Kotlini ülevaade

Kotlin on staatiliselt tüübitud keel (st muutujate tüüp on üheselt määratletud), mis kompileerub Java baitkoodiks ja kasutab Java virtuaalmasinat (JVM). Kotlin on ekspressiivne (st kergesti arusaadav) ja täpne programmeerimiskeel, mis võimaldab kirjutada lühemat koodi. Kotlini tugevus on ka täielik ühilduvus Javas kirjutatud koodiga.

Kotlini arendab JetBrains ja esimene versiooni avaldati 2016. aastal. Kotlinist on saanud üks olulisemaid keeli Androidi rakenduste loomisel ning alates 2019. aastast on see Google'i eelistatud Androidi arenduskeel.

Muutujad, deklareerimine ja kasutamine

Muutujate deklareerimine

Muutujaid saab deklareerida kahte moodi, val ja var abil. val abil deklareeritakse muutujat, mille väärtus ei muutu kunagi. var-i kasutatakse muutuja puhul, mille väärtus võib muutuda.

Muutujate tüübid

Int - tähistab täisarvu, üks paljudest numbritüüpidest, mida saab Kotlinis kasutada, saab salvestada täisarve vahemikus -2147483648 kuni 2147483647
Byte - salvestab täisarve vahemikus -128 kuni 127. Seda saab kasutada Int-i või muude täisarvutüüpide asemel, et säästa mälu
Short - salvestab täisarve vahemikus -32768 kuni 32767
Long - saab salvestada täisarve vahemikus -9223372036854775807 kuni 9223372036854775807. Seda kasutatakse juhul, kui Int ei ole väärtuse salvestamiseks piisavalt suur
Float - kasutatakse ujukomaarvude salvestamiseks, kümnendkohti 6-7
Double - kasutatakse ujukomaarvude salvestamiseks, kümnendkohti 15-16

Boolean - saab võtta ainult väärtuse true või false
String - kasutatakse märgijada (teksti) salvestamiseks. Stringi väärtused peavad olema ümbritsetud jutumärkidega
Char - kasutatakse ühe märgi salvestamiseks. Tähemärgi väärtus peab olema ümbritsetud üksikute jutumärkidega
Array - massiive kasutatakse mitme väärtuse salvestamiseks ühte muutujasse, selle asemel, et deklareerida iga väärtuse jaoks eraldi muutujaid

Funktsioonid

Funktsioonid on koodiplokid, mida saab konkreetse ülesande täitmiseks kutsuda. Kotlinis defineeritakse funktsioonid märksõnaga "fun", millele järgneb funktsiooni nimi, valikulised parameetrid ja tagastustüüp.

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Kotlini funktsioonidel võivad olla parameetri vaikeväärtused, mis tähendab, et kui parameetri jaoks väärtust ei edastata, kasutab see selle asemel vaikeväärtust. Näiteks:

```
fun greeting(name: String = "world") {
    println("Hello, $name!")
}
```

Lisades return parameetrile “?” on võimalik ka tagastada NULL-i

Tingimuslaused ja korduslaused

If tingimuslause

Kotlini If tingimuslause sarnaneb teistes keeltes kasutatavatele if tingimusalusele. If lauset saab koostada ühe rea siseselt, ilma loogeliste sulgudeta, kujul:

```
if (a < b) max = b
```

Kasutada võib ka else ja if else väiteid:

```
if (a > b) {
    max = a
} else {
    max = b
}

val maxOrLimit = if (maxLimit > a) maxLimit else if (a > b) a else b
```

If lause abil on võimalik ka väärtuseid omistada:

```
max = if (a > b) a else b
```

When tingimuslause

When tingimus sarnaneb C keele Switch tingimusele, kus erinevate tingimuste korral väljastatakse erinevad vastused:

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> {
        print("x is neither 1 nor 2")
    }
}
```

Iga haru käiakse läbi, kuniks leitakse õige tingimus. Kui sobivat tingimust ei leita, kasutatakse else tingimust.

Kui ühel vastel on mitu tingimust, tuleb need eraldada komaga:

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

In ja !In tingimustega saab kontrollida, kas väärtus kuulub või ei kuulu mingisse hulka:

```
when (x) {
    in 1..10 -> print("x is in the range")
    in validNumbers -> print("x is valid")
    !in 10..20 -> print("x is outside the range")
    else -> print("none of the above")
}
```

When tingimust võib edukalt kasutada If ja If-else tingimuse asendusena.

For korduslause

For ahel on Kotlini keeles sarnane näiteks C# keele foreach ahelaga. See tähendab, et süntaks käib läbi kõik itereeritavad sisendid:

```
for (item in collection) print(item)
```

Mingis väärtuste vahemikus itereerimiseks kasutatakse vahemiku tingimusi:

```
for (i in 1..3) {
        println(i)
}
for (i in 6 downTo 0 step 2) {
        println(i)
}
```

While korduslause

While ja do-while ahel käitavad süntaksit seniks, kuni on saavutatud eelmääratud tingimus.

While kontrollib tingimust ja käivitab seejärel süntaksi.
Do-while käivitab süntaksi ja kontrollib seejärel tingimusi.

```
while (x > 0) {
        x–
}
do {
        val y = retrieveData()
} while (y != null) // y is visible here!
```

Kotlin kasutab ahelatest väljumiseks ja jätkamiseks traditsioonilis break ja continue väiteid.

break: lõpetab käesoleva ahela töö.
continue: liigub edasi ahela järgmise sammuni.

Exceptions

Kotlini erandid pärandavad klassi Throwable. Igal erandil on tingimus ja sõnum.
Erand väljastatakse järgnevalt:

```
throw Exception("Hi There!")
```

Erandolukorra tabamiseks kasutatakse try ja catch väljendeid:

```
try {
        // some code
} catch (e: SomeException) {
        // handler
} finally {
        // optional finally block
}
```

Catch ja finally ei ole kohustuslikud, aga vähemalt üks neist peab olema esindatud.

Klassid

Inheritance/pärinemine

Kotilinis on kõigil klassidel sama superklass “Any”, mis rakendub isegi siis, kui supertüüp pole defineeritud. “Any”’l on 3 meetodi: equals(), hashCode(), toString(). Vaikimisi on kõik klassid *final*, ehk ei saa pärandada, kui selleks on soovi, siis tuleb klass märkida *open*, nt:

```
class Example // inherits from Any, isn’t open for inheritance

open class Example // Class is open for inheritance
```

Selge supertüübi defineerimiseks peab klassi päisesse kirjutama tüübi peale koolonit:

```
open class Base(p: Int)

class Derived(p: Int) : Base(p)
```

Kui tuletatud klassil on primaarne konstruktor, siis baasklass saab (ja peab) olla initsialiseeritud primaarses konstruktoris vastavalt parameetritele. Kui tuletatud klassil ei ole primaarset konstruktorit, siis iga teisejärguline konstruktor peab initsialiseerima baastüüpi kasutades *super* võtmesõna või see peab delegeerima järgmisele konstruktorile mis seda kasutab:

```
class MyView : View {
    constructor(ctx: Context) : super(ctx)

    constructor(ctx: Context, attrs: AttributeSet) : super(ctx, attrs)
}
```

Meetodite ülekirjutamiseks on *override* võtmesõna.
Atribuutide ülekirjutamiseks kasutatakse ka *override* võtmesõna. Saab *val* atribuuti üle kirjutada *var* atribuudiga, aga mitte vastupidi.

Superklassi meetodile ligi *super* võtmesõnaga, nt kui *inner class* peab ligi pääsema vanemklassi meetodile - sellisel juhul peab ka viitama välise klassi nimele: *super@Outer* (Outer on klassi nimi, milles meetod asub).

Mitmelt klassilt pärimise korral peab kasutama *super<Base>* võtmesõna, et näidata millisest vanemklassist meetod võetakse.

Properties/omadused

Atribuutide deklareerimine käib samamoodi nagu eelnevalt mainitud: *var* on muudetav muutuja, *val* on ainult loetav muutuja, lisades “?” andmetüübi kõrvale saab väärtus olla nullitav.
Klassi atribuudi kasutamiseks tuleb sellele lihtsalt viidata, siinkohal tasub tähele panna, et Kotlinis pole uue objekti loomisel *new* võtmesõna nagu Javas.

Interface/liides

Liidesed saavad sisaldada abstraktseid meetodeid ja ka meetodite implementeerimist. Abstraktsetest klassidest eristab see, et liidesed ei saa endas sisaldada olekut.
Klass või objekt saab pärida ühelt või mitmelt liideselt.
Liides saab pärida ka vanemliideselt.

Visibility modifiers/nähtavuse võtmesõnad

Kotlinis on 4 nähtavuse modifitseerijat: *private, protected, internal, public.*

Top-level:

- Kui modifitseerijat ei kasutata, siis vaikeväärtuseks on *public***.**
- *private* - ainult ligipääsetav failis
- *internal* - sama mooduli sees ligipääsetav
- *protected* - pole saadaval *top-level* deklaratsioonidele

Klassid:

- *private* - ainult klassi sees ligipääsetav
- *protected* - sama nagu *private*, aga lisaks ka nähtav alamklasside jaoks
- *internal* - mooduli sees deklareeritud klassi puhul kättesaadav
- *public* - ükskõik kus deklareeritud klassi puhul kättesaadav

Extensions

Kotlinis on võimalik pikendada klassi või liidest uute funktsionaalsustega ilma seda pärimata. Näiteks saab seda kasutada mingi kolmanda osapoole teegi klassile või liidesele uue funktsionaalsuse kirjutamiseks.

Data class

Andmeklassid on mõeldud vaid andmetega seotud operatsioonidega tegemiseks. Kotlin pakub sisseehitatud võimalust neid luua.

```
data class User(val name: String = "", val age: Int = 99)
```

Selline koodirida tekitab klassi User, millel on automaatselt olemas järgnevad funktsioonid:
equals() - kahe Useri võrdlemiseks omaduste haaval
hashCode() - tagastab arvulise väärtuse Useri sisust, kaks sama sisuga Userit peaksid tagastama sama väärtuse, ehk saab ka seda võrdlemiseks kasutada
toString() - Useri väärtused stringi kujul
componentN() - muutujate destruktureerimiseks ehk

```
val person = User("admin", 123)
val (name, age) = person
```

koodirida kompileerub järgnevaks:

```
val name = person.component1()
val age = person.component2()
```

copy() - ühe kasutaja andmete teisele kopeerimine

Enum class

Enum klasside eesmärk on pakkuda arendajale mõistetavaid konstantseid väärtuseid, kus arvuti loeb neid (vaikimisi) kui täisarve.

```
enum class Direction {
    NORTH,
    EAST
    SOUTH,
    WEST
}
val dir = Direction.NORTH
```

Tegelik **dir** väärtus masina arust on näiteks 0. On võimalik luua ka enum klass, millel on küll need konstandid, aga saab ka neist erinevaid väärtuseid kasutada.

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

Aga on võimalik ka:

```
var color = Color(0xABCDEF)
```

Anonüümseid klasse ja abstraktseid funktsioone kasutades saab ka midagi järgnevat luua, mis on harjumuspärase enum klassiga võrreldes päris erinev.

```
enum class ProtocolState {
    WAITING {
        override fun signal() = TALKING
    },
    TALKING {
        override fun signal() = WAITING
    };

    abstract fun signal(): ProtocolState
}
var state = ProtocolState.TALKING
state = state.signal()
```

Ehk peale **state.signal()** kutsumist on **state** väärtus nüüd **ProtocolState.WAITING**.

Object expressions

Vahest on vaja klassi funktsionaalsusi, kuid ei soovi selleks eraldi klassi tekitada. Siis on võimalik kasutada objekte, millega on võimalik luua anonüümseid klasse.

```
val helloWorld = object {
    val hello = "Hello"
    val world = "World"

    override fun toString() = "$hello $world"
}
```

Anonüümne klass saab ka pärineda mingist olemasolevast klassist.

```
val anonymousFromUser = object: User(“username”, 123) {
    override fun toString(): String {
        /* ... */
    }
}
```

Delegation

Delegatsiooni kasutades on võimalik edasi arendada erinevaid klasse, mis kõik täidavad mingi liidese/*interface* nõudeid.

```
interface Software {
        fun getLicense(): String
}

class OpenSource(): Software {
        override fun getLicense(): String {
        return "Open Source Software"
        }
}

class ClosedSource(): Software {
        override fun getLicense(): String {
        return "Closed Source Software"
        }
}
```

Kui nüüd tahta edasist funktsionaalsust **Software** liidest täitvale klassile anda, saab seda teha delegatsiooniga.

```
class PrintableSoftware(sw: Software): Software {
        override fun getLicense(): String {
                return sw.getLicense()
        }

        fun printLicense() {
                print(sw.getLicense())
        }
}
```

Täpselt sama funktsionaalsuse saavutamiseks on aga Kotlinil kiirem viis: **by** võtmesõna.

```
class PrintableSoftware(sw: Software): Software by sw {
   fun printLicense() {
      print(getLicense())
   }
}
```

Tühiviida kaitse (*null safety*)

Kotlini tüübisüsteemi eesmärk on välistada tühiviitadega seotud ohud, mis nt Javas moodustavad suure osa kompileerimisel tekkivatest vigadest.
Vaikimisi ei saa muutujale *null*'i algväärtuseks panna (muutuja on vaikimisi mitte-*null*'itav). Selleks tuleb nt stringi tüüpi muutuja deklareerimimisel kirjutada andmetüübi järel küsimärk:

```
var b: String? = "abc"
```

Null'itavaid muutujaid on turvaliselt võimalik välja kutsuda kas *null*'i eelnevalt tingimuslausega kontrollides või turvalise väljakutse operaatori ?. abil:

```
println(b?.length)
```

Kui b väärtus on *null*, siis prindib “null”, vastasel juhul muutuja b pikkuse.

Operaatori !! abil on võimalik välja kutsuda tühiviida erand (*NullPointerException*).

```
val l = b!!.length
```

Kui b väärtus on *null*, siis viskab kompilaator tühiviida erandi, vastasel juhul omistab muutujale l muutuja b pikkuse.

Kaasrutiinid (*coroutines*)

Kotlinis on asünkroonne tegevus lahendatud kaasrutiinina, mille puhul on sisuliselt tegu lõimega. Seda on kirjeldatud ka funktsioonina, mille täitmine on võimalik pausile panna. Kaasrutiin luuakse *suspend* märksõna abil ja selle funktsionaalsus põhineb peamiselt teekidel.

```
fun postItem(item: Item) {
        launch {
        val token = preparePost()
        val post = submitPost(token, item)
        processPost(post)
        }
}

suspend fun preparePost(): Token {
        // makes a request and suspends the coroutine
        return suspendCoroutine { /* ... */ }
}
```