# Ordenación arrays y listas

#### Fundamentos sobre ordenación

- Arrays y listas pueden ser ordenados según el orden natural de los objetos
- El orden natural se define a través de la interfaz Comparable, que deberá ser implementada por los objetos a ordenar
- Si las clases no implementan Comparable, se deberá definir el orden natural en otra interfaz llamada Comparator.

## Interfaz Comparable

>Se encuentra en el paquete java.lang:

```
interface Comparable<T>{
  int compareTo(T obj);
}
si objeto>obj resultado>0
  si objeto=obj resultado=0
  si objeto<obj resultado<0</pre>
```

- Es implementada por clases de envoltorio y String
- >Para poder ordenar listas y arrays de otro tipo de objetos, sus clases deberán implementar esta interfaz

# Ejemplo de implementación

```
class Persona implements Comparable < Persona > {
  private String nombre;
  private int edad;
  //constructores, setter y getter
  public int compareTo(Persona p){
     if(this.nombre.compareTo(p.getNombre()) = = 0){
          return ((Integer)this.edad).compareTo(p.getEdad());
     }else{
          return this.nombre.compareTo(p.getNombre());
```

Se delega el orden natural de la persona al nombre de la misma y, en caso de igualdad, al de la edad

## Ordenación de arrays

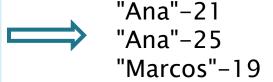
➤Se emplea el método sort(T[] datos) de la clase Arrays:

```
int [] numeros={7,2,34,11,6};
Arrays.sort(numeros);
for(int n:numeros){
    System.out.println(n);
}
2,6,7,11,34
```

#### Ordenación de colecciones

➤ Se emplea el método sort(List<T> datos) de Collections:

```
List<Integer> nums=new ArrayList<>();
nums.add(23);nums.add(8);nums.add(13);
Collections.sort(nums);
for(int n:nums){
    System.out.println(n);
}
```



## Interfaz Comparator

- Define el orden natural para aquellos tipos de objetos cuyas clases no implementan Comparable.
- >Se encuentra en java.util:

```
interface Comparator<T>{
  int compare (T obj1, T obj2);
}
si obj1>obj2 resultado >0
  si obj1==obj2 resultado=0
  si obj1<obj2 resultado <0</pre>
```

- >Se utiliza en los siguientes métodos:
  - Arrays.sort(T[] datos, Comparator<T> comp)
  - sort(Comparator<T> comp) método de List

### Ejemplo de ordenación Comparator

>Ordenar la siguiente lista de cadenas por número de caracteres de menor a mayor:

```
List<String> textos=new ArrayList<>();
textos.add("mi texto"); textos.add("hello");textos.add("es el más largo");
```

>Implementación Comparator:

```
public class Criterio implements Comparator<String>{
   public int compare(String ob1, String ob2){
     return ob1.length()-ob2.length();
   }
}
```

>Ordenación:

textos.sort(new Criterio());

## Búsqueda binaria

- La clase Arrays proporciona el siguiente método para realizar una búsqueda en un array:
  - int binarySearch(tipo[] array, tipo valor). Devuelve la posición del valor dentro del array, que previamente debe estar ordenado
- >Consideraciones sobre el método:
  - Si el array no está ordenado, el resultado es impredecible
  - Si el array está ordenado y el elemento no se encuentra, se devuelve -pins-1.
     Donde pins es la posición que le correspondería al elemento

```
int [] a1 = {3,5,7,9,15,20};
System.out.println(Arrays.binarySearch(a1, 9)); //3
System.out.println(Arrays.binarySearch(a1, 10)); //-5
```

### Comparación de arrays

- La clase Arrays incorpora en Java 11 el siguiente método para comparar arrays:
  - int compare(tipo[] array1, tipo[] array2). Devuelve el resultado de la comparación lexicográfica de ambos arrays
- >El resultado de la comparación será:
  - -1 Si array1 es menor que array2
  - 0 si ambos arrays son iguales
  - 1 Si array1 es mayor que array2

```
int[] a1 = {1,2,5};
int[] a2 = {1,2,1,4};
int[] a3 = {1,2,1,4,1};
System.out.println(Arrays.compare(a1, a2)); //1
System.out.println(Arrays.compare(a2, a3)); //-1
```