Tvorba mobilních aplikací

2. seminář

Radek Janoštík

Univerzita Palackého v Olomouci

24. 2. 2024

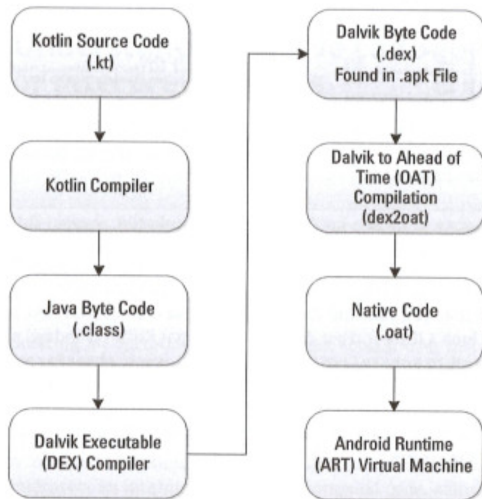
Nativní vývoj v Androidu

- Většina částí Android Software Development Kit (SDK) je napsána v Javě
- Dlouhou dobu byla Java jediným jazykem pro nativní vývoj
- Nekompilujeme ale do Java Bytekodu, ale do Dalvik Executable (DEX)
  - Dalvik = bývalé běhové prostředí
  - Art (Android Runtime) = současně používané běhové prostředí
- Při instalaci aplikace na zařízení je DEX kód optimalizován pro dané zařízení
- $\Rightarrow$ je možné psát i v jiném jazyce $\rightarrow$ Kotlin

Kotlin – úvod

- Moderní programovací jazyk vyvíjený společností JetBrains
  - `https://kotlinlang.org/`
  - Verze 1.0 z roku 2016
  - Nyní 1.8.10 (únor 2023)
- Úzce spjat s Javou
  - Interně jsou třídy namapovány na třídy z Java Class Library
  - Možnost kompilace a běhu v JVM (klasický JAR)
  - Syntakticky odlišný ale sémanticky velmi podobný (podobnost i se Swift)
  - Spousta moderních prvků, které by se do Javy tak rychle nedostaly
  - Multiparadigmový – OOP, funkcionální, krátké skripty, asynchronní programování
  - Interoperabilita
- Důraz na čitelnost, nepsaní zbytečné omáčky(středník)
- U Android vývojářů se natolik ujal, že jej Google uznal jako oficiální jazyk pro Android
  - Aplikace Netflix, Twitter, Pinterest, . . .
- Transpilace

Překlad Kotlinu do výsledné mobilní aplikace



Kotlin v kostce

- Proč nový jazyk?
- Probereme velmi stručný úvod do jazyka Kotlin bez návaznost na mobilní aplikace
- Nelze se naučit nový jazyk za jeden seminář $\rightarrow$ spíš jen ukázka
- Zbytek bude na vás $\rightarrow$ během vývoje komunikovat s dokumentací
- Není to jen o syntaxi, ale i drobná změna v uvažování

Typy

- Kotlin je staticky typovaný

Typy

- Kotlin je staticky typovaný (= každá proměnná má pevně daný typ), kontrola v čase kompilace
- Máme „chytré odvození typů“ – kompilátor z kontextu typ odvodí

```
val greet = "Ahoj svete!"
println(greet)
println(greet::class)
println(greet.javaClass)
```

- Implicitní specifikace typu:

```
val pi: Double = 3.1415
```

- „Všechno je objekt“ = můžeme „přes tečku“ volat funkce a přistupovat k vlastnostem
- Základní typy – speciální interní reprezentace
  https://kotlinlang.org/docs/basic-types.html

Val vs. var (1 / 2)

- Proměnné deklarované pomocí `val` jsou *immutable*

```
val pi = 3.1415
pi = 3.14159 // Chyba: val cannot be reassigned
```

- Proměnné deklarované pomocí `var` jsou *mutable*

```
var itemsRemaining = 31
itemsRemaining--
println(itemsRemaining)
```

- Preference používat `val` všude, kde je to možné (JIdea škaredě podtrhává)
  - Mutovatelnost může přinášet nechtěné chyby
  - Méně problému při synchronizaci paralelních aplikací
- Pozor na chování `val` u referenčních typů
  - Je neměnná reference, ale objekt v ní se měnit může

Val vs. var (2 / 2)

- Co vypíše kód a proč?

```
var factor = 2
fun doubleIt(n: Int) = n * factor
factor = 0
println(dopubleIt(2))
```

Val vs. var (2 / 2)

- Co vypíše kód a proč?

```
var factor = 2
fun doubleIt(n: Int) = n * factor
factor = 0
println(dopubleIt(2))
```

- Reference stejná, objekt může změnit stav

```
val message = StringBuilder("Ahoj ")
message.append("svete!")
println(message.toString)
```

Funkce

- Deklarace klíčovým slovem `fun` s názvem, argumenty a tělem
- Návratovou hodnotu opět nemusíme specifikovat, nemáme `void`, vždy něco vrací (`Unit`)
- Funkce může vracet pouze *výraz*

```
fun createHello(name: String) = "Hello $name"
fun sayHello(name: String) = println(createHello(name))
sayHello("Carl")
```

- Funkce s tělem:

```
fun max(nums: IntArray) : Int {
    var curr = Int.MIN_VALUE;
    for (num in nums) {
        if (num>curr) curr=num
    }
    return curr
}
```

Výchozí hodnota argumentu, pojmenovaní, pořadí

- Výchozí argument

```
fun createHello2(name: String, greet: String = "Hello")
                                   = "$greet $name"
println (createHello2("Carl"))
println (createHeloo2("Carl", "Ciao"))
```

- Možno zadávat argumenty pojmenováním

```
println (createHello2(greet = "Zdar", name="Karle"))
```

Řízení toku programu

- `If` jako *statement* (*příkaz*) stejný jako v jiných jazycích
  - Ve větvích se provedou nějaké akce
- Pozor – v Kotlinu může být `If` i *expression* (*výraz*)
  - Z větví se vrátí nějaká hodnota/výraz
- Můžeme například použít:

```kotlin
val caption = if (itemsRemaining>3) "je toho dost" else "nejak to udelame"

sayHello(if (Random.nextInt(0, 100)>10) "Carl" else "Charlie")
```

- Jde tohle v Javě? C#? Kde ano?

Řízení toku programu – when

- `when` – obdoba klasického switch s pokročilým *pattern matchingem*

```
fun printWhatToDo(dayOfWeek: Any) {
    when (dayOfWeek) {
        "Saturday", "Sunday" -> println("Relax")
         in  listOf ("Monday", "Tuesday", "Wednesday", "Thursday")
                                  -> println("Work Hard")
         in 1..4  -> println("Work Hard with numbers")
         "Friday", 5 -> println("Party")
         is String -> println("I don't know this day")
         else -> println("Are you from Mars?")
    }
}
```

- I `when` může být jako *statement* i *expression*
- Nemusí mít argument, „najde“ proměnné ve svém rozsahu

Kolekce

- Mutovatelné kolekce z Javy mají dvě rozhraní – mutovatelné a nemutovatelné

```
val names = mutableListOf("Thomas", "Gordon", "James", "Emily")
val imNames = listOf("Percy", "Henry", "Edward", "Toby")
println (names.javaClass)
println (imNames.javaClass)
names.add("Hiro")
imNames.add("Charlie") // chyba
```

- Kolekce za Javy: `Array, List, Set, Map`
- Navíc `Pair` a `Triple`

```
val maxSpeed = Pair("Thomas", 30)
val bigOnes = Triple("Gordon", "James", "Hiro")
println (maxSpeed.first)
println (bigOnes.third)
```

Cykly

- „Starý for“ typu `for (int i =0; i<30; i++)` v Kotlinu nemáme
- Vždy by se mělo iterovat „přes něco“
- Rozsahy – Abstrakce pro iteraci přes nějaký rozsah

```
val smallIntRange = 1..10
val firstLetters = 'a'..'g'
val allLetters = 'A'..'z'

for (l in allLetters) println(l)
for ((i, l) in allLetters.withIndex()) println("$l -- $i")
for (i in 5..10) println(i)
for (i in 10 downTo 3) println(i) // infixova notace
for (i in 5 until 10) println(i) // infix "krome posledniho"
for (i in 1 until 30 step 3) println(i)// kazdy treti
```

- `while` a `do...while` je stejný jako v Javě

Lambda výrazy

- = anonymní funkce, které můžeme brát jako hodnoty
  - argumenty metod, návratové hodnoty, volání
- Definice lambda výrazu ve tvaru, povinné je pouze `telo`

```
val jmeno : Typ = { seznamArgumentu -> telo }
```

- Funguje odvození typů

```
val tenTimes = { i : Int -> i * 10}
val areaOfTriangle = { a : Int, va : Int -> a * va / 2 }
println (tenTimes(30))
println (areaOfTriangle(10, 5))
```

- Velmi důležitý koncept (práce s kolekcemi, paralelní programování, callbacky, . . .)

Práce s kolekcemi – pole

- Kolekce prvků fixní délky, používat spíše `List` (optimalizace)
- Na pozadí generická třída `Array<T>`
- Vytvoření top-level funkcí:

```
val machines = arrayOf("Thomas", "Gordon", "James", "Emily")
val numbers = arrayOf(6, 7, 8)
val betterNumbers = intArrayOf(6,7,8) // java int vs. Integer
```

- Vytvoření konstruktorem – velikost + lambda pro naplnění

```
val anotherArray = Array(5, { i -> i*10 } )
val yetAnotherArray = Array(10) { it * 10 } // co se to deje?
```

- Přístup klasicky přes operátor `[]`

```
println (yetAnotherArray[5])
```

Práce s kolekcemi – seznam

- Proměnlivá délka – volba mutovatelnosti
- Vytvoření stejně jako pole

```
val iList : MutableList<Int> = MutableList(5) { it }
println ( iList [4])
println ( iList .get(4)) ;
iList .add(6)
println ( iList .contains(8))
```

- U nemutovatelných vytváření kopií při „přidání prvku“

```
val imNames2 = imNames − "Edward"
val imNames3 = imNames2 + "Rose"
```

- Množinu nechám na vás

Práce s kolekcemi – slovník (Map)

- Kolekce pro uchování **párů** typu `klíč→hodnota`
- Každý objekt má rozšiřující metodu `to`, která vytvoří pár

```
val p = "Radek" to "C#" // class kotlin.Pair
```

- Vytvoření opět podobné

```
val map = mapOf("Radek" to "C#", "Petr" to "Java", "Tomas" to
    "Haskell")
println (map.size) // 3
```

- Dotaz na přítomnost prvků

```
println (map.containsKey("Tomas"))
println (map.containsValue("C#"))
```

- Iterace

```
for (e in map) println("${e.key} -> ${e.value}")
for ((key, value) in map) println("$key -> $value")
```

Úkol (1 / 2)

V Jazycích Kotlin a Swift naprogramujte funkce

1. **fun** geometricSum(start: **Double**, ratio: **Double**, n: **Int**) : **Double**
   - Která vypočítá součet prvních n prvků geometrické posloupnosti začínající `start` s kvocientem `ratio`
2. **fun** square(n: **Int**)
   - Pro sudá `n` vykreslí do konzole pomocí ⋆ čtverec dané velikosti
   - Pro lichá `n` vykreslí do konzole pomocí ⋆ křížek dané velikosti
   - Například n=4 a n=5

```
****
*  *
*  *
****
```

```
  *
  *
*****
  *
  *
```

Úkol (2 / 2)

3. **fun** myPow(base: **Double**, n: **Int**)
   - Bez použití matematických funkcí (násobení povoleno)
   - Pomocí rekurze vypočítá `n` mocninu čísla `base`
4. **fun** onlyChosen(firstArray: IntArray, secondArray: IntArray, decider : (**Int**, **Int**) -> **Int**) : IntArray
   - První dva parametry jsou stejně dlouhá číselná pole
   - Posledním parametrem je lambda výraz, který má ze dvou čísel nějaké vybrat
   - Funkce vrátí pole vybraných prvků
5. Vytvořte pole obsahující všechny násobky 7 menší než 7000
   - Z tohoto pole sestupně vypište ty, které jsou dělitelné 13 a 17