Kotlin alapú szoftverfejlesztés - 1. előadás

Bevezetés

- JDK 8 ajánlott
- Kotlin → modern nyelvi elemeket ezen keresztül fogjuk megtanulni
- sokféle helyen használható
  - fut JVM felett → dektop, android
  - webfejlesztésre (js-re is fordul), react alkalmazás is írható
  - ios-en és androidon futó kód is írható

Jellemzők

1. **tömör nyelv**
   - nem kell felesleges dolgokat írni
   - átláthatóbb
2. **biztonságos nyelv**
   - null pointer exception-t nehéz benne csinálni
3. **interoperable**
   - bármelyik platformon futtatjuk, könnyen működik együtt az ottani natív kóddal
4. **Tool-friendly**
   - jó fejlesztőkörnyezetekben lehet használni

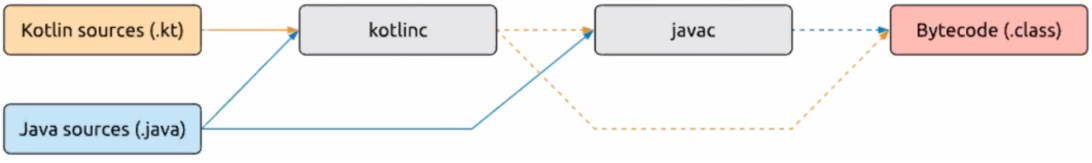
Történelme

- 2010 → JetLang néven elkezdi fejleszteni az IntelliJ
- 2016 → első stabil verzió
- azóta évente jön ki új verzió
- Kotlin szigetről van elnevezve (sziget Szentpéterváron)

Build tool-ok

- pl.: Gradle, Maven
- projektépítő eszköz

Fordítás

- forráskód → fordító → bytecode
- Kotlin Sources (.kt) → kotlinc → .class fájlok
- ezeket adjuk a JVM-nek
- 

Ismerkedés

- buildgradle fájl
  - verzió
  - milyen java-val van használva
  - dependencies rész (függőségek megadása)
    - kotlin standard library
    - junit (teszteléshez)

Hello Word program

```
fun main() {
    println("Hello word")
}
```

- `ctrl` + `shift` + `a` (Action kereső) → skb → show kotlin byte code
- *decompile* gomb → java kódot csinál a kotlin-ból
  - fontos: ez a kód nem mindig helyes!
- Java-ból kotlinba
  - Code → Convert Java from Kotlin
- alkalmazás belépő pontja → main függvény

Scratch fájlok → scriptelésre

- Action keresőből → Scratch
  - kis külön álló fájlok, modult kiválasztva tudjuk használni függvényeket, példányosítani osztályokat

REPL

- parancssor jellegű, kotlin kód írható, soronként fordítódik

Kotlin szintaktika

Változók - `var` vs `val`

```
var x: Int = 0
//var név: Típus = érték
//nincs pontosvessző
```

- `val` → mintha `final` lenne, nem változtatható az értéke
- `var` → változtatható az értéke
- FONTOS: ami csak lehet, az `val` legyen!
- nem kell pontos vessző
- ```
  val z = 2
  //fix értékű változó (mivel val), kitalálja magától, hogy ez Int
  ```
- `ctrl` + `q` → dokumentáció
- `ctrl` + `p` → type info
  ```

- ```
  //java verzió:
  //final FileInputStream fis = new FileInputStream("")
  val fis = FileInputStream("")
  //nem kell kiírni a new-t
  ```
- Változó típusának megadása postfix-ekkel
- ```
  val myLong = 1L
  val myFloat = 1f
  val myDouble = 1.0
  ```

Mi is az az Int

- választ a fordító, hogy `int` (primitív) vagy `Integer` (osztály) kell e

String-ek

- ```
  val name = "Sarah"
  val x = 3, y = 5, z = 7
  val name = "$x + $y = ${x + y}" //változók értékének belefűzésbe string-be
  ```
- e mögött java-s `string` összefűzés van
- `Alt` + `Enter` → minden fontos funkció

Függvények

- ```
  fun addSima(x: Int, y: Int): Int {
      return x + y
  }
  fun addRovid(x: Int, y: Int) = x + y
  ```
- `void` visszatérési érték
- ```
  fun noReturnValue(): Unit { //nem kötelező kiírni a Unit-ot
      println("Semmi vagyok")
  }
  //tudunk ilyet is csinálni:
  val result: Unit = noReturnValue()
  println(result)
  ```
- ennek előnye, hogy nincs megkülönböztetés a `void`-os függvények közt
  - így együtt kezelhetően
  ```

- ```
  //több féle paraméterlistával is szeretném
  fun register(
      username: String,
      password: String = "123456",
      email: String = ""
  ) {
      println("Registered $username, $password, $email")
  }
  register("alma", "alma123", "alma@alma.hu")
  register("korte", "korte123") //2 paraméterrel is meghívható
  register("narancs")
  register("tigtis", email = "tigris@tigris.hu")
  ```

Irányítási struktúrák

- Sima `if`
- ```
  val age = 15
  val drinkingAge = 18

  if(age < drinkingAge) {
      println("Nope")
  } else {
      println("Have a beer")
  }
  ```
- értékadás `if`-fel (?: operátor helyett)
- ```
  kedvezmeny = if (age < drinkingAge) {
      val diff = drinkingAge - age
      100 - diff * 5
  } else {
      0
  }

  val max = if (a > b) a else b // cserébe ?: operátor nincs!
  ```
- `switch` alternatíva → `when`
- ```
  val greade: Int = 2
  val description: String = when (greade) {
      1 -> { "Elégtelen" }
      2 -> { "Adequate" }
      3 -> { "Adequate" }
      4 -> { "Adequate" }
      5 -> { "Adequate" }
      else -> "Érvénytelen"
  }
  ```

- ```
  when (val rating = calculateRating()) {
      0 -> println("Katasztóvális") //ha 1 dolgot akarunk, nem kell {}
      1, 2, 3 -> println("Rossz") //tudunk több értéket megadni
      in 4..6 -> println("Átlagos") //teljes tartomány (4, 5, 6)
      in 7 until 10 -> println("Jó") //jobbról nyitott tartomány (7, 8, 9)
      10 -> println("Kiváló")
  }
  ```

- ```
  val x = 10
  val y = 2
  when { //if-else-if-else ág, az első igaz lesz kiértékelve!
      x <40 -> println("x is too small")
      y in 0..50 -> println("í is in invalid range")
  }
  ```

Kivételek

- mint java-ban → `try - catch - (finally)` blokk
- értékadás is lehet vele
- ```
  val input: String = readUserInput()
  val value: Double = try {
      input.toDouble()
  } catch (e: NumberFormatException) {
      0.0
  }
  ```

- semmit nem kötelező elkapni kotlin-ban
  ```