02

02-4. 컬렉션 타입

Basic Syntax

컬렉션 타입

배열

- Array로 표현되며 Array는 get, set. size등의 함수를 포함하고 있는 클래스
- arrayOf ( ) 함수를 이용 배열 선언

```
fun main(args: Array<String>) {
    //arrayOf 함수 이용
    var array = arrayOf(1, "kkang", true)
    array[0]=10
    array[2]="world"
    println("${array[0]} .. ${array[1]} .. ${array[2]}")
    println("size : ${array.size} .. ${array.get(0)} .. ${array.get(1)} .. ${array.get(2)}")

}
```

```
var arrayInt = arrayOf<Int>(10, 20, 30)
```

컬렉션 타입

배열

- 기초 데이터 타입과 관련된 xxxArrayOf( ) 함수들을 제공하고 이를 이용하여 배열을 만들어 사용할 수 있습니다.

- booleanArrayOf()
- byteArrayOf()
- charArrayOf()
- doubleArrayOf()
- floatArrayOf()
- intArrayOf()
- longArrayOf()
- shortArrayOf()

```
var arrayInt2= intArrayOf(10, 20, 30)
var arrayDouble= doubleArrayOf(10.0, 20.0, 30.0)
```

컬렉션 타입

배열

- arrayOf ( ), intArrayOf ( ) 함수들은 Array 클래스 타입의 배열 객체를 반환
- arrayOf ( ) 함수를 이용하지 않고 Array 클래스를 직접 이용하여 배열을 만들 수도 있습니다.

- Array(size: Int, init: (Int) -> T)

- 생성자의 매개변수가 두 개인데 첫 번째 매개변수가 배열의 크기입니다. 그리고 두 번째 매개변수가 배
- 열의 데이터입니다.

```
var array3=Array(3,  { i -> i*10})
```

```
var array4=Array<Int>(3, {i -> i * 10})
var array5=IntArray(3, { i -> i *10})
```

컬렉션 타입

배열

- Array<Int>처럼 제네릭을 이용하여 타입을 명시해도 되며, IntArray라는 클래스를 이용해도 됩니다. 코틀린에서는 각 타입의 배열을 위한 클래스를 따로 제공합니다

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

```
var array4=Array<Int>(3, {i -> i * 10})
var array5=IntArray(3, { i -> i *10})
```

컬렉션 타입

배열

- 배열을 정의할 때 크기만 지정하고 데이터는 대입하지 않는 빈 상태로 정의하는 방법

```
var array2= arrayOfNulls<Any>(3)
array2[0]=10
array2[1]="hello"
array2[2]=true
println("${array2[0]} .. ${array2[1]} .. ${array2[2]}")

var emptyArray=Array<String>(3,{""})
emptyArray[0]="hello"
emptyArray[1]="world"
emptyArray[2]="kkang"
println("${emptyArray[0]} .. ${emptyArray[1]} .. ${emptyArray[2]}")
```

컬렉션 타입

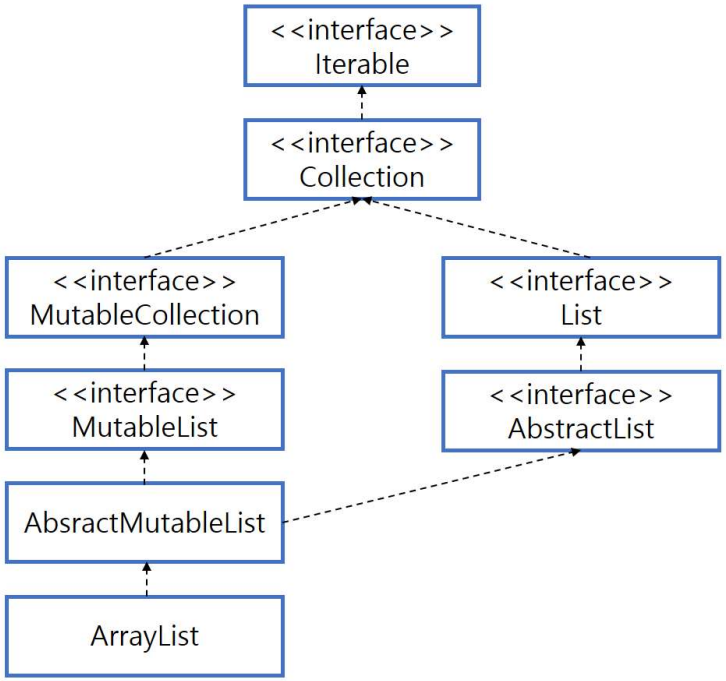
List, Set, Map

- List, Set, Map 모두 Collection이라는 인터페이스 타입으로 표현되는 클래스이며 통칭해서 컬렉션 타입의 클래스들이라고 부릅니다.
- List: 순서가 있는 데이터 집합. 데이터의 중복 허용
- Set: 순서가 없으며 데이터 중복을 허용하지 않음
- Map: 키와 값으로 이루어지는 데이터 집합. 순서가 없으며 키의 중복은 허용하지 않음

- Collection 타입의 클래스들은 mutable 클래스와 immutable 클래스로 구분
- kotlin.collection.List 인터페이스로 표현되는 객체는 immutable 이며 size(), get() 함수만 제공
- kotlin.collection.MutableList 인터페이스로 표현되는 객체는 mutable 이며 size(), get() 함수 이외에 add(), set() 함수 제공

컬렉션 타입

List, Set, Map



컬렉션 타입

List, Set, Map

- 불변과 가변 List를 만들기 위한 함수가 listOf ( ), mutableListOf ( )로 제공되며 이 함수들을 이용해 List 객체를 만들어 이용합니다.

	타입	함수	특징
List	List	listOf()	Immutable
	MutableList	mutableListOf()	mutable
Map	Map	mapOf()	Immutable
	MutableMap	mutableMapOf()	mutable
Set	Set	setOf()	Immutable
	MutableSet	mutableSetOf()	mutable

```
fun main(args: Array<String>) {
    val immutableList: List<String> = listOf("hello", "world")
    println("${immutableList.get(0)} .. ${immutableList.get(1)}")
}
```

```
val mutableList: MutableList<String> = mutableListOf("hello", "world")
mutableList.add("kkang")
mutableList.set(1, "korea")
println("${mutableList.get(0)} .. ${mutableList.get(1)} .. ${mutableList.get(2)}")
```

컬렉션 타입

List, Set, Map

- Map 은 mapOf( ), mutableMapOf() 함수 이용
- 데이터를 추가하는 방법은 01번 줄처럼 Pair 클래스를 이용하는 방법과 to 를 이용하는 방법

```
val immutableMap1= mapOf<String, String>(Pair("one","hello"), Pair("two", "world"))
println("${immutableMap1.get("one")} .. ${immutableMap1.get("two")}")
val immutableMap2= mapOf<String, String>("one" to "hello", "two" to "kkang")
println("${immutableMap2.get("one")} .. ${immutableMap2.get("two")}")

val mutableMap= mutableMapOf<String, String>()
mutableMap.put("one", "hello")
mutableMap.put("two","map")
println("${mutableMap.get("one")} .. ${mutableMap.get("two")}")
```

컬렉션 타입

List, Set, Map

- Set은 setOf ( ) 함수 혹은mutableSet( ) 함수를 이용하여 가변 객체를 만든 예입니다.

```
val immutableSet= setOf<String>("hello","hello","world")
println("${immutableSet.elementAt(0)} .. ${immutableSet.elementAt(1)}")
val mutableSet= mutableSetOf<String>()
mutableSet.add("hello")
mutableSet.add("set")
println("${mutableSet.elementAt(0)} .. ${mutableSet.elementAt(1)}")
```

컬렉션 타입

Iterator

- hasNext() 함수와 next() 함수를 이용해 순차적 이용

```
val list1= listOf<String>("hello","list")
    val iterator1=list1.iterator()
    while (iterator1.hasNext()){
        println(iterator1.next())
    }
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare