



Kotlin Collection

Eko Kurniawan Khannedy



License

- Dokumen ini boleh Anda gunakan atau ubah untuk keperluan non komersial
- Tapi Anda wajib mencantumkan sumber dan pemilik dokumen ini
- Untuk keperluan komersial, silahkan hubungi pemilik dokumen ini

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow



Pengenalan Collection



Sebelum Belajar Materi Ini

- Kotlin Dasar
- Kotlin Object Oriented Programming
- Kotlin Generic
- <https://www.udemy.com/course/pemrograman-kotlin-pemula-sampai-mahir/?referralCode=98BE2E779EB8A0BEC230>



Pengenalan Collection

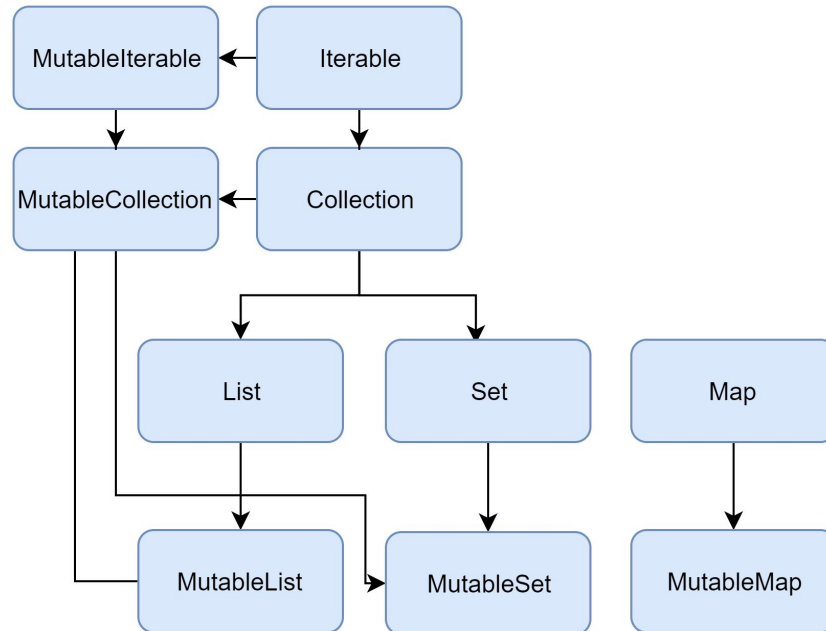
- Collection adalah hal umum yang biasa dimiliki di bahasa pemrograman, seperti Java, Python, Ruby dan lain-lain
- Sebuah collection biasanya berisikan sejumlah data, bahkan bisa kosong
- Kotlin standard library menyediakan data Collection yang sudah bisa langsung kita gunakan, tanpa tambahan library



Tipe Collection

- Secara garis besar, ada 2 jenis collection di Kotlin
 1. Immutable, collection yang hanya mendukung operasi read only
 2. Mutable, collection turunan immutable, namun mendukung operasi write juga (add, edit, dan remove)

Kotlin Collection Diagram



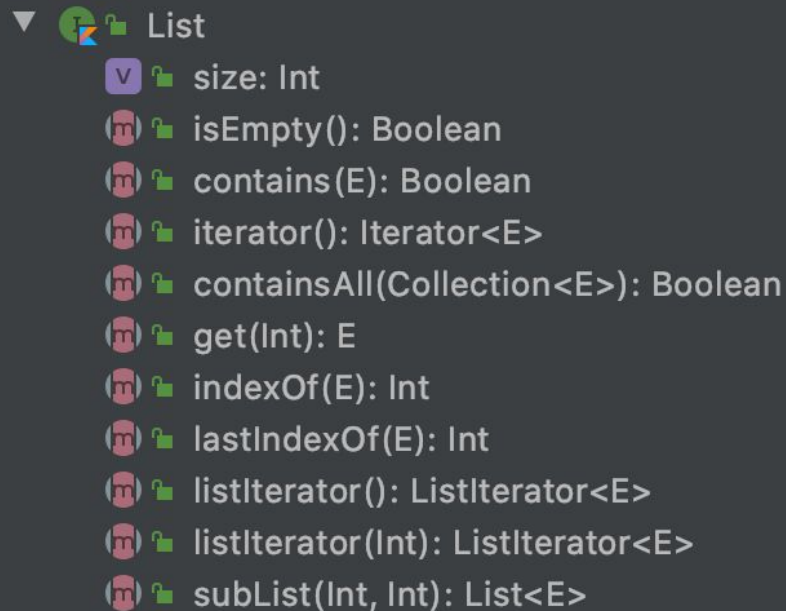
List



List

- List, adalah collection yang datanya seperti Array dan memiliki akses data menggunakan index
- Data di List boleh duplicate, artinya kita boleh memasukkan data yang sama berkali-kali ke dalam sebuah List yang sama

Function di List



```
▼ List
  size: Int
  isEmpty(): Boolean
  contains(E): Boolean
  iterator(): Iterator<E>
  containsAll(Collection<E>): Boolean
  get(Int): E
  indexOf(E): Int
  lastIndexOf(E): Int
  listIterator(): ListIterator<E>
  listIterator(Int): ListIterator<E>
  subList(Int, Int): List<E>
```

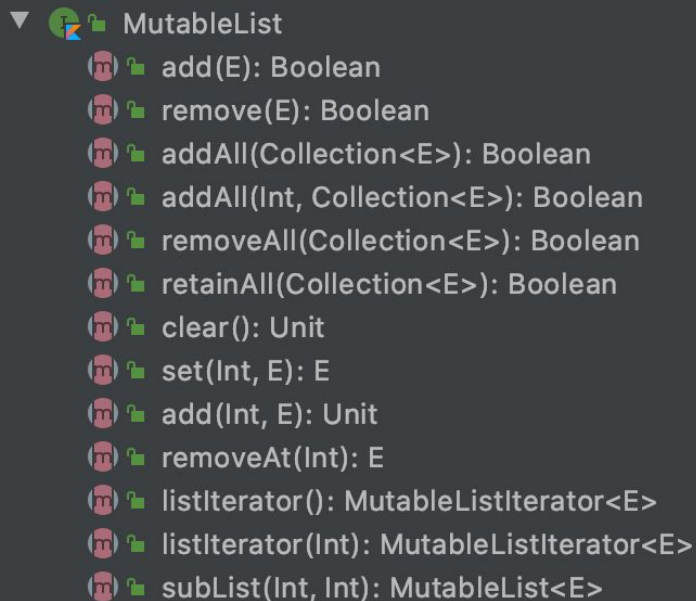
A screenshot of a code editor showing the List interface methods. The list is expanded, showing a dropdown arrow, a lock icon, and a multi-colored icon. The methods listed are: size: Int (with a value icon), isEmpty(): Boolean, contains(E): Boolean, iterator(): Iterator<E>, containsAll(Collection<E>): Boolean, get(Int): E, indexOf(E): Int, lastIndexOf(E): Int, listIterator(): ListIterator<E>, listIterator(Int): ListIterator<E>, and subList(Int, Int): List<E>. Each method has a lock icon and a multi-colored icon.



Code Program : List

```
3 ▶ fun main() {  
4     val list: List<String> = listOf("Eko", "Kurniawan", "Khannedy")  
5     println(list[0])  
6     println(list[1])  
7     println(list[2])  
8     println(list.isEmpty())  
9     println(list.indexOf("Kurniawan"))  
10    println(list.contains("Khannedy"))  
11 }  
12
```

Function di Mutable List



A screenshot of the IntelliJ IDEA IDE showing the `MutableList` class and its methods. The class is expanded, showing a list of methods with their signatures. Each method is preceded by a small icon consisting of a pink circle with a white 'm' and a green folder icon.

- ▼ `MutableList`
 - `add(E): Boolean`
 - `remove(E): Boolean`
 - `addAll(Collection<E>): Boolean`
 - `addAll(Int, Collection<E>): Boolean`
 - `removeAll(Collection<E>): Boolean`
 - `retainAll(Collection<E>): Boolean`
 - `clear(): Unit`
 - `set(Int, E): E`
 - `add(Int, E): Unit`
 - `removeAt(Int): E`
 - `listIterator(): MutableListIterator<E>`
 - `listIterator(Int): MutableListIterator<E>`
 - `subList(Int, Int): MutableList<E>`

Code Program : Mutable List

```
3 ▶ fun main() {  
4     val mutableList: MutableList<String> = mutableListOf()  
5     mutableList.add("Eko")  
6     mutableList.add("Kurniawan")  
7     mutableList.add("Khannedy")  
8     println(mutableList[0])  
9     println(mutableList[1])  
10    println(mutableList[2])  
11    println(mutableList.isEmpty())  
12 }  
13
```

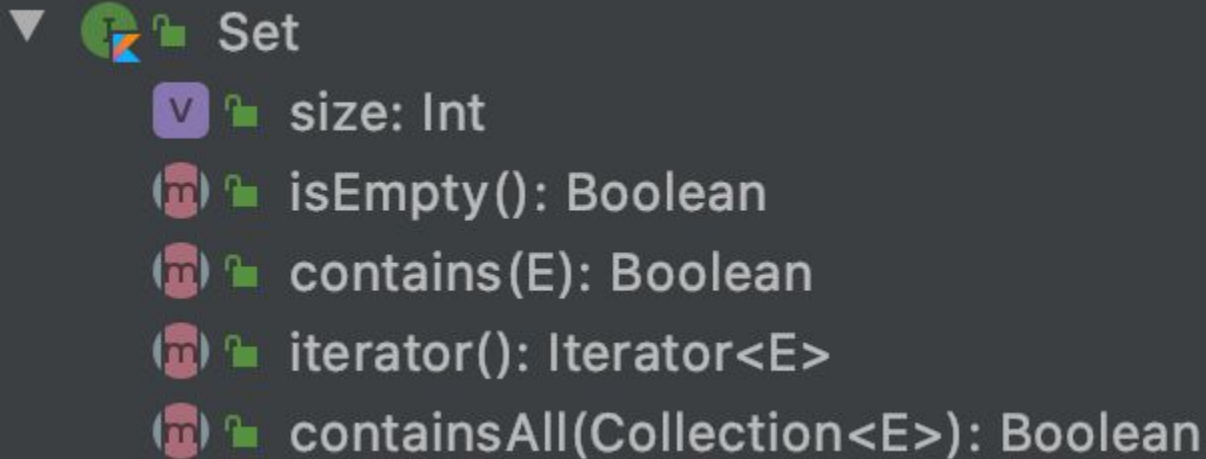
Set



Set

- Set, adalah collection yang datanya harus unik dan tidak pasti berurut
- Saat kita memasukkan data duplicate ke Set, maka data hanya akan disimpan satu, data duplicatenya tidak akan ditambahkan ke dalam Set
- Set sangat cocok untuk menyimpan data yang memang unik, tidak boleh sama
- Set menggunakan function hashCode() dan equals() untuk membandingkan apakah sebuah object sama atau tidak, jika hashCode() dan equals() nya sama, maka dianggap data tersebut duplicate, dan tidak akan diterima oleh Set

Function di Set



A screenshot of a Scala IDE snippet showing the methods of the `Set` class. The snippet is displayed on a dark background with a light gray border. It includes a dropdown arrow, a Scala logo, and a lock icon. The methods listed are:

- `size: Int` (with a purple 'v' icon)
- `isEmpty(): Boolean` (with a red 'm' icon)
- `contains(E): Boolean` (with a red 'm' icon)
- `iterator(): Iterator<E>` (with a red 'm' icon)
- `containsAll(Collection<E>): Boolean` (with a red 'm' icon)

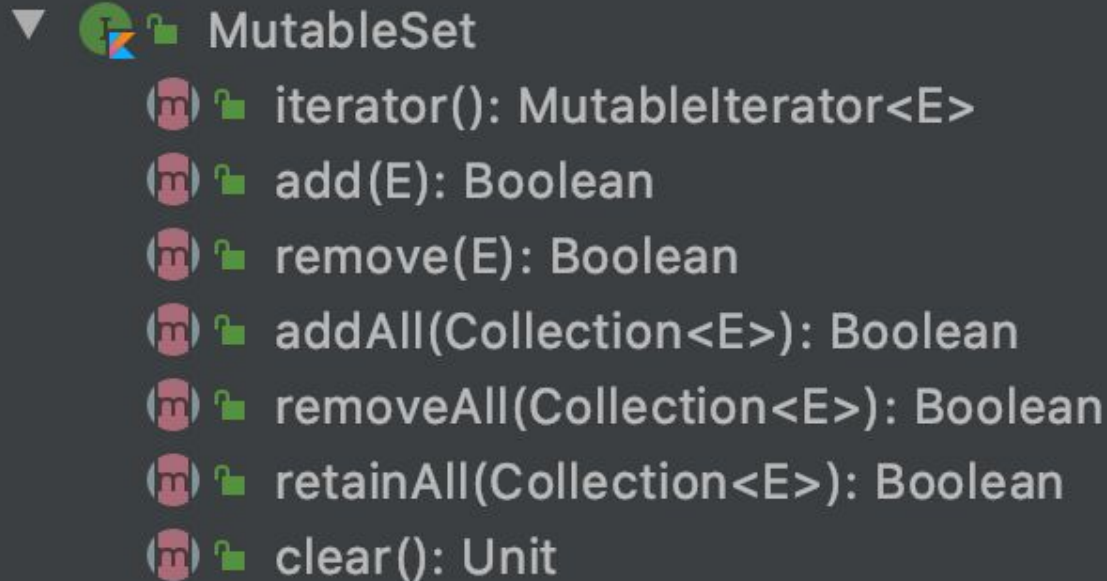
Code Program : Set (1)

```
3  class Person(val name: String) {  
4      override fun hashCode(): Int {  
5          return name.hashCode()  
6      }  
7  
8      override fun equals(other: Any?): Boolean = when (other) {  
9          is Person -> name == other.name  
10         else -> false  
11     }  
12 }  
13
```

















Kode Program : Set (2)

```
13
14 ▶ fun main() {
15     val set: Set<Person> = setOf(
16         Person("Eko"), Person("Kurniawan"), Person("Eko"), Person("Khannedy")
17     )
18
19     println(set.size)
20     println(set.contains(Person("Eko")))
21 }
22
```

Function di Mutable Set



A screenshot of a Scala IDE snippet showing the `MutableSet` class and its methods. The snippet is displayed on a dark background. At the top, there is a small icon of a green circle with a white 'I' and a green lock, followed by the text `MutableSet`. Below this, there are seven method signatures, each preceded by a small icon of a pink circle with a white 'm' and a green lock. The methods are: `iterator(): MutableIterator<E>`, `add(E): Boolean`, `remove(E): Boolean`, `addAll(Collection<E>): Boolean`, `removeAll(Collection<E>): Boolean`, `retainAll(Collection<E>): Boolean`, and `clear(): Unit`.

```
▼   MutableSet  
    iterator(): MutableIterator<E>  
    add(E): Boolean  
    remove(E): Boolean  
    addAll(Collection<E>): Boolean  
    removeAll(Collection<E>): Boolean  
    retainAll(Collection<E>): Boolean  
    clear(): Unit
```

Code Program : Mutable Set

```
3 ▶ fun main() {  
4     val mutableSet: MutableSet<Person> = mutableSetOf()  
5     mutableSet.add(Person("Eko"))  
6     mutableSet.add(Person("Kurniawan"))  
7     mutableSet.add(Person("Khannedy"))  
8     mutableSet.add(Person("Eko"))  
9  
10    println(mutableSet.size)  
11    println(mutableSet.contains(Person("Eko")))  
12 }
```

—

Pair



Pair

- Sebelum kita bahas collection Map, ada baiknya kita mengenal Pair
- Pair adalah representasi dari data yang berpasangan
- Di Map, Pair digunakan sebagai data Map ketika pertama kali Map dibuat

Data di Pair

```
22  * @property first First value.
23  * @property second Second value.
24  * @constructor Creates a new instance of Pair.
25  */
26  public data class Pair<out A, out B>(
27      public val first: A,
28      public val second: B
29  ) : Serializable {
30
31      /**
32       * Returns string representation of the [Pair] including its [first] and [second]
33       */
```




Code Program : Pair

```
3 ▶ fun main() {  
4     val pair = Pair("Eko", "Kurniawan")  
5     println(pair.first)  
6     println(pair.second)  
7  
8     val pair2: Pair<String, String> = "Eko" to "Kurniawan"  
9     println(pair2.first)  
10    println(pair2.second)  
11 }  
12  
13
```

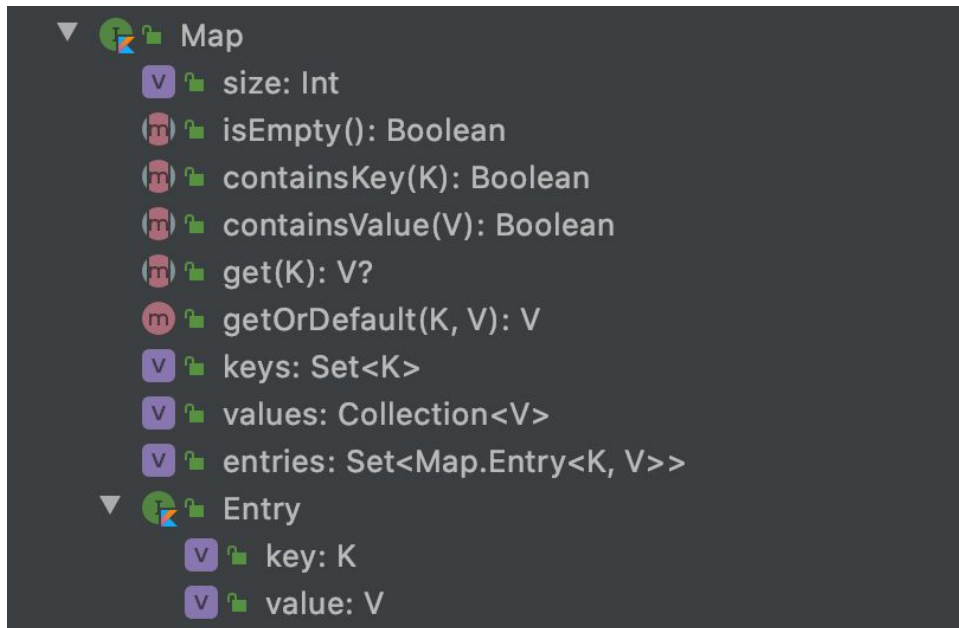
Map



Map

- Map (atau dictionary), adalah collection yang berbentuk key-value, dimana key berperan sebagai index.
- Key harus unik, jika kita menggunakan Key yang sama, maka data sebelumnya akan di replace oleh data yang baru
- Tipe data key bisa menggunakan tipe data apapun sesuai yang kita inginkan

Function di Map



Code Program : Map

```
4  val map: Map<String, String> = mapOf(  
5      "a" to "Eko",  
6      "b" to "Kurniawan",  
7      "c" to "Khannedy"  
8  )  
9  
10     println(map.size)  
11     println(map["a"])  
12     println(map["b"])  
13     println(map["c"])  
14 }
```

Function di Mutable Map

```
▼ MutableMap
  (m) put(K, V): V?
  (m) remove(K): V?
  (m) remove(K, V): Boolean
  (m) putAll(Map<out K, V>): Unit
  (m) clear(): Unit
  (v) keys: MutableSet<K>
  (v) values: MutableCollection<V>
  (v) entries: MutableSet<MutableMap.MutableEntry<K, V>>
  ▼ MutableEntry
    (m) setValue(V): V
```

Code Program : Mutable Map

```
3  ▶ fun main() {  
4      val mutableMap: MutableMap<String, String> = mutableMapOf()  
5      mutableMap["a"] = "Eko"  
6      mutableMap["b"] = "Kurniawna"  
7      mutableMap["c"] = "Khannedy"  
8  
9      println(mutableMap["a"])  
10     println(mutableMap["b"])  
11     println(mutableMap["c"])  
12 }
```



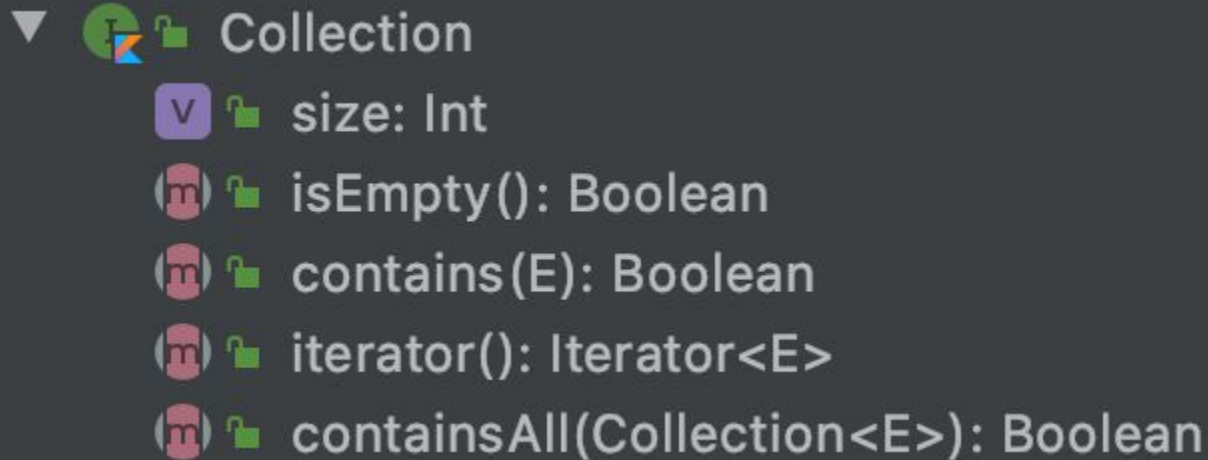
Collection



Collection

- Semua data collection adalah turunan dari interface Collection
- Collection digunakan sebagai general operation terhadap struktur data collection, seperti membaca dan menulis data di collection

Function di Collection



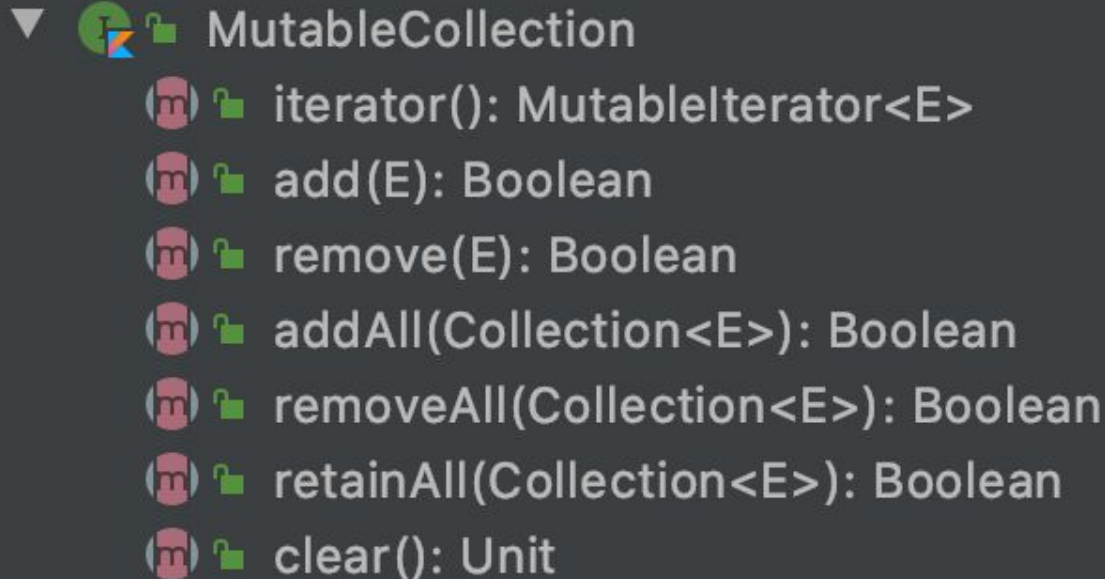
```
▼ Collection  
  V size: Int  
  (m) isEmpty(): Boolean  
  (m) contains(E): Boolean  
  (m) iterator(): Iterator<E>  
  (m) containsAll(Collection<E>): Boolean
```

A screenshot of a Scala IDE snippet showing the `Collection` trait and its methods. The snippet is displayed on a dark background. The `Collection` trait is expanded, showing its methods: `size: Int`, `isEmpty(): Boolean`, `contains(E): Boolean`, `iterator(): Iterator<E>`, and `containsAll(Collection<E>): Boolean`. Each method is preceded by a small icon: a green circle with a white 'V' for the `size` property, and a red circle with a white 'm' for the methods. A small green lock icon is also present next to each method name.

Kode Program : Collection

```
3 fun <T> displayCollection(collection: Collection<T>) {  
4     for (item in collection) println(item)  
5 }  
6  
7 fun main() {  
8     displayCollection(listOf("Eko", "Kurniawan", "Khannedy"))  
9     displayCollection(setOf("Eko", "Kurniawan", "Khannedy"))  
10    displayCollection(mapOf("a" to "Eko").entries)  
11    displayCollection(mapOf("a" to "Eko")) // Error, Map bukan Collection  
12 }
```

Function di Mutable Collection



```
▼ MutableCollection  
  iterator(): MutableIterator<E>  
  add(E): Boolean  
  remove(E): Boolean  
  addAll(Collection<E>): Boolean  
  removeAll(Collection<E>): Boolean  
  retainAll(Collection<E>): Boolean  
  clear(): Unit
```

A screenshot of a Scala IDE snippet showing the methods of the `MutableCollection` trait. The snippet is displayed on a dark background with a light gray border. It includes a dropdown arrow and a Scala logo icon. The methods listed are `iterator(): MutableIterator<E>`, `add(E): Boolean`, `remove(E): Boolean`, `addAll(Collection<E>): Boolean`, `removeAll(Collection<E>): Boolean`, `retainAll(Collection<E>): Boolean`, and `clear(): Unit`. Each method is preceded by a small icon representing a mutable collection (a circle with an 'm') and a lock icon.

Code Program : Immutable Collection

```
3 fun <T> displayMutableCollection(collection: MutableCollection<T>) {  
4     for (item in collection) println(item)  
5 }  
6  
7 fun main() {  
8     displayMutableCollection(mutableListOf("Eko", "Kurniawan", "Khannedy"))  
9     displayMutableCollection(mutableSetOf("Eko", "Kurniawan", "Khannedy"))  
10    displayMutableCollection(mutableMapOf("a" to "Eko").entries)  
11 }
```




Iterable









Iterable

- Iterable adalah superclass dari Collection interface
- Iterable adalah general operation untuk melakukan iterasi seluruh data di collection, atau menghapus data di collection

Function di Iterable





▼   Iterable
   iterator(): Iterator<T>





▼   Iterator
   next(): T
   hasNext(): Boolean

Code Program : Iterable

```
3 fun <T> displayIterable(iterable: Iterable<T>) {  
4     val iterator = iterable.iterator()  
5     while (iterator.hasNext()) println(iterator.next())  
6 }  
7  
8 fun main() {  
9     displayIterable(listOf("Eko", "Kurniawan", "Khannedy"))  
10    displayIterable(setOf("Eko", "Kurniawan", "Khannedy"))  
11 }
```

Function di Mutable Iterable

▼   MutableIterable
   iterator(): MutableIterator<T>

▼   MutableIterator
   remove(): Unit

Code Program : Mutable Iterable

```
3 fun <T> displayMutableIterable(iterable: MutableIterable<T>) {  
4     val iterator = iterable.iterator()  
5     while (iterator.hasNext()) println(iterator.next())  
6 }  
7  
8 fun main() {  
9     displayMutableIterable(mutableListOf("Eko", "Kurniawan", "Khannedy"))  
10    displayMutableIterable(mutableSetOf("Eko", "Kurniawan", "Khannedy"))  
11 }  
12  
13 |
```

—


Iterator















Iterator

- Sebelumnya kita sudah bahas tentang Iterator di Iterable dan MutableIterator di MutableIterable
- Iterator juga memiliki child interface, yaitu ListIterator dan MutableListIterator
- ListIterator digunakan oleh List, dan MutableListIterator digunakan oleh MutableList
- Set tetap menggunakan Iterator
- Sedangkan Map tidak mendukung Iterator

Function di ListIterator


▼  ListIterator

-   next(): T
-   hasNext(): Boolean
-   hasPrevious(): Boolean
-   previous(): T
-   nextIndex(): Int
-   previousIndex(): Int

Code Program : ListIterator

```
3  fun <T> displayListIterator(listIterator: ListIterator<T>) {  
4      while (listIterator.hasNext()) println(listIterator.next())  
5      while (listIterator.hasPrevious()) println(listIterator.previous())  
6  }  
7  
8  fun main() {  
9      displayListIterator(listOf("Eko", "Kurniawan", "Khannedy").listIterator())  
10 }  
11  
12
```

Function di MutableListIterator

- ▼   MutableListIterator
 -   next(): T
 -   hasNext(): Boolean
 -   remove(): Unit
 -   set(T): Unit
 -   add(T): Unit

Code Program : MutableListIterator

```
3 fun <T> displayMutableListIterator(mutableListIterator: MutableListIterator<T>) {  
4     while (mutableListIterator.hasNext()) println(mutableListIterator.next())  
5 }  
6  
7 fun removeOddNumber(mutableListIterator: MutableListIterator<Int>) {  
8     while (mutableListIterator.hasNext()) {  
9         val item = mutableListIterator.next()  
10        if (item % 2 == 1) mutableListIterator.remove()  
11    }  
12 }  
13
```

Konversi Collection



Konversi Collection

- Di kotlin kita bisa melakukan konversi collection ke jenis collection lain secara mudah
- Misal kita ingin mengubah dari array ke list, atau dari range ke set, atau dari set ke list, bisa dilakukan dengan mudah
- Semua function dimulai dengan kata “to”, misal toList, toSet, toMutableList, dan lain-lain

Kode Program : Konversi Collection

```
2
3 ▶ fun main() {
4     val array = arrayOf(1, 2, 3, 4, 5)
5     val range = 1..100
6     val list = range.toList()
7     val mutableList = array.toMutableList()
8     val set = list.toSet()
9     val mutableSet = list.toMutableSet()
10    val sortedSet = list.toSortedSet()
11 }
```

Collection Operations



Collection Operations

- Kotlin sudah menyediakan berbagai macam operation yang bisa kita lakukan di collection
- Sebelumnya kita sudah bahas banyak sekali operasi yang tersedia sebagai function bawaan collection
- Kotlin juga menyediakan operations tambahan berubah extension function
- Collection operations berupa extension function selalu menghasilkan collection baru, tanpa mengubah data di collection aslinya
- Oleh karena itu, collection operations di Kotlin sangat aman ketika dilakukan karena tidak akan mengubah collection aslinya



Jenis Operations

- Transformations, melakukan transformasi terhadap collection
- Filtering, melakukan filtering atau seleksi terhadap collection
- plus and minus operators, operasi + dan - terhadap collection
- Grouping, melakukan pemisahan dalam group terhadap collection
- Retrieving collection parts, mengambil sebagian data di collection
- Retrieving single elements, mengambil satu data di collection
- Ordering, melakukan pengurutan data di collection
- Aggregate operations, melakukan agregasi terhadap data yang ada di collection

Code Program : Collection Operations

```
3 ▶ fun main() {  
4     listOf("Eko", "Kurniawan").forEach { value -> println(value) }  
5     listOf("Eko", "Kurniawan").forEachIndexed { index, value ->  
6         println("$index = $value")  
7     }  
8 }  
9 |
```

Mapping



Transformations : Mapping

- Transformasi mapping digunakan untuk mengubah data ke data lain.
- Mapping akan membutuhkan lambda function sebagai parameter, dan akan mengembalikan collection hasil result lambda nya
- Urutan collection baru hasil dari mapping adalah sama dengan urutan collection aslinya.



Transformations : Mapping Operations

Mapping Operations	Keterangan
<code>map(transform: (T) -> R)</code>	Mengubah collection menjadi collection baru sesuai lambda transform
<code>mapIndexed(transform: (index, T) -> R)</code>	Mengubah collection menjadi collection baru, dengan tambahan parameter index
<code>mapNotNull(transform : (T) -> R)</code>	Sama seperti <code>map()</code> , namun menghiraukan hasil null
<code>mapIndexedNotNull(transform : (index, T) -> R)</code>	Sama seperti <code>mapIndexed()</code> , namun menghiraukan nilai null

Code Program : Transformations Mapping

```
3  ▶ fun main() {  
4      val list1: List<String> = listOf("Eko", "Kurniawan", "Khannedy")  
5      val list2: List<String> = list1.map { item -> item.toUpperCase() }  
6      val list3: List<Int> = list1.map { item -> item.length }  
7      val list4: List<String> = list1.mapNotNull { value ->  
8          if (value == "Eko") value else null  
9      }  
10 }
```



Mapping Operations untuk Map

Mapping Operations	Keterangan
<code>mapKeys(transform: (Entry<K,V>) -> R)</code>	Mengubah map menjadi map baru dengan nilai key baru sesuai dengan lambda transform
<code>mapValues(transform: (Entry<K,V>) -> R)</code>	Mengubah map menjadi map baru dengan nilai value baru sesuai dengan lambda transform

Code Program : Mapping Map

```
11  val map1: Map<Int, String> = mapOf(  
12      1 to "Eko",  
13      2 to "Kurniawan",  
14      3 to "Khannedy"  
15  )  
16  val map2: Map<Int, String> = map1.mapKeys { it.key * 10 }  
17  val map3: Map<Int, String> = map1.mapValues { it.value.toUpperCase() }  
18  
19  println(map2)  
20  println(map3)  
21
```

Ziping



Transformations : Zipping

- Zipping adalah transformasi yang digunakan untuk menggabungkan dua buah collection.



Transformations : Zipping Operations

Mapping Operations	Keterangan
<code>zip(collection)</code>	Menggabungkan dua collection, sehingga menghasilkan <code>Pair<T1, T2></code>
<code>zip(collection, transform: (T1, T2) -> R)</code>	Menggabungkan dua collection dengan lambda transform sehingga menghasilkan collection baru dari result lambda transform

Kode Program : Transformations Zipping

```
3  ▶ fun main() {  
4      val list1 = listOf("Eko", "Kurniawan", "Khannedy")  
5      val list2 = listOf("Programmer", "Zaman", "Now")  
6  
7      val list3: List<Pair<String, String>> = list1.zip(list2)  
8      val list4: List<String> = list1.zip(list2) { item1, item2 ->  
9          item1 + item2  
10     }  
11  
12     println(list3)  
13     println(list4)
```



Transformations : Unzip Operations

Mapping Operations	Keterangan
unzip()	Mengubah collection Pair<T1, T2> menjadi dua collection, collection T1 dan collection T2

Kode Program : Transformations Unzip

```
14
15  val list: List<Pair<String, String>> = listOf(
16      "Eko" to "Programmer",
17      "Kurniawan" to "Zaman",
18      "Khannedy" to "Now"
19  )
20
21  val pair: Pair<List<String>, List<String>> = list.unzip()
22 }
```

Association



Transformations : Association

- Association adalah operasi transformasi dari collection menjadi map
- Dalam beberapa operasi association, element hasil bisa menjadi key atau value



Transformations : Association Operation

Association Operation	Keterangan
<code>associate((T) -> Pair<K, V>)</code>	Merubah collection T menjadi Map<K, V>
<code>associateWith(T -> V)</code>	Merubah collection T menjadi Map<T, V>
<code>associateBy(T -> K)</code>	Merubah collection T menjadi Map<K, T>

Code Program : Transformations Association

```
2
3 ▶ fun main() {
4     val list1 = listOf("Eko", "Kurniawan", "Khannedy")
5     val map1: Map<String, Int> = list1.associate { Pair(it, it.length) }
6     val map2: Map<String, Int> = list1.associateWith { it.length }
7     val map3: Map<Int, String> = list1.associateBy { it.length }
8
9     println(map1)
10    println(map2)
11    println(map3)
12 }
```

Flattening



Transformations : Flattening

- Flattening adalah operasi transformasi untuk menjadikan nested collection menjadi flat



Transformations : Flattening Operation

Flattering Operation	Keterangan
flatten	Mengubah nested collection menjadi flat collection
flatMap((T) -> Iterable<R>)	Mengubah collection T menjadi flat collection R

Code Program : Transformations Flattening (1)

```
2
3 ▶ fun main() {
4     val list1: List<List<String>> = listOf(
5         listOf("Eko", "Kurniawan", "Khannedy"),
6         listOf("Programmer", "Zaman", "Now")
7     )
8     val list2: List<String> = list1.flatten()
9 }
```

Code Program : Transformations Flattening (2)

```
2
3  class Member(val name: String, val hobbies: List<String>)
4
5  fun main() {
6      val members: List<Member> = listOf(
7          Member("Eko", listOf("Reading", "Gaming")),
8          Member("Budi", listOf("Sport", "Traveling"))
9      )
10     val hobbies: List<String> = members.flatMap { it.hobbies }
```

String Representation



Transformations : String Representation

- String Representations adalah operasi transformasi untuk mengubah collection menjadi string



String Representation Operation

String Representation Operation	Keterangan
<code>joinToString(separator, prefix, suffix)</code>	Mengubah collection menjadi string dengan separator, prefix dan suffix
<code>joinToString(separator, prefix, suffix, transform)</code>	Mengubah collection menjadi string dengan separator, prefix dan suffix, namun akan diubah dalam transform terlebih dahulu
<code>joinTo(Appendable, separator, prefix, suffix, transform)</code>	Sama seperti <code>joinToString</code> , namun string ditambahkan ke <code>Appendable</code>

Code Program : String Representation

```
2
3 ▶ fun main() {
4     val names = listOf("Eko", "Kurniawan", "Khannedy")
5     println(names.joinToString(" ", "|", "|"))
6     println(names.joinToString(" ", "|", "|") { string -> "Item $string" })
7
8     val appendable = StringBuilder()
9     names.joinTo(appendable, ",", "|", "|") { it }
10    println(appendable.toString())
11 }
```

Filtering



Filtering by Predicate

- Filtering adalah salah satu operasi collection yang sangat populer
- Di kotlin, filtering bisa menggunakan predicate berupa lambda yang berisi kondisi filter nya
- Jika return dari predicate tersebut true, maka data di terima, jika false maka data dibuang



Filtering by Predicate Operation

Filtering Operation	Keterangan
<code>filter((T) -> Boolean)</code>	Melakukan filtering terhadap collection
<code>filterIndexed((index, T) -> Boolean)</code>	Melakukan filtering terhadap collection, dengan tambahan informasi index
<code>filterNot((T) -> Boolean)</code>	Kebalikan dari <code>filter((T) -> Boolean)</code>
<code>filterIsInstance<T>()</code>	Mengambil hanya data instant dari T
<code>filterNotNull()</code>	Mengambil hanya data yang tidak null

Code Program : Filtering by Predicate

```
2
3 ▶ fun main() {
4     val list1 = listOf("Eko", "Kurniawan", "Khannedy")
5     val list2 = list1.filter { it.length > 5 }
6     val list3 = list1.filterIndexed { index, value -> index % 2 == 0 }
7
8     val list4: List<Any?> = listOf(null, 1, "Eko", "Kurniawna", 2, null)
9     val list5: List<String> = list4.filterIsInstance<String>()
10    val list6: List<Any> = list4.filterNotNull()
11
12    println(list1)
```

Partitioning



Filtering : Partitioning

- Pada filtering by predicate, kita akan menghilangkan data yang tidak masuk kondisi filter
- Dengan filtering partition, kita akan membagi collection menjadi dua, dimana collection pertama yang masuk filtering, dan collection kedua yang tidak masuk filtering



Filtering : Partitioning Operation

Partitioning Operation	Keterangan
<code>partition((T) -> Boolean): Pair<List<T>, List<T>></code>	Mempartisi collection dengan filtering sehingga menghasilkan Pair collection T yang masuk filter, dan collection T yang tidak masuk filter

Code Program : Filtering Partitioning

```
2
3 ▶ fun main() {
4     val list1 = listOf("Eko", "Kurniawna", "Khannedy")
5     val (listMatch, listNotMatch) = list1.partition { it.length > 5 }
6     println(list1)
7     println(listMatch)
8     println(listNotMatch)
9 }
```

Testing



Filtering : Testing

- Filtering testing adalah operation yang bisa digunakan untuk mengecek isi data dari collection
- Hasil dari filtering testing adalah boolean, dimana true jika sesuai kondisi, dan false jika tidak sesuai kondisi



Filtering : Testing Operation

Testing Operation	Keterangan
<code>any((T) -> Boolean) : Boolean</code>	Mengecek apakah minimal ada satu data sesuai dengan kondisi
<code>none((T) -> Boolean) : Boolean</code>	Mengecek apakah tidak ada satupun data sesuai dengan kondisi
<code>all((T) -> Boolean) : Boolean</code>	Mengecek apakah semua data sesuai dengan kondisi
<code>any()</code>	Apakah collection memiliki data
<code>none()</code>	Apakah collection tidak punya data

Code Program : Testing Operation

```
2
3 ▶ fun main() {
4     val list = listOf("Eko", "Kurniawan", "Khannedy")
5     println(list.any { it.length > 5 })
6     println(list.none { it.length > 5 })
7     println(list.all { it.length > 5 })
8     println(list.any())
9     println(list.none())
10 }
```

Plus & Minus Operators



Plus & Minus Operators

- Di kotlin, ada operator extension function plus dan minus untuk collection
- Parameter pertama adalah collection, dan parameter selanjutnya bisa collection atau element, dan akan menghasilkan collection baru



Plus & Minus Operators

Plus & Minus Operators	Keterangan
plus(element)	Membuat collection baru dengan menambahkan element
plus(collection)	Membuat collection baru dengan menambahkan collection
minus(element)	Membuat collection baru dengan menghapus element
minus(collection)	Membuat collection baru dengan menghapus collection

Kode Program : Plus & Minus Operators

```
2
3 ▶ fun main() {
4     val list1 = listOf("Eko", "Kurniawan", "Khannedy")
5     val list2 = list1 + "Programmer"
6     val list3 = list1 + listOf("Programmer", "Zaman", "Now")
7     val list4 = list1 - "Eko"
8     val list5 = list1 - listOf("Kurniawan", "Khannedy")
9
10    val map1 = mapOf("a" to "Eko", "b" to "Kurniawan")
11    val map2 = map1 + ("c" to "Khannedy")
12    val map3 = map1 - "a"
```

Grouping



Grouping

- Grouping adalah operasi untuk meng-grup kan element-element yang ada di collection.



Grouping Operators

Grouping Operator	Keterangan
<code>groupBy((T) -> K) : Map<K, List<T>></code>	Menggabungkan collection menjadi group <code>Map<K, List<T>></code>
<code>groupingBy((T) -> K) : Grouping<K, T></code>	Menggabungkan collection menjadi object <code>Grouping<K, T></code> Grouping interface akan dibahas lebih detail di materi terpisah

Code Program : Grouping

```
2
3 ▶ fun main() {
4     val list1 = listOf("a", "a", "b", "b", "c", "c")
5     val map: Map<String, List<String>> = list1.groupBy { it }
6     val grouping: Grouping<String, String> = list1.groupingBy { it }
7
8     println(map)
9     println(grouping)
10 }
```

Slicing



Slice Operator

- Slice adalah operator yang digunakan untuk mengambil sebagian element yang ada di collection yang sesuai dengan parameter range



Slice Operators

Slice Operator	Keterangan
<code>slice(range)</code>	Mengambil sebagian element yang ada di collection sesuai dengan index range

Code Program : Slice

```
3  ▶ fun main() {  
4      val list1 = (0..100).toList()  
5      val list2 = list1.slice(0..50)  
6      val list3 = list1.slice(0..100 step 2)  
7      val list4 = list1.slice(100 downTo 0 step 2)  
8  
9      println(list1)  
10     println(list2)  
11     println(list3)  
12     println(list4)  
13 }
```

—

Take & Drop



Take & Drop Operators

- Untuk mengambil sebagian element di collection, selain menggunakan slice, kita juga bisa menggunakan operator take dan drop



Take Operators

Slice Operator	Keterangan
<code>take(n)</code>	Mengambil collection diawal sejumlah n
<code>takeLast(n)</code>	Mengambil collection diakhir sejumlah n
<code>takeWhile((T) -> Boolean)</code>	Mengambil element dari awal selama kondisi bernilai true, jika false, maka berhenti
<code>takeWhileLast((T) -> Boolean)</code>	Sama seperti <code>takeWhile</code> , namun dimulai dari element akhir



Drop Operators

Slice Operator	Keterangan
<code>drop(n)</code>	Menghapus collection diawal sejumlah n
<code>dropLast(n)</code>	Menghapus collection diakhir sejumlah n
<code>dropWhile((T) -> Boolean)</code>	Menghapus element dari awal selama kondisi bernilai true, jika false, maka berhenti
<code>dropWhileLast((T) -> Boolean)</code>	Sama seperti <code>dropWhile</code> , namun dimulai dari element akhir



Code Program : Take & Drop Operators

```
4  val chars = ('a'..'z').toList()
5
6  println(chars.take(3)) // [a, b, c]
7  println(chars.takeWhile { it < 'f' }) // [a, b, c, d, e]
8  println(chars.takeLast(2)) // [y, z]
9  println(chars.takeLastWhile { it > 'w' }) // [x, y, z]
10
11 println(chars.drop(23)) // [x, y, z]
12 println(chars.dropLast(23)) // [a, b, c]
13 println(chars.dropWhile { it < 'x' }) // [x, y, z]
14 println(chars.dropLastWhile { it > 'c' }) // [a, b, c]
```

Chunked



Chunked Operator

- Chunked adalah operasi untuk memotong collection menjadi beberapa collection



Chunked Operator

Chunked Operator	Keterangan
<code>chunked(n) : List<List<T>></code>	Memotong collection menjadi beberapa collection dengan jumlah data n
<code>chunked(n, (List<T>) -> R) : List<R></code>	Memotong collection menjadi beberapa collection dengan jumlah data n, lalu melakukan transformasi List<T> menjadi R, sehingga menghasilkan List<R>

Code Program : Chunked

```
2  
3 ▶ fun main() {  
4     val range = (1..100).toList()  
5     val list1 = range.chunked(10)  
6     val list2 = range.chunked(10) { list: List<Int> ->  
7         var total = 0  
8         for (number in list) {  
9             total += number  
10        }  
11        total ^chunked  
12    }  
13
```

Windowed



Windowed Operation

- Windowed adalah operasi untuk mendapatkan semua kemungkinan collection sesuai dengan range yang telah ditentukan



Windowed Operator

Windowed Operator	Keterangan
<code>windowed(size, step, partialWindow)</code>	Mengambil semua kemungkinan collection sebesar size, dilanjut dengan step, dan diakhiri dengan partialWindow
<code>windowed(size, step, partialWindow, transform)</code>	Sama seperti <code>windowed()</code> sebelumnya, namun hasil collection di transform

Code Program : Windowed

```
fun main() {  
    val range = (1..5).toList()  
    println(range.windowed(3, 1, false)) // [[1, 2, 3], [2, 3, 4], [3, 4, 5]]  
    println(range.windowed(3, 1, false) { it.size }) // [3, 3, 3]  
    println(range.windowed(3, 1, true)) // [[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5], [5]]  
    println(range.windowed(3, 1, true) { it.size }) // [3, 3, 3, 2, 1]  
    println(range.windowed(3, 2, false)) // [[1, 2, 3], [3, 4, 5]]  
    println(range.windowed(3, 2, true)) // [[1, 2, 3], [3, 4, 5], [5]]  
}
```

Retrieve by Position



Retrieve Single Elements by Position

- Di kotlin kita bisa mengambil satu element di collection menggunakan posisi



Retrieve Single Element by Position

Retrieve Single Element Operator	Keterangan
<code>elementAt(index)</code>	Mengambil element pada posisi index.
<code>first()</code>	Mengambil element pertama
<code>last()</code>	Mengambil element terakhir
<code>elementAtOrNull(index)</code>	Mengambil element pada posisi index. Null jika tidak ada index
<code>elementAtOrElse(index, defaultValue)</code>	Mengambil element pada posisi index. Atau default value jika tidak ada index



Code Program : Retrieve Single by Position

```
fun main() {  
    val list = listOf("Eko", "Kurniawan", "Khannedy")  
    println(list.first())  
    println(list.last())  
    println(list.elementAt(1))  
    println(list.elementAtOrNull(10))  
    println(list.elementAtOrElse(10) { "" })  
}
```

Retrieve by Condition



Retrieve Single Elements by Condition

- Selain mengambil satu element di collection berdasarkan posisi, kita juga bisa mengambil single element berdasarkan kondisi



Retrieve Single Element by Position

Retrieve Single Element Operator	Keterangan
<code>first((T) -> Boolean)</code>	Mengambil element pertama yang sesuai kondisi
<code>last((T) -> Boolean)</code>	Mengambil element terakhir yang sesuai dengan kondisi
<code>firstOrNull((T) -> Boolean) / find()</code>	Mengambil element pertama yang sesuai kondisi, atau null jika tidak ada
<code>lastOrNull((T) -> Boolean) / findLast()</code>	Mengambil element terakhir yang sesuai kondisi, atau null jika tidak ada

Code Program : Single Element by Condition

```
2
3 ▶ fun main() {
4     val range = (1..20).toList()
5     println(range.first { it > 10 }) // 11
6     println(range.last { it > 10 }) // 20
7     println(range.firstOrNull { it > 50 }) // null
8     println(range.lastOrNull { it > 50 }) // null
9     println(range.find { it > 50 }) // null
10    println(range.findLast { it > 50 }) // null
11 }
12
13
```

Retrieve by Random



Retrieve Single Element by Random

- Kotlin juga mendukung operasi untuk mengambil element secara random di dalam collection



Retrieve Single Element by Position

Retrieve Single Element Operator	Keterangan
<code>random()</code>	Mengambil element secara random



Code Program : Single Element by Random

```
2
3 ▶ fun main() {
4     val range = (1..100).toList()
5     println(range.random())
6     println(range.random())
7     println(range.random())
8     println(range.random())
9     }
```

Checking Existence



Checking Existence

- Kotlin mendukung operasi untuk mengecek apakah dalam collection terdapat data yang kita cari



Checking Existence Operator

Checking Existence Operator	Keterangan
<code>contains(element) : Boolean</code>	Mengecek apakah terdapat element di collection
<code>containsAll(Collection<T>) : Boolean</code>	Mengecek apakah terdapat element collection T di collection
<code>isEmpty() : Boolean</code>	Mengecek apakah collection kosong
<code>isEmpty() : Boolean</code>	Mengecek apakah collection tidak kosong



Code Program : Checking Existence

```
2
3 ▶ fun main() {
4     val range = (1..10).toList()
5     println(range.contains(5))
6     println(range.containsAll(listOf(3, 4, 5)))
7     println(range.isEmpty())
8     println(range.isNotEmpty())
9     }
10
11
```

Ordering



Ordering : Natural

- Mengurutkan data sudah menjadi salah satu kebiasaan yang kita lakukan saat membuat program
- Kotlin mendukung operasi untuk mengurutkan collection
- Pada materi Kotlin Generic kita sudah belajar tentang interface Comparable, yaitu interface yang digunakan untuk melakukan perbandingan dua buah object, yang biasanya digunakan untuk mengurutkan data
- Secara default tipe data yang ada di kotlin sudah memiliki Comparable (seperti Number, Char atau String). Oleh karena itu kita tidak perlu lagi membuat Comparable sendiri untuk mengurutkan data tersebut



Ordering : Natural Operator

Ordering Natural Operator	Keterangan
<code>sorted()</code>	Mengurutkan collection secara ascending
<code>sortedDescending()</code>	Mengurutkan collection secara descending



Code Program : Ordering Natural

```
2
3 ▶ fun main() {
4     val numbers = listOf(1, 4, 2, 4, 6, 3, 5)
5     println(numbers.sorted()) // [1, 2, 3, 4, 4, 5, 6]
6     println(numbers.sortedDescending()) // [6, 5, 4, 4, 3, 2, 1]
7 }
```

Custom Order



Ordering : Custom

- Selain menggunakan comparable bawaan kotlin, kita juga bisa membuat comparable sendiri
- Ini cocok untuk mengurutkan data yang kita buat sendiri



Ordering : Custom Operator

Ordering Natural Operator	Keterangan
<code>sortedBy(selector: (T) -> R)</code>	Mengurutkan collection secara ascending menggunakan selector
<code>sortedByDescending(selector: (T) -> R)</code>	Mengurutkan collection secara descending menggunakan selector
<code>sortedWith(Comparator<T>)</code>	Mengurutkan collection dengan <code>Comparator<T></code>

Code Program : Ordering Custom

```
3      data class Fruit(val name: String, val quantity: Int)
4
5  ▶ fun main() {
6      val fruits = listOf(Fruit("Apple", 10), Fruit("Orange", 5))
7      println(fruits.sortedBy { it.quantity })
8      println(fruits.sortedByDescending { it.quantity })
9      println(fruits.sortedWith(compareBy { it.quantity }))
10     println(fruits.sortedWith(compareByDescending { it.quantity }))
11     println(fruits.sortedWith(Comparator { a, b ->
12         a.quantity.compareTo(b.quantity)
13     }))
```

Reverse Order



Ordering : Reverse

- Di Kotlin, kita bisa mendapatkan collection dengan urutan data terbalik dari data aslinya



Ordering : Reverse Operator

Ordering Reverse Operator	Keterangan
<code>reversed()</code>	Membuat collection baru dengan urutan terbalik dari collection aslinya
<code>asReversed()</code>	Membuat view terhadap collection dengan urutan terbalik, jika collection aslinya berubah, view pun akan ikut berubah

Kode Program : Ordering Reverse

```
2
3 ▶ fun main() {
4     val list1 = mutableListOf("Eko", "Kurniawan", "Khannedy")
5     val listReverse = list1.reversed()
6     val listAsReverse = list1.asReversed()
7     list1.add("Programmer")
8     println(listReverse) // [Khannedy, Kurniawan, Eko]
9     println(listAsReverse) // [Programmer, Khannedy, Kurniawan, Eko]
10 }
```

Random Order



Ordering : Random

- Di Kotlin, kita juga bisa mengacak isi element di collection



Ordering : Random Operator

Ordering Random Operator	Keterangan
shuffled()	Membuat collection baru dengan posisi element random



Code Program : Ordering Random

```
2  
3 ▶ fun main() {  
4     val numbers = (1..10).toList()  
5     println(numbers.shuffled())  
6     println(numbers.shuffled())  
7     println(numbers.shuffled())  
8     println(numbers.shuffled())  
9 }
```

Aggregate



Aggregate

- Kotlin memiliki berbagai operasi untuk melakukan aggregate



Aggregate Operator (1)

Aggregate Operator	Keterangan
max() dan min()	Mengambil nilai maksimal dan minimum
average()	Mengambil nilai rata-rata
sum()	Mengambil nilai jumlah seluruh element
count()	Mengambil berapa banyak element



Aggregate Operator (2)

Aggregate Operator	Keterangan
maxBy(selector) dan minBy(selector)	Mengambil nilai maksimal dan minimum
maxWith(Comparator) dan minWith(Comparator)	Mengambil nilai maksimal dan minimum sesuai comparator
sumBy(selector)	Mengambil nilai jumlah seluruh element sesuai selector
sumByDouble(selector)	Sama dengan sumBy() namun menghasilkan Double



Code Program : Aggregate

```
2
3 ▶ fun main() {
4     val numbers = (1..100).toList()
5     println(numbers.max())
6     println(numbers.min())
7     println(numbers.average())
8     println(numbers.sum())
9     }
```

Fold & Reduce



Fold & Reduce

- Jika kita ingin membuat aggregate secara manual, kita bisa menggunakan operator fold dan reduce
- Fold dan reduce merupakan operasi yang dilakukan pada element-element yang ada di collection secara iterasi sequential dan mengembalikan hasil.
- Hasil dari fold dan reduce akan digunakan untuk iterasi selanjutnya.



Fold & Reduce Operator (1)

Fold & Reduce Operator	Keterangan
$\text{reduce}((R, T) \rightarrow R) : R$	Pada iterasi pertama, R adalah element pertama, T adalah element selanjutnya. Iterasi selanjutnya, R adalah hasil dari iterasi sebelumnya
$\text{fold}(R, (R, T) \rightarrow R) : R$	fold sama dengan reduce, yang membedakan adalah, fold memiliki inisial value R yang bisa kita masukkan secara manual



Fold & Reduce Operator (2)

Fold & Reduce Operator	Keterangan
<code>reduceRight()</code> & <code>foldRight()</code>	Sama seperti <code>reduce()</code> dan <code>fold()</code> namun dimulai dari element akhir
<code>reduceIndexed()</code> & <code>foldIndexed()</code>	Sama seperti <code>reduce()</code> dan <code>fold()</code> namun terdapat parameter index
<code>reduceRightIndexed()</code> & <code>foldRightIndexec()</code>	Sama seperti <code>reduceIndexec()</code> dan <code>foldIndexec()</code> namun dimulai dari element akhir

Code Program : Fold & Reduce

```
3 ▶ fun main() {  
4     val numbers = (1..100).toList()  
5     val max = numbers.reduce { acc, i ->  
6         if (acc < i) i ^reduce  
7         else acc ^reduce  
8     }  
9     val sum = numbers.fold(0) { acc, i ->  
10         acc + i  
11     }  
12     println(max)  
13     println(sum)
```

Grouping Interface



Grouping Interface

- Sebelumnya kita sudah bahas tentang operasi Grouping, salah satunya adalah groupingBy yang menghasilkan object dari interface Grouping



Grouping Interface Operator

Grouping Interface Operator	Keterangan
<code>eachCount()</code>	Menghitung jumlah data di tiap group
<code>reduce()</code> dan <code>fold()</code>	Operasi <code>reduce()</code> dan <code>fold()</code> di tiap group
<code>aggregate(key: K, accumulator: R?, element: T, first: Boolean)</code>	Operasi aggregate data di tiap group, dimana jika data pertama mana parameter <code>first</code> bernilai <code>true</code> , dan accumulator akan bernilai <code>null</code>

Code Program : Grouping Interface

```
3  ▶ fun main() {  
4      val list1 = listOf("a", "a", "b", "b", "c", "c")  
5      val grouping: Grouping<String, String> = list1.groupingBy { it }  
6      println(grouping.eachCount())  
7      println(grouping.fold("") { accumulator, element -> accumulator + element })  
8      println(grouping.reduce { key, accumulator, element -> accumulator + element })  
9      println(grouping.aggregate { key, accumulator: String?, element, first ->  
10         if (first) element ^aggregate  
11         else accumulator + element ^aggregate  
12     })  
13 }
```

List Specific Operations



List Specific Operation

- Di kotlin ada banyak operasi yang khusus dibuat untuk List



List Specific Operator (1)

List Specific Operator	Keterangan
<code>getOrElse(index, (index) -> T)</code>	Mengambil element di index, jika tidak ada, mengembalikan nilai lambda
<code>getOrNull(index)</code>	Mengambil element di index, jika tidak ada, mengembalikan null
<code>subList(from, to)</code>	Mengambil semua element dari from (inclusive) ke to (exclusive)
<code>binarySearch(value)</code>	Melakukan pencarian menggunakan algoritma binary search

Code Program : List Specific Operator (1)

```
3 ▶ fun main() {  
4     val list = listOf("Eko", "Kurniawan", "Khannedy")  
5     println(list.getOrNull(1)) // Kurniawan  
6     println(list.getOrElse(5) { i -> "Ups" }) // Ups  
7     println(list.subList(0, 2)) // [Eko, Kurniawan]  
8  
9     val sortedList = list.sorted()  
10    println(sortedList.binarySearch("Eko")) // 0  
11 }
```




List Specific Operator (2)

List Specific Operator	Keterangan
<code>indexOf(value)</code>	Mengambil index element yang sama dengan value
<code>lastIndexOf(value)</code>	Mengambil index element terakhir yang sama dengan value
<code>indexOfFirst((T) -> Boolean)</code>	Mengambil index element yang sama dengan kondisi
<code>indexOfLast((T) -> Boolean)</code>	Mengambil index element terakhir yang sama dengan kondisi

Code Program : List Specific Operator (2)

```
11
12     val numbers = listOf(1, 1, 2, 2, 3, 3, 4, 4, 5, 5)
13     println(numbers.indexOf(2)) // 2
14     println(numbers.lastIndexOf(3)) // 5
15     println(numbers.indexOfFirst { it > 3 }) // 6
16     println(numbers.indexOfLast { it > 3 }) // 9
17 }
```

Set Specific Operations



Set Specific Operations

- Di kotlin juga terdapat operations yang khusus digunakan untuk Set



Set Specific Operator

Set Specific Operator	Keterangan
union(collection)	Mengembalikan semua element dari kedua set
intersect(collection)	Mengembalikan semua element yang terdapat di kedua set
substract(collection)	Mengembalikan semua element yang tidak dimiliki oleh set

Kode Program : Set Specific Operator

```
2
3 ▶ fun main() {
4     val numbers1 = (1..10).toSet()
5     val numbers2 = (6..15).toSet()
6
7     println(numbers1 union numbers2)
8     // [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
9     println(numbers1 intersect numbers2)
10    // [6, 7, 8, 9, 10]
11    println(numbers1 subtract numbers2)
12    // [1, 2, 3, 4, 5]
13 }
```

Map Specific Operations



Map Specific Operations

- Di kotlin juga terdapat banyak operasi yang khusus untuk Map



Map Specific Operator

Map Specific Operator	Keterangan
<code>getOrElse(key, (key) -> V)</code>	Mengambil value sesuai key, jika tidak ada eksekusi lambda
<code>getValue(key)</code>	Mengambil value sesuai key, atau throw exception
<code>filter((Entry<K,V>) -> Boolean)</code>	Filter sesuai kondisi
<code>filterKeys((K) -> Boolean)</code>	Filter dengan kondisi key
<code>filterValues((V) -> Boolean)</code>	Filter dengan kondisi value

Code Program : Map Specific Operations

```
2
3 ▶ fun main() {
4     val map = mapOf("a" to "Eko", "b" to "Kurniawan", "c" to "Khannedy")
5     println(map.getValue("a")) // Eko
6     println(map.getOrElse("d") { "Ups" }) // Ups
7     println(map.filter { (key, value) -> value.length > 5 }) // {b=Kurniawan, c=Khannedy}
8     println(map.filterKeys { key -> key > "b" }) // {c=Khannedy}
9     println(map.filterValues { value -> value.length > 5 }) // {b=Kurniawan, c=Khannedy}
10 }
```

Sequence



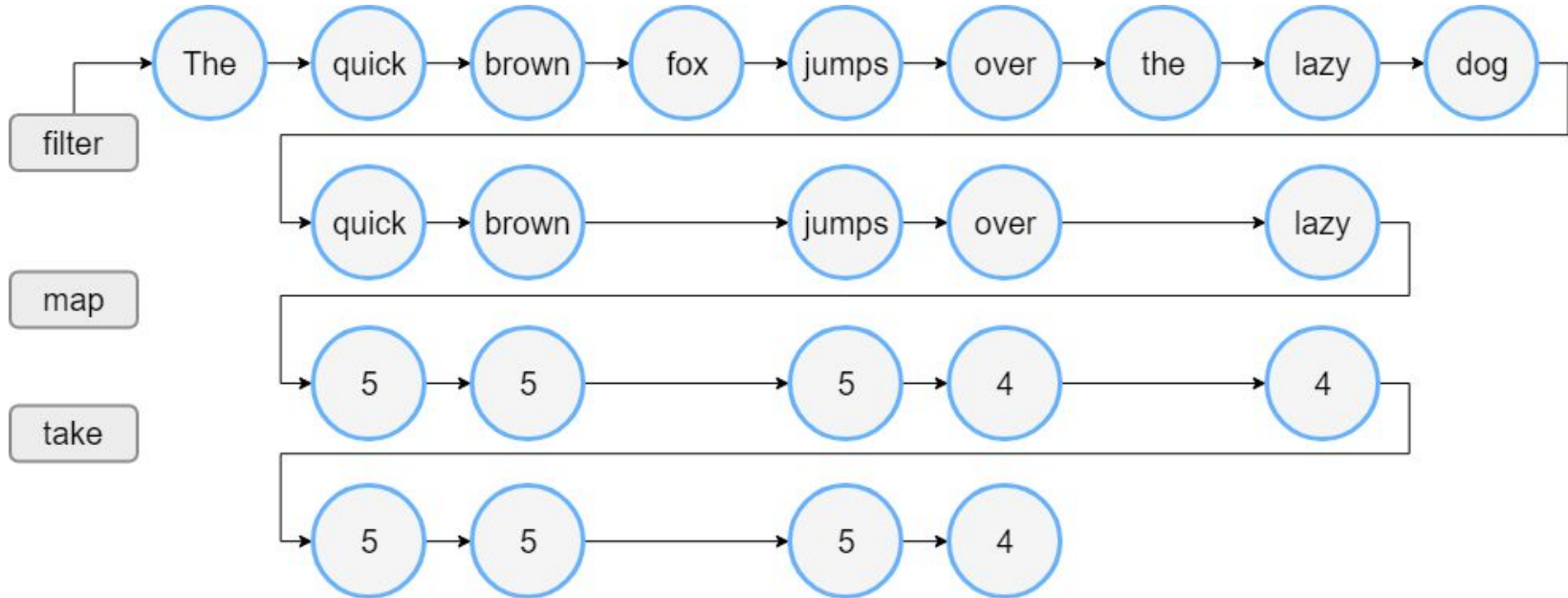
Sequence

- Selain List, Set dan Map, di Kotlin juga terdapat collection yang bernama Sequence
- Sequence sekilas sama dengan collection lain, namun berbeda cara kerjanya ketika kita menambahkan banyak operasi ke sequence
- Collection lain, ketika kita menambahkan beberapa operasi, semua akan dieksekusi secara eager. Setiap proses akan mengembalikan collection baru.
- Berbeda dengan sequence, sequence mengeksekusi semua operasi secara lazy. Hanya akan dieksekusi ketika dibutuhkan

Code Program : Bukan Sequence

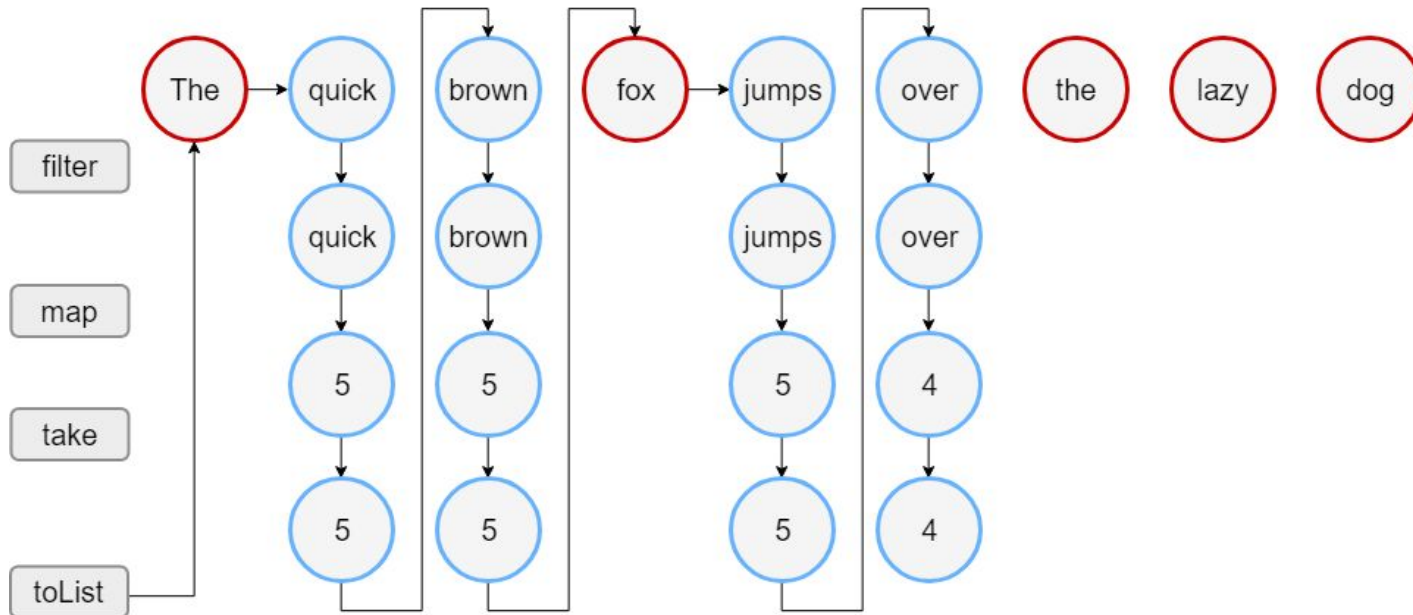
```
3 ▶ fun main() {  
4     val words = "The quick brown fox jumps over the lazy dog".split(" ")  
5     val lengthsList = words  
6         .filter { println("filter: $it"); it.length > 3 } ^filter  
7         .map { println("length: ${it.length}"); it.length } ^map  
8         .take(4)  
9  
10    println("Lengths of first 4 words longer than 3 chars:")  
11    println(lengthsList)  
12 }
```

Flow Bukan Sequence



Code Program : Sequence

```
3 fun main() {
4     val words = "The quick brown fox jumps over the lazy dog".split(" ")
5     val wordsSequence = words.asSequence()
6
7     val lengthsSequence = wordsSequence
8         .filter { println("filter: $it"); it.length > 3 } ^filter
9         .map { println("length: ${it.length}"); it.length } ^map
10        .take(4)
11
12    println("Lengths of first 4 words longer than 3 chars")
13    println(lengthsSequence.toList())
14 }
```





Kapan Menggunakan Sequence?

Gunakan ketika :

- Data collection besar
- Operasi collection kompleks dan banyak

Jangan gunakan ketika :

- Data collection tidak terlalu besar, karena ada overhead lazy object di Sequence
- Operasi yang dilakukan sederhana

Properties di Map



Properties di Map

- Salah satu hal yang biasa dilakukan dalam membuat program adalah, menyimpan value properties di Map
- Kadang ini dilakukan ketika parsing data JSON atau hal-hal dinamis lainnya.
- Dalam keadaan seperti ini, kita bisa melakukan delegate properties di Map

Kode Program : Properties di Map

```
3  class Application(map: Map<String, Any>) {
4      val name: String by map
5      val version: Int by map
6  }
7
8  fun main() {
9      val application = Application(mapOf(
10         "name" to "Kotlin App",
11         "version" to 1
12     ))
13     println(application.name)
```

Deconstructing Declarations di Map



Destructuring Declarations di Map

- Sebelumnya kita telah belajar destructuring declaration
- Map menyimpan datanya dalam `Entry<K,V>`, dimana `Entry` memiliki function `component1()` untuk mendapatkan Key, dan `component2()` untuk mendapatkan Value
- Hal ini jadi memungkinkan kita melakukan destructuring di `Entry Map`

Kode Program : Destructuring di Map

```
4  val map = mapOf(  
5      "a" to "Eko", "b" to "Kurniawan", "c" to "Khannedy"  
6  )  
7  
8  for ((key, value) in map) {  
9      println("$key : $value")  
10 }  
11  
12 map.forEach { (key, value) -> println("$key : $value") }  
13 map.forEach { entry -> println("${entry.component1()} : ${entry.component2()}") }  
14 }
```

Materi Selanjutnya



Materi Selanjutnya

- Gradle
- Kotlin Unit Test
- Kotlin Coroutine



Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/khannedy
- Twitter : twitter.com/khannedy
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com