2CEIT5EP5-Mobile Application Development

Kotlin Basics

Prepared By:

Prof. Nishi Patwa

Assistant Professor

Computer Engineering Department

U.V.Patel College of Engineering

Ganpat University

Contents

1.Kotlin main function

The main function is the entry point to the Kotlin program.

main_fun.kt

```
fun main() {

   println("main function is an entry point")
}
```

The program prints a simple message to the terminal.

1. Kotlin Data Types

Kotlin data type is a classification of data which tells the compiler how the programmer intends to use the data. For example, Kotlin data could be numeric, string, boolean etc.

Kotlin treats everything as an object which means that we can call member functions and properties on any variable.

Kotlin is a statically typed language, which means that the data type of every expression should be known at compile time.

Kotlin built in data type can be categorized as follows:

- Number
- Character
- String

- Boolean
- Array

2.1 Kotlin Number Data Types

Kotlin number data types are used to define variables which hold numeric values and they are divided into two groups: (a) Integer types store whole numbers, positive or negative (b) Floating point types represent numbers with a fractional part, containing one or more decimals.

Following table list down all the Kotlin number data types, keywords to define their variable types, size of the memory taken by the variables, and a value range which can be stored in those variables.

Data Type	Size (bits)	Data Range
Byte	8 bit	-128 to 127
Short	16 bit	-32768 to 32767
Int	32 bit	-2,147,483,648 to 2,147,483,647
Long	64 bit	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bit	1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64 bit	4.94065645841246544e-324 to 1.79769313486231570e+308

If we will try to store a value more than permitted value in a variable of particular data type, the Kotlin compiler will complain because an overflow would occur at runtime.

Following example shows how to define and access different Kotlin number data types:

```
fun main(args: Array<String>) {
   val a: Int = 10000
   val d: Double = 100.00
   val f: Float = 100.00f
   val l: Long = 1000000004
   val s: Short = 10
   val b: Byte = 1

   println("Int Value is " + a)
   println("Double  Value is " + d)
   println("Float Value is " + f)
   println("Long Value is " + l )
   println("Short Value is " + s)
   println("Byte Value is " + b)
}
```

When you run the above Kotlin program, it will generate the following output:

```
Int Value is 10000
Double Value is 100.0
Float Value is 100.0
Long Value is 1000000004
Short Value is 10
Byte Value is 1
```

2.2 Kotlin Character Data Type

Kotlin character data type is used to store a single character and they are represented by the type Char keyword. A Char value must be surrounded by single quotes, like 'A' or '1'.

Following example shows how to define and access a Kotlin Char data type:

```
fun main(args: Array<String>) {
   val letter: Char    // defining a Char variable
   letter = 'A'        // Assigning a value to it
   println("$letter")
}
```

When you run the above Kotlin program, it will generate the following output:

```
A
```

Kotlin supports a number of escape sequences of characters. When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in the following statement is a valid character and it is called a new line character

```
println('\n') //prints a newline character

println('\$') //prints a dollar $ character

println('\\') //prints a back slash \ character
```

The following escape sequences are supported in Kotlin: \t, \b, \n, \r, \', \", \\ and \$.

2.3 Kotlin String Data Type

The String data type is used to store a sequence of characters. String values must be surrounded by double quotes (" ") or triple quote (""" """).

We have two kinds of string available in Kotlin - one is called Escaped String and another is called Raw String.

- Escaped string is declared within double quote (" ") and may contain escape characters like '\n', '\t', '\b' etc.
- Raw string is declared within triple quote (""" """) and may contain multiple lines of text without any escape characters.

```
fun main(args: Array<String>) {
   val escapedString : String  = "I am escaped String!\n"
   var rawString :String  = """This is going to be a
   multi-line string and will
   not have any escape sequence""";

   print(escapedString)
   println(rawString)
}
```

When you run the above Kotlin program, it will generate the following output:

```
I am escaped String!
This is going to be a
multi-line string and will
not have any escape sequence
```

2.4 Kotlin Boolean Data Type

Boolean is very simple like other programming languages. We have only two values for Boolean data type - either true or false.

```
fun main(args: Array<String>) {
   val A: Boolean = true   // defining a variable with true value
   val B: Boolean = false   // defining a variable with false value

   println("Value of variable A "+ A )
   println("Value of variable B "+ B )
}
```

When you run the above Kotlin program, it will generate the following output:

```
Value of variable A true
Value of variable B false
```

Boolean has a nullable counterpart Boolean? that can store a null value as below:

```
val boolNull: Boolean? = null
```

2.5 Kotlin Array Data Type

Kotlin arrays are a collection of homogeneous data. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

We will study array in a separate chapter, for now let's look at one example to define an array of integers and then access its one of the elements.

```
fun main(args: Array<String>) {
   val numbers: IntArray = intArrayOf(1, 2, 3, 4, 5)
   println("Value at 3rd position : " + numbers[2])
}
```

When you run the above Kotlin program, it will generate the following output:

```
Value at 3rd position : 3
```

2. Kotlin Data Type Conversion

Type conversion is a process in which the value of one data type is converted into another type. Kotlin does not support direct conversion of one numeric data type to another, For example, it is not possible to convert an Int type to a Long type:

```
fun main(args: Array<String>) {
   val x: Int = 100
   val y: Long = x  // Not valid assignment

   println(y)
}
```

When you run the above Kotlin program, it will generate the following output:

```
main.kt:3:18: error: type mismatch: inferred type is Int but Long was expected
val y: Long = x // Not valid assignment
```

To convert a numeric data type to another type, Kotlin provides a set of functions:

- toByte()
- toShort()
- toInt()
- toLong()
- toFloat()
- toDouble()
- toChar()

Now let's rewrite above example once again and try to run it:

```
fun main(args: Array<String>) {
   val x: Int = 100
   val y: Long = x.toLong()

   println(y)
}
```

When you run the above Kotlin program, it will generate the following output:

```
100
```

3. Kotlin Basic Input/Output

In this article, you will learn to display output to the screen, and take input from the user in Kotlin.

4.1 Koltin Output

You can use println() and print() functions to send output to the standard output (screen). Let's take an example:

```
fun main(args : Array<String>) {
    println("Kotlin is interesting.")
}
```

When you run the program, the output will be:

```
Kotlin is interesting.
```

Here, println() outputs the string (inside quotes).

Difference Between println() and print()

- print() - prints string inside the quotes.
- println() - prints string inside the quotes similar like print() function. Then the cursor moves to the beginning of the next line.

When you use println() function, it calls System.out.println() function internally. (System.out.println() is used to print output to the screen in Java).

If you are using IntelliJ IDEA, put your mouse cursor next to println and go to Navigate > Declaration (Shortcut: Ctrl + B . For Mac: Cmd + B), this will open Console.kt (declaration file). You can see that println() function is internally calling System.out.println().

Similarly, when you use print() function, it calls System.out.print() function.

Example 4.1.1: print() and println()

```
fun main(args : Array<String>) {
    println("1. println ");
    println("2. println ");

    print("1. print ");
    print("2. print");
}
```

When you run the program, the output will be:

```
1. println
2. println
1. print 2. print
```

Example 4.1.2: Print Variables and Literals

```
fun main(args : Array<String>) {
    val score = 12.3

    println("score")
    println("$score")
    println("score = $score")
    println("${score + score}")
    println(12.3)
}
```

When you run the program, the output will be:

```
score
12.3
score = 12.3
24.6
12.3
```

4.2 Kotlin Input

In this section, you will learn to take input from the user..

To read a line of string in Kotlin, you can use readline() function.

Example 4.2.1: Print String Entered By the User

```
fun main(args: Array<String>) {
    print("Enter text: ")

    val stringInput = readLine()!!
    println("You entered: $stringInput")
}
```

When you run the program, the output will be:

```
Enter text: Hmm, interesting!
You entered: Hmm, interesting!
```

It's possible to take input as a string using readLine() function, and convert it to values of other data type (like Int) explicitly.

If you want input of other data types, you can use Scanner object.

For that, you need to import Scanner class from Java standard library using:

```
import java.util.Scanner
```

Then, you need to create Scannerobject from this class.

```
val reader = Scanner(System.`in`)
```

Now, the reader object is used to take input from the user.

Example 4.2.2: Getting Integer Input from the User

```
import java.util.Scanner

fun main(args: Array<String>) {

    // Creates an instance which takes input from standard input (keyboard)
    val reader = Scanner(System.`in`)
    print("Enter a number: ")

    // nextInt() reads the next integer from the keyboard
    var integer:Int = reader.nextInt()

    println("You entered: $integer")
}
```

When you run the program, the output will be:

```
Enter a number: -12
You entered: -12
```

Here, reader object of Scanner class is created. Then, the nextInt() method is called which takes integer input from the user which is stored in variable integer.

4.Kotlin Operators

Kotlin has a set of operators to perform arithmetic, assignment, comparison operators and more. You will learn to use these operators in this article.

Operators are special symbols (characters) that carry out operations on operands (variables and values). For example, `+` is an operator that performs addition. In Java variables article, you learned to declare variables and assign values to variables. Now, you will learn to use operators perform various operations on them.

5.1 Arithmetic Operators

Here's a list of arithmetic operators in Kotlin:

Kotlin Arithmetic Operators

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Modulus Operator

Example: Arithmetic Operators

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

    val number1 = 12.5
    val number2 = 3.5
    var result: Double

    result = number1 + number2
    println("number1 + number2 = $result")

    result = number1 - number2
    println("number1 - number2 = $result")

    result = number1 * number2
    println("number1 * number2 = $result")

    result = number1 / number2
    println("number1 / number2 = $result")

    result = number1 % number2
    println("number1 % number2 = $result")
}
```

When you run the program, the output will be:

```
number1 + number2 = 16.0
number1 - number2 = 9.0
number1 * number2 = 43.75
number1 / number2 = 3.5714285714285716
number1 % number2 = 2.0
```

The `+` operator is also used for the concatenation of `String` values.

Example: Concatenation of Strings

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

    val start = "Talk is cheap. "
    val middle = "Show me the code. "
    val end = "- Linus Torvalds"

    val result = start + middle + end
    println(result)
}
```

When you run the program, the output will be:

```
Talk is cheap. Show me the code. - Linus Torvalds
```

How arithmetic operators actually work?

Suppose, you are using `+` arithmetic operator to add two numbers `a` and `b`.

Under the hood, the expression `a + b` calls `a.plus(b)` member function.

The `plus` operator is overloaded to work with `String` values and other basic data types (except Char and Boolean).

```
// + operator for basic types

operator fun plus(other: Byte): Int

operator fun plus(other: Short): Int

operator fun plus(other: Int): Int

operator fun plus(other: Long): Long

operator fun plus(other: Float): Float

operator fun plus(other: Double): Double


// for string concatenation

operator fun String?.plus(other: Any?): String
```

You can also use `+` operator to work with user-defined types (like objects) by overloading `plus()` function.
Here's a table of arithmetic operators and their corresponding functions:

Expression	Function name	Translates to
a + b	plus	a.plus(b)
a - b	minus	a.minus(b)
a * b	times	a.times(b)
a / b	div	a.div(b)
a % b	mod	a.mod(b)

5.2 Assignment Operators

Assignment operators are used to assign value to a variable. We have already used simple assignment operator `=` before.

```
val age = 5
```

Here, 5 is assigned to variable `age` using `=` operator.
Here's a list of all assignment operators and their corresponding functions:

Expression	Equivalent to	Translates to
a +=b	a = a + b	a.plusAssign(b)
a -= b	a = a - b	a.minusAssign(b)
a *= b	a = a * b	a.timesAssign(b)
a /= b	a = a / b	a.divAssign(b)

Expression	Equivalent to	Translates to
a %= b	a = a % b	a.modAssign(b)

Example: Assignment Operators

```
fun main(args: Array<String>) {
    var number = 12

    number *= 5   // number = number*5
    println("number  = $number")
}
```

When you run the program, the output will be:

```
number = 60
```

Recommended Reading: *Overloading assignment operators in Kotlin*.

5.3 Unary prefix and Increment / Decrement Operators

Here's a table of unary operators, their meaning, and corresponding functions:

Operator	Meaning	Expression	Translates to
+	Unary plus	+a	a.unaryPlus()
-	Unary minus (inverts sign)	-a	a.unaryMinus()
!	not (inverts value)	!a	a.not()

Operator	Meaning	Expression	Translates to
++	Increment: increases value by1	++a	a.inc()
--	Decrement: decreases value by 1	--a	a.dec()

Example: Unary Operators

```
fun main(args: Array<String>) {
    val a = 1
    val b = true
    var c = 1

    var result: Int
    var booleanResult: Boolean

    result = -a
    println("-a = $result")

    booleanResult = !b
    println("!b = $booleanResult")

    --c
    println("--c = $c")
}
```

When you run the program, the output will be:

```
-a = -1
!b = false
--c = 0
```

Recommended Reading: *Overloading Unary Operators*

5.4 Comparison and Equality Operators

Here's a table of equality and comparison operators, their meaning, and corresponding functions:

Operator	Meaning	Expression	Translates to
>	greater than	a > b	a.compareTo(b) > 0
<	less than	a < b	a.compareTo(b) < 0
>=	greater than or equals to	a >= b	a.compareTo(b) >= 0
<=	less than or equals to	a < = b	a.compareTo(b) <= 0
==	is equal to	a == b	a?.equals(b) ?: (b === null)
!=	not equal to	a != b	!(a?.equals(b) ?: (b === null))

Comparison and equality operators are used in control flow such as *if expression*, *when expression*, and *loops*.

Example: Comparison and Equality Operators

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

    val a = -12
    val b = 12

    // use of greater than operator
    val max = if (a > b) {
        println("a is larger than b.")
        a
    } else {
        println("b is larger than a.")
        b
    }

    println("max = $max")
}
```

When you run the program, the output will be:

```
b is larger than a.
max = 12
```

Recommended Reading: *Overloading of Comparison and Equality Operators in Kotlin*

5.5 Logical Operators

There are two logical operators in Kotlin: `||` and `&&`

Here's a table of logical operators, their meaning, and corresponding functions.

Operator	Description	Expression	Corresponding Function
||	`true` if either of the Boolean expression is `true`	`(a>b)||(a<c)`	`(a>b)or(a<c)`
&&	true if all Boolean expressions are `true`	`(a>b)&&(a<c)`	`(a>b)and(a<c)`

Note that, `or` and `and` are functions that support infix notation.

Logical operators are used in control flow such as *if expression*, *when expression*, and *loops*.

Example: Logical Operators

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

    val a = 10
    val b = 9
    val c = -1
    val result: Boolean

    // result is true is a is largest
    result = (a>b) && (a>c) // result = (a>b) and (a>c)
    println(result)
}
```

When you run the program, the output will be:

```
true
```

Recommended Reading: *Overloading of Logical Operators in Kotlin*

5.6. in Operator

The `in` operator is used to check whether an object belongs to a collection.

Operator	Expression	Translates to
in	a in b	b.contains(a)
!in	a !in b	!b.contains(a)

Example: in Operator

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

    val numbers = intArrayOf(1, 4, 42, -3)

    if (4 in numbers) {
        println("numbers array contains 4.")
    }
}
```

When you run the program, the output will be:

```
numbers array contains 4.
```

Recommended Reading: *Kotlin in Operator Overloading*

5.7. Index access Operator

Here are some expressions using index access operator with corresponding functions in Kotlin.

Expression	Translated to
`a[i]`	`a.get(i)`
`a[i, n]`	`a.get(i, n)`
`a[i1, i2, ..., in]`	`a.get(i1, i2, ..., in)`
`a[i] = b`	`a.set(i, b)`
`a[i, n] = b`	`a.set(i, n, b)`
`a[i1, i2, ..., in] = b`	`a.set(i1, i2, ..., in, b)`

Example: Index access Operator

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

    val a  = intArrayOf(1, 2, 3, 4, - 1)
    println(a[1])
    a[1]= 12
    println(a[1])
}
```

When you run the program, the output will be:

```
2
12
```

Recommended Reading: *Kotlin Index access operator Overloading*

5.8. Invoke Operator

Here are some expressions using invoke operator with corresponding functions in Kotlin.

Expression	Translated to
`a()`	`a.invoke()`
`a(i)`	`a.invoke(i)`
`a(i1, i2, ..., in)`	`a.inkove(i1, i2, ..., in)`
`a[i] = b`	`a.set(i, b)`

In Kotlin, parenthesis are translated to call `invoke` member function.

Recommended Reading: *Invoke Operator Overloading in Kotlin*

5.9 Bitwise Operation

Unlike Java, there are no bitwise and bitshift operators in Kotlin. To perform these task, various functions (supporting infix notation) are used:

- `shl` - Signed shift left
- `shr` - Signed shift right
- `ushr` - Unsigned shift right
- `and` - Bitwise and
- `or` - Bitwise or
- `xor` - Bitwise xor
- `inv` - Bitwise inversion

5. Flow Control in Kotlin

6.1 Kotlin if expression

Unlike Java (and other many programming languages), `if` can be used an expression in Kotlin; it returns a value.

Here is an example:

Example: Kotin if expression

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

    val number = -10

    val result = if (number > 0) {
        "Positive number"
    } else {
        "Negative number"
    }

    println(result)
}
```

When you run the program, the output will be:

```
Negative number
```

The `else` branch is mandatory when using `if` as an expression.

The curly braces are optional if the body of `if` has only one statement. For example,

```
fun main(args: Array<String>) {
    val number = -10
    val result = if (number > 0) "Positive number" else "Negative number"
    println(result)
}
```

Example: if...else...if Ladder

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

    val number = 0

    val result = if (number > 0)
        "positive number"
    else if (number < 0)
        "negative number"
    else
        "zero"

    println("number is $result")
}
```

Example: Nested if Expression

This program computes the largest number among three numbers.

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

    val n1 = 3
    val n2 = 5
    val n3 = -2

    val max = if (n1 > n2) {
        if (n1 > n3)
            n1
        else
            n3
    } else {
        if (n2 > n3)
            n2
        else
            n3
    }

    println("max = $max")
```

6.2 Kotlin when Expression

The `when` construct in Kotlin can be thought of as a replacement for [Java switch Statement](). It evaluates a section of code among many alternatives.

Example: Simple when Expression

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

    val a = 12
    val b = 5

    println("Enter operator either +, -, * or /")
    val operator = readLine()

    val result = when (operator) {
        "+" -> a + b
        "-" -> a - b
        "*" -> a * b
        "/" -> a / b
        else -> "$operator operator is invalid operator."
    }

    println("result = $result")
}
```

6.3 Kotlin while Loop

The syntax of `while` loop is:

```
while (testExpression) {

    // codes inside body of while loop

}
```

6.4 Kotlin do...while Loop

The `do...while` loop is similar to `while` loop with one key difference. The body of `do...while` loop is executed once before the test expression is checked.
Its syntax is:

```
do {

   // codes inside body of do while loop

} while (testExpression);
```

6.5 Kotlin for Loop

The for loop in Kotlin iterates through anything that provides an iterator. In this article, you learn to create for loop (with the help of examples).

There is no traditional for loop in Kotlin unlike Java and other languages.
In Kotlin, `for` loop is used to iterate through ranges, arrays, maps and so on (anything that provides an iterator).

The syntax of `for` loop in Kotlin is:

```
for (item in collection) {

    // body of loop

}
```

Example: Iterate Through a Range

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

    for (i in 1..5) {
        println(i)
    }
}
```

6.6 : Kotlin break and Continue

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

    for (i in 1..10) {
        if (i == 5) {
            break
        }
        println(i)
    }
}
fun main(args: Array<String>) {

    for (i in 1..5) {
        println("$i Always printed.")
        if (i > 1 && i < 5) {
            continue
        }
        println("$i Not always printed.")
    }
}
```

7. Kotlin functions

Kotlin functions tutorial shows how to use functions in Kotlin language.

Functions in Kotlin are declared with the fun keyword. The body of a function is called a block and is enclosed within { } curly brackets. Functions may return values with the return keyword.

Functions in Kotlin are first-class citizens: they can be stored in variables and data structures, passed as arguments to and returned from other higher-order functions. We can use functions in any way that is possible for other non-function values.

7.1 Kotlin function definition

A function is a code block containing a series of statements. It is a good programming practice for functions to do only one specific task. Functions bring modularity to programs. Functions have the following advantages:

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

A *function signature* is a unique identification of a function for the Kotlin compiler. The signature consists of a function name, its parameters, and the return type.

7.2 Kotlin function naming

Any legal character can be used in the name of a function. By convention, function names begin with an lowercase letter. The function names are verbs or verbs followed by adjectives or nouns. Each subsequent word starts with an uppercase character. The following are typical names of functions in Kotlin:

- execute
- findId
- setName
- getName
- checkIfValid
- testValidity

Kotlin uses lowerCamelCase convention for function names. Other programming languages might use CamelCase or snake_case.

7.3 Simple Function Program

```
fun callMe() {
    println("Printing from callMe() function.")
    println("This is cool (still printing from inside).")
}

fun main(args: Array<String>) {
    callMe()
    println("Printing outside from callMe() function.")
}
```

```
fun addNumbers(n1: Double, n2: Double): Int {
    val sum = n1 + n2
    val sumInteger = sum.toInt()
    return sumInteger
}

fun main(args: Array<String>) {
    val number1 = 12.2
    val number2 = 3.4
    val result: Int

    result = addNumbers(number1, number2)
    println("result = $result")
}
```