

Taller Estructuras de Datos en Kotlin

ADSI 2469283

Santiago Sánchez Luna

El objetivo de este taller es que los aprendices sean capaces de comprender y utilizar las principales estructuras de datos en Kotlin, incluyendo arreglos, listas, conjuntos, mapas y pares.

El aprendiz deberá realizar un informe donde se evidencien los siguientes puntos:

1. Introducción a las estructuras de datos en Kotlin
   1. ¿Qué son las estructuras de datos y para qué se utilizan?

Las estructuras de datos son aquellas que nos permiten como desarrolladores, organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.

Una estructura de datos nos permite trabajar en un algo nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla

* 1. Ventajas de utilizar estructuras de datos en Kotlin

Las ventajas de utilizar las estructuras de datos son las siguientes:

* Facilidad de uso -> Kotlin tiene una gran cantidad de estructuras de datos integradas, lo que significa que no es necesario crearlas desde cero cada vez que se necesita almacenar datos.
* Seguridad de tipo -> Kotlin es un lenguaje fuertemente tipado, lo que significa que el tipo de una variable debe ser declarado explícitamente.
* Flexibilidad -> Kotlin ofrece una amplia gama de estructuras de datos, desde simples listas y mapas, hasta estructuras más complejas como árboles y grafos.
* Eficiencia -> Las estructuras de datos en Kotlin están optimizadas para ser eficientes en términos de tiempo y espacio.
* Interoperabilidad -> Kotlin es compatible con Java, lo que significa que se pueden utilizar estructuras de datos Java en Kotlin y viceversa.
  1. Diferencias entre las estructuras de datos en Kotlin y Java

Kotlin y java comparten muchas similitudes en cuantos, a sus estructuras de datos, pero hay algunas diferencias importantes a tener en cuenta:

* Nullabilidad -> Kotlin tiene la característica de nullabilidad que no existe en Java, los tipos de datos pueden ser declarados como nullable o no-nullable. Esto significa que las estructuras de datos de Kotlin pueden contener valores nulos de manera segura.
* Operaciones de colecciones: En Kotlin, se proporcionan varias funciones de extensión en las colecciones que permiten realizar operaciones más eficientes y expresivas en comparación con Java. Por ejemplo, Kotlin tiene funciones como map() y filter() que permiten transformar y filtrar colecciones de manera más sencilla.
* Sintaxis: Aunque Kotlin y Java comparten muchas similitudes en su sintaxis, hay algunas diferencias notables en la forma en que se definen y se usan las estructuras de datos.

1. Arreglos en Kotlin
   1. ¿Qué es un arreglo?

un arreglo es una estructura de datos que contiene un conjunto de elementos. Los elementos están almacenados en una secuencia ordenada y se pueden acceder a ellos por su índice.

* 1. Creación de arreglos en Kotlin

Para declarar un arreglo en Kotlin, se utiliza la palabra clave "arrayOf", seguida de los elementos separados por comas entre corchetes.

Por ejemplo:

val arr = arrayOf(1,2,3,4,5,6)

* 1. Accediendo a los elementos de un arreglo

Para acceder a un arreglo en Kotlin se utiliza el índice del elemento al que se quiere acceder, Por ejemplo:

val arr = arrayOf(1,2,3,4,5,6)

val element = arr[2]

Esto nos tomaría el valor de la posición 2 el cual su valor es 3.

* 1. Modificando los elementos de un arreglo

Para modificar un elemento de un arreglo en Kotlin, primero se debe acceder a este y tomar el valor que se quiere modificar, Por ejemplo:

val arr = arrayOf(1,2,3,4,5,6)

arr[2] = 23

esto cambiaría el valor del elemento en la posición 2 pase de 3 a ser 23.

* 1. Recorriendo un arreglo

La manera más común de recorrer un arreglo es por medio de una iteración, como puede ser for, for Each.

* 1. Funciones útiles para trabajar con arreglos en Kotlin

existen multiples funciones útiles para trabajar con arreglos en Kotlin algunas de estas son:

1. size, se utiliza para obtener el tamaño del arreglo.

2. get, se usa para acceder a un elemento por su índice

3. set, se utiliza para modificar un elemento en un arreglo por su índice

4. contains, se usa para saber si hay un elemento en un arreglo

5. indexOf, Se utiliza para obtener el índice de un elemento en el arreglo.

6. sorted, se usa para ordenar los elementos de un arreglo de forma ascendente.

1. Listas en Kotlin
   1. ¿Qué es una lista?

Una lista es una colección genérica de elementos que se caracteriza por almacenarlos de forma ordenada, donde pueden existir duplicados y se indexan los elementos con base 0.

* 1. Creación de listas en Kotlin

Primero se crea una variable llamada numbers de tipo List<Int> porque contendrá una lista de solo lectura de números enteros. Crea una nueva clase List con la función de la biblioteca estándar de Kotlin listOf() y pasa los elementos de la lista como argumentos separados por comas. listOf(1, 2, 3, 4, 5, 6) y esto quedaría de la siguiente manera:

val numbers: List<Int> = listOf(1, 2, 3, 4, 5, 6)

* 1. Accediendo a los elementos de una lista

Una de las maneras más comunes para acceder a una lista es usando un iterador de la siguiente manera:

fun main() {

val list: List<String> = listOf("Uno", "Dos", "Tres")

val itr = list.listIterator() // o usa `iterator()`

while (itr.hasNext()) {

println(itr.next())

}

}

* 1. Modificando los elementos de una lista

Para modificar los elementos se puede acceder a través de su índice para asignarles un nuevo valor:

Val frutas = mutableListOf(“manzana”, “naranja”, “banana”)

Y se modificaria de la siguiente forma:

Frutas[1] = “uva”

Esto lo que hace es modificar la palabra naranja que se encuentra en la posición 1 por la palabra uva

* 1. Recorriendo una lista

Para recorrer una lista se puede utilizar bucles como for o while dependiendo de las necesidades:

Val frutas = listOf(“manzana”, “naranja”, “banana”)

For ( frutas in frutas ) {

Println(fruta)

}

Esto lo que hace es imprimir cada elemento de la lista frutas en la consola

* 1. Funciones útiles para trabajar con listas en Kotlin

Algunas de las funciones mas comunes para trabajar con listas son las siguientes:

* Filter: se utiliza para filtrar elementos de una lista según un predicado dado.
* Map: se usa para transformar cada elemento de una lista en otro valor.
* Reduce: se utiliza para combinar los elementos de una lista en un solo valor.
* Sorted: se utiliza para ordenar los elementos de una lista en orden ascendente.
* GroupBy: esta función se utiliza para agrupar los elementos de una lista según una clave dada.
* Take: se usa para tomar los primeros n elementos de una lista.
* Drop: se utiliza para eliminar los primeros n elementos de una lista.

1. Conjuntos en Kotlin
   1. ¿Qué es un conjunto?

En Kotlin, un conjunto es una colección de elementos únicos sin orden específico. Los elementos en un conjunto son únicos, lo que significa que no puede haber duplicados.

* 1. Creación de conjuntos en Kotlin

Para declarar un conjunto en Kotlin, se utiliza la palabra clave "setOf", seguida de los elementos separados por comas entre paréntesis.

Por ejemplo:

Val conj = setOf(“sebas”,” diego”,”juanes”,”santi”)

* 1. Accediendo a los elementos de un conjunto

para acceder a los elementos de un conjunto en Kotlin se debe utilizar el bucle for-each o la función forEach para recorrer los elementos.

Por ejemplo:

Val conj = setOf(“sebas”,” diego”,”juanes”,”santi”)

Conj.forEach{ element ->

Print(element)

}

Esto imprimiría cada elemento del conjunto.

* 1. Modificando los elementos de un conjunto

los conjuntos por defecto son inmutables, lo que significa que no se pueden modificar los datos que están dentro de estos, pero existen arreglos de tipo mutables, para su declaración en vez de utilizar setOf se utilizaría mutableSetOf, con estos conjuntos mutables se pueden modificar de distintas maneras con las funciones add para agregar un elemento, remove para eliminar un elemento o clear que se usa para eliminar todos los elementos del conjunto.

* 1. Recorriendo un conjunto

para recorrer los conjuntos en Kotlin se debe utilizar el bucle for-each o la función forEach para recorrer los elementos.

Por ejemplo:

Val conj = setOf(“sebas”,” diego”,”juanes”,”santi”)

For (element in conj){

Print(element)

}

* 1. Funciones útiles para trabajar con conjuntos en Kotlin

existen múltiples funciones útiles para trabajar con conjuntos en Kotlin algunas de estas son:

1. add, se usa para agregar un elemento al conjunto.

2. remove, se utiliza para elimina un elemento del conjunto.

3. contains, verifica si un elemento está presente en el conjunto.

4. union, se utiliza para crear un nuevo conjunto que contiene todos los elementos de este conjunto y otro conjunto.

5. intersect, se usa para crear un nuevo conjunto que contiene solo los elementos que están presentes en este conjunto y otro conjunto.

6. subtract, crea un nuevo conjunto que contiene solo los elementos que están en este conjunto, pero no en otro conjunto.

7. clear,: elimina todos los elementos del conjunto.

8. size, devuelve el número de elementos en el conjunto.

9. isEmpty, se utiliza para verificar si el conjunto está vacío.

10. max, retorna el elemento máximo en el conjunto.

11. min, retorna el elemento mínimo en el conjunto.

12. forEach, ejecuta una acción en cada elemento del conjunto.

1. Mapas en Kotlin
   1. ¿Qué es un mapa?

Un mapa es una colección que almacena sus elementos en formas de pares clave-valor, lo que esto quiere decir que cada clave le corresponde un solo valor y será única como si se tratase de un identificador

* 1. Creación de mapas en Kotlin

Con el mapOf() crea un mapa inmutable compuesto por una lista de pares, donde el primer valor es la clave y el segundo el valor:

Val frutas: Map<Int, String> = mapOf( 34 to “mandarina”, 1 to “uva”)

For ((key, value) in frutas) {

Println(“$key es la llave y la fruta es $value”)

}

* 1. Accediendo a los elementos de un mapa

Se puede acceder a un mapa usando la función get():

Print (frutas.get (34))

O con los corchetes como un atajo para get

Print(frutas[34])

* 1. Modificando los elementos de un mapa

En Kotlin, se pueden modificar los elementos de un mapa si este es mutable de no serlo no se podrá, un ejemplo con mapa mutable es el siguiente:

Val frutas = mutableMapOf(“key1” to “mandarina”, “key2” to “uva”)

map.put(“key3”, “naranja”) // con este se agrega un nuevo elemento

map[“key2”] = “nueva fruta” // actualiza un elemento existente en el mapa utilizando la clave

map.remove(“key1) // elimina un elemento del mapa

* 1. Recorriendo un mapa

Para recorrer un mapa se puede utilizar el método forEach o el bucle for

val mapa = mapOf("a" to 1, "b" to 2, "c" to 3)

mapa.forEach { (llave, valor) ->

println("$llave -> $valor")

}

Y con el método for seria de la siguiente forma

val mapa = mapOf("a" to 1, "b" to 2, "c" to 3)

for ((llave, valor) in mapa) {

println("$llave -> $valor")

}

* 1. Funciones útiles para trabajar con mapas en Kotlin

En Kotlin existen varias funciones útiles como las siguientes:

* mapOf: esta función crea un mapa inmutable con las entradas especificas.
* mutableMapOf: esta función crea un mapa mutable con las entradas especificadas.
* Put: agrega una nueva entrada al mapa. Si la llave ya existe, se sobrescribe el valor
* Get: obtiene el valor asociado con una llave en el mapa. Si la llave no existe, retorna null.
* getOrElse: obtiene el valor asociado con una llave en el mapa. Si la llave no existe, retorna un valor por defecto especificado.
* Keys: retorna una lista con las llaves del mapa.
* Values: retorna una lista con los valores del mapa.
* Filter: esta función retorna un nuevo mapa con las entradas que cumplen una condición especificada en una lambda (un lambda es una función que puede ser usado como expresión, esto quiere decir que no está ligada a un identificador y que se puede usar como un valor).

1. Pares en Kotlin
   1. ¿Qué es un par?

un par es una estructura de datos que representa dos valores relacionados.

* 1. Creación de pares en Kotlin

Un par se define utilizando la clase "Pair", que tiene dos propiedades llamadas "first" y "second" que pueden contener cualquier tipo de datos.

Por Ejemplo:

Val user = Pair("Santi", 32)

* 1. Accediendo a los elementos de un par

Para acceder a los elementos de un par se utilizan las propiedades first y second, que devuelven el primer y segundo valor respectivamente.

Por ejemplo:

val user = Pair("Santi", 32)

val name = user.first // esto equivaldría a “Santi”

val number = user.second // esto equivaldría a 32

* 1. Modificando los elementos de un par

Los pares son inmutables, esto significa que sus elementos no pueden ser modificados, sin embargo, se puede utilizar la función copy, que crea una copia del par existente con los elementos reemplazados por los valores especificados.

Por ejemplo:

val user = Pair("Santi", 32)

val newUser = user.copy(first = "Juanes", second = 12)

println("Tengo ${newUser.second} ${newUser.first}")

* 1. Recorriendo un par

Los pares no se pueden recorrer como tal, ya que son estructuras de solo 2 datos, la manera para acceder a estos es por medios de las propiedades first y second.

* 1. Funciones útiles para trabajar con pares en Kotlin

existen múltiples funciones útiles para trabajar con conjuntos en Kotlin algunas de estas son:

1. first y second, son propiedades para acceder a los elementos del par.

2. copy, esta función crea una copia del par existente con los elementos reemplazados por los valores especificados.

3. toList, convierte los pares en una lista.

4. to, esta convierte el par en un objeto de tipo Pair.

5. let, esta función se puede utilizar para realizar una operación en un par y luego devolver el resultado.

1. Prácticas de estructuras de datos en Kotlin
   1. Ejercicios prácticos para aplicar los conceptos aprendidos
   2. Solución a los ejercicios prácticos Recursos adicionales:

Presento enlace de los ejercicios

Documentación oficial de Kotlin: [**https://kotlinlang.org/docs/reference/**](https://kotlinlang.org/docs/reference/)

**Entrega.**

Se deberá realizar la entrega de un informe con la solución de los puntos anteriores, el aprendiz acompañará la investigación con ejemplos practicos de cada estructura y deberá publicar el código fuente en un repositorio en GitHub.