



UNIVERSIDAD  
COMPLUTENSE  
DE MADRID



**ntic**  
master  
**School**

**Scala**

Colecciones

Enero de 2023



# Agenda

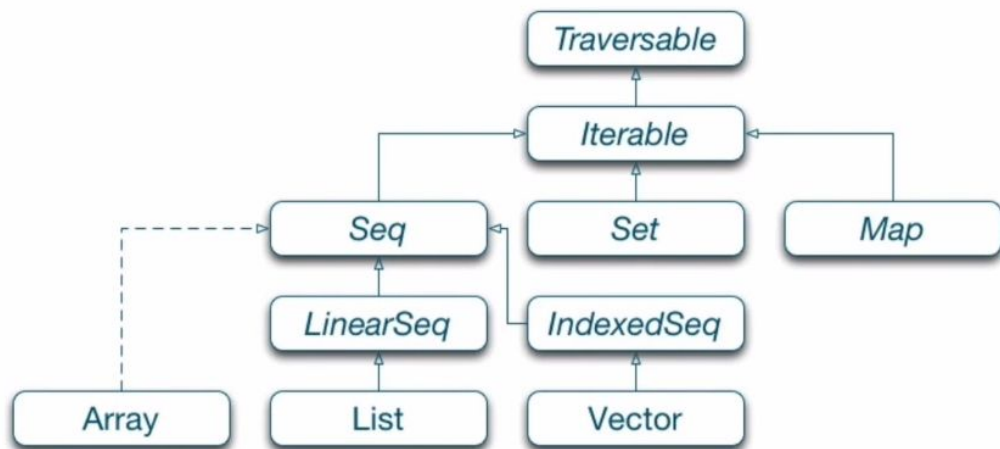
- **Esquema de colecciones**
- **Creando colecciones**
- **Colecciones destacables y sus métodos**
- **Las Tuplas**
- **Colecciones & (in)mutabilidad**



# 1 Esquema de colecciones



# Orientado a objetos en Scala



- La librería de colecciones de Scala es muy intuitiva y comprensible.



## 2 Creando colecciones

# Mis primeras colecciones



```
scala> Vector(1, 2, 3)
res10: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)

scala> Seq(1, 2, 3)
res11: Seq[Int] = List(1, 2, 3)

scala> Set(1, 2, "3")
res12: scala.collection.immutable.Set[Any] = Set(1, 2, 3)
```

- Se instancian sin hacer uso de la palabra reservada *new*.
- Cada colección tiene un companion object definido con su método *apply*.



# Colecciones y su azúcar sintáctico



```
scala> Vector(1, 2, 3)
res13: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)

scala> Vector.apply(1, 2, 3)
res14: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)
```

- Cada llamada a los objetos se traduce en una llamada a *apply*.
- *apply* es una método de factoría



# Parámetros de tipos para las colecciones



```
scala> Vector(1, 2, 3)
res15: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)

scala> Vector[Int](1, 2, 3)
res16: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)
```

- Los parámetros de tipo se definen entre corchetes `[]`
- Los tipos se pueden inferir o establecer explícitamente.





# 3 Colecciones destacables y sus métodos

## Métodos destacados de colecciones

- `++` -> Concatena dos colecciones
- `toSeq, toSet, etc` -> Convierte entre tipos de colecciones
- `isEmpty, size` -> Nos dan información del tamaño
- `contains` -> Verifica si un elemento es contenido en la colección
- `head` -> Devuelve el primer elemento
- `last` -> Devuelve el último elemento
- `tail` -> Devuelve todos los elementos excepto el primero
- `init` -> Devuelve todos los elementos excepto el último
- `take n` -> Devuelve los primeros *n* elementos
- `drop n` -> Elimina los primeros *n* elementos
- `zip` -> Mezcla los elementos de las colecciones en parejas
- `groupBy` -> Particiona la colección en un mapa de colecciones

# Sequences



```
scala> val numeros = Seq(1, 2, 3)
numeros: Seq[Int] = List(1, 2, 3)

scala> numeros(0)
res21: Int = 1

scala> numeros :+ 4
res22: Seq[Int] = List(1, 2, 3, 4)

scala> 0 +: numeros
res23: Seq[Int] = List(0, 1, 2, 3)

scala> Seq(1, 2, 2, 3).distinct
res24: Seq[Int] = List(1, 2, 3)
```

- Colección lineal de elementos.
- Respeta orden de inserción.
- Colección que es un mapeo de una posición a un elemento.
- Apto para accesos por índice, pero menos óptimo.



# Sets



```
scala> val numeros = Set(1, 2, 3)
numeros: scala.collection.immutable.Set[Int] = Set(1, 2, 3)

scala> numeros(1)
res5: Boolean = true

scala> numeros.contains(1)
res6: Boolean = true

scala> numeros + 4
res7: scala.collection.immutable.Set[Int] = Set(1, 2, 3, 4)

scala> numeros + 3
res8: scala.collection.immutable.Set[Int] = Set(1, 2, 3)
```

- Colección que no acepta duplicados.
- Acceso por índice (*apply*) == *contains*.
- Función + añade siempre al final.



# 4 Las Tuplas y más colecciones

# Tuplas



```
scala> Tuple2(1, "a")
res9: (Int, String) = (1,a)

scala> Tuple2(1, 2)
res10: (Int, Int) = (1,2)

scala> (1, "b")
res11: (Int, String) = (1,b)
```

- No son colecciones, sino un contenedor finito de elementos.
- Las Tuples son case clases con tipo parametrizada para cada atributo.
- Scala nos provee implementaciones de Tuple1 hasta Tuple 22.
- Nos aprovechamos del azúcar sintáctico para crear tuplas más sencillo.



## Tuplas II



```
scala> val pair = (1, "b")
pair: (Int, String) = (1,b)

scala> pair._2
res12: String = b

scala> 1 -> "b"
res13: (Int, String) = (1,b)

scala> val pair = 1 -> "b"
pair: (Int, String) = (1,b)
```

- Los atributos de las tuplas son identificados por posición: `_1`, `_2`, etc.
- Un par (Tuple2) se puede instanciar con el operador `->`



# Maps



```
scala> val map = Map(1 -> "a", 2 -> "b")
map: scala.collection.immutable.Map[Int,String] = Map(1 -> a, 2 -> b)

scala> map(1)
res14: String = a

scala> map.get(9)
res15: Option[String] = None

scala> map.getOrElse(1, "z")
res16: String = a

scala> map.getOrElse(9, "z")
res17: String = z
```

- Colección de clave-valor.
- Pares con clave única.
- Misma idea de los diccionarios en Python.





# 5 Colecciones & (in)mutabilidad

# Colecciones y la inmutabilidad



- Las colecciones de Scala residen en:
  - *scala.collection* - clases abstractas
  - *scala.collection.immutable*
  - *scala.collection.mutable*
  - Existen las colecciones paralelizables.
    - *scala.collection.parallel*
- Por defecto (sin tener que hacer *imports*) se usan las colecciones inmutables.
- Para hacer uso de colecciones mutables se debe importar la clase mutable.



# Mutando colecciones inmutables.



```
scala> val numeros = Vector(1, 2, 3)
numeros: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)

scala> 0+:numeros
res1: scala.collection.immutable.Vector[Int] = Vector(0, 1, 2, 3)

scala> numeros
res2: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3)
```

- Las colecciones inmutables no se pueden modificar.
- Las operaciones de modificación (adición, por ejemplo) siempre devuelven una nueva instancia.
- Las colecciones inmutables son persistentes (comparten datos estructurales).



## Ejercicio: Usa sequence



- Define una `case class` que se llame `Station` y que tenga como parámetro:
  - nombre: `name`
  - tipo: `String`
- Añade un parámetro de clase a `Train` de nombre `schedule`, que sea inmutable y de tipo `Seq` de `Station`.
  - Asegúrate de que `schedule` tenga como mínimo 2 elementos.

