YALOVA UNIVERSITY

HUAWEI

Android Programming with Huawei Mobile Services

Berk Ozyurt
Mehmet Yozgatli
Cengiz Toru

W2:
KOTLIN BASICS

- **VCS**
- **Kotlin Basics**
  1. var / val
  2. Numbers
  3. Booleans
  4. Strings
- **Collections**
- **Control Flow**

Version Control System

VCS

Version
Control
System

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.

VCS

Version
Control
System

VCS software keeps track of every modification to the code in a special kind of database.

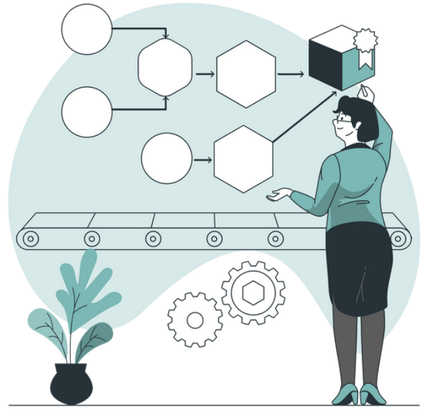
VCS protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

VCS helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.

Benefits of VCS

Fast

Change History



Branching and Merging

Traceability

What is Git?

Git is the most popular VCS in the world.

It works well on a wide range of operating systems and IDEs.

In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

What is Git?

Kotlin Basics

Basic Types - var / val

var

If the variable you created is a variable that can be changed later, "var" should be used.

```
var myString  :  String = "Huawei Mobile Services"
myString  =  "HMS"
```

```
var myString = "Huawei Mobile Services"
myString = "HMS
```

val

If the variable you created is a variable that will not change later, "val" should be used.

```
val myString : String = "Huawei Mobile Services"
myString= "HMS" //ERROR
```

```
val myString= "Huawei Mobile Services"
myString= "HMS" //ERROR
```

Basic Types - Numbers

Integer Types

Kotlin provides a set of built-in types that represent numbers.

For integer numbers, there are four types with different sizes and, hence, value ranges.

Type	Size (bits)	Min value	Max value
`Byte`	8	-128	127
`Short`	16	-32768	32767
`Int`	32	-2,147,483,648 (-2^{31})	2,147,483,647 ($2^{31} - 1$)
`Long`	64	-9,223,372,036,854,775,808 (-2^{63})	9,223,372,036,854,775,807 ($2^{63} - 1$)

```
val one = 1 // Int
val threeBillion = 3000000000 // Long
val oneLong = 1L // Long
val oneByte: Byte = 1
```

Basic Types - Numbers

Floating-Point Types

For real numbers, Kotlin provides floating-point types Float and Double.
According to the IEEE 754 standard, floating point types differ by their decimal place, that is, how many decimal digits they can store.

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
`Float`	32	24	8	6-7
`Double`	64	53	11	15-16

```
val pi = 3.14 // Double
// val one: Double = 1 // Error: type mismatch
val oneDouble = 1.0 // Double

val e = 2.7182818284 // Double
val eFloat = 2.7182818284f // Float, actual value is 2.7182817
```

Basic Types - Numbers

Types Changes

All number types support conversions to other types.

- **toByte():** Byte
- **toShort():** Short
- **toInt():** Int
- **toLong():** Long
- **toFloat():** Float
- **toDouble():** Double
- **toChar():** Char

Operations

Kotlin supports the standard set of arithmetical operations over numbers: **+, -, *, /, %.**
They are declared as members of appropriate classes.

Basic Types - Booleans

The type Boolean represents boolean objects that can have two values: true and false. Boolean has a nullable counterpart Boolean? that also has the null value.

- || – disjunction (logical OR)
- && – conjunction (logical AND)
- ! - negation (logical NOT)

```
val myTrue: Boolean = true
val myFalse: Boolean = false
val boolNull: Boolean? = null

println(myTrue || myFalse)
println(myTrue && myFalse)
println(!myTrue)
```

```
true
false
false
```

Basic Types - Strings

Strings in Kotlin are represented by the type String. Generally, a string value is a sequence of characters in double quotes (").

```
val str = "abcd 123"
```

String Concatenation

val string1 = "Huawei "
val string2 = "Mobile"
val string3 = "Services"

val string4 = string1 + string2 +string3 // "$string1 $string2 $string3"

print(string4)

Basic Types - Strings

- get(int index)
- substring(int beginIndex)
- substring(int beginIndex, int endIndex)
- toUpperCase()
- toLowerCase()
- isEmpty()
- replace(char old, char new)

Collections

Collections - Arrays

Array is a data structure that stores a large number of data of the same type (String, Int, Double or any Class type we will create).
Array size is given only at creation. It cannot be changed later.

```
val intArray = Array<Int>( size: 10){0}
val stringArray = Array<String>( size: 10){""}
val doubleArray = Array<Double>( size: 10){1.0}
val charArray = Array<Char>( size: 10){'A'}
```

Collections - Arrays

Assigning Value to Array:

In Kotlin, we access the data in an array with index and set method and assign a value.

1.index

```
val myArray = Array<Int>( size: 4){0}
myArray[0] = 12
myArray[1] = 8
myArray[2] = 34
myArray[3] = 19
```

2.set

```
val myArray = Array<Int>( size: 4){0}
myArray.set(0,12)
myArray.set(1,8)
myArray.set(2,34)
myArray.set(3,19)
```

Collections - Arrays

Reading Value from Array:

Similar to assigning value to array, we can read values with index and get method.

1.index

```
println(myArray[0])
println(myArray[1])
println(myArray[2])
println(myArray[3])
```

2.get

```
println(myArray.get(0))
println(myArray.get(1))
println(myArray.get(2))
println(myArray.get(3))
```

myArray.size
myArray.indexOf()
last

Collections - Arrays

Declaring an Array with arrayOf():

arrayOf is another array creation method used in Kotlin.

- val names = arrayOf(“Aysel”, “Aydin”, “Barış”)
- val numbers = arrayOf(15, 25, 36)
- val mixArray=arrayOf<Any>(15, “Aysel”, true)
- val name = arrayOfNull<String>(4)

Collections - Arrays

- ByteArray
- CharArray
- ShortArray
- IntArray
- LongArray
- BooleanArray
- FloatArray
- DoubleArray

```
val intListe = IntArray(8)
val doubleListe = DoubleArray(5)
```

Collections - List

They are sequential Collections whose elements can be accessed with their indexes. An element can appear more than once in the list.

They are read-only structures that cannot be added, deleted or updated.

- var list = listOf<String>("Test1","Test2","Test3")
- val numbers = listOf("Halil", "Murat", "Sevban", "Ömer")

Collections - List

- list.size
- list.get()
- list.indexOf()
- list.add()
- list[i] = "value"
- list.remove("value")
- list.removeAt(1)

Control Flow

Control Flow - If - Else

In Kotlin, if is an expression: it returns a value. Therefore, there is no ternary operator (condition ? then : else) because ordinary if works fine in this role.

If you're using if as an expression, for example, for returning its value or assigning it to a variable, the else branch is mandatory.

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

// With else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}

// As expression
val max = if (a > b) a else b
```

Control Flow - When

when defines a conditional expression with multiple branches. It is similar to the switch statement in C-like languages. Its simple form looks like this.

when matches its argument against all branches sequentially until some branch condition is satisfied.

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

Control Flow - For

The for loop iterates through anything that provides an iterator. This is equivalent to the foreach loop in languages like C#. The syntax of for is the following:

for (item in collection) print(item)

for (item: Int in ints) {
    // ...
}

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

Control Flow - While

while and do-while loops execute their body continuously while their condition is satisfied. The difference between them is the condition checking time:

- while checks the condition and, if it's satisfied, executes the body and then returns to the condition check.
- do-while executes the body and then checks the condition. If it's satisfied, the loop repeats. So, the body of do-while executes at least once regardless of the condition.

```
while (x > 0) {
    x--
}

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

Control Flow - Returns and Jumps

Kotlin has three structural jump expressions:

- **return** by default returns from the nearest enclosing function or anonymous function.
- **break** terminates the nearest enclosing loop.
- **continue** proceeds to the next step of the nearest enclosing loop.

LET'S TALK
CONTACT INFORMATION

Berk Ozyurt
berk.ozyurt1@huawei.com

Mehmet Yozgatli
mehmet.yozgatli1@huawei.com

Cengiz Toru
cengiz.toru@huawei.com

Telegram Channel
will be created

Official Website
https://developer.huawei.com