Learning Kotlin.

Shafiqur Rhaman

October 13, 2022

Build command for `kotlin` code compile and Execute.

```
kotlinc basic-operation.kt -include-runtime -d basi-opration.jar
java -jar basi-opration.jar
```

Variables and Data Types

- There is two types of variables in `kotlin`
  - Mutable

    ```
    var aquarium = 1
    aquarium = 50
    ```

  - Immutable

    ```
    val fish = "Nemo"
    ```

- Immutable list can not reassign new value but can be manipulate list.

  ```
  val myList = mutalbleListOf("tuna", "salmon", "shark")
  myList.remove("shark")
  ```

- Byte

  ```
  val myByte: Byte = 13
  ```

- Short

  ```
  val myShort: Short = 125
  ```

- Int

  ```
  val myInt: Int = 12345678
  ```

- Long

  ```
  val myLong: Long = 12_123_123_123_1234
  ```

- Float

  ```
  val myFloar: Float = 13.43F
  ```

- Double

  ```
  val myDouble: Double = 3.1233445535445
  ```

- Boolean

  ```
  val myBoolean: Boolean = true
  ```

- Char

  ```
  val myChar: Char = 'A'
  ```

- String

  ```
  val myString: String = "Shafiqur Rahman"
  ```

Arithmetic Operators

- (+, -, /, *, %)
- int divide by int returns int
- float divide by float returns float

  ```
  1/2 // Will return 0
  1.0/20. // Will return 0.5
  ```

- Arithmetic Methods

  ```
  val fish = 2
  fish.times(6)
  fish.div(10)
  fish.plus(3)
  fish.minus(3)
  ```

- Boxing

  ```
  1.toLong()

  val boxed: Number = 1
  boxed.toLong()
  ```

Comparison Operators

- `(==, !=, <, >, <=, >=)`

Null Safety

- Add `?` to indicate variable can contain null value

  ```
  var marble: Int? = null
  var lotsOfFish: List<String?> = listOf(null, null)
  var evenMoreFish: List<String>? = null
  var definitelyFish: List<String?>? = null
  ```

- Force a null able type in `kotlin`

  ```
  goldfish!!.eat()
  ```

- Null check operator

  ```
  return fishFoodTreats?.dec() ?: 0
  ```

Switch

```
var welcomeMessage = "Hello and welcome to Kotlin"
when (welcomeMessage.length) {
   0 -> println("Nothing to say?")
   in 1..50 -> println("Perfect")
   else -> println("Too long!")
}
```

Array

- Typed Array

  ```
  val numbers = intArrayOf(1, 2, 3)
  ```

- Non Typed Array

  ```
  import java.utils.*

  val school = arrayOf("tuna", "salmon", "shark")
  println(Arrays.toString(school))
  ```

- Arrays of arrays

```
import java.util.*

var fish = 12
var plants = 5

val swam = listOf(fish, plants)

val bigSwarm = arrayOf(swam, arrayOf("Dolphin", "Whale", "orka"))


println(Arrays.toString(bigSwarm))
```

- Array comprehension

  ```
  val array = Array(5) { it * 2 }
  println(array.asList())
  ```

-

List

Map

For Loop

- looping without index

  ```
  for (element in swarm) println(element)
  ```

- looping with index

  ```
  for ((index, element) in swarm.withIndex()){
      println("Fish at $index is $element")
  }
  ```

- Ranges print

  ```
  for (i in 'b'..'g') println(i)

  for (i in 1..120) println(i)

  for (i in 5 downTo 1) println(i)

  for (i in 3..6 step 2) println(i)
  ```

For Each

While Loop

Repeat Loop

Filer

- Eager Filter (Create a new list)

  ```
  val decorations = listOf(
      "rock", "pagoda", "plastic plant", "alligator", "flowerpot"
  )
  val eager = decorations.filter { it[0] == 'p'}
  println(eager)
  ```

- Lazy Filter

  ```
  val decorations = listOf(
      "rock", "pagoda", "plastic plant", "alligator", "flowerpot"
  )
  val filtered = decorations.asSequence().filter() { it[0] == 'p' }
  println(filtered)
  println(filtered.toList())
  ```

`lambda`

- A value assigned at compile time, and the value never changes when the variable is accessed.
- a lambda assigned at compile time, and the lambda is executed every time the variable is referenced, returning a different value.

Class

```
class Aquarium (
    var width: Int = 20,
    var height: Int = 40,
    var length: Int = 100
) {
    var volume: Int
get() = width * height * length / 1000
set(value) { height = (value * 1000) / (width  * length) }

    var water = volume * 0.9
```

```
    constructor(numberOfFish: Int): this() {
val water: Int = numberOfFish * 2000
val tank: Double = water + (water * 0.1)
height = (tank / (length * width)).toInt()
    }

}


fun main() {
    val smallAquarium = Aquarium(numberOfFish = 9)
    println(
" Length: ${smallAquarium.length} " +
" Width: ${smallAquarium.width} " +
" height: ${smallAquarium.height} "
    )

    println("Volume: ${smallAquarium.volume}")
}
```

Package Visibility

- public - Default Everywhere.
- private - File
- internal - Module

Class Visibility

- public - Default. Class and public member
- private - Inside class. Sub classes `can't` see.
- protected - Inside class. Sub classes can see.
- internal - Module

Inheritance

- We have to add `open` to class to make sub class from it.
- We have to add `override` to sub class to override properties or methods.

  ```
  import kotlin.math.PI

  open class BaseAquarium (
  ```

```
        var lenght: Int = 100,
        var width: Int = 20,
        var height: Int = 40
    ){
        open var volume: Int
    get() = (width * height * lenght) / 1000
    set(value) { height = (value * 1000) / (width * lenght) }

        open var water = volume * 0.9

        constructor(numberOfFish: Int): this(){
    val water: Int = numberOfFish * 2000
    val tank: Double = water + (water * 0.1)
    height = (tank / (lenght * width)).toInt()
        }
    }


    class TowerTank(): BaseAquarium() {
        override var water = volume * 0.8

        override var volume: Int
    get() = ((width * height * lenght) / 1000 * PI).toInt()
    set(value) { height = (value * 1000) / (width * lenght) }
    }


    fun main() {
        var myTowerTank = TowerTank()

        println("Volume of new tower tank aquarium ${myTowerTank.volume}")

    }
```

Abstract Class, Interface, Singleton Object

```
fun main() {
    delegate()
}

fun delegate(){
    val pleco = Plecostomus()
    println("Fish has color ${pleco.color}")
    pleco.eat()
}

interface FishAction{
    fun eat()
}

interface FishColor {
    val color: String
}

class Plecostomus(fishColor: FishColor = GoldColor):
    FishAction by PrintingFishAction("a lot of Food"),
    FishColor by fishColor


object GoldColor: FishColor {
    override val color = "gold"
}


object RedColor: FishColor {
    override val color = "red"
}


class PrintingFishAction(val food: String): FishAction {
    override fun eat() {
println(food)
    }
}
```