

INTRO TIL KOTLIN

Android teamet hos Forsvaret

Martin Bondkall Gjerde

Erik Stensrud Larsen

Jan Olav Kjøde



bouvet

Intro

- Utviklet av JetBrains
- Inspirert av Scala og Groovy
- Alternativ til Java, introdusert i 2011
- Populært på Android
- Foretrukket av Google siden 2019

Variabler

- Alt er et objekt
 - Int, Byte, Short, Long, Float, Double, Boolean og Char
- Primitiver finnes ikke
 - int, byte, short, long, float, double, boolean og char
- Null-safe
 - Som standard kan ikke et objekt være null
- ; er optional

Variabler

- `val`
 - Kan ikke endres når verdien er tilordnet.
Ligner på `final` i Java.
- `var`
 - Kan endres senere i programmet.
- Type er valgfritt, blir inferred av kompilatoren

```
var changeableName: String = "Variable"
val finalName: String = "Value"

changeableName = "Can be changed"
//finalName = "Immutable"    -> compiler error

val typeInferred = "Compiler recognizes the String"

println("Value" == finalName) // java: .equals()
println("Value" === finalName) // java: ==
```

Strings

- Inline støtte med \$ og \${}

```
val firstName = "Kalle"
val lastName = "Anka"
val fullName = "${lastName.toUpperCase()}, $firstName"
val multiLineString = """
    SELECT *
    FROM my_table
    WHERE id = :id
    -- can use " without escaping
    """.trimIndent()
```

Funksjoner og metoder

- Nøkkelord: fun
- Argumenttypen kommer etter argumentnavnet
- Returtype på slutten av signatur

```
fun squareWithBlock(n: Int): Int {  
    return n * n  
    //always 'return' in a block  
}  
  
fun squareOneLiner(n: Int) = n * n  
println("square(3) == ${squareOneLiner(n: 3)}") //> 9
```

Unit som returtype

- Unit er det samme som void i Java
- Unit kan bli inferred

```
fun printer(n: Int) {  
    println("The number is $n")  
}  
  
fun printerWithUnit(n: Int): Unit {  
    println("The number is $n")  
}
```

Argumenter

- Kan ha defaultverdi
- Argumenter kan navngis

```
fun printTemperature(degrees: Double, unit: String = "Celcius") {  
    println("temperature is $degrees $unit" )  
}
```

```
printTemperature( degrees: 37.0, unit: "Celcius")  
printTemperature( degrees: 37.0)
```

```
// > temperature is 37.0 Celcius  
// > temperature is 37.0 Celcius
```

```
fun confusing(name: String, isActive: Boolean, isAdmin: Boolean){}  
confusing( name: "Ole", isActive: true, isAdmin: true)  
confusing(name = "Ole", isAdmin = true, isActive = false)
```


Nullable typer

- Hver type har en komplementær nullable type
 - Eksempler: String vs String? og Int vs Int?
- Kompilatoren forhindrer at man tildeler null til et objekt som ikke kan være null

```
// var middleName: String = null //will not compile
var middleName: String? = null
if (middleName != null) {
    //No need to .get() as in Optional
    println("Middle name: $middleName")
}
```

Nullsafety og Elvis

- Håndtering av null
 - ?. -> verdien dersom ikke null
 - ?: -> verdien dersom uttrykket før var null (elvis)

```
val middleName: String? = null
val upperMiddleName: String? = middleName?.toUpperCase()
val defaultIfNull: String = middleName?.toUpperCase() ?: ""
```

Smart cast

- Any er super-klassen til alle klasser
- Trenger ikke cast etter at en type har blitt sjekket
- `is` = `instanceof`

```
val something: Any = getObject()
if (something is String) {
    println(something.toUpperCase())
}
```

When = switch++

- Smart casting
- Bruke when i et uttrykk
- Kan ha default (else)
- Trenger ikke argumenter

```
val surprise: Any = getSomething()
var whatIsIt: String? = null
when (surprise) {
    is String -> whatIsIt = surprise.toUpperCase()
    42         -> whatIsIt = "Life"
    3.14       -> whatIsIt = "PI"
}
```

```
val surprise: Any = getSomething()
val whatIsIt: String = when (surprise) {
    is String -> surprise.toUpperCase()
    42         -> "Life"
    3.14       -> "PI"
    else      -> "Whatever"
}
```

```
val char = 'c'
val result = when {
    char == 'A' || char == 'a' -> 1
    else -> -1
}
```

Klasser

- Ingen new ved opprettelse
- Gettere og Settere blir autogenerated
- «Nice to have»-features
 - Default verdier
 - Navngitte argumenter

```
class Person(val firstName: String, var lastName: String) {}  
  
// bruk av person i kotlin  
val erik = Person("Erik", "Larssen")  
println(erik.firstName)  
erik.lastName = "Larsen"
```

```
class Person(val lName: String, val fName: String, val mName: String? = null)  
  
val defaultMiddleName = Person( lName: "Kjøde", fName: "Jan" )  
val namedArguments = Person(fName = "Jan", mName = "Olav", lName = "Kjøde")
```

Arv

- Klassen som det arves fra må være open
- Bruker : istedenfor extends

```
open class Person(val name: String)
class IdentifiablePerson(val ssn: String, name: String) : Person(name)

val citizen = IdentifiablePerson(ssn: "0123456789", name: "Martin")
println("Name: ${citizen.name}, ssn: ${citizen.ssn}")

// > Name: Martin, ssn: 0123456789
```

Interface

- Bruker også :
(slik som arv)

```
open class Person(val name: String)

interface PersonService {
    fun addPerson(personToAdd: Person)
}

class PersonServiceImpl : PersonService {
    override fun addPerson(personToAdd: Person) {
        println("Persisting to database: $personToAdd")
    }
}
```

Dataklasser

- Immutable (hvis val)
- Autogenererte metoder:
 - copy()
 - toString()
 - equals()

```
data class Person(  
    val firstName: String,  
    val middleName: String,  
    val age: Int,  
)  
  
val bob = Person( firstName: "Bob", middleName: "Kåre", age: 52)  
println(bob)  
  
// > Person(firstName=Bob, middleName=Kåre, age=52)  
  
val otherBob = bob.copy(age = 25)  
println(otherBob)  
  
// > Person(firstName=Bob, middleName=Kåre, age=25)
```


Collections

- Immutable eller mutable
- Elementer nås med [index] eller .get(index)
- Metoder for å kopiere til mutable/immutable
- + tilføyer et element (sist / til høyre)

```
val fruits = listOf("Apple", "Banana")
val apple = fruits[0]
val banana = fruits.get(1)

val moreFruits = fruits + "Orange"
val mutableFruits = moreFruits.toMutableList()
mutableFruits.add("Kiwi")

val mList = mutableListOf("Bobbine", "Erika", "Thomesine")
val immutableList = mList.toList()
// immutableList.add("Georgine") // Does not compile

val set = setOf("Bobbine", "Erika", "Thomesine")
val map = mapOf("B" to "Bobbine", "E" to "Erika")
```

Scope functions

- let
- run
- with
- apply
- also

```
val petra = Person(name: "Petra", age: 30, place: "Paris")

petra?.let { p ->
    println(p)
    p.moveTo(place: "London")
    p.incrementAge()
    println(p)
}
```

Extension functions

- Legge til metoder i allerede eksisterende klasser

```
fun String.echo(): String {  
    return "$this $this"  
}  
  
val someString = "Hello"  
  
println(someString.echo()) // > Hello Hello
```