КОЛЕКЦІЇ В KOTLIN

МАСИВИ В KOTLIN. СТВОРЕННЯ

```kotlin
fun main() {
    // масив з елементів
    val numbers = arrayOf(1, 2, 3, 4, 5)
    val fruits = arrayOf("Apple", "Banana", "Cherry")
    // зповнення масиву заданого розміру
    val zeros = Array(5) { 0 } // [0, 0, 0, 0, 0]
    val squares = Array(5) { it * it } // [0, 1, 4, 9, 16]
    // специфікація типу масиву
    val intArray = intArrayOf(1, 2, 3)
    val doubleArray = doubleArrayOf(1.1, 2.2, 3.3)
    val charArray = charArrayOf('a', 'b', 'c')
}
```

МАСИВИ В KOTLIN. ДОСТУП ДО ЕЛЕМЕНТІВ

```
fun main() {
    val items = arrayOf("Kotlin", "Java", "Swift")
    println(items[0])          // Kotlin
    items[1] = "Python"
    println(items.joinToString()) // Kotlin, Python, Swift
    // перегляд елементів
    for (item in items) {    println(item)    }
    // за індексами
    for (i in items.indices) {
        println("Index $i: ${items[i]}")
    }
    // метод forEach
    items.forEach { println(it) }
}
```

- The Kotlin Standard Library provides a comprehensive set of tools for managing collections – groups of a variable number of items (possibly zero) that are significant to the problem being solved and are commonly operated on.
- Collections are a common concept for most programming languages, so if you're familiar with, for example, Java or Python collections, you can skip this introduction and proceed to the detailed sections.
- A collection usually contains a number of objects of the same type (and its subtypes). Objects in a collection are called elements or items. For example, all the students in a department form a collection that can be used to calculate their average age.

Kotlin Documentation

The following collection types are relevant for Kotlin:

- List is an ordered collection with access to elements by indices – integer numbers that reflect their position. Elements can occur more than once in a list.
- Set is a collection of unique elements. It reflects the mathematical abstraction of set: a group of objects without repetitions. Generally, the order of set elements has no significance.
- Map (or dictionary) is a set of key-value pairs. Keys are unique, and each of them maps to exactly one value. The values can be duplicates.

Kotlin Documentation

COLLECTION HIERARCHY IN KOTLIN

Kotlin collections are categorized into **read-only (immutable)** and **mutable** collections.

Immutable (Read-only) Collections.

Interfaces:
Collection, List, Set, Map

Cannot be modified after creation

Mutable Collections

Interfaces:
MutableCollection, MutableList, MutableSet, MutableMap

Support modification: add, remove, put, etc.

LIST DETAILS

Immutable List

```
fun main() {
    val list: List<String> =
        listOf("Kotlin", "Java", "Swift")
    println(list.size)       // 3
    println(list.contains("Java"))
    println(list)
    list.add("Pascal")
}
```

Mutable List

```
fun main() {
    val list: MutableList<String> =
        mutableListOf("Kotlin", "Java")
    println(list.size)       // 3
    list.add("Pascal")
    println(list)
    list.remove("Kotlin")
    println(list)
}
```

SETS

Set (Immutable)

```
fun main() {
    val uniqueValues: Set<Int> = setOf(1, 2, 3, 3, 4)
    println(uniqueValues) // [1, 2, 3, 4]
    println(uniqueValues.contains(11)) // false
    println(uniqueValues.contains(1)) // true
}
```

Mutable Set

```
fun main() {
    val mutableSet: MutableSet<String> = mutableSetOf("Apple", "Banana")
    mutableSet.add("Orange")
    mutableSet.remove("Banana")
    mutableSet.add("Apple")
    println(mutableSet) // [Apple, Orange]
}
```

MAPS

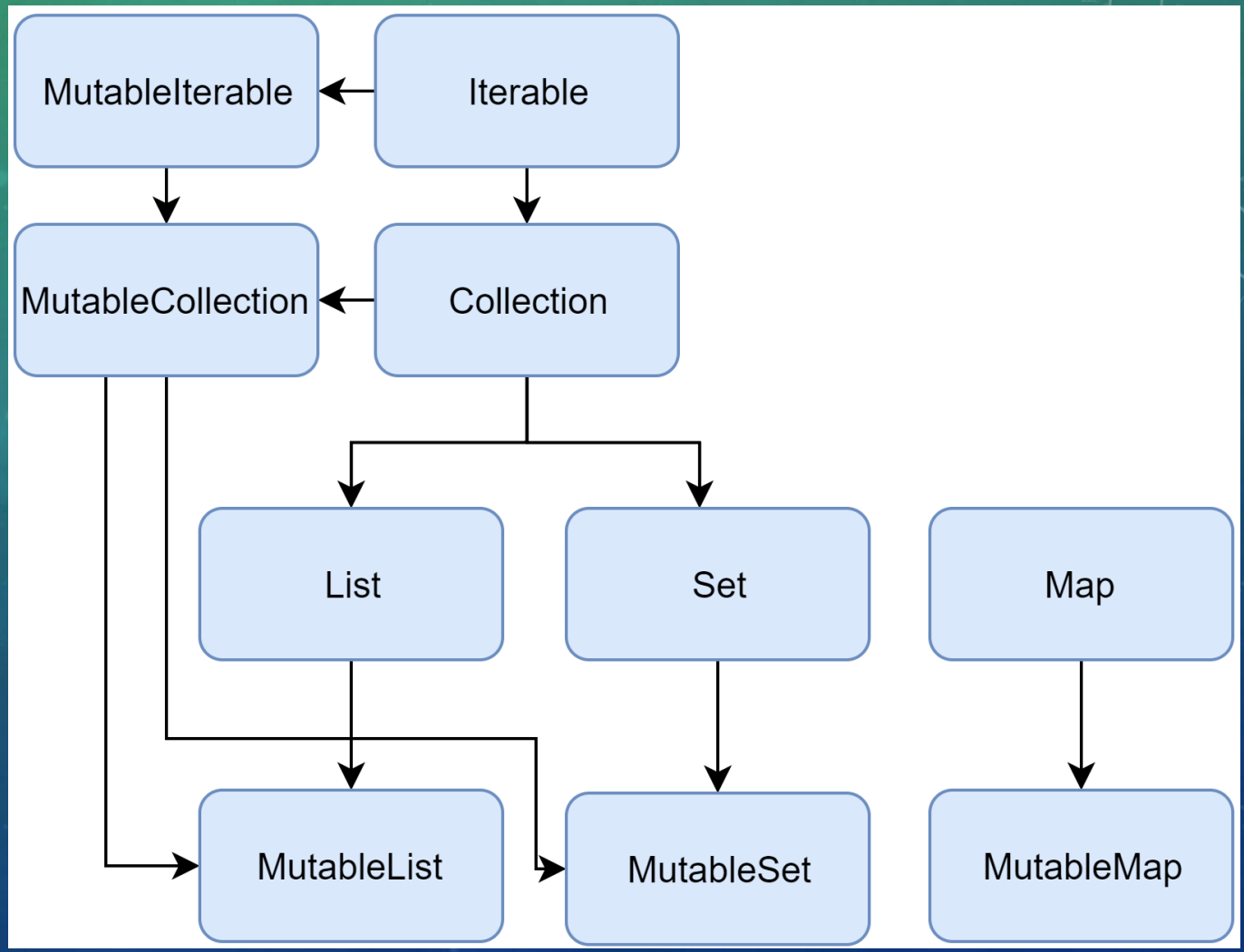
Mutable Map

```
fun main() {
    val capitals =
        mutableMapOf("USA" to "Washington",
            "UK" to "London")
    capitals["Germany"] = "Berlin" // Add
    capitals["UK"] = "Manchester"  // Update
    capitals.remove( key: "USA")
    println(capitals)
}
```

Map (Immutable)

```
fun main() {
    val countryCodes = mapOf(
        "US" to "United States",
        "FR" to "France",
        "IN" to "India"
    )
    println(countryCodes["FR"]) // France
}
```

COLLECTION HIERARCHY IN KOTLIN



CONVERSION BETWEEN COLLECTIONS

```
fun main() {
    var set = setOf(1, 2, 3, 77)
    val listFromSet = set.toMutableList()
    println(listFromSet)
    listFromSet.add(77)
    println(listFromSet)
    set = listFromSet.toMutableSet()
    println(set)
    set.add(55)
    println(set)
    set = set.toSet()
    set.add(33)
}
```

```
fun main() {

    val map = mapOf("a" to 1, "b" to 2)
    println(map)
    val keysList = map.keys.toList()
    println(keysList)
    val valuesSet = map.values.toSet()
    println(valuesSet)

}
```

FUNCTIONAL STYLE

```
fun main() {
    data class Person(val name: String, val age: Int)

    val people = listOf(
        Person("Alice", 25),
        Person("Bob", 32),
        Person("Charlie", 29)
    )

// Find people older than 30
    val older = people.filter { it.age > 30 }
    println(older) // [Person(name=Bob, age=32)]

// Get list of names
    val names = people.map { it.name }
    println(names) // [Alice, Bob, Charlie]
}
```

KOTLIN COLLECTIONS VS JAVA STREAM API

Java Stream API

```java
List<String> names = List.of("Alice", "Bob", "Charlie");
List<String> upper = names.stream()
                          .filter(name -> name.length() > 3
                          .map(String::toUpperCase)
                          .collect(Collectors.toList());
```

Kotlin Collection

```kotlin
val names = listOf("Alice", "Bob", "Charlie")
val upper = names.filter { it.length > 3 }
                 .map { it.uppercase() }

println(upper) // [ALICE, CHARLIE]
```

JAVA STREAM API VS KOTLIN COLLECTIONS

Java Stream API	Kotlin Equivalent
filter()	filter { }
map()	map { }
flatMap()	flatMap { }
sorted()	sortedBy { }
limit() / skip()	take() / drop()
collect(Collectors.toList())	toList()
anyMatch() / allMatch()	any { } / all { }
reduce()	reduce { acc, it -> }

LAZY EVALUATION. KOTLIN SEQUENCES.

Eager (List)

```kotlin
val list = listOf(1, 2, 3, 4, 5)
val result = list.map { it * 2 }.filter { it > 5 }
println(result) // [6, 8, 10]
```

Lazy (Sequence)

```kotlin
val seq = listOf(1, 2, 3, 4, 5).asSequence()
val result = seq.map { it * 2 }
                .filter { it > 5 }
                .toList()

println(result) // [6, 8, 10]
```