ENG. INFORMÁTICA E DE COMPUTADORES

Algoritmos e Estruturas de Dados

(parte 13)

2° Semestre 2021/2022

Instituto Superior de Engenharia de Lisboa
Paula Graça

ITERADORES

- Um *iterador* permite percorrer todos os elementos de uma coleção independentemente da sua implementação específica

- Permite a uma aplicação cliente, explorar a sua coleção de objetos sem expor os detalhes da implementação

ITERADORES

- A interface *Iterator* está definida na biblioteca padrão do Kotlin
- Disponibiliza dois métodos: *hasNext()* e *next()*

Método	Descrição
hasNext()	Devolve um *boolean* que indica se a coleção ainda tem mais elementos para devolver fun hasNext(): Boolean
next()	Devolve o próximo objecto da colecção operator fun next(): E

ITERADORES

- Interface *Iterator*

```
val cars = listOf(“Mercedes", “Ford", “Audi")

for (car in cars) {
        println(car)
}


val team = mapOf(“João" to 33, “Ana" to 3, “Maria" to 10)

for (player in team) {
        println("${player.key} wears ${player.value}")
}
```

ITERADORES

- Classe *StringIterator*

```
class StringIterator(val str: String): Iterator<Char> {

        var current = 0

        override fun hasNext() = current < str.length

        override fun next(): Char {
                if (current == str.length) throw NoSuchElementException()
                current++
                return str[current - 1]
        }
}
```

ITERADORES

- Função *Main* que usa o iterador

```
fun main() {
        val str = "This is a collection of chars."
        strColl = StringIterator("This is a collection of chars.")

        for (c in strColl)
                print(c)
        println()
}
```

ITERADORES

- A interface *Iterable* está definida na biblioteca padrão do Kotlin
- As classes que estão de acordo com a interface *Iterable* podem ser representadas como uma sequência de elementos que podem ser percorridos
- Disponibiliza o método *iterator()*

Método	Descrição
iterator()	Devolve um iterador para a coleção que implementa a interface Iterator operator fun iterator(): Iterator<E>

ITERADORES

- Classe *StringCollection*

```
class StringCollection(val str: String) : Iterable<Char> {

        override fun iterator(): Iterator<Char> {
                return StringIterator(str)
        }
}
```

- Classe *StringIterator*
- (apresentada anteriormente)

```
class StringIterator(val str: String): Iterator<Char> {
        private var current = 0

        override fun hasNext() = current < str.length

        override fun next(): Char {
                if (current == str.length) throw NoSuchElementException()
                current++
                return str[current - 1]
        }
}
```

ITERADORES

- Função *Main* que usa a coleção iterável

```
fun main() {
        val str = "This is a collection of chars."

        val it = StringCollection(str).iterator()
        while (it.hasNext())
                println(it.next())
        println()
}
```

ITERADORES

- As interfaces *Iterator* e *Iterable* podem ser definidas na mesma classe
- Classe *StringIterableCollection*

```
class StringIterableCollection(val str: String) : Iterator<Char>, Iterable<Char> {
        private var current = 0

        override fun hasNext() = current < str.length

        override fun next(): Char {
                if (current == str.length) throw NoSuchElementException()
                current++
                return str[current - 1]
        }

        override fun iterator(): Iterator<Char> {
                return StringIterableCollection(str)
        }
}
```

ITERADORES

- Função *Main* que usa a coleção iterável

```
fun main() {
        val str = "This is a collection of chars.“

        val strColl2 = StringIterableCollection(str)

        for (c in strColl2)
                println(c)
        println()

        val it = strColl2.iterator()
        while (it.hasNext())
                println(it.next())
        println()
}
```