LECTURE 5.1: CONTROL FLOW CONTD.

BY LINA HAMMAD & AHMAD BARGHASH

In this part of the lecture, you will learn about when construct in Kotlin with the help of various examples.

WHEN EXPRESSION

- The **when** expression in Kotlin works same as switch case in other programming languages such as C, C++ and Java.
- **when** operator matches the variable value against the branch conditions. If it is satisfying the branch condition, then it will execute the statement inside that scope.
- We can also use **ranges** in **when expression**.

WHEN EXPRESSION

- When matches its argument against all branches sequentially until some branch condition is satisfied. When can be used either as an expression or as a statement. If it is used as an expression, the value of the satisfied branch becomes the value of the overall expression. If it is used as a statement, the values of individual branches are ignored. (Just like with if, each branch can be a block, and its value is the value of the last expression in the block.)

- The **(optional)** else branch is evaluated if none of the other branch conditions are satisfied. If when is used as an expression, the else branch is mandatory, unless the compiler can prove that all possible cases are covered with branch conditions (as, for example, with enum class entries and sealed class subtypes).

WHEN EXPRESSION EXAMPLE

```
fun main(){
    var ch = 'A'

    when(ch){

        'A' -> println("A is a Vowel")
        'E' -> println("E is a Vowel")
        'I' -> println("I is a Vowel")
        'O' -> println("O is a Vowel")
        'U' -> println("U is a Vowel")

        else -> println("$ch is a Consonant")
    }
}
```

The output is:

```
A is a Vowel
```

WHEN EXPRESSION EXAMPLE

- We can also rewrite the same code while merging some of the cases

```
fun main(){

    var ch = 'A'

    when(ch){

        'A', 'E', 'I', 'O', 'U' -> println("$ch is a Vowel")

        else -> println("$ch is a Consonant")
    }
}
```

The output is:

```
A is a Vowel
```

WHEN EXPRESSION EXAMPLE

- The following program takes an input string from the user. Suppose the user entered *. In this case, the expression a * b is evaluated, and the value is assigned to variable result.
- If none of the branch conditions are satisfied (user entered anything except +, -, *, or /) , else branch is evaluated.

```
fun main() {
    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")
}
```

The output is:

```
Enter operator either +, -, * or /
*
result = 60
```

WHEN EXPRESSION EXAMPLE

- In the previous example, we used when as an expression. However, it's not mandatory to use when as an expression. For example,

```
fun main( ) {
    val a = 12
    val b = 5
    println("Enter operator either +, -, * or /")
    val operator = readLine()
    when (operator) {
        "+" -> println("$a + $b = ${a + b}")
        "-" -> println("$a - $b = ${a - b}")
        "*" -> println("$a * $b = ${a * b}")
        "/" -> println("$a / $b = ${a / b}")
        else -> println("$operator is invalid")
    }
}
```

The output is:

```
Enter operator either +, -, * or /
*
12 * 5 = 60
```

- Here, when is not an expression (return value from when is not assigned to anything). In this case, the else branch is not mandatory.

WHEN EXPRESSION EXAMPLE

```
fun main( ) {
    var x = 1
    when (x) {
        1 -> print("x == 1")
        2 -> print("x == 2")
        else -> { // Note the block
            print("x is neither 1 nor 2")
        }
    }
}
```

The output is:

```
x == 1
```

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

The output is:

```
x == 0 or x == 1
```

WHEN EXPRESSION EXAMPLE

```
fun main() {
    val n = -1
    when (n) {
        1, 2, 3 -> println("n is a positive integer less than 4.")
        0 -> println("n is zero")
        -1, -2 -> println("n is a negative integer greater than -3.")
    }
}
```

The output is:

```
n is a negative integer greater than -3.
```

WHEN EXPRESSION EXAMPLE

```
fun main() {
    val x:Int = 5
    when (x) {
        1,2 -> print(" Value of X either 1,2")

        else -> { // Note the block
            print("x is neither 1 nor 2")
        }
    }
}
```

The output is:

```
x is neither 1 nor 2
```

WHEN EXPRESSION EXAMPLE

```
fun main() {
    val a = 11
    val n = "11"
    when (n) {
        "cat" -> println("Cat? Really?")
        12.toString() -> println("Close but not close enough.")
        a.toString() -> println("Bingo! It's eleven.")
    }
}
```

The output is:

```
Bingo! It's eleven.
```

KOTLIN WHEN EXPRESSION WITH RANGES

- Range is declared by two dots ..
- We can also check a value for being **in** or **!in** a range or a collection , In the following example we have used multiple ranges inside the when expression to find out the digits in the given number.

```
fun main() {
    var x = 30
    when (x) {
        in 1..10 -> print("x is in the range") // from 1 to 10
        !in 10..20 -> print("x is outside the range") // x < 20
        else -> print("none of the above") // x > 10 and x <= 20
    }
}
```

The output is:

```
x is outside the range
```

KOTLIN WHEN EXPRESSION WITH RANGES EXAMPLE

```
fun main( ){

    var num = 78
    when(num) {
        in 1..9 -> println("$num is a single digit number")
        in 10..99 -> println("$num is a two digit number")
        in 100..999 -> println("$num is a three digit number")
        else -> println("$num has more than three digits")
    }
}
```

The output is:

```
78 is a two digit number
```

KOTLIN WHEN EXPRESSION WITH RANGES EXAMPLE

```
fun main(){

    var age = 16

    when(age) {
        in 1..17 -> {
            val num = 18 - age
            println("You will be eligible for voting in $num years")
        }
        in 18..100 -> println("You are eligible for voting")
    }
}
```

The output is:

```
You will be eligible for voting in 2 years
```

LECTURE 5.1: KOTLIN TYPE CONVERSION

BY LINA HAMMAD & AHMAD BARGHASH

In this part of the lecture, you will learn about type conversion; how to convert a variable of one type to another with the help of example.

KOTLIN TYPE CONVERSION

- In Kotlin, a numeric value of one type is not automatically converted to another type even when the other type is larger. This is different from how Java handles numeric conversions. For example;
- In Java,

```
int number1 = 55;
long number2 = number1;     // Valid code
```

- Here, value of number1 of type int is automatically converted to type long and assigned to variable number2.
- In Kotlin,

```
val number1: Int = 55
val number2: Long = number1    // Error: type mismatch.
```

- Though the size of Long is larger than Int, Kotlin doesn't automatically convert Int to Long.
- Instead, you need to convert to type Long explicitly. Kotlin does it for type safety to avoid surprises.

TYPE CONVERSION

- Here's a list of functions in Kotlin used for type conversion:
  - toByte()
  - toShort()
  - toInt()
  - toLong()
  - toFloat()
  - toDouble()
  - toChar()

  - Note, there is no conversion for Boolean types.

CONVERSION FROM LARGER TO SMALLER TYPE

- The functions mentioned in the previous slide can be used in both directions (conversion from larger to smaller type and conversion from smaller to larger type).
- Conversion from smaller type to a larger one is straight forward as the value remains the same.
- However, conversion from larger to smaller type may truncate the value. For example, when converting from integer to Byte, the resulting Byte value is represented by the least significant 8 bits of this Int value. For example,

```
fun main() {
    val number1: Int = 545344
    val number2: Byte = number1.toByte()
    println("number1 = $number1")
    println("number2 = $number2")
}
```

- When you run the program, the output will be:

```
number1 = 545344
number2 = 64
```

CONVERT CHAR TO STRING (SMALL TO LARGE)

```
fun main( ) {
    val ch = 'c'
    var st = ch.toString();
    println("The string is: $st")
}
```

The output is:

```
The string is: c
```

CONVERT DOUBLE TO INT

```
fun main( ) {
    println(1.1.toInt())
    println(1.7.toInt())
    println(-2.3.toInt())
    println(-2.9.toInt())
}
```

The output is:

```
 1
 1
-2
-2
```

SIMPLE TYPE CASTING EXAMPLES

```
fun main( ){

    /**
     * Double to int type casting
     */
    println("4.554 to int: " + (4.554.toInt()))

    /**
     * int to Char type casting
     */
    println("65 to Char: " + (65.toChar()))

    /**
     * Char to int type casting
     */
    println("B to int: " + ('B'.toInt()))
}
```

The output is:

```
4.554 to int: 4
65 to Char: A
B to int: 66
```

ALSO CHECK OUT THESE ARTICLES RELATED TO TYPE CONVERSION

- Sring to Int, and Int to String Conversion
- Long to Int, and Int to Long Conversion
- Double to Int, and Int to Double Conversion
- Long to Double, and Double to Long Conversion
- Char to Int, and Int to Char
- String to Long, and Long to String Conversion
- String to Array, and Array to String Conversion
- String to Boolean, and Boolean to String Conversion
- String to Byte, and Byte to String Conversion
- Int to Byte, and Byte to Int Conversion

ASCII CHARACTER SET

- Character data is represented in a computer by using standardized numeric codes which have been developed. The most widely accepted code is called the **American Standard Code for Information Interchange** (**ASCII**).
- The ASCII code associates an integer value for each symbol in the character set, such as letters, digits, punctuation marks, special characters, and control characters.

ASCII printable characters	ASCII Code
A – Z	65 – 90
a – z	97 - 122
@	64
Space	32
$	36

LECTURE 4.3: KOTLIN USER INPUT

BY LINA HAMMAD & AHMAD BARGHASH

In this part, you will learn to take input from the user in Kotlin.

REVISION FROM LAST LECTURE

- Sring to Int, and Int to String Conversion
- Long to Int, and Int to Long Conversion
- Double to Int, and Int to Double Conversion
- Long to Double, and Double to Long Conversion
- Char to Int, and Int to Char
- String to Long, and Long to String Conversion
- String to Array, and Array to String Conversion
- String to Boolean, and Boolean to String Conversion
- String to Byte, and Byte to String Conversion
- Int to Byte, and Byte to Int Conversion

Note: there is no way to convert the String to the Character but remember that each String is a set of character and you can reach any character of the string using the index if that character:
Ex:

```
var name= "Alex"
var char1 = name[0]
var char2 = name[2]
println("char1= $char1, char2= $char2")
```

The output is:

```
Char1= A, char2= e
```

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.
- Kotlin has readLine() function which returns a line from standard input stream.
- readLine() returns String which can be converted to other types as required.

KOTLIN INPUT EXAMPLE

```
fun main() {
    print("Enter text: ")
    val stringInput = readLine()
    println("You entered: $stringInput")
}
```

The output is:

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

GETTING INTEGER OR OTHER DATA TYPES FROM READLINE() METHOD

- Getting integer or other data types from readLine() method
- The output returned by readLine() function is String and can be converted to other types.
- here's how you can convert String returned by readLine() to an Integer.

```
println("Enter number 1: ")
val x = readLine()!!.toInt()
println("Enter number 2: ")
val y = readLine()!!.toInt()
val sum = x + y
println("Sum = $sum")
```

The output is:

```
Enter number 1:
5
Enter number 2:
3
Sum = 8
```

```
fun main() {
    print("Enter the first number: ")
    var number1 = readLine()!!
    print("Enter the second number: ")
    var number2 = readLine()!!
    var sum = number1.toInt() + number2.toInt()

    println("$number1 + $number2 = $sum")
}
```

The output is:

```
Enter the first number: 12
Enter the second number: 12
12 + 12 = 24
```

GETTING INTEGER OR OTHER DATA TYPES FROM READLINE() METHOD

- If you examine the code carefully, you'd find another operator **!!**, the double exclamation operator.
- This operator is part of null pointer safely offered by Kotlin.
- Converts any value to a non-null type and throws an exception if the value is null.
- **Note**: readLine() may return null if input is redirected to a file and end of file is reached.

GETTING ANY DATA TYPE INPUT FROM THE USER USING THE SCANNER CLASS

- 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 Scanner object from this class.

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

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

GETTING ANY DATA TYPE INPUT FROM THE USER USING THE SCANNER CLASS EXAMPLE

```
import java.util.Scanner
fun main() {
    // 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 num1 = reader.nextInt()
    var sum= num1 + 10
    println("You entered: $num1")
    println("$num1 + 10 = $sum")
}
```

The output is:

```
Enter a number: 20
You entered: 20
20 + 10 = 30
```

GETTING ANY DATA TYPE INPUT FROM THE USER USING THE SCANNER CLASS EXAMPLE

```
import java.util.Scanner
fun main() {
// Creates an instance which takes input from standard input (keyboard)
    val reader = Scanner(System.`in`)
    print("Enter a letter: ")
    // next() reads the next String from the keyboard
    // single() to convert the one-digit string to a character
    var c = reader.next().single()
    var sum= c.toInt() + 10
    println("You entered: ${c.toInt()} )
    println("${c.toInt()} + 10: $sum")

}
```

The output is:

```
Enter a letter: w
You entered: 119
119 + 10 = 129
```

GETTING ANY DATA TYPE INPUT FROM THE USER USING THE SCANNER CLASS

- The 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.

- To get Long, Float, double and Boolean input from the user, you can use nextLong(), nextFloat(), nextDouble() and nextBoolean() methods respectively.