Taller Estructuras de Datos en Kotlin

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 evidencian los siguientes puntos:

1. Introducción a las estructuras de datos en Kotlin

    a. ¿Qué son las estructuras de datos y para qué se utilizan?

    Los tipos de colección (a veces llamados estructuras de datos) te permiten almacenar varios valores, generalmente del mismo tipo de datos, de manera organizada. Una colección puede ser una lista ordenada, una agrupación de valores únicos o una asignación de valores de un tipo de datos a valores de otro tipo.

    b. Ventajas de utilizar estructuras de datos en Kotlin

    cuenta con curva de aprendizaje, La sencillez de la sintaxis permite una fácil curva de aprendizaje, ideal para aprender tu primer lenguaje de programación. Un ejemplo muy habitual al trabajar con listas en Java, para acceder a la primera posición tendrás que buscar en la posición 0, en Kotlin simplemente llamarías a la función ***first().*** Además, el hecho de que todo funcione tan bien desde el minuto cero para Android es una ventaja potencial. Ya que sin saber nada de Kotlin puedes tener un proyecto montado y listo para trabajar en menos de 10 minutos sin ningún problema. A partir de ahí todo funciona como si trabajaras con Java: puedes ejecutar desde el IDE, depurar sin problema, hacer refactors, utilizar instant run, etcétera.

    c. Diferencias entre las estructuras de datos en Kotlin y Java

    Ambos tienen casi la misma velocidad de codificación, pero Java requiere escribir más código. Sin embargo, se dedica menos tiempo a pensar la solución con Java. Kotlin tiene construcciones más concisas, por lo que requiere escribir menos código. Sin embargo, se tarda más tiempo en encontrar la solución a una tarea.

2. Arreglos en Kotlin

    a. ¿Qué es un arreglo?

    Un arreglo es una colección de elementos del mismo tipo, a los que se accede a través de su índice.

    b. Creación de arreglos en Kotlin

    Para declarar un array en kotlin usamos la función ***arrayOf():***

    Toma los elementos del array como parámetros:

```
val animals = arrayOf("Bird", "Cat", "Dog", "Bear", "Fish", "Lion")
```

    c. Accediendo a los elementos de un arreglo

    Para acceder a los elementos de un array usamos su índice:

- El nombre del array
- Un corchete angular de apertura ([ ) y uno de cierre ( ])
- El índice del elemento entre los corchetes.

```
println(animals[0])
println(animals[2])
println(animals[4])
```

    d. Modificando los elementos de un arreglo

    Para modificar los elementos de un arreglo seleccionamos el índice que deseamos modificar, de la siguiente manera:

```
animals[1] = "Wolf"
animals[3] = "Chickens"
animals[5] = "Fox"
```

e. Recorriendo un arreglo

Para recorrer un arreglo se usa un ciclo ***for*** y el operador ***in***. En este caso recorre el arreglo por valor:

```
for (position in animals) {
     println(position)
}
```

Si queremos recorrer el arreglo por índices debemos iterar en array.indices:

```
for (position in animals.indices) {
     println("In the index ${position} we have ${animals[position]}")
}
```

f. Funciones útiles para trabajar con arreglos en Kotlin

Un arreglo es una estructura con valores de datos, que están almacenados de forma contigua en memoria. Todos los elementos son referenciados por un mismo nombre y tienen el mismo tipo de dato.

Los elementos estarán indexados tomando como base el 0 y el tamaño declarado del arreglo será fijo.

Kotlin usa la clase genérica ***Array<T>*** para representar arreglos. Crear instancias con un tipo parametrizado usa los siguientes métodos:

arrayOf(vararg elements:T): recibe un argumento variables con elementos de tipo T y retorna el arreglo que los contiene.

arrayOfNulls(size:Int): crea un arreglo de tamaño size con elementos de tipo T e inicializa los valores con null.

emptyArray(): crear un arreglo vacío con el tipo T

3. Listas en Kotlin

a. ¿Qué es una lista?

Una lista es una colección ordenada de elementos. Para trabajar con listas se utiliza la interfaz ***List***.

b. Creación de listas en Kotlin

Usamos ***listOf()*** para crear una lista inmutable. Podremos tener todos los elementos de un mismo tipo de dato o de diferentes tipos de dato:

```
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Para crear una lista mutable usamos la función ***mutableListOf()***:

```
val mutableNumbers: MutableList<Int> = mutableListOf<Int>(1, 21, 32, 43, 55, 66)
```

c. Accediendo a los elementos de una lista

```
println(numbers)
println(numbers[0])
println(numbers[2])
println(numbers[4])
println(numbers[6])
println(numbers[8])
```

```
println(mutableNumbers[0])
println(mutableNumbers[2])
println(mutableNumbers[4])
```

d. Modificando los elementos de una lista

```
mutableNumbers[1] = 11
mutableNumbers[3] = 40
mutableNumbers[5] = 5
```

e. Recorriendo una lista

```
for (position in numbers.indices) {
    println("In the index ${position} we have ${numbers[position]}")
}
```

```
for (position in mutableNumbers.indices) {
    println("In the index ${position} we have ${mutableNumbers[position]}")
}
```

f. Funciones útiles para trabajar con listas en Kotlin

Las colecciones se pueden clasificar en dos grandes grupos, las mutables e inmutables. Es decir, las que se pueden editar (mutables) y las que son solo de lectura (inmutable).

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

Para acceder al estado de la lista puedes usar los siguientes miembros básicos:

size: para obtener la cantidad de elementos de la lista.
list[index]: para obtener el elemento ubicado en index. Esta es la construcción para el operador de acceso posicional get(index).
indexOf(element): para obtener el índice de la primera ocurrencia de element.
lastIndexOf(element): para obtener el índice de la última ocurrencia del element.
subList(fromIndex, toIndex): para obtener una porción de la lista en el rango [fromIndex, toIndex).

4. Conjuntos en Kotlin

a. ¿Qué es un conjunto?

Un conjunto es una colección sin orden de elementos únicos y no puede tener ningún duplicado.

b. Creación de conjuntos en Kotlin

Para la creación de conjuntos de solo lectura realizamos lo siguiente:

```
val months = setOf("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug")
```

Si necesitamos remover o modificar los elementos de un conjunto usamos ***mutableSetOf()***:

```
val mutableMonths = mutableSetOf("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug")
```

c. Accediendo a los elementos de un conjunto

d. Modificando los elementos de un conjunto

e. Recorriendo un conjunto

En Kotlin, podemos hacer un recorrido con la ayuda del método forEach()

f. Funciones útiles para trabajar con conjuntos en Kotlin

Sets De Solo Lectura

Un conjunto o set es una colección de elementos sin ordenar que no soporta duplicados. Puedes ver este diseño conceptual como el modelo de los conjuntos matemáticos.

La interfaz genérica Set<E> es la que representa a los conjuntos de solo lectura en el paquete kotlin.collections. Al igual que List, Set extiende de Collection<E>.

Igualdad De Conjuntos

Dos conjuntos son iguales aunque sus elementos hayan sido incluidos en un orden diferente o la inicialización tenga copias.

El Método contains()

Para expresar la notación «a pertenece a A» usa el método contains(element) sobre el conjunto, para determinar si element pertenece. El operador in también cumple con esta evaluación.

Sets Mutables

Al igual que setOf(), la función mutableSetOf() recibe los elementos que habitaran en el conjunto, también puedes optar por crear un conjunto vacío pero especificando el tipo parametrizado

Añadir elementos

Agrega elementos a través del método add() o usando los operadores de adición (+) o adición compuesta (+=) de las colecciones

Recuerda que no se permiten los duplicados para los tipos Set, por lo que añadir un elemento existente no tendrá efecto.

Remover elementos

como es normal usa el método remove() para remover elementos de un conjunto. o similar a la agregación, usa el operador de resta

resta (-) o resta compuesta para conseguir el mismo resultado, tanto add() como remove() retornan un tipo boolean, por lo que si las

operaciones fueron exitosas tendrás true, de lo contrario false.

Operaciones entre conjuntos

La función union() toma como argumentos dos colecciones y retorna en conjunto con todos los elementos que pertenezcan a ambas.

La función intersect()

Si deseas aislar solo los elementos que estén presentes en dos colecciones, entonces usa la función intersect() el resultado será un

conjunto intermedio de coincidencias

La función subtract()

cuando necesites calcular la diferencia entre dos colecciones usa la función subtract() el valor de retorno de A subtract B es el conjunto

que resulta de eliminar de A cualquier elemento que esté en B. Tanto si no especificas el argumento del tipo explicitamente para SetB

el compilador de kotlin se quejara de la ausencia de información para la interferencia.

5. Mapas en Kotlin

a. ¿Qué es un mapa?

Un mapa es una colección que almacena sus elementos (entradas) en forma de pares clave-valor. Esto quiere decir que a cada clave le corresponde un solo valor y será única como si se tratase de un identificador. La ilustración anterior muestra la correspondencia entre una colección de claves a una de valores.

b. Creación de mapas en Kotlin

Si queremos crear un mapa que contenga el nombre y la edad de una persona, se realiza de la siguiente manera:

```
val namesMap = mutableMapOf(
  "James" to 17,
  "Camila" to 22,
  "Samuel" to 9,
  "Sara" to 12
)
```

c. Accediendo a los elementos de un mapa

Para crear un Map en Kotlin debemos definir una variable e indicar de qué tipo de datos son la clave del mapa y el valor que almacena. Hemos creado un Map que almacena tres entradas.

d. Modificando los elementos de un mapa

Para crear un Map en Kotlin debemos definir una variable e indicar de qué tipo de datos son la clave del mapa y el valor que almacena. Hemos creado un Map que almacena tres entradas.

Añadir Y Actualizar Entradas

Usa el método ***put(key, value)*** para asociar la clave ***key*** con el valor ***value***. Si la clave no existe la entrada es añadida al mapa, de lo contrario el valor es actualizado.

```
namesMap.put("Camila", 22)
    namesMap["Camila"] = 25
```

e. Recorriendo un mapa

Un ejemplo de esto es recorrer sobre los elementos de un mapa en un ***bucle for***:

```
for ((name, edad) in namesMap) {
     println("$name -> $edad")
}
```

f. Funciones útiles para trabajar con mapas en Kotlin

La Función map()

La función de orden superior map() te permite aplicar una función sobre todos los elementos de una colección con el fin de una nueva colección con el cálculo final.

La solución consistió en pasar una función lambda que invoque a la función de extensión String.toUpperCase() de cada elemento. El elemento es representado por la referencia it.

Recuerda que la sintaxis para aplicar la función map en mapas usa como parámetro cada entrada Entry<K, V>, por lo que puedes acceder a cada componente en tu transformación.

6. Pares en Kotlin

a. ¿Qué es un par?

Son una representación genérica (cualquier tipo de datos) de dos valores (pares). Podemos emplearlos para guardar una pareja de valores por ejemplo un usuario y contraseña o cualquier pareja de valores que tengan alguna relación.

b. Creación de pares en Kotlin

Para emplear valores pares realizamos lo siguiente:

```
val userPassword = Pair("Clarita22", 51300)
```

c. Accediendo a los elementos de un par

Para acceder a cada uno de los elementos dentro de un ***Pair*** usamos las palabras reservadas ***first*** y ***second*** respectivamente de la siguiente manera:

```
println(userPassword.first)
println(userPassword.second)
```

d. Modificando los elementos de un par

e. Recorriendo un par

f. Funciones útiles para trabajar con pares en Kotlin

7. Prácticas de estructuras de datos en Kotlin

a. Ejercicios prácticos para aplicar los conceptos aprendidos

1. Se desea guardar los sueldos de cinco operadores (Estructura: Arreglos)
2. Crear una lista inmutable con los días de la semana. Probar las propiedades y métodos principales para administrar la lista.
3. Crear un conjunto inmutable que almacene las fechas de este año que son feriados.Ingresar luego por teclado una fecha y verificar si se encuentra en el conjunto de feriados
4. Crear un mapa que permita almacenar 5 artículos, utilizar como clave el nombre de productos y como valor el precio del mismo.Desarrollar además las funciones de:

   1) Imprimir en forma completa el diccionario

   2) Mostrar la cantidad de artículos con precio superior a 20.
5. Declarar una clase Persona con las propiedades nombre y edad, definir como métodos su impresión y otra que retorna true si es mayor de edad o false en caso contrario. En la función main define un arreglo con cuatro elementos de tipo Persona. Calcular cuantas personas son mayores de edad.

b. Solución de los ejercicios práctico

1. Declaramos el arreglo y creamos espacio para cinco componentes. Para acceder a cada componente del arreglo utilizamos for y mediante un subíndice indicamos que componente estamos procesando. Cuando i valore 0 estamos cargando la primer componente del arreglo. Las componentes comienzan a numerarse a partir de cero y llegan hasta el tamaño que le indicamos menos 1. Una vez que cargamos todas sus componentes podemos imprimirlas una a una dentro de otro for:

```
fun main() {
    val sueldos = IntArray(5)
    // Carga de los elementos por teclado
    for (i in 0..4) {
        println("Ingrese sueldo: ")
        sueldos[i] = readLine()!!.toInt()
    }
    // Impresion de sus elementos
    println("Lista de Sueldos registrados: ")
    for (i in 0..4) {
        println(sueldos[i])
    }
}
```

2. Para crear una lista inmutable podemos llamar a listOf y pasar como parámetro los datos a almacenar, debemos indicar el tipo de datos que almacena luego de List.

   Lo que podemos hacer con una lista inmutable es acceder a sus elementos, por ejemplo recorrerla con un for,Acceder a cualquier elemento por un subíndice.

```
fun main() {
    var lista1: List<String> = listOf("lunes", "martes", "miercoles", "jueves", "viernes", "sabado", "domingo")
    println("Imprimir la lista completa")
    println(lista1)
    println("Imprimir el primer elemento de la lista")
    println(lista1[0])
    println("Imprimir el primer elemento de la lista")
    println(lista1.first())
    println("Imprimir el último elemento de la lista")
    println(lista1.last())
    println("Imprimir el último elemento de la lista")
    println(lista1[lista1.size-1])
    println("Imprimir la cantidad de elementos de la lista")
    println(lista1.size)
    println("Recorrer la lista completa con un for")
    for(elemento in lista1) {
        print("$elemento ")
    }
    println()
    println("Imprimir el elemento y su posicion")
    for(posicion in lista1.indices) {
        print("[$posicion]${lista1[posicion]} ")
    }
}
```

3.Declaramos un data class que representa una fecha, data class Fecha(val dia: Int, val mes: Int, val año: Int)

Definimos un conjunto inmutable de tipo Fecha y guardamos dos fechas mediante la llamada a la función setOf.

Cargamos una fecha cualquiera por teclado, Mediante el operador in verificamos si la fecha ingresada se encuentra en el conjunto de feriados.

```
data class Fecha(val dia: Int, val mes: Int, val año: Int)
fun main() {
    var feriados: Set<Fecha> = setOf(Fecha(1, 1, 2017),
                                     Fecha(25, 12, 2017))
    println("Ingrese una fecha")
    print("Ingrese el dia:")
    val dia = readln().toInt()
    print("Ingrese el mes:")
    val mes = readln().toInt()
    print("Ingrese el año:")
    val año = readln().toInt()
    if (Fecha(dia, mes, año) in feriados){
        println("La fecha ingresada es feriado")
    } else {
        println("La fecha ingresada no es feriado")
    }
}
```

4. En este caso creamos un mapa cuya clave es de tipo String y su valor es un Float.

Para mostrar el mapa en forma completo lo hacemos recorriendo por medio de un for,Para contar la cantidad de productos que tienen un precio superior a 20 llamamos al método count y le pasamos una función lambda analizando el parámetro it en la propiedad value si cumple la condición de superar al valor 20.

```
data class Producto(val nombre: String, val precio: Float, val cantidad: Int)
fun cargar(productos: MutableMap<Int, Producto>) {
    productos[1] = Producto("Papas", 13.15f, 200)
    productos[15] = Producto("Manzanas", 20f, 0)
    productos[20] = Producto("Peras", 3.50f, 0)
}
fun listadoCompleto(productos: MutableMap<Int, Producto>) {
    println("Listado completo de productos")
    for((codigo, producto) in productos) {
        println("Codigo: $codigo  Descripcion ${producto.nombre} Precio: ${producto.precio} Stock Actual:
${producto.cantidad}")
    }
println()
}
fun consultaProducto(productos: MutableMap<Int, Producto>) {
    print("Ingrese el codigo de un producto:")
    val codigo = readln().toInt()
    if (codigo in productos){
        println("Nombre: ${productos[codigo]?.nombre} Precio: ${productos[codigo]?.precio} Stock:
${productos[codigo]?.cantidad}")
    } else {
        println("No existe un producto con dicho codigo")
    }
}
fun sinStock(productos: MutableMap<Int, Producto>) {
```

```
    val cant = productos.count { it.value.cantidad == 0 }
    println("Cantidad de artículos sin stock: $cant")
}
fun main() {
    val productos: MutableMap<Int, Producto> = mutableMapOf()
    cargar(productos);
    listadoCompleto(productos)
    consultaProducto(productos)
    sinStock(productos)
}
```

5.La declaración de la clase Persona define 2 propiedades en el mismo constructor y sus dos métodos.

En la función main definimos una variable llamada personas que es un Array con componentes de tipo Persona. Para definir sus componentes utilizamos la función arrayOf que nos provee la librería estándar de Kotlin, A la función arrayOf se le pasa cada uno de los objetos de tipo Persona.

Un Array una vez creado no puede cambiar su tamaño.

En cada ciclo del for en la variable per se almacena una de las componentes del arreglo.

```
class Persona(val nombre: String, val edad: Int) {
    fun imprimir() {
        println("Nombre: $nombre  Edad: $edad")
    }

    fun esMayor() = if (edad >= 18) true else false
}

fun main(parametro: Array<String>) {
    val personas: Array<Persona> = arrayOf(Persona("ana", 22), Persona("juan", 13), Persona("carlos", 6),
Persona("maria", 72))
    println("Listado de personas")
    for(per in personas)
        per.imprimir()
    var cant = 0
    for(per in personas)
        if (per.esMayor())
            cant++
    println("Cantidad de personas mayores de edad: $cant")
}
```

Recursos adicionales:

Documentación oficial de Kotlin: **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 prácticos de cada estructura y deberá publicar el código fuente en un repositorio en GitHub.

Taller Estructuras de Datos en Kotlin 2