TEMA 10 COLECCIONES

INDICE

1 - INTRODUCCIÓN

TIPOS DE SISTEMAS DE ALMACENAMIENTO

ESTRUCTURAS ESTATICAS: Se establece un tamaño en el momento de la creación o definición y su tamaño no puede variar después. Ejemplos arrays y las matrices.

ESTRUCTURAS DINAMICAS (contenedores): El tamaño es variable pudiendo crecer o decrecer. Ejemplos: listas, árboles, conjuntos y algunos tipos de cadenas de caracteres

CONTENEDORES

COLECCIONES: Grupos de elementos individuales

- **List** – no contiene reglas y es conjunto de elementos que siguen una secuencia concreta.
- **HashSet** – Conjunto de Objetos a partir de una matriz que contiene un índice a partir del código hash del objeto. Es una colección de elementos únicos, pero desordenados. Implementa elementos serializables y clonables.
- **Conjunto** – permite la inserción de un elemento de cada tipo.

MAPAS: Grupo de pares objeto-valor. Devuelven un conjunto (set) que no puede tener elementos duplicados. Se les llaman Arrays asociativos.

- **HashMap** – Esctructura que almacena pares clave-valor. Permite nulos y no se encuentra ordenada.

METODOS SOBRECARGADOS

- **fill()** - rellenan colecciones y mapas.
- **toString()** - método impresor bastante legible.

- Las colecciones de imprimen entre corchetes**{}**

- Los mapas se imprimen entre llaves asociados el par con el signo igual **(clave = valor)**

2 – COLECCIONES

INTERFACES

- En Java no es una clase sino un **interface**.
- Una **interfaz** es un elemento del lenguaje que actúa como plantilla que permite preestablecer el contenido de una o varias clases.
- Una **interfaz** contiene una colección de métodos abstractos y propiedades constantes que permiten activar las herencia múltiple:
    - Se pueden declarar mediante **static**.
    - Pueden implementar otras interfaces.

EJEMPLO DE INTERFACES

```
public interface Vehiculo {
      public String matricula = “”;
public void arrancar();
public void detener();
}
public class Coche implements Vehiculo {
public void arrancar(){
      System.out.println(“arrancando motor…”); }
      public void detener(){
      System.out.println(“deteniendo motor…”); }
}
```

- Se trata de una clase declarativa que ofrece una plantilla para que cada jugador pueda diseñar su propio coche.
- Por debajo del **<interface List>** hace referencia a un conjunto de elementos accesibles por la posición.
- El interface incluye el método **get(i)**, donde se accede a cualquier elemento de la lista.
- La clase **AbstractList** es una clase abstracta que contiene las clases instánciales concretas:
    1. **ArrayList**
    2. **LinkedList**
    3. **Vector**
    - Hereda una clase que define el concepto de pilas o **stack**.

JAVA SET

- Es otra interface de Java que define un conjunto de elementos que no contiene ningún elemento repetido.
- Contienen las clases instánciales concretas:
  1. **HashSet**
  2. **TreeSet**

JAVA Queue

- Las colas tienen la peculiaridad de que el primer elemento que entra es el primero que sale.
- Contienen las clases instánciales concretas:
  1. **DeQueue**
  2. **BlockingQueue**

INTERFACE JAVA MAP

- Es un interface de mapas o diccionario, que no están relacionados con las colecciones.
- Definen un par clave-valor, equivale a una palabra y su descripción.
- Tienen 4 clases:
  1. **HashMap**
  2. **HashTable**
  3. **TreeMap**
  4. **Properties**

JAVA 8: mejoras

- Añade el interface Iterable por encima de la interface Collection.
- Ventaja: Permite integrar de forma más natural la programación en el lenguaje JAVA.

Métodos de Iterable:

1. **Iterator**
2. **forEach**
3. **SplitIterator**

Afecta a todas las API de colecciones de Java, incluidos mapas o diccionarios.

ITERACIONES <iterators>

¿Qué es un iterador?

Es un patrón de diseño de comportamiento que permite el recorrido secuencial por una estructura de datos compleja sin exponer sus detalles internos.

Nota: No confundirlo con **interación**, que es la repetición de un bloque de sentencias un número determinado de veces o hasta que se cumpla una condición.

¿Cómo funciona un iterador?

Los iteradores implementan varios algoritmos de recorrido. Varios objetos iteradores pueden recorrer la misma colección al mismo tiempo.

Además de implementar el propio algoritmo, un objeto iterador encapsula todos los detalles del recorrido, como la posición actual y cuantos elementos quedan hasta el final.

COLECCIONES

Métodos de Collection:

`Iterator<E> iterator()` -> crea un iterador para recorrer los elementos de la colección

`object [ ] toArray()` -> Pasa una colección a un array de objetos de tipo Object

`addAll (Collection<? Extends E> c)` -> Añade todos los elementos de una colección a otra, siempre que sean del mismo tipo.

`containsAll(Collection<?> c)` -> Permite comprobar si una colección contiene los elementos existentes en otra, devolviendo TRUE.

`boolean removeAll(Collection<?>` c) -> Borra todos los elementos pasados como parámetros de nuestra colección.

`boolean retainAll(Collection<?> c)` -> Elimina los elementos que no están en la colección pasada como argumento.

Métodos de ArrayList:

MÉTODO	DESCRIPCIÓN
`size()`	Devuelve el número de elementos (int)
`add(X)`	Añade el objeto X al final. Devuelve true.
`add(posición, X)`	Inserta el objeto X en la posición indicada.
`get(posicion)`	Devuelve el elemento que está en la posición indicada.
`remove(posicion)`	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
`remove(X)`	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
`clear()`	Elimina todos los elementos.
`set(posición, X)`	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
`contains(X)`	Comprueba si la colección contiene al objeto X. Devuelve true o false.
`indexOf(X)`	Devuelve la posición del objeto X. Si no existe devuelve -1