Przedmiot specjalnościowy I Aplikacje mobilne dla systemu Android

Dawid Bitner PAM 1A
Sprawozdanie

Porównanie języka Kotlin do innych języków, różnice

1) Brak średników – mały szczegół, ale bardzo widoczny który odróżnia ten język od większości innych popularnych języków, za wyjątkiem np. Pythona.

```
fun main() {
    println("Hello, world!!!")
}
```

2) Definiowanie zmiennych – var i val. Var – variable umożliwia nam zmienianie wartości tej zmiennej w dalszej części programu, val – value natomiast nam na to nie pozwala. Jest to odpowiednikiem final z Javy.

```
fun main(){
    val finalInt = 66
    println("finalInt. Int type? ${finalInt is Int}. Value: $finalInt")
    // finalInt = 67 nie zadziała

    var notFinalInt = 70
    notFinalInt = 71
    println("notFinalInt. Int type? ${notFinalInt is Int}. Value: $notFinalInt")
}
```

3) Domyślnie nie możemy przypisać wartości null do zmiennej, żeby było to możliwe musimy użyć znaku „?”, rozwiązanie jest podobne do tego z C#.

```
fun main(){
    val finalInt = 66
    println("finalInt. Int type? ${finalInt is Int}. Value: $finalInt")
    // finalInt = 67 nie zadziała

    var notFinalInt = 70
    notFinalInt = 71
    println("notFinalInt. Int type? ${notFinalInt is Int}. Value: $notFinalInt")
    notFinalInt = null // nie zadziała
}
```

```
Null can not be a value of a non-null type Int
```

4) W Kotlinie nie potrzebujemy jak np. w Javie klasy do tego by wywołać funkcję, dodatkowo da się to robić w bardzo prosty sposób, poprzez np. przypisanie wartości po znaku „=” jak ma

to miejsce np. podczas definiowania zmiennych

```
fun beLikePi() = 3.14

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

5) Optionale
„?.” – operator bezpiecznego wywołania,
„!!” – sprawdza czy obiekt jest null’em i rzucające NullPointerException,
„?:” – pozwala zdecydować co zrobić, gdy obiekt jest null’e

```
fun checkNull1(data: Any?): String {
  return data?.toString() ?: "" //Odpowiednik wersji z Javy
}
fun checkNull2(data: Any?) {
  data?.toString() // Nigdy się nie wykona jeżeli data jest nullem
  data!!.toString() // rzuci NPE gdy data jest nullem
}
```

6) Występowanie when , które zastąpiło switch z rodziny języków C, wydaje się być bardzo wygodne w użyciu i czytelniejsze od switcha.

```
when (x) {
    0, 1 -> print("x == 0 or x == 1")
    else -> print("otherwise")
}
```

7) W Kotlinie zreturnować wynik ifa możemy np. w następujący sposób:

```
val max = if (a > b) {
    print("Choose a")
    a
} else {
    print("Choose b")
    b
}
```

8) Klasy - w Kotlinie klasy tworzy się i operuje się na nich w dużo bardziej przejrzysty sposób niż w klasycznej Javie. Dzięki słowu kluczowemu “data” uzyskujemy automatycznie

wygenerowane metody takie jak equals(), czy toString().

```
data class Person(
    val firstName: String,
    val lastName: String,
    val age: Int,
    val female: Boolean
)

fun main(args: Array<String>) {
    val person = Person("Dawid", "Bitner", 22, false)
    val personDouble = Person("Barack", "Obama", 52, false)
    println("Person: $person")
    println("Person double: $personDouble")

    println("Comparision by equals: ${person == personDouble}")
    println("Comparision by reference: ${person === personDouble}")

    val femalePerson = person.copy(female = true, firstName = "Barbara")
    println("Female person: $femalePerson")

    println("Comparison by equals: ${femalePerson == personDouble}")
    println("Comparison by reference: ${femalePerson === personDouble}")
}
```

9) =, ==, === - w powyższym przykładzie widzimy też porównanie przez metodę equals, za którą odpowiada “==”, natomiast potrójny znak równości porównuje przez refrencję, czyli zachowuje się tak jak “==” w Javie.

10) Singleton - tworzenie go w Kotlinie jest bardzo proste. By go stworzyć, to przy definicji klasy zamiast słówka class, wystarczy użyć object, dzięki temu mamy tylko jedną instancję klasy.