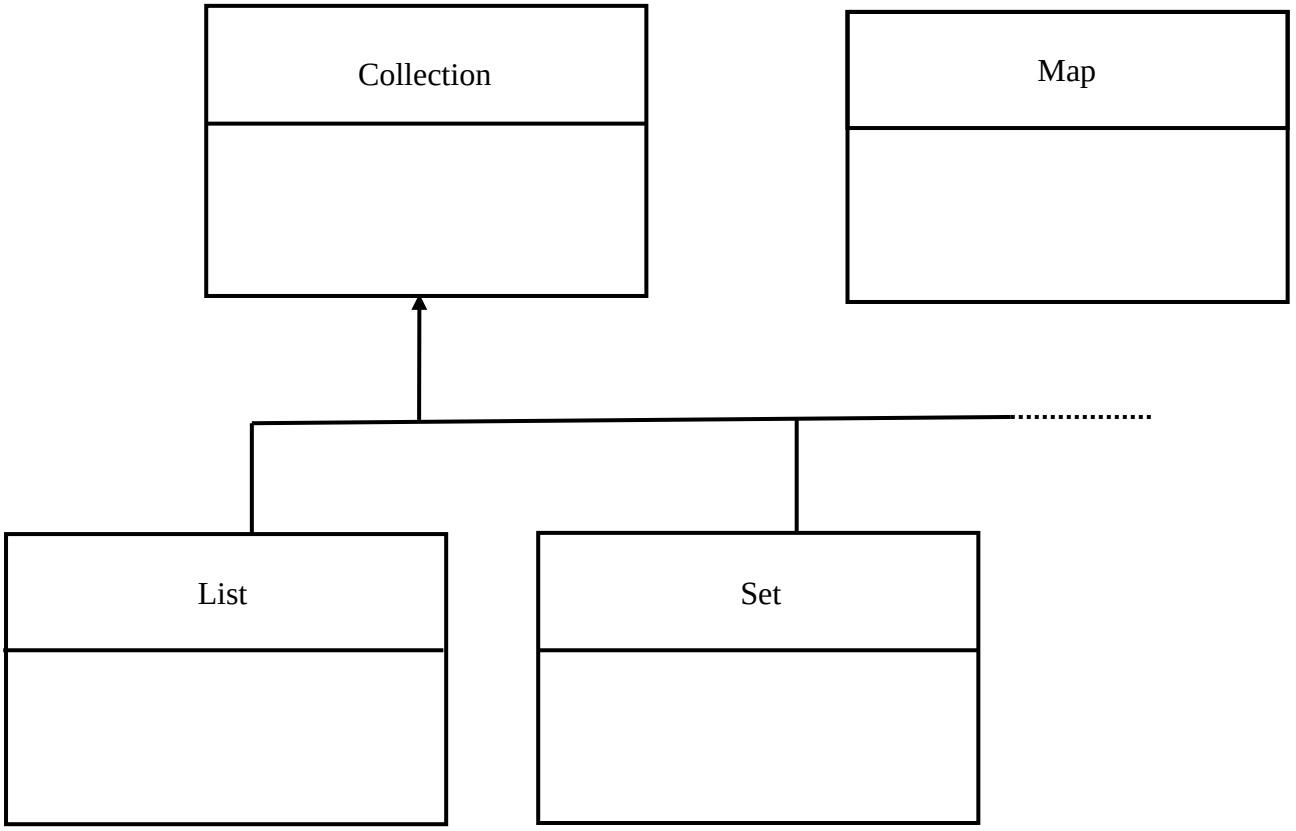
Collections

Una colección representa un grupo de objetos. Cuando trabajamos con un conjunto de elementos muchas veces necesitamos un "almacén" para poder guardarlos. Java emplea una interfaz genérica *collection* con tal fin.
Gracias a esta interfaz podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes: agregar, eliminar, tamaño, etc.
Partiendo de la interfaz *Collection* extienden otra serie de interfaces genéricas, la diferencia entre estas sub-interfaces es que aportan distinta funcionalidad a la interfaz anterior.

Tipos principales de Collections:



List:

Define sucesión de objetos. Permite duplicarlos. A parte de los métodos heredados de *Collection* añade:

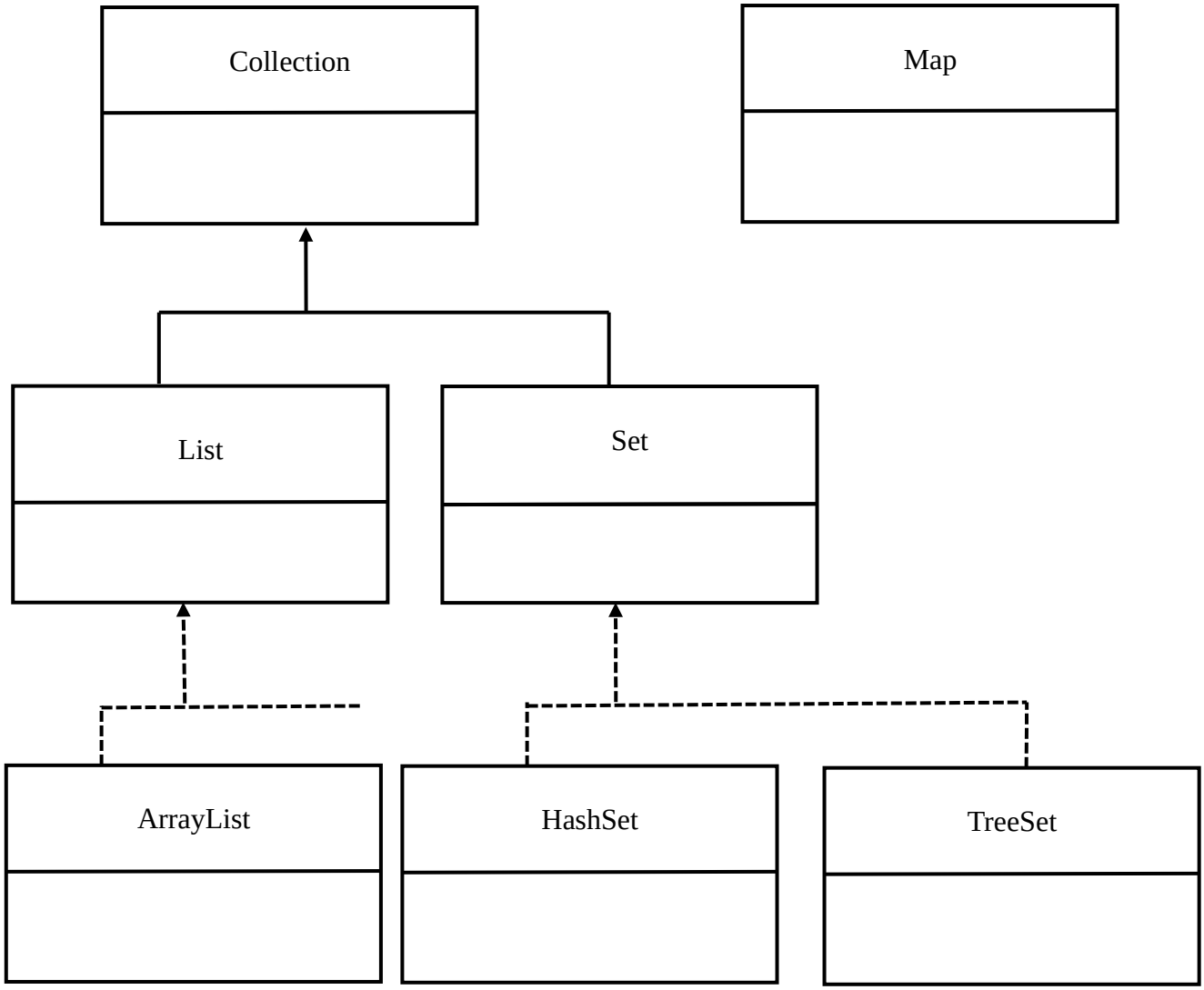
- Acceso posicional de objetos o elementos.
- Búsqueda de elementos.
- Iteración (no hacer hincapié), es decir, que se usaba antes de Java 8.

Implementación típica ***ArrayList.***

Set:

Define una colección que no puede contener elementos duplicados. Contiene únicamente los métodos heredados de *Collection*, añadiendo la restricción de prohibir duplicados.

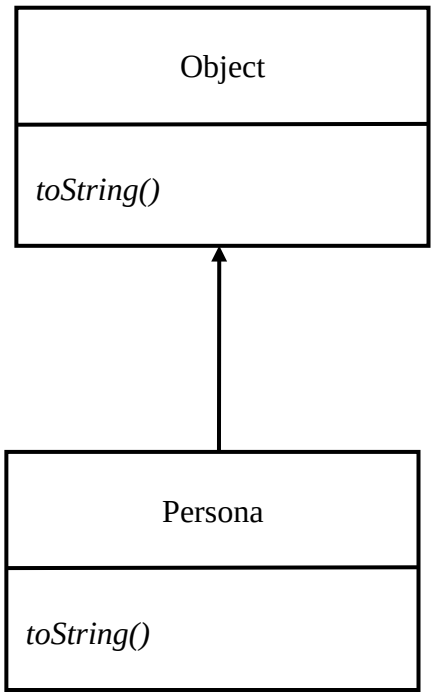
Implementación típica : ***HashSet.***



Map:
No hereda *Collection*, por lo tanto no lo es, pero resulta ser muy útil en muchas ocasiones. La interfaz Map asocia claves de valores, no puede contener claves duplicadas y cada una de dichas claves solo pueden tener un valor asociado como máximo.

Implementación típica ***HashMap***.

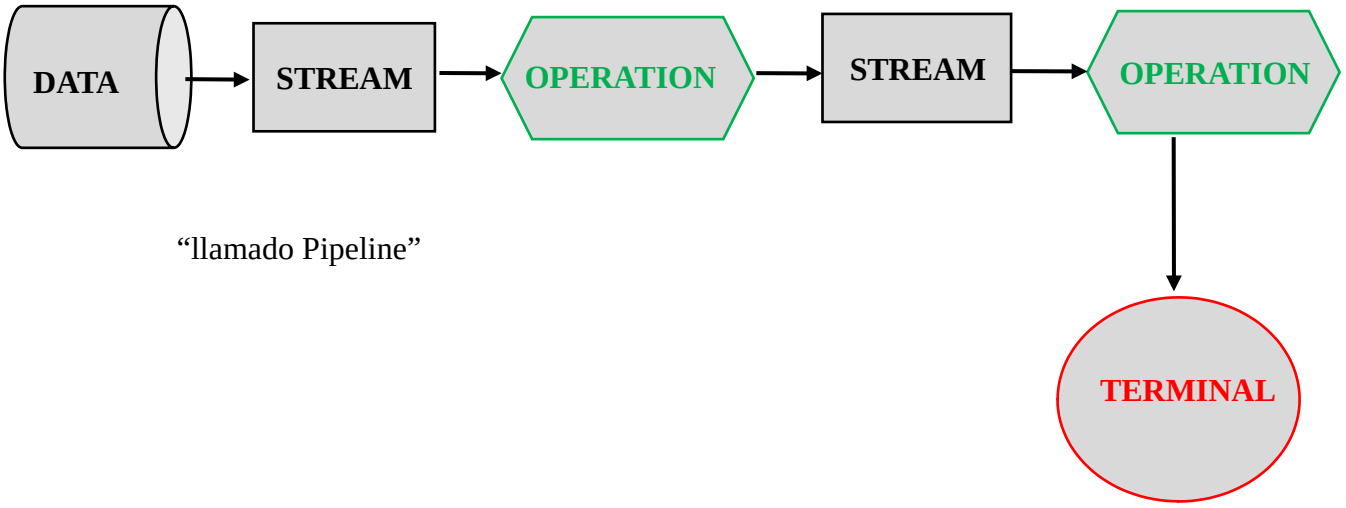
Proyecto: ***ejemplos***
Class: *CollectionEjemplo1*
1 - Explicar redefinición de método *toString()*



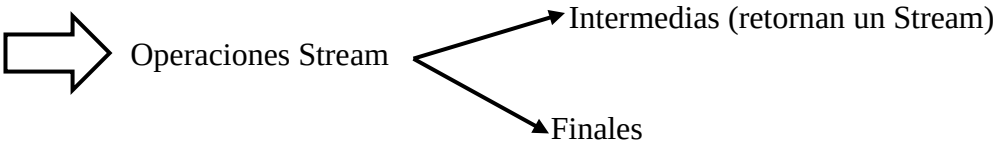
Proyecto: ***ejemplos***
Class: *CollectionEjemplo3*
1 - Explicar redefinición de métodos *equals()* y *hashcode()*, para el caso de Set, bien explicado en el ejemplo.

Java 8 - Stream y lambda

Podemos definir *Streams* como una secuencia de funciones que se ejecutan una detrás de la otra, de forma anidada.
Un *Stream* "parece una Collection" permitiendo realizar operaciones sobre el *Stream*, y cada operación devuelve un nuevo Stream sobre el cual podemos seguir encadenando otras operaciones, hasta llegar a una operación final.



"llamado Pipeline"



Algunas intermedias: filter, sorter
Alguna final: forEach

Ahora bien, antes de continuar y pasar a un ejemplo debemos mencionar Lambda.

Una expresión Lambda tiene parámetros de ésta forma:

(parámetros) ⟶ expresión
(parámetros) ⟶ {Statements}

Ejemplo Lambda expresión:

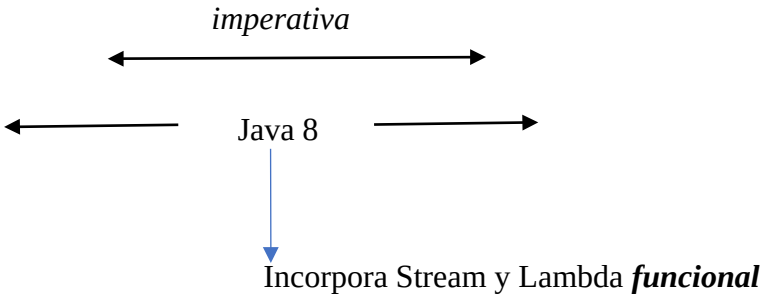
() ⟶ 5 retorna 5
X ⟶ 2*y duplica el valor de x y lo retorna
(x,y) ⟶ x-y toma dos valores y retorna diferencia
(int x, int y) ⟶ x+y toma dos enteros y retorna uno
(string s) ⟶ System.out.print(s) toma sting y lo imprime

- Observar return implícito (1 linea)
- Observar si la expresión Lambda es de 1 linea
- Observar si no se requieren llaves ni return
- Observar si tiene 1 parámetro, no obligatorio paréntesis
- Observar si no tiene parámetro o tiene mas de 1, usa ()

Proyecto: ***teoria-stream***
Class: *Ejemplo1*

Dar concepto: Imperativa

Java 8 mantiene compatibilidad con imperativa



Imperativa: COMO hacerlo

Funcional: QUE quiero