Colecciones: de Wollok a Java

Este resumen es un conjunto de recetas para "traducir" mensajes de colecciones de Wollok a Java. Este trabajo es una fuente de soluciones comunes para problemas comunes, pero no sustituye la consulta propia a Internet, libros y apuntes.

Hay muchas formas de solucionar un problema. Entre las que encontré, elegí las más expresivas y las que usaran Streams de Java 1.8.

Si bien las soluciones se pueden copiar y pegar, varias de ellas requieren para su comprensión total entender cómo funcionan los streams de Java, métodos estáticos de clases, expresiones lambda, comparadores y el tipo Optional, cuyas explicaciones están fuera del alcance de este trabajo.

```
class Persona {
  var nombre
  var edad
  var amigos = []

  constructor(_nombre, _edad){
    nombre = _nombre
    edad   = _edad
  }

  method agregarAmigos(unasPersonas){
    amigos.addAll(unasPersonas)
  }

  method envejecer(){
    edad = edad++
  }

  method nombre() = nombre
  method edad()   = edad
  method amigos() = amigos
}
```

```
public class Persona {
  protected String nombre;
  protected int edad;
  protected List<Persona> amigos = new ArrayList<>();

  public Persona(String nombre, int edad){
    super();
    this.nombre = nombre;
    this.edad   = edad;
  }
  public void agregarAmigos(List<Persona> unasPersonas){
    this.amigos.addAll(unasPersonas);
  }
  public void envejecer(){
    this.edad++;
  }
  public String nombre(){
    return this.nombre;
  }
  public int edad(){
    return this.edad;
  }
  public List<Persona> amigos(){
    return this.amigos;
  }
}
```

ORDENES QUE ENTIENDEN TODAS LAS COLECCIONES		
Descripción	**Wollok**	**Java**
Ejecuta una acción sobre cada uno de los elementos de la colección.	`personas.forEach {p => p.envejecer()}`	`personas.forEach(p -> p.envejecer());`
Agrega un objeto a una colección. Si es una lista, lo agrega al final. Si la colección es un conjunto y el objeto ya está en el conjunto, entonces no hace nada	`var lorena = new Persona ("Lorena",21)` `personas.add(lorena)`	`Persona lorena = new Persona ("Lorena",21);` `personas.add(lorena);`
Agrega todos los elementos de la colección pasada por parámetro a la colección que recibe el mensaje	`personas.addAll(personas2)`	`personas.addAll(personas2);`
Quita un objeto de la colección. Si la colección es una lista, elimina el primer objeto igual que encuentra.	`personas.remove(roman)`	`personas.remove(roman);`
Quita todos los elementos de la colección pasada como parámetro que esten en la que recibe el mensaje	`personas.removeAll(personas2)`	`personas.removeAll(personas2);`
Quita todos los elementos de la coleccion. La colección queda vacía	`personas.clear()`	`personas.clear();`

CONSULTAS QUE ENTIENDEN TODAS LAS COLECCIONES		
Descripción	**Wollok**	**Java**
Devuelve la cantidad de elementos de la colección	`personas.size()`	`personas.size()`
Devuelve `true` si la colección no tiene elementos	`personas.isEmpty()`	`personas.isEmpty()`
Devuelve `true` si el objeto pasado por parametro está en la colección	`personas.contains(sebastian)`	`personas.contains(sebastian)`
Devuelve la cantidad de veces que un objeto se encuentra en la colección	`personas.occurrencesOf(natalia)`	`Collections.frequency(personas, natalia)`

CONSULTAS QUE ENTIENDEN TODAS LAS COLECCIONES		
Descripción	**Wollok**	**Java**
Devuelve una nueva colección con los elementos pertenecientes a dos colecciones pasadas como parámetros. En Java hay que declarar que tipo de colección resultante se quiere obtener	`personas + personas2`	`Stream.concat(personas.stream(),personas2.stream())` `.collect(Collectors.toSet())`
Devuelve `true` si ambas colecciones tienen los mismos elementos. Si es una lista también se fija que estén en el mismo orden	`personas == personas1`	`personas.equals(personas1)`
Devuelve un elemento cualquiera de la coleccion. Si la colección está vacía produce un error	`personas2.anyOne()`	`personas2.iterator().next()`

Devuelve una lista con el resultado de aplicar una transformación a cada elemento de la colección	`personas.map {p -> p.edad()}`	`personas.stream()`
		`.map(p -> p.edad() )`
		`.collect(Collectors.toSet());`

Devuelve una nueva colección con todos los objetos en la colección original que cumplen con una condición	`personas.filter {p => p.edad() < 18}`	`personas.stream()`
		`.filter(p -> p.edad() < 18)`
		`.collect(Collectors.toSet());`

Devuelve la cantidad de elementos de la colección que cumplen una condición	`personas.count {p => p.edad() < 18}`	`personas.stream()`
		`.filter(p -> p.edad() < 18)`
		`.count();`

Devuelve un elemento de la colección que es el mayor según la ordenación dada	`personas.max {p => p.edad()}`	`personas.stream()`
		`.max (Comparator.comparing(p -> p.edad()))`
		`.get();`

Devuelve un elemento de la colección que es el menor según la ordenación dada	`personas.min {p => p.edad()}`	`personas.stream()`
		`.min (Comparator.comparing(p -> p.edad()))`
		`.get();`

Devuelve la suma de la transformación de cada uno de los elementos de la colección	`personas.sum {p => p.edad()}`	`personas.stream()`
		`.mapToInt(p -> p.edad())`
		`.sum();`

CONSULTAS QUE ENTIENDEN TODAS LAS COLECCIONES		
Descripción	**Wollok**	**Java**
Devuelve true si hay algún objeto en la colección que cumple con la condición dada	`personas.any {p=> p.nombre() == "Juan"}`	`personas.stream()` `.anyMatch( p -> p.nombre() == "Juan");`
Devuelve true si todos los objetos en la colección cumplen con la condición dada	`personas.all {p => p.edad() > 5}`	`personas.stream()` `.allMatch( p -> p.edad() > 5);`

Devuelve un objeto de la colección que cumple con la condición dada. Si no encuentra lanza error	`personas.find {p => p.edad()==18)}`	`personas.stream()`
		`.filter( p -> p.edad() == 18)`
		`.findAny()`
		`.get();`

Devuelve un objeto de la colección que cumple con la condición dada. Si no hay ninguno devuelve el objeto especificado	`personas.findOrDefault`
	`({p => p.edad()==80}, cristina)`
		`personas.stream()`
		`.filter( p -> p.edad() == 80)`
		`.findAny()`
		`.orElse(cristina);`

Dada una colección de cuyos elementos se puede producir una nueva subcolección de elementos, devuelve una colección con los elementos de las subcolecciones	`personas2.flatMap`
	`{p => p.amigos()}.asSet()`
		`personas2.stream()`
		`.map ( p -> p.amigos())`
		`.flatMap(l -> l.stream())`
		`.collect(Collectors.toSet());`

CONSULTAS QUE ENTIENDEN SOLAMENTE LAS LISTAS		
Descripción	**Wollok**	**Java**
Devuelve el primer elemento de la lista	`natalia.amigos().first()`	`natalia.amigos().get(0)`
Devuelve el último elemento de la lista	`natalia.amigos().last()`	`natalia.amigos().get(natalia.amigos().size()-1)`
Devuelve el elemento ubicado en la posición n-esima de la lista. El primer elemento está en la posición 0.	`natalia.amigos().get(1)`	`natalia.amigos().get(1)`
Devuelve la sublista entre una posicion inicial y una final. En Java la posición final no es inclusiva	`natalia.amigos().subList(1,2)`	`natalia.amigos().subList(1,3)`
Devuelve una sublista con los primeros n elementos de la lista	`natalia.amigos().take(2)`	`natalia.amigos().subList(0,2)`

CONSULTAS QUE ENTIENDEN SOLAMENTE LAS LISTAS		
Descripción	**Wollok**	**Java**
Devuelve una sublista sin los primeros n elementos de la lista	`natalia.amigos().drop(2)`	`natalia.amigos().subList(2,natalia.amigos().size())`
Devuelve una lista con los mismos elementos que la original, pero en el orden inverso. [1]	`natalia.amigos().reverse()`	`List<Persona> xs = new ArrayList(natalia.amigos());` `Collections.reverse(xs);`

Dada una lista, devuelve una nueva lista con los elementos ordenados. Se especifica el orden por medio de una expresión lamba de dos parametros que indica cuando un elemento es menor que otro[1]	`natalia.amigos().sortBy`
	`{p1,p2 => p1.edad() <= p2.edad()}`
		`natalia.amigos().stream()`
		`.sorted((p1,p2)->Integer.compare(p1.edad(),p2.edad()))`
		`.collect(Collectors.toList());`

Devuelve un conjunto con todos los elementos de la lista	`natalia.amigos().asSet()`	`new HashSet(natalia.amigos())`

CONSULTAS QUE ENTIENDEN SOLAMENTE LOS CONJUNTOS		
Descripción	**Wollok**	**Java**
Devuelve un nuevo conjunto con los elementos que están en alguno de los dos conjuntos[1].	`personas.union(personas2)`	`Set<Persona> s = new HashSet<Persona>(personas);` `s.addAll(personas2);`
Devuelve un nuevo conjunto con los elementos que están coincidentes de ambos conjuntos[1]	`personas.intersection(personas2)`	`Set<Persona> s = new HashSet<Persona>(personas);` `s.retainAll(personas2);`
Devuelve un conjunto con todos los elementos del receptor salvo aquellos que esten en el pasado por parámetro[1]	`personas.difference(personas2)`	`Set<Persona> s = new HashSet<Persona>(personas);` `s.removeAll(personas2);`
Devuelve una lista con los elementos del conjunto. El orden es incierto	`personas.asList()`	`new HashSet(natalia.amigos())`

[1]Es necesario crear una nueva colección para simular el comportamiento de Wollok porque Java NO genera una nueva copia, sino que modifica la colección original.