Bloque 3

Colecciones y Mapas

List, Set, SortedSet y Map

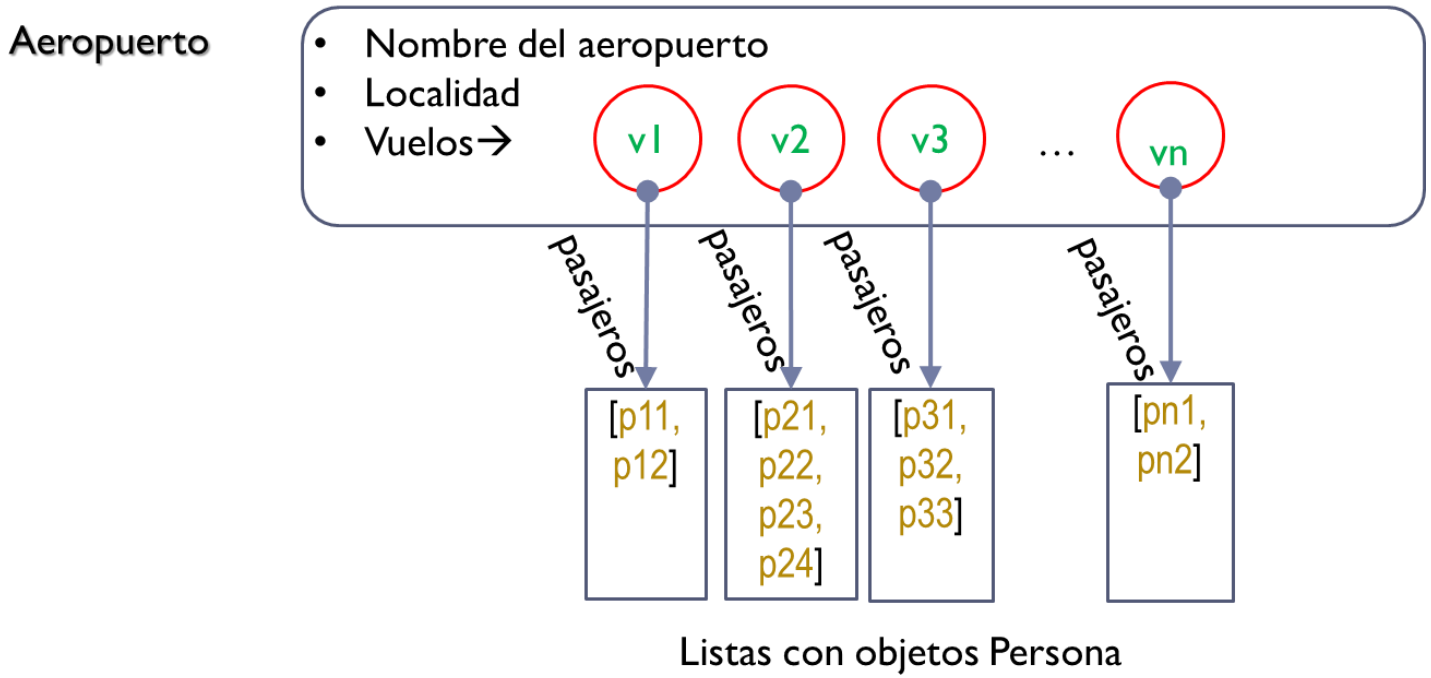
Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

Ejercicio Aeropuerto

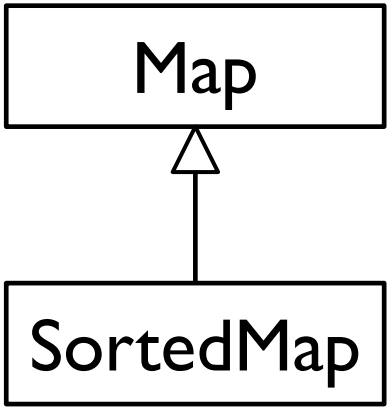
- Realice los ejercicios del ***EnunciadoAeropuerto03***

  Apartados 9 y 10



Tipo Map (*Interfaces*)

Jerarquía de interfaces.



2023/24

Tipo Map (*definición y construcción*)

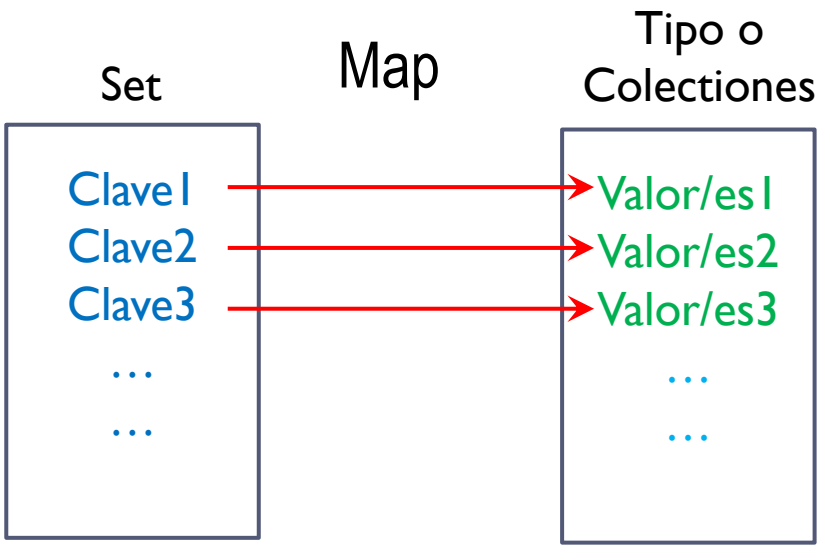
El tipo ***Map*** es una interfaz definida en el paquete "java.util".

Es el mismo concepto de diccionario de Python que permite guardar ***pares*** de datos asociando a una clave un valor o valores.

IMPORTANTE: No es un subtipo de ***Collection*** ni de ***Iterable***

Aunque veremos métodos que permiten recorrelos

Gráficamente:



Tipo Map (*definición y construcción*)

La interfaz *Map* necesita *instanciarse* con dos tipos de objetos:

- El primero para las *claves* (T1) y el segundo con los valores (T2): *Map*<*T1*,*T2*>. (*asocia a los objetos de un tipo T1 los de un tipo T2*)
- Las *claves* están organizadas como un conjunto por lo que no hay claves repetidas. →El método Set<K> *keySet*(), devuelve el conjunto de claves
- Los *valores* están organizados como una colección.→El método Collection<V> *values*(), devuelve una colección de los valores

<u>Ejemplos de instanciación de mapas :</u>

-Asocia al DNI/NIE/Pasaporte de una persona la edad: Map<String, Integer>
-Agrupar vuelos *por* la fecha de salida: Map<LocalDate,List<Vuelo>>
-Agrupar objetos Canción *por* la duración: Map<Duration, List<Cancion>>
-Agrupar número de vuelos *por* precio: Map<Double, Integer>

2023/24

Tipo Map (*definición y construcción*)

Construcción :La interfaz *Map* se puede implementar con la clase *HashMap*, mediante las siguientes sintaxis:

- *Map*<*T1, T2*> mapa=new *HashMap*<*T1, T2*> ();

2023/24

Tipo Map (*Métodos*)

Métodos: Los métodos del tipo *Map* están en

A continuación, los 14 más habituales:

- void *clear*(). Elimina todos los elementos (*pares o entradas*) del mapa.
- boolean *isEmpty*(). Devuelve *true* si el mapa no contiene ningún *par* (es decir, está vacío).
- int *size*(). Devuelve el número de *pares* del mapa.
- boolean *containsKey*(Object key). Devuelve *true* si el mapa contiene la *clave* especificada.
- boolean *containsValue*(Object value). Devuelve *true* si una o más *claves* del mapa tienen asociadas el *valor* especificado.

Tipo Map (*Métodos*)

- V *get*(Object key). Devuelve el *valor* asociado con la *clave* especificada o *null* si esa *clave* no está en el mapa.
- V *put*(key, value). *Inserta o reescribe* el *par* clave-valor en el mapa. Devuelve el *valor* previamente asociado con la *clave* si está ya estaba en el Map o *null*, en caso contrario. Es decir, si ya existe la *clave*, sustituye el valor anterior por el nuevo y devuelve el antiguo.
- void *putAll*(Map<? extends K, ? extends V> **m**). Añade o reescribe en el mapa que invoca al método los *pares* contenidos en **m**. Es equivalente a hacer, uno por uno, *put* de todos los elementos de "m".
- Set<K> *keySet*(). Devuelve un *Set* con las *claves* que contiene el mapa. (*OJO:* es una vista, por lo que, si se cambia un dato en el conjunto, se están cambiando las *claves* del mapa).

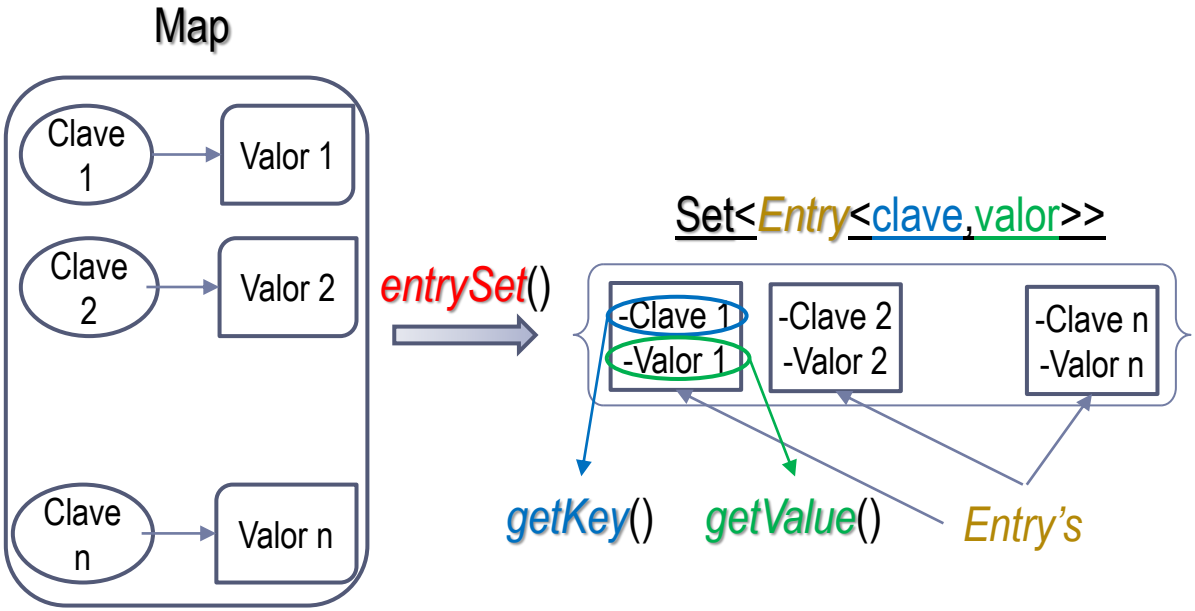
2023/24

Tipo Map (*Métodos*)

- Collection<V> *values*(). Devuelve una *Collection* con los *valores* del mapa. (*OJO* es una vista, por lo que, si se cambia un dato en la colección, se están cambiando los *valores* del mapa).
- V *remove*(Object key). Elimina el *par* que tiene como *clave* el parámetro especificado. Devuelve el *valor* previamente asociado con la *clave*, o *null* si la *clave* no existía.

Tipo Map (*Métodos*)

- Set<*Entry*<K, V>> *entrySet*(). Devuelve un conjunto con objetos tipo *Entry* que a su vez contiene dos atributos con los "*pares*" del mapa. Dado que es una colección se puede recorrer con for (equivalente a *items() de Python*).
- K *getKey*(): Devuelve la parte del "*par*" que corresponde a la *clave*.
- V *getValue*(): Devuelve la parte del "*par*" que corresponde al valor.



Lectura de fichero y carga en una colección
-Factorías-

En general, manejaremos colecciones, generalmente *listas,* que suelen ser el resultado de cargar en ellas los registros de un fichero. Al estilo de lo que hacíamos en Python

Esquema: La clase factoría tiene, al menos, dos métodos principales. Uno público que recibe la *ruta del fichero* y otro privado que convierte una *String* en un objeto del tipo.

```
public class FactoriaTipo {
    public static List<Tipo> LeerObjetosDelTipos(String ruta){
        …
    }
    private static Tipo stringATipo (String línea){
        …
        aquí se "parseará" cada línea del fichero
        …
    }
}
```

Dos métodos

Lectura de fichero y carga en una colección -Factorías-

```
public class FactoriaTipo{

   public static List<Tipo> leerObjetosTipo (String ruta) {
      List<Tipo> res = new ArrayList<Tipo>();
      try {
         List<String> líneas = Files.readALLLines(Paths.get(ruta));
         for (String línea:líneas.subList(n,líneas.size())) {
              res.add(parseaTipo(línea));
         }
      } catch (IOException e) {
           System.out.println("Error al abrir el fichero " + ruta);
           e.printStackTrace();
      }
      return res;
   }
...
```

Número de líneas de cabecera

El método *readAllLines* lee de una vez todas las línea y devuelve una lista de String en que cada registro se almacena en un elemento de la lista.

2023/24

Lectura de fichero y carga en una colección
-Factorías-

```
public class FactoriaTipo {
...

  private static Tipo parseaTipo(String línea) {
    Checkers.checkNoNull(línea);
    String[] trozos=línea.split(";");
    Checkers.check("Las líneas no se trocean bien",
                                   trozos.length==número de campos);

    Tipo1 campo1=conversión al Tipo1 del trozos[0].trim();
    Tipo2 campo2=conversión al Tipo2 del trozos[1].trim();
    Tipo3 campo3=conversión al Tipo3 del trozos[2].trim();
    ...
    return new Tipo(campo1, campo2, campo3,...);
  }
}
```

Se construye el objeto a partir del constructor con parámetros

2023/24

Lectura de fichero y carga en una colección
-Factorías-

```
public class FactoríaAnimal {

  public static List<Animal> leerAnimales (String ruta) {
    List<Animal> res = new ArrayList<Animal>();
    try {
      List<String> líneas = Files.readAllLines(Paths.get(ruta));
      for (String línea:líneas.subList(1,líneas.size())) {
        res.add(parseaAnimal(línea));
      }
    } catch (IOException e) {
        System.out.println("Error al abrir el fichero " + ruta);
        e.printStackTrace();
    }
    return res;
  }
...
```

2023/24

Lectura de fichero y carga en una colección
-Factorías-

```
public class FactoríaAnimal{
...
  private static Animal parseaAnimal(String línea) {
    Checkers.checkNoNull(línea);
    String[] trozos=línea.split(";");
    Checkers.check("Las líneas no se trocean bien",
                          trozos.length==5);
    Familia familia=Familia.valueOf(trozos[0].trim());
    String nombre=trozos[1].trim();
    Double pesoMedio=Double.valueOf(trozos[2].trim());
    Integer edadMedia=Integer.valueOf(trozos[3].trim());
    Boolean doméstico=(trozos[4].trim().equals("SI"));
    return new Animal(familia, nombre, pesoMedio, edadMedia,
                doméstico);
    }
}
```