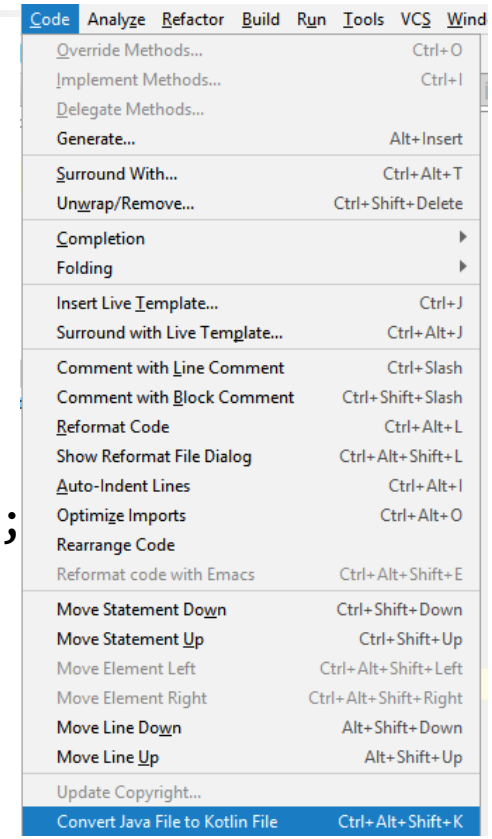
- ▸ Introduction
- ▸ Conventions
- ▾ Collections
  - ✓ Introduction
  - ✓ Filter map
  - ✓ All Any and other predicates
  - ✓ FlatMap
  - ✓ Max min
  - ✓ Sort
  - ✓ Sum
  - ✓ GroupBy
  - ✓ Partition
  - ✓ Fold
  - ✓ Compound tasks
  - ✓ Get used to new style
    - TestShop.kt
    - Shop.kt

Cvičenie

Pošli screenshot s Koans, dostaneš Math.floor(koans/10) resp. Math.floor(% / 20).

Java -> Kotlin

```
public class fib {
    static Integer[] table = new Integer[100];
    private static int fib(int n) {
        Integer result = table[n];
        if (result == null) {
            if (n < 2)
                result = 1;
            else
                result = fib(n - 2) + fib(n - 1);
            table[n] = result;
        }
        return result;
    }
    public static void main(String[] args) {
        for(int i = 0; i<20; i++)
            System.out.println("fib(" + i + ")=" + fib(i));
    }
}
```



Java -> Kotlin

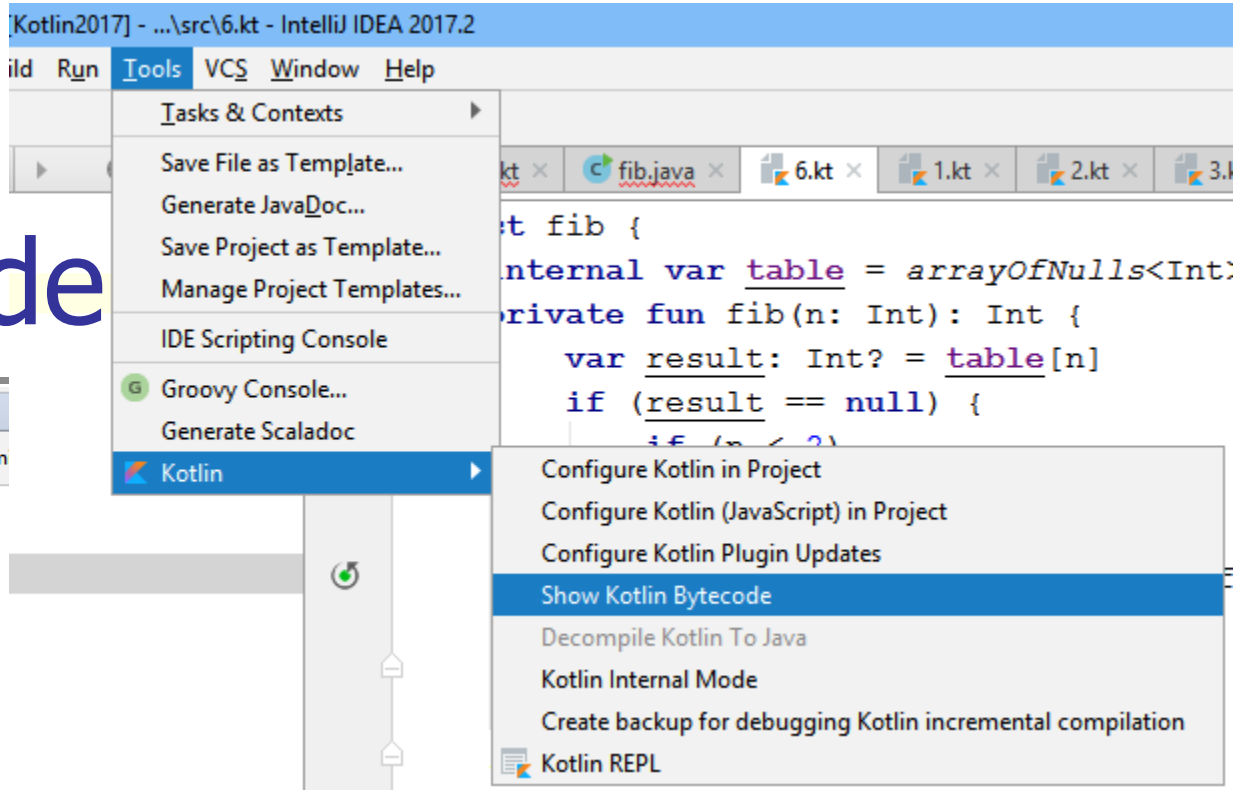
Čo nás prekvapilo

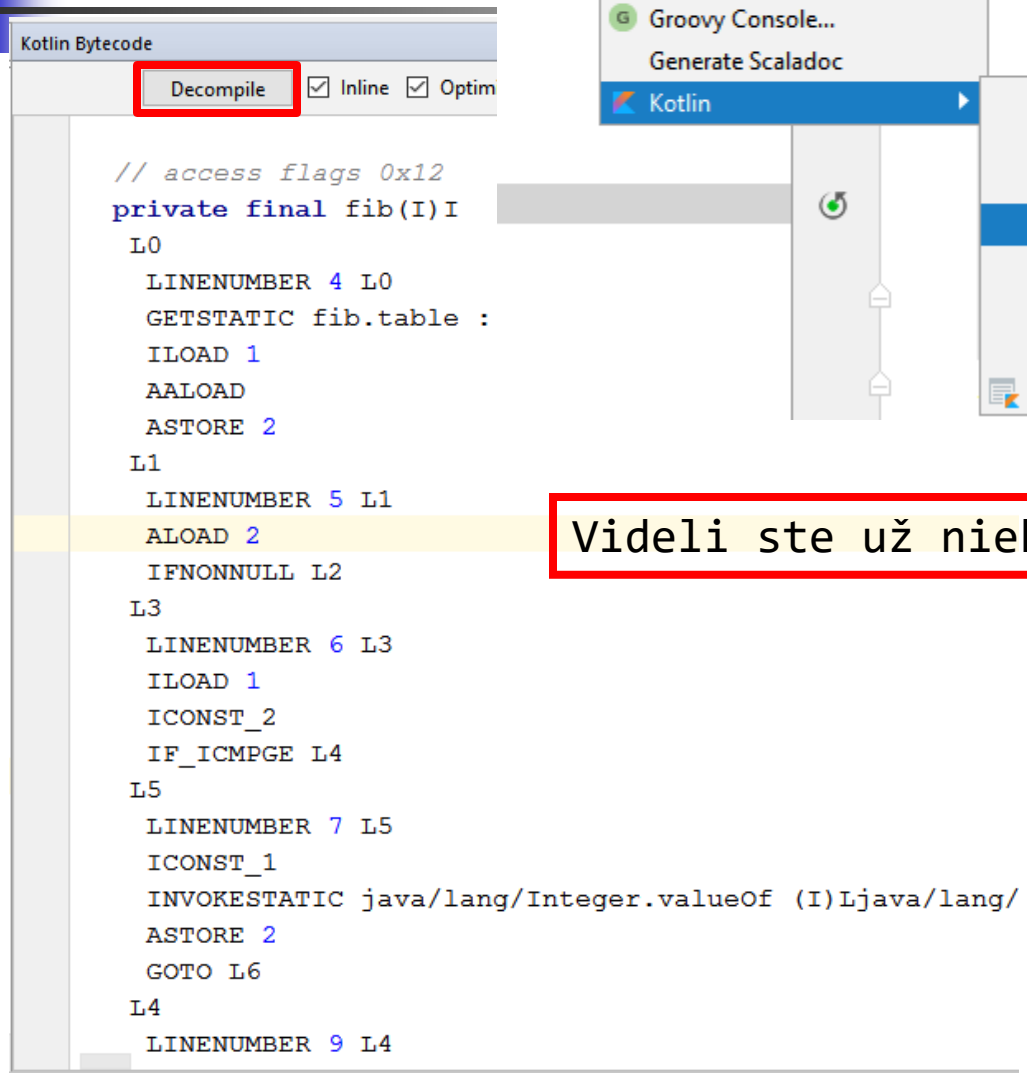
```
object fib {
    internal var table = arrayOfNulls<Int>(100)
    private fun fib(n: Int): Int {
        var result: Int? = table[n]
        if (result == null) {
            if (n < 2)
                result = 1
            else
                result = fib(n - 2) + fib(n - 1)
            table[n] = result
        }
        return result
    }
    @JvmStatic fun main(args: Array<String>) {
        for (i in 0..19)
            println("fib(" + i + ")=" + fib(i))
    }
}
```

Už nenájdete pôvodný zdroják

DÚ podobne vygenerované sa neuznajú

ByteCode





Videli ste už niekedy kompilát vášho kódu

From Java to Kotlin
(if-when)

- **if** je výraz

```
fun binCifSum(n : Int) : Int =
   if (n <= 0) 0
   else binCifSum(n/2) + if (n % 2 == 0) 0 else 1


fun main(args:Array<String>):Unit {
   for (n in 0..10)
       println("binCifSum $n je ${binCifSum(n)}")
}


val kategoria =
       if (vek < 6) "predskolsky"
       else if (vek <= 11) "1.stupen"
       else if (vek <= 18) "2.stupen"
       else "mimo"
```

0.kt

From Java to Kotlin
(when)

- **when** je výraz

```
val kategoria1 =
        when (vek) {
            in 0..5 -> "predskolsky"
            in 5..11 -> "1.stupen"
            in 12..18 -> "2.stupen"
            else -> "mimo"
        }
```

```
for (x in 1..10) println(x)                    // 1, 2, …, 10
for (x in (1..10).toList()) println(x)         // 1, 2, …, 10
for (x in (10 downTo 1).toList()) println(x) // 10, 9, …, 1
for (x in 1 until 10) println(x)               // 1, 2, …, 9
for (x in 'a'..'z') println(x)                 // a, b, …, z
```

0.kt

From Java to Kotlin

- Operátory porovnania, podobne ako Java `<=, <, >=, >, !=`

ale

`==` je porovnanie hodnôt

`===` je porovnanie referencii

```
val a = "kot"
val b = "lin"
val c = (a+b).trim()
val d = "kotlin"
println("c==d ${c==d}, c===d ${c===d}")
```

c==d true, c===d false

0.kt

From Java to Kotlin

- Operátory porovnania, podobne ako Java `<=, <, >=, >, !=`

ale

`==` je porovnanie hodnôt

`===` je porovnanie referencii

```
val a = "kot"
val b = "lin"
val c = (a+b).trim()
val d = "kotlin"
println("c==d ${c==d}, c===d ${c===d}")
```

c==d true, c===d false

0.kt

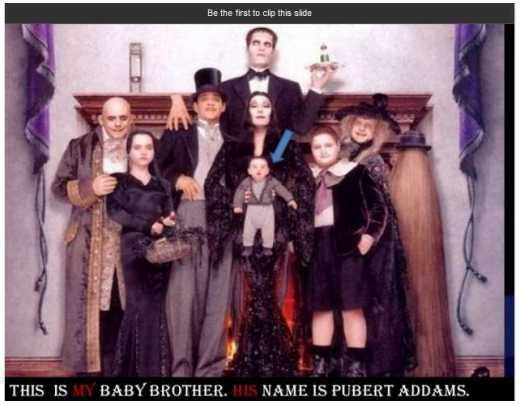
Decompile

(vidíte len časť main – ale je to .java)


```
@JvmStatic
    public static final void main(@NotNull String[] args) {
        Intrinsics.checkParameterIsNotNull(args, "args");
        int i = 0;
        byte var2 = 19;
        if (i <= var2) {
            while(true) {
                String var3 = "fib(" + i + ")=" + INSTANCE.fib(i);
                System.out.println(var3);
                if (i == var2) {
                    break;
                }
                ++i;
            }
        }
    }
```

6.kt

Addams Kotlin family




```
data class Person(val first : String, val name: String,
                  val age: Int? = null,
                  val father : Person?, val mother : Person?)
```

Data class je class s predgenerovanými equals, hashCode, toString, copy

```
fun main(args: Array<String>) {
        val father = Person("Gomez", "Addams", 156, null, null)
        val mother = Person("Morticia", "Addams", 136, null, null)
        val daugther  = Person("Wednesday", "Addams", 46, father, mother)
        val son  = Person("Pugsley", "Addams", 36, father, mother)
        val family = listOf( father, mother, daugther, son,
              Person("Fester", "Addams", 174, null, null), // uncle
              Person("Pubert", "Addams", null, null, null) // on the picture
            )
        val oldest = family.maxBy { it.age ?: 0 }
        println("The oldest is: $oldest")
    }
```

1.kt

Číselné funkcie, String template

```
fun fib(n: Int): Int {
        return if (n < 2) 1 else fib(n-1) + fib(n-2)
}
fun fib1(n: Int): Int {
    fun fib(n: Int, a : Int = 0, b : Int = 1): Int {
        return if (n < 0) a else fib(n-1, b, a+b)
    }
    return fib(n)
}
fun main(args: Array<String>) {
    val lst = listOf(1,2,3,4,5,6,7,8,9,10)
    println(lst.map { n -> fib(n) })
    println(lst.map { fib1(it) })
    lst.forEach { println("fib($it) = ${fib1(it)}")}
    for(i in 1..11) println("fib($i) = ${fib1(i)}" )
    println("Maximum:${lst.map { fib(it) }.max()}")
}
```

2.kt

Properties

```
data class Rectangle(val height: Int, val width: Int) {
    val isSquare: Boolean
            get() {
                return height == width
            }
    fun size():Int {
        return height * width
    }
}

fun main(args: Array<String>) {
    val rect = Rectangle(41, 43)
    println("Toto $rect je stvorec: ${rect.isSquare}")
    println("Obsah $rect je: ${rect.size()}")
}
```

3.kt

Enumerables, when

(sú aj v Jave 5+)

```
enum class Color { RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET }
```

```
enum class Colour(val r: Int, val g: Int, val b: Int) {
    WHITE(0, 0, 0), RED(255, 0, 0),  YELLOW(255, 255, 0),
    GREEN(0, 255, 0), BLUE(0, 0, 255), BLACK(255, 255, 255);
    fun rgb() = (r * 256 + g) * 256 + b
}
```

```
fun getMnemonic(c : Colour) = {
    when (c) {
        Colour.WHITE -> "Biela"
        Colour.BLACK -> "Biela"
        else         -> "Seda"
    }
}
```

4.kt

Enumerables, when

(when alias switch)

```
fun mix(c1: Color, c2: Color) =
    when (setOf(c1, c2)) {
        setOf(Color.RED, Color.YELLOW) -> Color.ORANGE
        setOf(Color.YELLOW, Color.BLUE) -> Color.GREEN
        setOf(Color.BLUE, Color.VIOLET) -> Color.INDIGO
        else -> throw Exception("Dirty color")
    }

fun mixOptimized(c1: Color, c2: Color) =
    when {
        (c1 == Color.RED && c2 == Color.YELLOW)  -> Color.ORANGE
        (c1 == Color.YELLOW && c2 == Color.BLUE) -> Color.GREEN
        (c1 == Color.BLUE && c2 == Color.VIOLET) -> Color.INDIGO
        else -> throw Exception("Dirty color")
    }
```

4.kt

Derivácia

```
interface Expr                                      // abstract class
enum class Operator { Plus, Times }
data class Num(val value: Int) : Expr               // subclass, extends
data class Variable(val variable: String) : Expr
data class Op(val operator: Operator, val left:Expr, val right:Expr):Expr

fun derive(e: Expr, variable : String): Expr {
    if (e is Num) { return Num(0) }                 // typeof
    else if (e is Variable) {                       // typeof
        return if (e.variable == variable) Num(1) else Num(0) // typecast
    } else if (e is Op) {
        when(e.operator) {     // vie, že e:Expr je Op, ((Op)e).operator
            Plus -> return Op(Plus,
                        derive(e.left, variable), derive(e.right, variable))
            Times -> return Op(Plus,
                       Op(Times, derive(e.left, variable), e.right),
                       Op(Times, derive(e.right, variable), e.left))
        }
    }
    throw IllegalArgumentException("Unknown expression")
}
```

5.kt

Zjednodušovanie

```
fun simplify(e: Expr):Expr {
    when (e) {
      is Op -> {
          when (e.operator) {
            Operator.Plus -> {
              return
              if (e.left is Num && e.right is Num)
                 Num(e.left.value + e.right.value)
              else if (e.left == Num(0)) simplify(e.right)
              else if (e.right == Num(0)) simplify(e.left)
              else e.copy(left = simplify(e.left), right = simplify(e.right))
            }

          else -> return e
      }
      return e
}
```

5.kt

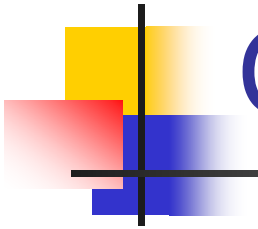
Metódy

```
sealed class Expression {
    data class Num(val value: Int) : Expression()
    data class Variable(val variable: String) : Expression()
    data class Op(val operator: Operator,
         val left: Expression, val right: Expression) : Expression()

    fun derive(variable : String): Expression {
        if (this is Num) {      // typeof
            return Num(0)
        } else if (this is Variable) {
            return if (this.variable == variable) Num(1) else Num(0)

    fun simplify(): Expression {
        when (this) {
            is Op -> {
```

9.kt



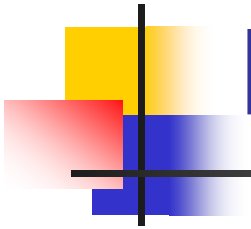
Cykly

```
fun main(args: Array<String>) {
    for(i in 0..10) println(i)
    for(i in 1 until 10) println(i)
    for(i in 10 downTo 0 step 3) println(i)
    for(i in 0..10) println(i)
    for (c in 'A'..'F') println(Integer.toBinaryString(c.toInt()))

    for (c in ' '..'z')
        if (c in 'a'..'z' || c in 'A'..'Z')
            print(c)

    for (c in ' '..'z')
        when (c) {
            in '0'..'9' -> println("digit")
            in 'a'..'z', in 'A'..'Z' -> println("letter")
        }
}
```

7.kt



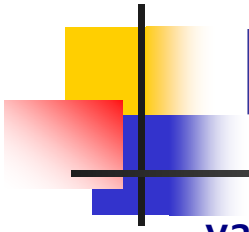
Kolekcie

```
val set = hashSetOf(2, 3, 5, 7, 11, 13, 17)
val list = arrayListOf(-1, 0, 1)
val map = hashMapOf("sedma" to 7, "osma" to 8, "dolnik" to 11,
                    "hornik" to 12, "kral" to 13, "eso" to 15)

println(set)
println(set.javaClass)
println(list)
println(list.javaClass)
println(map)
println(map.javaClass)

for(x in list)
   for(y in set)
      for((key, value) in map)
          println("$x $y $key $value")
```

8.kt



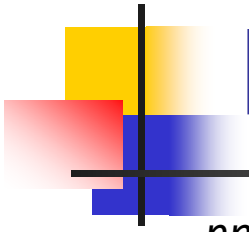
Funkcie

```
val fcia = { x:Int, y : Int -> println("sucet $x+$y"); x+y}
val proc = { x:Int, y : Int -> println("sucet $x+$y")}

println(fcia(12,7))
proc(13,9)
println({ x:Int -> x+1 }(2))
; // inak neopochopí, že nejde o blok, ale lambda konštantu
{ x:Int -> println(x)}(4)
       // preto jasnejší zápis
run {{ x:Int -> println(x)}(4)}

val delta = 5
println(listOf(1,2,3)
                .map { it + delta}   // x -> x + delta, clojure
                .filter {it % 2 == 0} )
```

10.kt



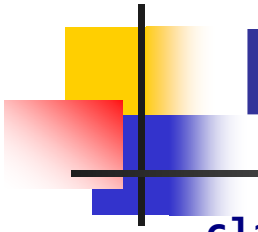
Funkcie

```
println(family.map { it.first })  // mapToObj
println(family.filter { it.age?:0 > 100 } )
println(family.all { it.age?:0 < 100 } )
println(family.all { it.name == "Dracula" } )
println(family.groupBy { it.father } )
println(family.filter {
    it.age == family.maxBy { person: Person -> person.age?:0 }?:0 } )
Ak by .age bol Int, nie Int?
    it.age == family.maxBy { person: Person -> person.age }?:0 } )

val numbers = mapOf(0 to "zero", 1 to "one")
for((father, persons) in family.groupBy { it.father })
    println("${persons.size} ma otca $father")

println(listOf("a", "aba", "b", "ba", "abba").groupBy { it.length })
println(listOf("a", "aba", "b", "ba", "abba").flatMap { it.toList() })
```

10.kt



Funkcie

```
class Book(val title: String, val authors: List<String>)
val books = listOf(
        Book("Action in Kotlin", listOf("Dmitry Jemerov", "Svetlana Isakova")),
        Book("Mort", listOf("Terry Pratchett")),
        Book("Good Omens", listOf("Terry Pratchett", "Neil Gaiman")),
        Book("Discworld", listOf("Terry Pratchett", "Paul Kidby")))
println(books.flatMap { it.authors }.toSet())

listOf(1, 2, 3, 4)
          .asSequence()
              .map { print("map($it) "); it * it }
              .filter { print("filter($it) "); it % 2 == 0 }
          .toList()

val nats = generateSequence(1) { it + 1 }
println(nats.takeWhile { it <= 100 }.sum())
println(nats.takeWhile { it <= 10 }.reduce({ x:Int, y : Int -> x*y}))
```

10.kt

Nullables

elvis je aj javascripte

V Jave je String skutočný retazec alebo null

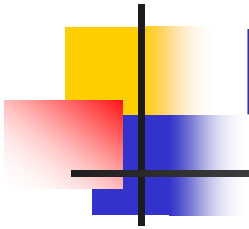
V Kotline String je LEN skutočný reťazec a null nepatrí do typu String

Existuje String? čo je String alebo null, vo všobecnosti: T? = T ∪ null

T? Podobne vo Swingu, Java Optional[T] =, Scala Option[T]

```
fun foo(str : String?) {
   println(str)
   if (str != null) println(str.toUpperCase())
   println(str?.toUpperCase())     // safe call operátor
                        // x?.m == if (x != null) x.m else null
}
fun stringLen(s: String?): Int = s?.length?:0 // Elvis operátor
if (if (s == null) then null else s.length) == null then 0 else s.length
fun nonEmptystringLen(s: String?): Int {
   val sNotNull: String = s!!         // určite nebude null,
              // ak bude tak exception kotlin.KotlinNullPointerException
   return sNotNull.length
}
```

11.kt



Nullables

- Safe call

  ```
  o ?. m() = if (o == null) null else o.m()
  ```

- Elvis operátor

  ```
  o ?: default = if (o == null) default else o
  ```

- Safe cast

  ```
  o as? T = if (o typeof T) o else null
  ```

- Not-null assertion

  ```
  o!! = if (o != null) o else N.P.E.
  ```

- let

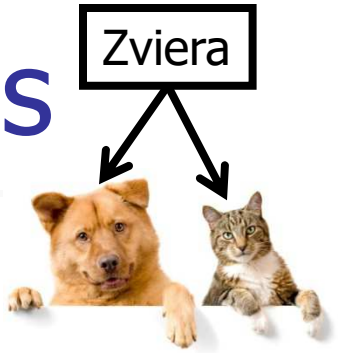
  ```
  o?.let {…it…} = if (o != null) {…it <- o…}
  ```

Immutables

Collection	Immutable	Mutable
List	listOf() *listOf*<String>("a", "b") .get(0)	arrayListOf() *arrayListOf*<String>("a", "b") .set(1, "Kotlin")
Set	setOf() *setOf*<String>("a", "b", "a") .contains("a")	hashSetOf() linkedSetOf() sortedSetOf() *hashSetOf*<String>("a", "b", "a") .remove("a")
Map	mapOf() *mapOf*("a" *to* 1, "b" *to* 100) **.keys**	hashMapOf() linkedMapOf() sertedMapOf() *hashMapOf*("a" *to* 1, "b" *to* 100) *.set*("b", 10)

11.kt

Podtriedy a polymorfizmus




```
open class Zviera {                 // open znamená nie final
    open fun pozdrav() { }          // open znamená nie final
}
class Macka : Zviera() {            // Macka je podtrieda Zviera
    override fun pozdrav() { println("mnau") }
}
class Pes : Zviera() {              // Pes je ine Zviera
    override fun pozdrav() { println("haf") }
}

class Stado<T : Zviera>() {                 // stádo implementujeme ako
    var lst: MutableList<T> = mutableListOf() // mutable list je zámer
    val size: Int get() = lst.size
    operator fun get(i: Int): T {           // T je v out pozícii
        return lst.get(i); }                // operátor dovolí stado[i]
    operator fun set(i: Int, v: T) {        // T je v in pozícii
        lst.set(i, v)                       // operátor dovolí stado[i] = v
    }
}
```

13.kt

Podtriedy a polymorfizmus

(variancie – covariancia a contravariancia – teória)

Macka je podtrieda Zviera, Macka <: Zviera
Pes je podtrieda Zviera, Pes <: Zviera

Stado<T : Zviera> je parametrický typ pre ľubovoľný podtyp T typu Zviera

Stado<Macka> ani Stado<Pes> ale **nie je** podtrieda Stado<Zviera>

Stado je na parameter T invariantné

Ak ale chceme, aby Stado<Macka>, Stado<Pes> BOLI podtrieda Stado<Zviera>,
horoví sa tomu covariancia, potom stado musí byť deklarované takto:

```
class Stado<out T : Zviera>() { ... }    // Stado[Macka] <: Stado[Zviera]
```

Ak chceme, aby XYZ<Zviera> BOLA podtrieda XYZ<Macka>,
horoví sa tomu contravariancia, potom stado musí byť deklarované takto:

```
class XYZ<in T : Zviera>() { ... }       // XYZ[Zviera] <: XYZ[Macka]
```

Podtriedy a polymorfizmus

(stado je invariantné)


```
fun pozdravitVsetky(zvery : Stado<Zviera>) {
    for (i in 0 until zvery.size)
        zvery[i].pozdrav()
}

fun pozdravitMacky(macky : Stado<Macka>) {
    for (i in 0 until macky.size)
        macky[i].pozdrav()          // macky[i] : Macka, preto .pozdrav()

    pozdravitVsetky(macky)          // toto nejde lebo macky : Stado<Macka>
                                    // to nie je podtyp Stado<Zviera>

    pozdravitVsetky(macky as Stado<Zviera>) // smart Cast
                // povie kompilátoru, že ver mi, macky : Stado<Zviera>
                // kompilátor uverí a zavolá funkciu

    pozdravitVsetky(macky)          // toto uz ide, lebo kompilátor uveril
                                    // že macky : Stado<Zviera>
}
```

13.kt

Podtriedy a polymorfizmus

(stado je invariantné a zneužijeme toho)


```
val stado = Stado<Macka>()          // main
stado.append(Macka())
stado.append(Macka())
stado[1] = Macka()                  // ilustrácia operátora set
val m = stado[0]                    // ilustrácia operátora get
pozdravitMacky(stado)

stado[1] = Pes()                    // nejde, lebo Macka nie je podtrieda Pes
stado.append(Pes())                 // nejde, lebo Macka nie je podtrieda Pes
pozdravitVsetky(stado)  // Stado<Macka> nie je podtrieda Stado<zviera>
                                    // tzv. Smart cast
pozdravitVsetky(stado as Stado<Zviera>)  // ale presvedčíme kompilátor
stado[1] = Pes()                    // a už nám verí
stado.append(Pes())                 // oklamali sme ho ☺ ☺ ☺ ☺ ☺

pozdravitVsetky(stado)              // stado as Stado<Zviera> to on už vie !
pozdravitMacky(stado)               // toto on kompilátor vie, ale keďže
                          // sme ho oklamali, vypomstí sa nám v runtime
Exception "main" java.lang.ClassCastException:Pes cannot be cast to Macka
Hádanka: na ktorom riadku to padlo ?
```

Covariancia

(prvý pokus - stado snáď bude covariantné)

Ak ale chceme, aby Stado<Macka>, Stado<Pes> BOLI podtriedy Stado<Zviera>, tak to nejde takto:

```
class Stado<out T : Zviera>() {          // v scale Stado[+T]
    var lst: MutableList<T> = mutableListOf()
                // T je deklarované ako out je v invariant pozícií
    val size: Int get() = lst.size
    operator fun get(i: Int): T { return lst.get(i); }
                // T je deklarované ako out je v out pozícií, ok ☺
    operator fun set(i: Int, v: T) { lst.set(i, v) }
                // T je deklarované ako out je v in pozícií
    fun append(v: T) { lst.add(v) }
                // T je deklarované ako out je v int pozícií
}
```

Scala: covariant argument in contravariant position ...

Veľmi zjednodušene:

out je výstupný argument, **in** je vstupný argument metódy

Viac: https://kotlinlang.org/docs/reference/generics.html

Covariancia

(druhý pokus - stado už bude covariantné za cenu ...)

Ak ale chceme, aby Stado<Macka>, Stado<Pes> BOLI podtriedy Stado<Zviera>:

- nesmie mať žiadnu metódu so vstupným argumentom :T, lebo ten je out
- ako štruktúru naplniť, modifikovať ? jedine v konštruktore
- ergo, je to nemodifikovateľná [**immutable**] štruktúra/trieda/typ

```
class Stado<out T : Zviera>(val lst : List<T>) {
        val size: Int get() = lst.size
        operator fun get(i: Int): T { return lst.get(i); }
                    // T je deklarované ako out je v out pozícií, ok ☺
        //operator fun set(i: Int, v: T) { lst.set(i, v) }
        //fun append(v: T) { lst.add(v) }
        // var lst2: MutableList<T> = mutableListOf()
    }
fun pozdravitMacky(macky : Stado<Macka>) {
    pozdravitVsetky(macky)          // toto ide lebo macky:Stado<Macka>,
}                                   // to je podtyp Stado<Zviera>
val stado = Stado<Macka>(listOf(Macka(), Macka()))
pozdravitVsetky(stado)
```

14.kt

Contravariancia

()


```
abstract class Zviera(val size : Int = 0) { }
data class Macka(val krasa : Int) : Zviera(1) { }
data class Pes(val dravost : Int) : Zviera(2) { }
// alias comparable
interface Compare<in T> {
    fun compare(z1: T, z2: T): Int
}
val MackaCompare : Compare<Macka> = object: Compare<Macka> {
    override fun compare(m1: Macka, m2: Macka): Int {
        println("macky$m1 a $m2 si porovnavaju  ${m1.krasa} a ${m2.krasa}")
        return m1.krasa - m2.krasa
    }
}
// val ZvieraCompare: Compare<Zviera> = MackaCompare // pre contravar…
val ZvieraCompare: Compare<Zviera> = object: Compare<Zviera> {
    override fun compare(z1: Zviera, z2: Zviera): Int {
        println("zviera $z1 a $z2 si porovnavaju ${z1.size} a ${z2.size}")
        return z1.size - z2.size
} }
```

Contravariancia (in):
Macka <: Zviera =>
Compare[Zviera] <: Compare[Macka]

podtyp **nadtyp**

15.kt

Zhrnutie

(covariancia, contravariancia, invariancia)

Covariant	Contravariant	Invariant
`Producer<out T>`	`Consumer<in T>`	MutableList<T>
$T_1 <: T_2 \Rightarrow G[T_1] <: G[T_2]$ Príklad: Producer<Macka> je podtyp Producer<Zviera> Skutočný príklad: interface List<out E>: Collection<E>	$T_1 <: T_2 \Rightarrow G[T_2] <: G[T_1]$ Príklad: Consumer<Zviera> je podtyp Consumer<Macka> Skutočný príklad: Interface Comparable<in E>	$T_1 <: T_2 \Rightarrow$ $G[T_1]$ a $G[T_2]$ nemajú ŽIADEN vzťah
T môže byť len v **out** pozícií, napr. výsledok fcie	T môže byť len v **in** pozícií, napr. vstup do fcie	T môže byť v ľubovoľnej pozícií

https://kotlinlang.org/docs/reference/generics.html

No fajn, hneď je to jasnejšie ☺

Kotlin: má pre co/contra-varianciu out/in

Scala: +/-

A ako to bolo v Jave ?

- Stado<? extends Zviera>
- Compare<? Super Macka>