Bloque 3

Colecciones y Mapas

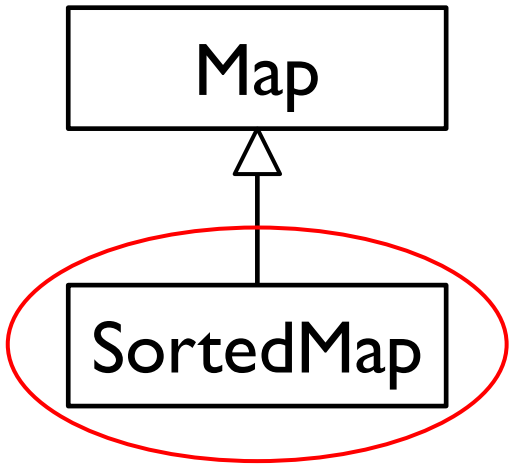
List, Set, SortedSet y Map

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

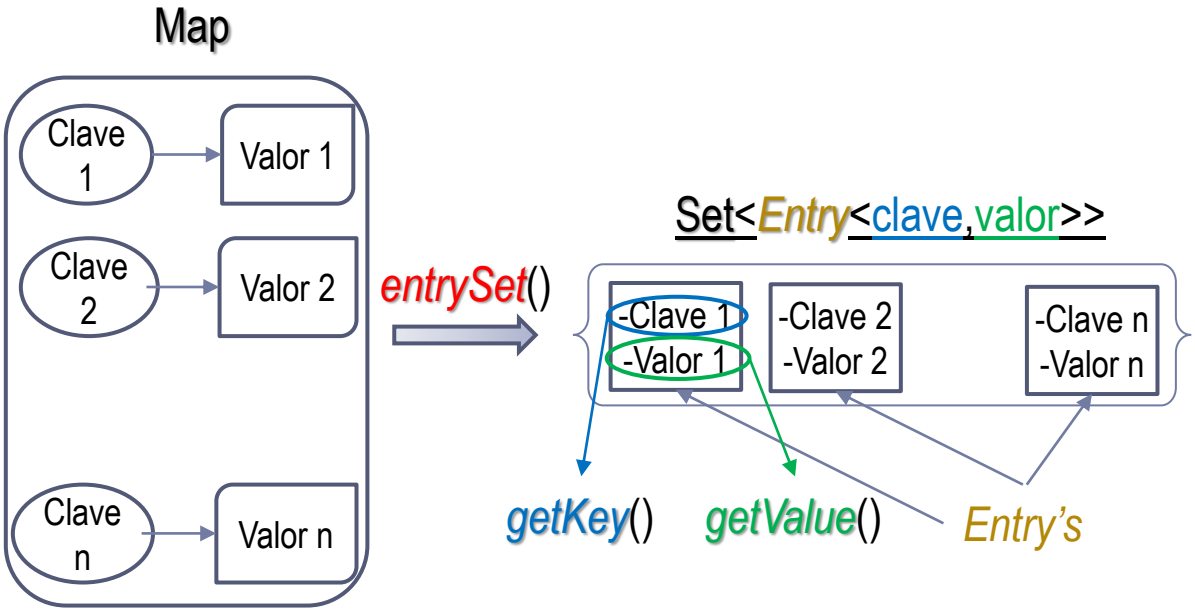
Tipo Map (*Interfaces*)

Jerarquía de interfaces.



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.



2023/24

Tipo SortedMap (*definición y construcción*)

Construcción

Un *SortedMap* es una interfaz subtipo de *Map* que se implementa con la clase *TreeMap*:

- *SortedMap*<T1, T2> mapa=new *TreeMap*<T1, T2> (); crea un mapa en el que las *claves* están ordenadas por el orden natural de los objetos de tipo T1.
- *SortedMap*<T1, T2> mapa=new *TreeMap*<T1, T2> (Comparator **cmp**); crea un mapa en el que las *claves* están ordenadas por el orden inducido por el objeto comparador **cmp** de los objetos de tipo T1.

Cuando recorremos el conjunto que devuelve el método *keySet* las claves se obtienen ordenadas según el orden inducido por el orden natural o por el comparador, según el caso.

Esquema de construcción de un Mapa

Esquema de Construcción de un mapa a partir de una colección de objetos (es igual que los diccionarios en Python)

1. Se crea un mapa vacío. Por ejemplo, "*m*"
2. Se recorre cada elemento de la colección. Para cada clave:
   - Si la clave NO ESTA en el mapa (*m.containsKey(clave)*)
     a) Inicializar valor
     b) Añadir al mapa el *par: m.put*(*clave*, *valor*)
   - Si la clave YA ESTÁ
     a) Obtener el valor asociado a esa clave: *m.get(clave)*
     b) Actualizar el valor con el *nuevoValor.*
     c) Reescribir en el mapa el nuevo *par: m.put*(*clave*, *nuevoValor*)

2023/24

Tipo Map
(*Ejemplo: ¿Cuántos Vuelos a cada destino?*)

```
public static void main(String[] args) {
  List<Vuelo>lv=…(es una lista de vuelos)

  Map<String,Integer> mapa=new HashMap<String,Integer>();
  for (Vuelo v:lv) {
    String clave=v.getDestino();
    if(!mapa.containsKey(clave)) {
      mapa.put(clave, 1);
    }
    else {
      Integer valor= mapa.get(clave);
      valor++;
      mapa.put(clave, valor);
      }
  }
…(¡Hecho el mapa!)
```

Tipo Map
(*Ejemplo: ¿Cuántos Vuelos a cada destino?*)

```
//Ver resultados del mapa

    for (Entry<String,Integer> par:mapa.entrySet()) {
        System.out.println(par);
        System.out.println(par.getKey()+"--->"+ par.getValue());
    }
}
```

Resultado para cada print:

```
Oviedo=4
Las Palmas=10
Barcelona=7
Madrid=5
Málaga=6
Valencia=5
```

```
Oviedo--->4
Las Palmas--->10
Barcelona--->7
Madrid--->5
Málaga--->6
Valencia--->5
```

Introducción a Collection (recorridos)

2023/24

Sentencia break, Esquemas de "todos cumplen"

La sentencia ***break*** está pensada para interrumpir (terminar) un bucle for o while.

Esquema que permite ver ***si todos*** los elementos de una colección cumplen una determinada *condición*:

```
public static boolean todosCumplenQue...(Collection<Tipo> c){
     boolean res=true;   //Se parte de la presunción que todos cumplen
     for (Tipo elemento : c){
         if (elemento no cumple){   //Si se encuentra uno que ya no cumple
             res=false;
             break;
         }
     }
     return res;
}
```

Introducción a Collection (recorridos)

Sentencia break, Esquemas de "existe alguno"

La sentencia *break* está pensada para interrumpir (terminar) un bucle for o while.

Esquema que permite ver *si existe alguno* de los elementos de una colección que cumple una determinada *condición*:

```
public static boolean existeAlgunoQue...(Collection <Tipo> c){
     boolean res=false;   //Se parte de la presunción que no hay ninguno
     for (Tipo elemento : c){
         if (elemento cumple){   //Si se encuentra uno que si cumple
             res=true;
             break;
         }
     }
     return res;
}
```

Introducción a Collection (recorridos)

Cálculo del máximo

Esquema que permite ver *encontrar el máximo* de los elementos de una colección por determinada *propiedad*:

```
public static Tipo máximoPor…(Collection <Tipo> c){
     Tipo res=null;   //se parte de que aún no hay máximo
     for (Tipo elemento : c){
         if (res==null || res.propiedad < elemento.propiedad){
             res=elemento;
          }
     }
     return res;
}
```

Nota.- Si se pidiese alguna *propiedad* del objeto que tiene el máximo, hay que cambiar el *Tipo* del método al de la propiedad a devolver y la última sentencia sería:

```
return res.propiedad;
```

Introducción a Collection (recorridos)

Cálculo del mínimo

Esquema que permite ver *encontrar el mínimo* de los elementos de una colección por determinada *propiedad*:

```
public static Tipo mínimoPor…(Collection <Tipo> c){
     Tipo res=null;   //se parte de que aún no hay mínimo
     for (Tipo elemento : c){
         if (res==null || res.propiedad > elemento.propiedad){
             res=elemento;
          }
     }
     return res;
}
```

Nota.- Si se pidiese alguna *propiedad* del objeto que tiene el mínimo, hay que cambiar el *Tipo* del método al de la propiedad a devolver y la última sentencia sería:

```
return res.propiedad;
```

Introducción a Collection (recorridos)

Cálculo del máximo (ejemplo)

Encontrar la persona de más edad

```
public static Persona másEdad (Collection <Persona> personas){
     Persona res=null;    //se parte de que aún no hay ninguna persona
     for (Persona p : personas){
          if (res==null || res.getEdad()<p.getEdad()){
               res=p;
          }
     }
     return res;
}
```

Nota.- Si se pidiese alguna *propiedad* (por ejemplo, *el peso*) del objeto que tiene el máximo, hay que cambiar ***en la cabecera del método*** el tipo: *Persona* por *Double* y la sentencia `return res.getPeso();`

2023/24

Ejercicio Aeropuerto

- Realice los ejercicios del ***EnunciadoAeropuerto04***

  Apartados 10, 11 y 12