UT07. 05-COLECCIONES EN JAVA. OTRAS

Programación de 1DAW

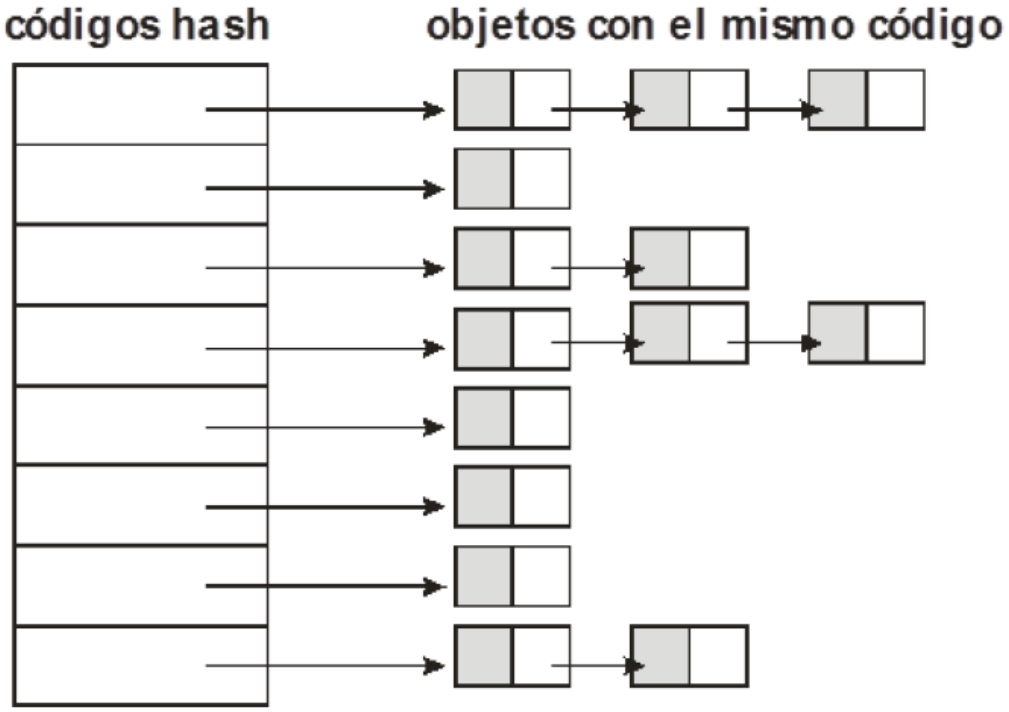
C.I.F.P. Carlos III - Cartagena

Set. Colección sin duplicados

- Interfaz **Set**
  - Para listas dinámicas de elementos sin duplicados.
  - Deriva de **Collection**
  - Sin duplicados: método **equals** → redefinir.

HashSet. Colección sin duplicados.

- Clase **HashSet**.
  - Implementa la clase **Set**.
  - Tiene los métodos de la clase **List**.
  - Internamente dispone de una tabla de tipo hash que asocian claves a conjunto de valores.
  - Tiene valores duplicados pero no se tiene en cuenta el orden



Colecciones sin duplicados

- Construcción **HashSet**:
  - Tamaño inicial de tabla (por defecto 16)
  - Factor de carga (por defecto 75%): indica cuándo se debe redimensionar el array.

Constructor	Uso
HashSet()	Construye una nueva lista vacía con tamaño inicial 16 y un factor de carga de 0,75
HashSet (Collection <? Extends E> lista)	Crea una lista Set a partir de la colección indicada
HashSet (int capacidad)	Crea una lista con el tamaño indicado y un factor de 0,16
HashSet (int capacidad , double factor)	Crea una lista con la capacidad y el factor indicados

Colecciones sin duplicados.

- **HashSet**.
  - Necesario redefinir:
    - **equals**: comprueba si dos elementos son iguales
    - **hashCode** (identificador de la lista hashSet).
  - Si dos objetos son iguales (equals) ➔ hashCode debe devolver el mismo entero.
  - Si dos objetos no son iguales (equals) ➔ hashCode puede o no devolver el mismo entero.
  - Diseñado específicamente para búsquedas rápidas.
  - Puede ser necesario implementar el interfaz **Comparable ➔ compareTo**.

```java
    public static void main(String[] args) {
        HashSet<Alumno> l = new HashSet<>();

        Alumno a1=new Alumno ("alberto",7);
        Alumno a2=new Alumno ("alberto",6);
        Alumno a3=new Alumno ("alberto",7);
        Alumno a4=new Alumno ("adrian",7);
        Alumno a5=new Alumno ("alberto",7);
        Alumno a6=new Alumno ("adrian",8);

        l.add(a1);
        l.add(a2);
        l.add(a3);
        l.add(a4);
        l.add(a5);
        l.add(a6);
        for(Alumno a:l){
            System.out.println(a);
        }
    }

}
```

```
tput - PruebaHashSet (run)
adrian.8.0
alberto.7.0
adrian.7.0
alberto.6.0
BUILD SUCCESSFUL (total time: 3 seconds)
```

```java
public class Alumno {
    protected String nombre;
    protected double nota;

    public Alumno(String nombre, double nota) {
        this.nombre = nombre;
        this.nota = nota;
    }

    @Override
    public boolean equals (Object obj){
        boolean iguales = false;
        if(obj instanceof Alumno){
            Alumno a = (Alumno)obj;
            iguales = a.nombre.equals(nombre) && a.nota==nota;
        }
        return iguales;
    }

    @Override
    public int hashCode(){
        return nombre.hashCode()+(int)nota*10000;
    }

    @Override
    public String toString(){
        return nombre+"."+nota;
    }
}
```

Mapas. Interfaz Map

- Definen colecciones de elementos que poseen pares de datos **clave-valor**.
- Se usa para localizar valores en función de la clave que poseen.
- No deriva de Collection → no usa iteradores.
- No permiten insertar objetos nulos → provocan excepciones de tipo **NullPointerException**
- Interfaz **Map<k, v>**.
  - Es la raíz de todas las clases de mapas.
  - K: tipo de datos de la clave.
  - V: tipo de valores.
- No se puede repetir el conjunto de claves

Método	Uso
V **get** (K clave)	Devuelve el objeto que posee la clave Indicada
V **put** (Object clave, V valor)	Coloca el par clave-valor en el mapa (asociando la clave a dicho valor). Si la clave ya existiera, sobrescribe el anterior valor y devuelve el objeto antiguo. Si esa clave no aparecía en la lista, devuelve null
V **remove** (Object clave)	Elimina de la lista el valor asociado a esa clave. Devuelve el valor que tuviera asociado esa clave o null si esa clave no existe en el mapa.
boolean **containsKey** (Object clave)	Indica si el mapa posee la clave señalada
boolean **containsValue** (Object valor)	Indica si el mapa posee el valor señalado
void **putAll** (Map<?extends K, extends V> mapa)	Añade todo el mapa indicado, al mapa actual
Set<K> **keySet** ()	Obtiene un objeto Set creado a partir de las claves del mapa
Collection<V> **values** ()	Obtiene la colección de valores del mapa, permite utilizar el HashMap como si fuera una lista normal al estilo de la clase Collection (por lo tanto se permite Recorrer cada elemento de la lista con un iterador)
int **size** ()	Devuelve el número de pares clave-valor del mapa
Set<Map.Entry <K,V>> **entrySet** ()	Devuelve una lista formada por objetos Map.Entry
void **clear** ()	Elimina todos los objetos del mapa

Interfaz Map.Entry

- La **interfaz Map.Entry** se define de forma interna a la interfaz Map y representa un objeto de par clave/valor. Es decir mediante esta interfaz podemos trabajar con una entrada del mapa.

método	uso
K **getKey** ()	Obtiene la clave del elemento actual Map.Entry
V **getValue** ()	Obtiene el valor
V **setValue** (V valor)	Cambia el valor y devuelve el valor anterior del objeto actual
boolean **equals** (Object obj)	Devuelve verdadero si el objeto es un Map.Entry cuyos pares clave-valor son iguales que los del Map.Entry actual