UNIDAD I2: COLECCIONES

CLASE 3: INTERFACES LIST Y SET / CLASE COLECTION

INTERFAZ LIST

La interfaz List hereda de Collection y es de esta que heredan las clases ArrayList y LinkedList

Lo que identifica a la interfaz List y sus implementaciones es el acceso directo a posiciones de la colección.

INTERFAZ LIST

Sus elementos pueden estar repetidos y el orden en el que se encuentran es relevante.

Podemos utilizar todos los métodos listados en la presentación anterior, y además nuevos métodos concretos de List.

INTERFAZ LIST

- **E get(int indice):** Devuelve el elemento que ocupa esa posición en la lista (empezando desde el 0)
- **E set(int indice, E elem):** Guarda elem en la posición indice machacando el valor que hubiera previamente.
- **E remove(int indice):** Elimina el elemento que esté en la posición *indice* y lo devuelve.

INTERFAZ LIST

- **void add(int indice, E elem):** Inserta *elem* en la posición *indice*, todos los elementos que se encuentren en esa posición, y en adelante, se mueven para hacer hueco.
- **boolean addAll(int indice, Collection<? extends E> c):** Inserta todos los elementos de la colección, en el mismo orden, comenzando por *indice*.

INTERFAZ LIST

Además tiene funciones de búsqueda, ordenación y comparación:

- **int indexOf**(**Object ob**): Devuelve la primera instancia que encuentre en la lista, sino -1.

- **int lastIndexOf**(**Object ob**): Mismo procedimiento que indexOf pero comenzando por el final.

INTERFAZ LIST

- **boolean equals**(**Object otraLista**): Compara las dos listas. Devuelve true si ambas tienen los mismos elementos, en el mismo orden y con las mismas repeticiones.

- **void sort**(**Comparator<? super E>**): Ordena la lista según el comparador indicado (puede ser comparador de tipo E o una clase superior), para ordenar por orden natural se utiliza Comparator.naturalOrder( );

INTERFAZ LIST: IMPLEMENTACIONES

ArrayList y LinkedList: Ambas listas cuya diferencia radica en implementación interna y rendimiento.

ArrayList: Todos los elementos están seguidos en memoria

LinkedList: Los elementos están distribuidos por la memoria

INTERFAZ SET

La interfaz Set trata los datos como un conjunto matemático, no existen las repeticiones ni un orden.

No tiene más métodos disponibles que los que ya vimos en Collection, añade la restricción de no duplicados.

INTERFAZ SET: IMPLEMENTACIONES

- **HashSet**: No garantiza ningún orden en la inserción.
- **TreeSet**: Garantiza orden basado en el valor de la inserción (comparación por compareTo)
- **LinkedHashSet**: Inserta los elementos al final de la lista.

CONVERSIONES ENTRE COLECCIONES

Existe la posibilidad de crear unas colecciones a partir de otra teniendo en cuenta sus constructores.

CONVERSIONES ENTRE COLECCIONES

HashSet o LinkedHashSet (conjuntos no ordenados) a TreeSet (conjuntos ordenados)

1. Se construye un TreeSet con el criterio de ordenación decidido (como parámetro del constructor del TreeSet) y se añaden todos los valores con un addAll( )

CONVERSIONES ENTRE COLECCIONES

```
public static void main(String[] args) {
    Set<Integer> conjEnt = new LinkedHashSet<>();
    conjEnt.add(4);
    conjEnt.add(1);
    conjEnt.add(5);
    conjEnt.add(10);
    conjEnt.add(3);
    System.out.println(conjEnt);
    Set<Integer> conjEntOrd = new TreeSet<>();
    conjEntOrd.addAll(conjEnt);
    System.out.println(conjEntOrd);
}
```

```
[4, 1, 5, 10, 3]
```

```
[1, 3, 4, 5, 10]
```

CONVERSIONES ENTRE COLECCIONES

HashSet o LinkedHashSet (conjuntos no ordenados) a TreeSet (conjuntos ordenados)

2. Se construye un TreeSet pasando como parámetro otro conjunto.

CONVERSIONES ENTRE COLECCIONES

```
public static void main(String[] args) {
    Set<Integer> conjEnt = new LinkedHashSet<>();
    conjEnt.add(4);
    conjEnt.add(1);
    conjEnt.add(5);
    conjEnt.add(10);
    conjEnt.add(3);
    System.out.println(conjEnt);
    Set<Integer> conjEntOrd = new TreeSet<>(conjEnt);
    System.out.println(conjEntOrd);
}
```

```
[4, 1, 5, 10, 3]
[1, 3, 4, 5, 10]
```

CONVERSIONES ENTRE COLECCIONES

```java
List<Integer> lista = new ArrayList<>();
lista.add(4);
lista.add(1);
lista.add(5);
lista.add(10);
lista.add(3);
lista.add(5);
lista.add(1);
System.out.println(lista);
Set<Integer> conjunto = new HashSet<>(lista);
System.out.println(conjunto);
lista = new ArrayList<>(conjunto);
System.out.println(lista);
```

Utilizando constructores es posible hacer conversiones entre colecciones.

```
[4, 1, 5, 10, 3, 5, 1]
[1, 3, 4, 5, 10]
[1, 3, 4, 5, 10]
```

CONVERSIONES ENTRE COLECCIONES

Utilizando una variable Collection podemos utilizar todos los métodos de la interfaz.

```
Collection<Integer> coleccion = new ArrayList<>();
coleccion.add(5);
coleccion.add(10);
coleccion.add(3);
coleccion.add(5);
coleccion.add(1);
coleccion = new LinkedHashSet<>(coleccion);
coleccion = new ArrayList<>(coleccion);
System.out.println(coleccion);
```

```
[5, 10, 3, 1]
```

CLASE COLLECTIONS

La clase **Collections** reúne una serie de utilidades en forma de métodos estáticos que trabajan con tipos genéricos.

CLASE COLLECTIONS

- **Métodos de ordenación:**
  - **static <T extends Comparable<? super T>> void sort** (**List <T> lista**)
    - Ordena la lista pasada por parámetro teniendo en cuenta su orden natural
  - **static <T extends Comparable<? super T>> void sort** (**List <T> lista, Comparator<? super T> c**)
    - Ordena la lista pasada por parámetro teniendo en cuenta el comparador pasado por el segundo parámetro

CLASE COLLECTIONS

- **Métodos de búsqueda:**
  - **static <T> int binarySearch** (**List <? extends Comparable<? super T>> lista, T clave**)
    - Hace una búsqueda binaria de clave en lista, esta lista debe estar ordenada previamente
    - Precisa la implementación de Comparable dentro de T

CLASE COLLECTIONS

- **Métodos de búsqueda:**
  - **static <T> int binarySearch (List <? extends T> lista, T clave, Comparator<? super T> c)**
    - Hace una búsqueda binaria de clave en lista, esta lista debe estar ordenada previamente
    - Como tercer parámetro se pasa el comparador con el que se ha ordenado la lista.

CLASE COLLECTIONS

- BinarySearch es muy eficiente y tiene tiempos de búsqueda muy cortos.
- El inconveniente es que la lista debe estar ordenada.
- Sólo es útil si vamos a realizar varias búsquedas bajo el mismo criterio.

CLASE COLLECTIONS

- **Métodos para la manipulación de datos:**
  - **static void swap (List <?> lista, int i, int j)**
    - Intercambia los elementos de los índices i y j entre sí.

```
List<Integer> datos = new ArrayList<>();
datos.add(1); //indice 0
datos.add(2);
datos.add(3);
datos.add(4); //indice 3
datos.add(5);
datos.add(6);
Collections.swap(datos, 0, 3);
System.out.println(datos);
```

```
[4, 2, 3, 1, 5, 6]
```

CLASE COLLECTIONS

- **Métodos para la manipulación de datos:**
  - **static <T> boolean replaceAll (List <?> lista, T antiguo, T nuevo)**
    - Remplaza todas las instancias del elemento antiguo por el nuevo

```
Collections.replaceAll(datos, 4, 100);
System.out.println(datos);
```

```
[4, 2, 3, 1, 5, 6]
```

```
[100, 2, 3, 1, 5, 6]
```

CLASE COLLECTIONS

- **Métodos para la manipulación de datos:**
  - **static <T> void fill (List <? super T> lista, T valor_relleno)**
    - Sustituye todos los elementos de la lista por el *valor_relleno*

```
Collections.fill(datos, 0);
System.out.println(datos);
```

```
[0, 0, 0, 0, 0, 0]
```

CLASE COLLECTIONS

- **Métodos para la manipulación de datos:**
  - **static <T> void copy (List <? super T> destino, List <? extends T> origen)**
    - Copia todos los valores de la lista origen en la lista destino, sobrescribe los valores que ya estuvieran. La lista destino deberá tener mínimo el tamaño de la tabla origen

```
[1, 2, 3, 0, 0, 0]
```

```java
List<Integer> otros = new ArrayList<>();
otros.add(1); otros.add(2); otros.add(3);
Collections.copy(datos, otros);
System.out.println(datos);
```

CLASE COLLECTIONS

- **Otros métodos de utilidad:**
  - **static void shuffle(List <?> lista)**
    - Desordena los elementos de la lista mediante una fórmula propia de Java

```java
List<Integer> datos = new ArrayList<>();
datos.add(1);
datos.add(2);
datos.add(3);
datos.add(4);
datos.add(5);
datos.add(6);
Collections.shuffle(datos);
System.out.println(datos);
```

```
[5, 1, 2, 4, 6, 3]
```

CLASE COLLECTIONS

- **Otros métodos de utilidad:**
  - **static int frequency(Collection <?> col, Object ob)**
    - Nos devuelve el número de veces que aparece un elemento en una colección.

```
int rep = Collections.frequency(datos, 0);
System.out.println(datos);
System.out.println("Cantidad de veces que se repite el 0: "+rep);
```

```
[1, 2, 3, 0, 0, 0]
Cantidad de veces que se repite el 0: 3
```

CLASE COLLECTIONS

- **Otros métodos de utilidad:**
  - **static <T extends Comparable<? super T>> T max (Collection<? extends T> col)**
    - Devuelve el elemento con valor máximo en la colección, las comparaciones se basan en el orden natural

```
int max = Collections.max(datos);
System.out.println(datos);
System.out.println("Mayor valor de la lista: "+max);
```

```
[1, 2, 3, 0, 0, 0]
Mayor valor de la lista: 3
```

CLASE COLLECTION

- **Otros métodos de utilidad:**
  - **static <T extends Comparable<? super T>> T max (Collection<? extends T> col, Comparator<? super T> comp)**
    - Devuelve el elemento con valor máximo en la colección, las comparaciones se basan en el orden del comparador pasado por parámetro

No es necesario que las listas estén ordenadas para buscar el máximo

CLASE COLLECTIONS

- **Otros métodos de utilidad:**
  - **static <T extends Comparable<? super T>> T min(Collection<? extends T> coll)**
    - Devuelve el elemento con valor mínimo en la colección, las comparaciones se basan en el orden natural

```
int min = Collections.min(datos);
System.out.println(datos);
System.out.println("Menor valor de la lista: "+min);
```

```
[1, 2, 3, 0, 0, 0]
Menor valor de la lista: 0
```

CLASE COLLECTION

- **Otros métodos de utilidad:**
  - **static void reverse** (**List <?> lista**)
    - Invierte la lista colocando los elementos en orden inverso. No devuelve una nueva lista, sino que modifica la original
  - **static <T> Set<T> singleton**(**T elem**)
    - Devuelve un conjunto con elem como único elemento. Este nuevo conjunto no se puede modificar. Se suele utilizar para eliminar las ocurrencias de un elemento repetido.

CLASE COLLECTION

```
System.out.println(datos);
datos.removeAll(Collections.singleton(0));
System.out.println(datos);
```

```
[1, 2, 3, 0, 0, 0]
[1, 2, 3]
```

EJERCICIOS

Ejercicio 1:

Elige en cada caso el tipo de Colección más apropiada para cada conjunto de datos:

- Se quieren insertar 20 números menores de 100 en una colección. Se ha especificado que los 20 números deben ser diferentes (no puede haber duplicados) y se interesa que se ordenen de menor a mayor conforme se añadan a la colección.
- Se quiere introducir una serie de nombres de animales en una colección. Estos nombres podrán repetirse pero lo que nos importa es que se inserten en el mismo orden en el que los insertamos en un inicio.

Crea ambas colecciones en Java y muestra los valores almacenados.

NO SE PUEDEN UTILIZAR MÉTODOS EXTERNOS

INTRODUCCIÓN

Ejercicio 2:

Crea una aplicación que tenga las siguientes clases:

- **Asignatura** con los siguientes atributos idasignatura y nombreasig
- **NotasAlumno** con los siguientes atributos asignatura de tipo Asignatura y nota
- **Alumno** con los siguientes atributos idalumno, nombre y una lista de NotasAlumno así como los métodos:
  - public int notaAsignatura(int as) recibe un idasignatura y devuelve la nota
  - public void insertaNota(NotasAlumno n) añade un objeto NotasAlumno a las notas del alumno.
  - public double notaMedia() devuelve la nota media del alumno.

INTRODUCCIÓN

Ejercicio 2:

- public String asignaturaMasNota() devuelve el nombre de asignatura con más nota.
- public String asignaturaMenosNota() devuelve el nombre de asignatura con menos nota.
- public int numeroSuspensos() devuelva el número de asignaturas suspensas que tiene el alumno (nota < 5)
- public int numeroAprobados() devuelva el número de asignaturas aprobadas que tiene el alumno (nota >=5).
- toString() con este formato:

id: 1 / Nombre: Maria

- MATEMÁTICAS, Nota: 5
- LENGUA, Nota: 8

INTRODUCCIÓN

Ejercicio 2:

En main, crea una lista de Alumnos y añade 5 alumnos con sus correspondientes notas. Después crea un menú que permita:

- Añadir un nuevo alumno sin notas
- Añadir notas a un alumno
- Listar a todos los alumnos
- Reorganizar la lista de alumnos
- Obtener la nota media más repetida
- Listar la cantidad de aprobados y suspensos de cada alumno.