M3 - Programació

Tipus complexos III - Llistes i matrius

Índex

- Llistes
  - Immutables
  - Mutables
  - Llistes multidimensionals

Llistes - Definició

- Les col·leccions es poden classificar en dos grans grups, mutables i immutables.
  - Mutables - Poden editar-se
  - Immutables - no poden editar-se

- Dintre de les col·leccions tenim les llistes.

- La llista és una col·lecció ordenada amb accés a elements per índexs: nombres enters que reflecteixen la seva posició.

- Els elements poden aparèixer més d'una vegada en una llista.

Llistes Immutables - Declaració

- La declaració d'una llista immutable sería així:

```
val immutableList: List<String> = listOf("red", "blue", "green", "black")
```

Llistes immutables - Funcions

Veurem que disposem de diverses funcionalitats per a manegar les llistes immutables:

- **size** - Mostra el tamany de la llista
- **get(pos)** - Torna el valor de la posició *pos*
- **first()** - Torna el primer valor de la llista
- **last()** - Torna l'últim valor de la llista

Llistes immutables - Funcions

Veurem que disposem de diverses funcionalitats per a manegar les llistes immutables:

```
fun main() {
    val readOnly: List<String> = listOf("Dilluns", "Dimarts",
"Dimecres", "Dijous", "Divendres", "Dissabte", "Diumenge")
    readOnly.size
    readOnly.get(3)
    readOnly.first()
    readOnly.last()
    println(readOnly)
}
```

Llistes mutables - Declaració

- Les llistes mutables, tot i que són més ineficients en quant a memòria que les llistes immutables, són molt més interessants a l'hora de donar-nos més opcions programatives.

- La declaració i inicialització d'una llista mutable sería així:

```
val mutableList: MutableList<String> = mutableListOf("red", "blue",
"green", "black")
println(mutableList) // [red, blue, green, black]
```

Llistes mutables - Declaració

- El fet de ser mutables ens permet anar afegint i eliminant elements un cop la llista ja ha estat inicialitzada, cosa que no se'ns permetia fer en els arrays o les llistes immutables.
- Per defecte s'afegeixen els elements al final de la llista, però també podem indicar la posició on el volem afegir.

```
val mutableList: MutableList<String> = mutableListOf("red", "blue",
"green", "black")
println(mutableList) // [red, blue, green, black]
mutableList.add("yellow")
println(mutableList) // [red, blue, green, black, yellow]
mutableList.add(1, "orange")
println(mutableList) // [red, orange, blue, green, black, yellow]
```

Llistes mutables - Nulabilitat

- Ara hem de tenir cura amb una llista mutable pel simple fet que podria estar buida o contenir un null.
- Un null és un valor nul en una de les seves posicions, que, si hi accedim i intentem operar, l'aplicació es trencarà, ens saltarà un error.
- Per això, tenim algunes funcions que ens permetrà treballar com una llista immutable, però amb seguretat.
  - **none()** - Ens torna true si la llista és buida
  - **firstOrNull()** - Ens torna el primer element de la llista, sinó null
  - **elementAtOrNull(pos)** - Ens torna l'element a la posició *pos* o null en cas que no existeixi
  - **lastOrNull()** -Ens torna l'últim element de la llista, sinó null

Llistes mutables - Funcions

- Veurem que disposem de diverses funcionalitats per a manegar les llistes mutables:
  - **clear()** - Elimina tots els elements de la llista
  - **remove(element)** - Elimina l'element de la llista, si existeix
  - **removeAt(pos)** - Elimina l'element de la posició *pos*
  - **set(pos, element)** - Reemplaça l'element de la posició *pos* per un nou element

Llistes mutables - Creació buida

Com ens ho fem per crear una llista mutable buida?

```
val mutableList: MutableList<Int> = mutableListOf()

list.add(4)
list.add(5)
list.add(15)
```

Iterar llistes

- La forma de recórrer les llistes és molt similar a la de recórrer els arrays.
- Podem fer-ho a través de les seves posicions

```
for (i in exampleList.indices) {
    println(exampleList.get(i))
}
```

- …o a través dels seus valors.

```
for (valor in exampleList) {
    println(valor)
}
```

Llistes - Multidimensional

- A vegades ens interessarà tenir llistes amb diferents nivells de profunditat.
- Què vol dir això?
- Imaginem el següent codi:

```
val colours: MutableList<String> = mutableListOf("red", "blue", "green", "black")
```

- Això ens generarà una llista: [red, blue, green, black]

Llistes - Multidimensional

- Però i si en comptes de tenir una MutableList d'Strings tenim una MutableList amb MutableLists d'Strings?

```
val colours: MutableList<MutableList<String>> = mutableListOf(
mutableListOf("red", "blue", "green"),
mutableListOf("white", "black", "grey"),
mutableListOf("violet", "orange", "brown"))
```

- Això ens generarà una llista de llistes:
  - **[** [red, blue, green],
    [white, black, grey],
    [violet, orange, brown] **]**

Llistes - Multidimensional

- Això ens generarà una llista: **[** [red, blue, green],
  [white, black, grey],
  [violet, orange, brown] **]**

- A cada nivell de la llista tenim una nova llista.

```
println(colours[0]) // [red, blue, green]
println(colours[2]) // ???
println(colours[2][1]) // orange
println(colours[1][2]) // ???
```

Llistes - Multidimensionals

- Com ens ho fem per crear una llista multidimensional buida?

```
var list : MutableList<MutableList<Int>> = mutableListOf()
var elementList : MutableList<Int> = mutableListOf()

list.add(elementList)
elementList.add(4)
elementList.add(3)

elementList = mutableListOf(8, 15)
list.add(elementList)
```