Como ordenar arrays en Java. Método Arrays.sort()

ARRAYS.SORT(), COLLECTIONS.REVERSEORDER()

Para ordenar arrays de cualquier tipo Java dispone del método sort de la clase Arrays.

Para utilizarlo es necesario incluir el import:

```
import java.util.Arrays;
```

Por ejemplo, dado el siguiente array de Strings:

```
String [] nombres = {"juan", "pedro", "ana", "maria", "felipe", "luis", "eduardo"};
```

para ordenarlo de forma ascendente escribiremos la instrucción:

```
Arrays.sort(nombres);
```

Si mostramos el array por pantalla, comprobaremos que está ordenado de forma ascendente:

```
for(String s : nombres)
    System.out.println(s);
```

Arrays.sort ordena de forma ascendente (de menor a mayor). Para ordenar un array de forma descendente (de mayor a menor) hay que indicarlo utilizando el método reverseOrder() de la clase Collections.

Para utilizar reverseOrder es necesario incluir el import:

```
import java.util.Collections;
```

Por ejemplo, para ordenar el array nombres de forma descendente escribimos la instrucción Arrays.sort de la siguiente forma:

```
Arrays.sort(nombres, Collections.reverseOrder());
```

También tenemos la opción de ordenar solo una parte del array, indicando la posición del elemento inicial y la del elemento final (que no se incluye en la ordenación).

Por ejemplo, para ordenar solo los elementos 1, 2 y 3 ("pedro", "ana", "maria") del array nombres escribimos la instrucción de esta forma:

```
Arrays.sort(nombres, 1, 4);
```

El 1 indica la posición del elemento donde comienza la ordenación y el 4 indica la posición del primer elemento que no entra en la ordenación.

El contenido del array después de esta ordenación es el siguiente:

juan
ana
maria
pedro
felipe
luis
eduardo

vemos que solo se han ordenado los elementos 1, 2 y 3. El resto quedan igual:

También podemos ordenar solo una parte del array en orden inverso. Por ejemplo, para ordenar solo los elementos 1, 2 y 3 en orden inverso:

```
Arrays.sort(nombres, 1,4, Collections.reverseOrder());
```

El contenido del array es ahora:

juan
pedro
maria
ana
felipe
luis
eduardo

Con Arrays.sort podemos ordenar arrays de cualquier tipo de datos. Por ejemplo, para ordenar un array de enteros:

```
int [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};
Arrays.sort(numeros);
//mostrarlo ordenado
for (int n : numeros) {
    System.out.println(n);
}
```

Collections.reverOrder() solo funciona para arrays de objetos. Por este motivo si queremos ordenar de forma descendente arrays de tipos de datos simples debemos utilizar la clase envolvente equivalente al tipo de dato básico. Puedes ver las clases envolventes que corresponden a cada tipo de dato en esta entrada.

Por ejemplo, para ordenar un array de enteros forma descendente hay que declararlo de tipo Integer en lugar de int.

```
Integer [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};
Arrays.sort(numeros, Collections.reverseOrder());
for (int n : numeros) {
    System.out.println(n);
}
```

Cómo ordenar un array en Java utilizando Comparable y Comparator

EJEMPLO DE USO DE LAS INTERFACES COMPARABLE Y COMPARATOR JAVA PARA ORDENAR UN ARRAY DE OBJETOS

En esta entrada vamos a explicar como se puede ordenar un array de objetos en Java mediante el uso de la interface Comparable. Vamos a crear un array de tipo Empleado y posteriormente lo ordenaremos por nombre de empleado y lo mostraremos ordenado. Continuaremos el ejemplo ordenando de nuevo el array esta vez por el campo sueldo. Para hacer esta ordenación se utilizará la interface Comparator.

Disponemos de la clase Empleado que contiene los siguientes atributos:

```
public class Empleado{
    private String nif;
    private String nombre;
    private double sueldo;

    ///Resto de métodos de la case
}
```

Esta clase la vamos a utilizar en un programa que pide los datos de los empleados y los guarda en un array de 20 elementos llamado *empleados*que se encuentra en la clase principal. Posteriormente los mostrará ordenados por nombre.

```
public class EjemploComparableComparator {
    static Empleado[] empleados = new Empleado[20];
    static int indice = 0;
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args)  {
        leerEmpleados(); //en este método se llena el array empleados
        Arrays.sort(empleados); //ordena los empleados por nombre
        mostrarEmpleados(); //muestra el array empleados
    }
}
```

Para ordenar hemos utilizado:

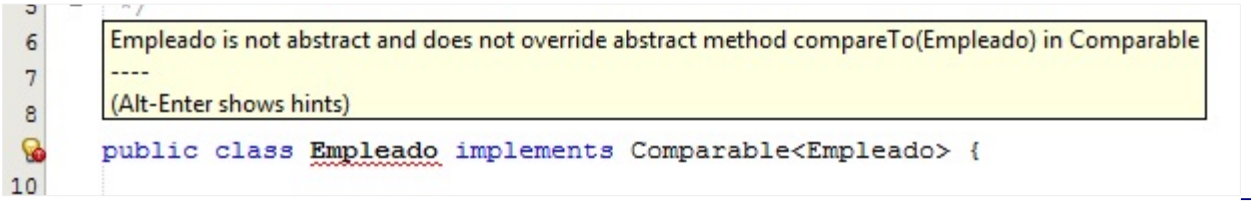
Arrays.sort(empleados);

Para que el método sort *sepa* que para ordenar el array tiene que comparar los nombres de los empleados, debemos indicárselo de alguna forma. La forma de indicarle a sort el criterio de ordenación es la siguiente:

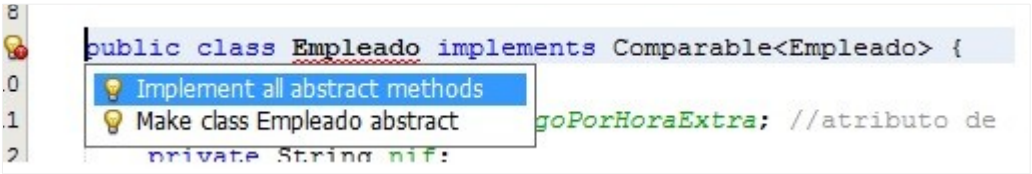
1. La Clase Empleado debe implementar la interface Comparable:

public class Empleado implements Comparable<Empleado> {

2. Nos aparecerá un error:



Que solucionamos pulsando en la opción *Implement all abstract methods*:



3. Comprobamos que al final de la clase se ha creado un nuevo método llamado *compareTo*

```
@Override
public int compareTo(Empleado t) {
    throw new UnsupportedOperationException("Not supported yet.");
}
```

4. Eliminamos la línea throw new….. y en su lugar escribimos el código para comparar los nombres de empleados:

```
@Override
public int compareTo(Empleado t) {
    return this.nombre.compareToIgnoreCase(t.nombre);
}
```

El método compareTo debe devolver 0 si son iguales, <0 si el de la izquierda (this) es menor que el de la derecha (t) ó >0 si el de la izquierda es mayor que el de la derecha.

En este caso como se trata de 2 String los podemos comparar con el método compareToIgnoreCase de String y devolver el resultado que obtengamos.

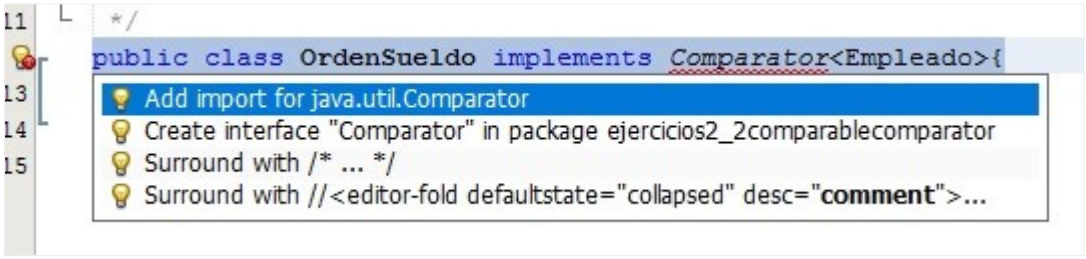
De esta forma cuando invoquemos al método sort en la instrucción *Arrays.sort(empleados) se* utilizará el método compareTo que acabamos de escribir para decidir el orden de los objetos. Este proceso es transparente para nosotros. El método *sort* aplicado a un array de objetos utiliza un algoritmo de ordenación mergesort. Va comparando los elementos del array según este algoritmo y decide si dos elementos están desordenados según el código que hemos escrito en el método compareTo.

Supongamos ahora que además de mostrar el array ordenado por nombre también queremos mostrarlo ordenado por sueldo. Para eso tendremos que utilizar la interface **Comparator** de la siguiente forma:

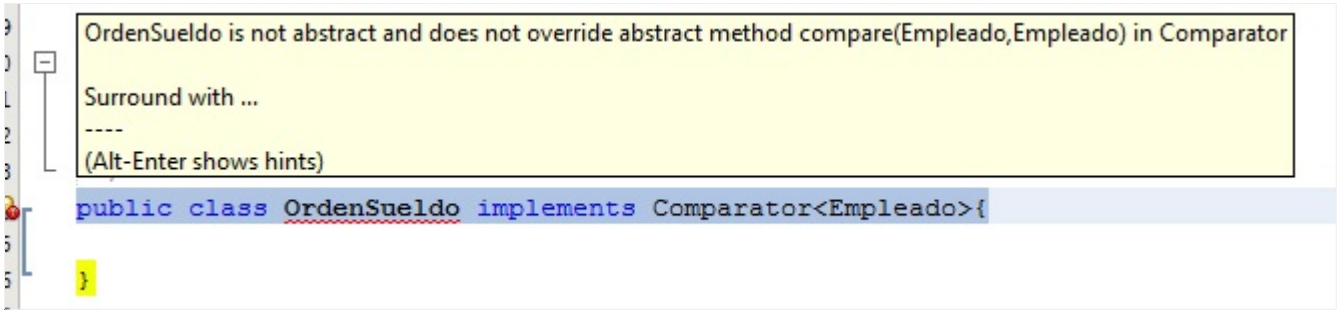
1. Creamos una clase nueva. La podemos llamar por ejemplo OrdenSueldo. Esta clase debe implementar la interface Comparator.

public class OrdenSueldo implements Comparator<Empleado>{

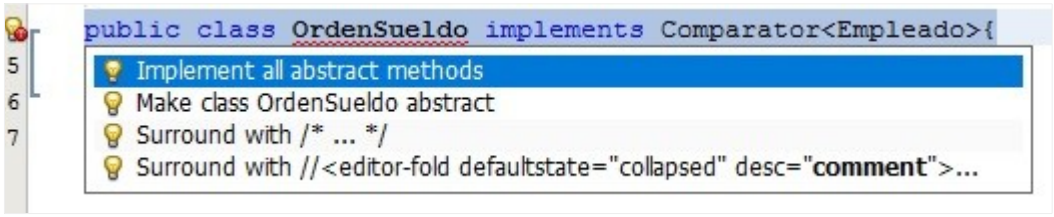
2. Nos indica que no tenemos el import. Lo añadimos:



3. A continuación nos indica otro error similar al que vimos en el ejemplo anterior



4. Lo solucionamos pulsando en *Implement all abstract methods*:



5. Esto crea el método compare:

```
public class OrdenSueldo implements Comparator<Empleado>{
    @Override
    public int compare(Empleado o1, Empleado o2) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

6. Eliminamos la línea throw new….. y en su lugar escribimos el código para comparar los sueldos.

```
public class OrdenSueldo implements Comparator<Empleado>{
    @Override
    public int compare(Empleado o1, Empleado o2) {
        if(o1.calcularSueldo() > o2.calcularSueldo()){
            return 1;
        }else if(o1.calcularSueldo() < o2.calcularSueldo()){
            return -1;
        }else{
            return 0;
        }
    }
}
```

El método compare debe devolver 0 si son iguales, <0 si el de la izquierda (o1) es menor que el de la derecha (o2) ó >0 si el de la izquierda es mayor que el de la derecha.
Para ordenar el array de empleados por sueldo debemos escribir ahora:
Arrays.sort(empleados, new OrdenSueldo());
Para utilizar el Comparator se crea un objeto de la clase que hemos creado. Este objeto se utiliza como parámetro cuando se invoca al método sort. En este caso sort utilizará como criterio de ordenación el método *compare* que hemos escrito en la clase OrdenSueldo.

Otra forma de hacer esto, sin tener que crear una nueva clase cada vez que queramos ordenar por un criterio distinto es escribir el código completo del comparator cuando se hace la llamada al método sort:

```
Arrays.sort(empleados, new Comparator<Empleado>() {
    @Override
    public int compare(Empleado o1, Empleado o2) {
        if (o1.calcularSueldo() > o2.calcularSueldo()) {
            return 1;
        } else if (o1.calcularSueldo() < o2.calcularSueldo()) {
            return -1;
        } else {
            return 0;
        }
    }
});
```

De esta forma nos evitamos añadir clases innecesarias al proyecto. Debemos tener en cuenta que podemos ordenar los objetos de un array por muchos criterios distintos y para cada uno habría que crear una clase nueva que implemente la interface Comparator con su método compare correspondiente.