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, settery 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:

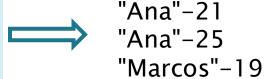
```
int [] numeros={7,2,34,11,6};
Arrays.sort(numeros);
for(int n:numeros){
    System.out.println(n);
}
2,6,7,11,34
```

Ordenación de listas

>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);
}
```





Revisión conceptos



Indica qué se mostrará al ejecutar el siguiente código:

```
List<Integer> enteros=List.of(6,1,4,0);
List<Integer> enteros2=List.copyOf(enteros);
Collections.sort(enteros);
System.out.println(enteros2);
```

- a. [6,1,4,0]
- b. [0,1,4,6]
- c. Excepción
- d. Error de compilación

Respuesta

La respuesta correcta es la C. Se produce una excepción en la tercera instrucción al intentar ordenar una colección inmutable.

Interfaz Comparator

- >Utilizada para poder definir 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)
 - wetodo 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());

Revisión conceptos



Tenemos una lista de objetos Curso, donde cada curso está caracterizado por un nombre, duración y precio. Si la colección está referenciada por la variable "cursos", escribe las instrucciones para ordenar la colección de menor a mayor duración

Respuesta

```
cursos.sort(new Comparator < Curso > (){
   public int compare(Curso c1, Curso c2){
     return c1.getDuracion().compareTo(c2.getDuracion());
   }
});
```

Búsqueda binaria en arrays

- >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 plns–1. Donde plns 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
```

Comparación de arrays II

- >La clase Arrays incorpora en Java 9 el siguiente método para comprobar diferencias entre arrays:
 - int mismatch(tipo[] array1, tipo[] array2). Devuelve la posición de la primera diferencia entre los dos arrays, o −1 si no hay diferencias

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

Revisión conceptos



Indica qué se mostrará al ejecutar el siguiente código:

```
int [] a1 = {9,5,7,1,4};
int [] a2 = {4,5,7,9,6,1};
Arrays.sort(a1);
Arrays.sort(a2);
System.out.println(Arrays.mismatch(a1, a2));
```

```
a. 3
b. 0
c. 2
d. -1
```

Respuesta

La respuesta correcta es la A. Al ordenar los arrays, la primera discrepancia se produce en el cuarto elemento, es decir, en la posición 3