Ordenar arrays y ArrayList de objetos utilizando las interfaces Comparable y Comparator

Disponemos de la clase Empleado que contiene los siguientes atributos:

```
public class Empleado{
    private String nif;
    private String nombre;
    private double sueldo;
    private int edad;
    ///Resto de métodos de la case
}
```

Disponemos de un array para almacenar 20 objetos de tipo Empleado:

```
static Empleado[] empleados = new Empleado[20];
```

Suponemos que durante la ejecución del programa se han guardado en el array los 20 empleados y a continuación queremos mostrarlos ordenados alfabéticamente por nombre del empleado.

La instrucción que escribimos para ordenar el array es:

```
Arrays.sort(empleados);
```

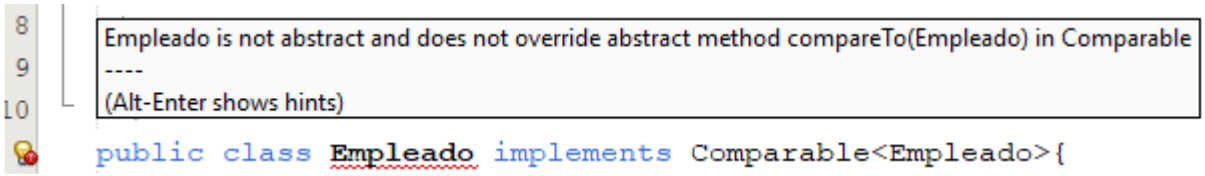
Pero para que esto funcione y realice la ordenación de forma correcta hay que indicarle a Arrays.sort que para ordenar los objetos tiene que comparar los nombres de los empleados.

La forma de indicarle al método sort el criterio de ordenación es la siguiente:

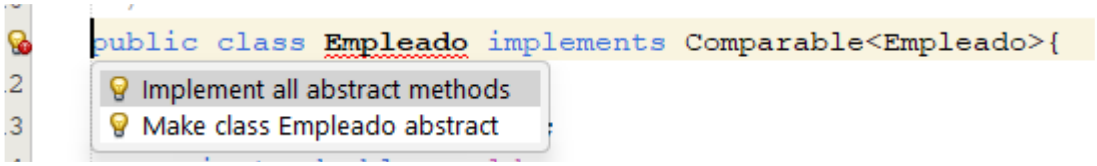
1. La Clase Empleado debe implementar la interface Comparable. Para ello la primera línea de la clase se escribe así:

```
public class Empleado implements Comparable<Empleado> {
```

2. Nos aparecerá un error:



El error lo solucionamos pulsando en la marca de error y seleccionando 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 o) {
    throw new UnsupportedOperationException(message: "Not supported yet."); // Generate
}
```

4. Dentro del método compareTo debemos escribir el código para indicar cómo se deben ordenar los objetos, en este caso debemos indicar que queremos realizar la ordenación por nombre.

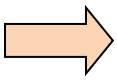
Eliminamos la línea throw new….. y en su lugar escribimos el código para comparar los nombres de empleados:

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

El método compareTo realiza la comparación de dos objetos: el objeto que invoca al método (this) y el objeto que recibe como parámetro (o). La comparación de objetos la realiza por alguno de sus atributos, en este caso por el nombre.

El método compareTo devuelve un número entero: devuelve 0 si son iguales, un número entero < 0 si el de la izquierda (this) es menor que el de la derecha (o) ó un número entero > 0 si el de la izquierda es mayor que el de la derecha.

En este caso como se trata de comparar dos nombres (2 String) los podemos comparar con el método compareToIgnoreCase de String y devolver directamente con el return el resultado que obtengamos de esa comparación.



Cuando una clase implementa la interface Comparable y por lo tanto contiene un método compareTo, al ordenar los objetos de la clase mediante Arrays.sort() o Collections.sort() se ordenarán según indique el método compareTo de la clase.

El método compareTo indica el orden natural de los objetos u orden por defecto.

Cuando invoquemos al método sort en la instrucción *Arrays.sort(empleados)* para que ordene el array, de forma interna el método sort invocará a su vez al método compareTo que acabamos de escribir para decidir el orden de los objetos.

El método *sort* aplicado a un array de objetos va comparando los elementos del array y decide si dos elementos están desordenados o no según el código que hemos escrito en el método compareTo. Este proceso de ordenación es transparente para nosotros.

Supongamos ahora que además de mostrar el array ordenado por nombre, también queremos mostrarlo **ordenado por edad** de los empleados.

Como el método compareTo ya lo tenemos hecho para que ordene por nombre, tenemos que utilizar otra interface para que ordene el array por un atributo distinto que no sea el nombre de empleado. Para esto utilizamos la interface **Comparator**.

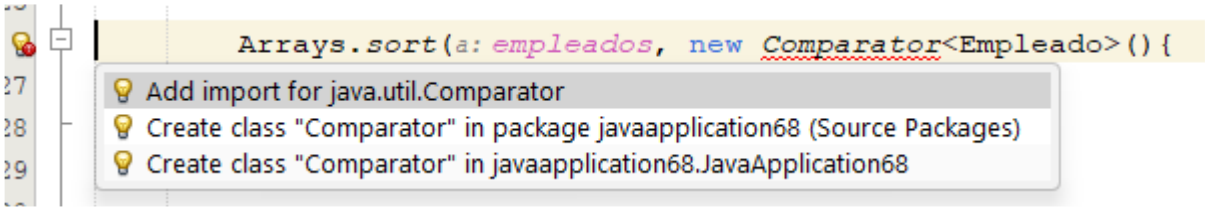
Se utiliza de la siguiente forma:

Escribimos la instrucción Arrays.sort para ordenar el array de empleados pero le añadimos un segundo parámetro, como aparece en la imagen

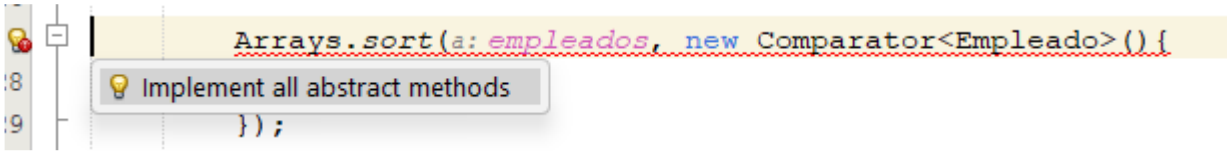
```
Arrays.sort(a: empleados, new Comparator<Empleado>(){

});
```

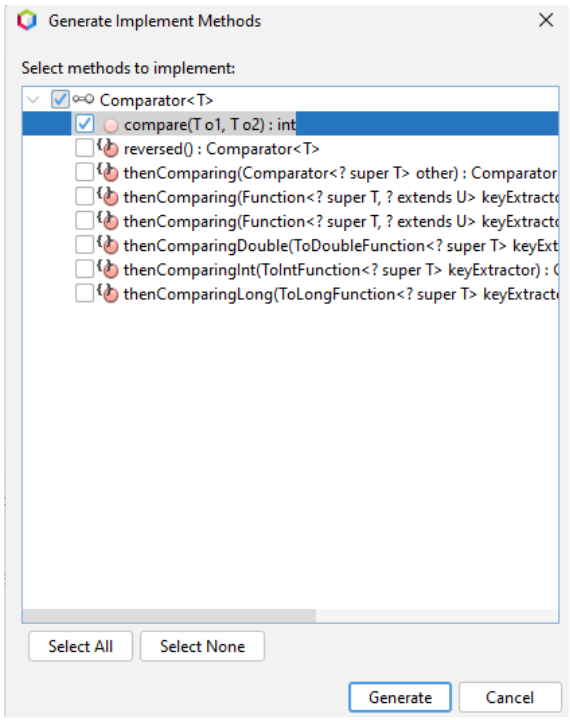
El error que aparece se debe a que tenemos que hacer el import:



Después nos aparecerá otro error que solucionamos pulsando en *Implement all abstract methods*



En la ventana que aparece, seleccionamos el método *compare* y pulsamos en *Generate*



Podemos ver el método compare que se ha generado:

```
Arrays.sort(a: empleados, new Comparator<Empleado>(){
    @Override
    public int compare(Empleado o1, Empleado o2) {
        throw new UnsupportedOperationException(message: "Not supported yet."); // Generated fr
    }
});
```

Dentro del método compare debemos escribir el código para indicar cómo se deben ordenar los objetos, en este caso debemos indicar que queremos realizar la ordenación por edad.

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

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

El método compare realiza la comparación de dos objetos que recibe como parámetros. La comparación de ambos objetos la realiza por alguno de sus atributos, en este caso por edad.

El método compare devuelve siempre un entero: devuelve 0 si son iguales, un número entero < 0 si el de la izquierda (o1) es menor que el de la derecha (o2) ó un número entero > 0 si el de la izquierda es mayor que el de la derecha.

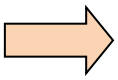
En este caso queremos ordenar por edad, por lo que comparamos las edades de o1 y o2 y devolvemos 0 si son iguales, 1 si la edad de o1 es mayor que la edad de o2 y -1 si la edad de o1 es menor que la edad de o2.

Como la edad es un atributo de tipo entero, en lugar de realizar la comparación de las edades con instrucciones if, podemos hacerlo esta forma:

```
Arrays.sort(a: empleados, new Comparator<Empleado>(){
    @Override
    public int compare(Empleado o1, Empleado o2) {
        return o1.getEdad() - o2.getEdad();
    }
});
```

Al restar las edades, si son iguales se devuelve 0, si la edad de o1 es mayor que la de o2 al restarlas se obtendrá un número entero > 0 y si la edad de o1 es menor que la de o2 al restarlas se obtendrá un número < 0.

Cuando el atributo es de tipo entero esta forma de hacerlo es más sencilla que las instrucciones if pero ambas son válidas.



Cada vez que queramos ordenar el array por un orden que no sea el orden natural (indicado en el método *compareTo* de la clase) tendremos que crear un nuevo *Comparator* en el momento de realizar la ordenación.

Mediante el Comparator estamos indicando al método sort el criterio de ordenación para esa ordenación en concreto.

Por ejemplo, supongamos que además nos piden que mostremos el array **ordenado por el sueldo** de los empleados. Para esto debemos crear un nuevo Comparator de la misma forma que hemos hecho en el ejemplo anterior para ordenarlo por edad. En el método compare tendremos que escribir ahora el código para ordenar los empleados por sueldo.

Tenemos que comparar los sueldos de o1 y o2 y devolvemos 0 si son iguales, 1 si el sueldo de o1 es mayor que el sueldo de o2 y -1 si el sueldo de o1 es menor que el sueldo de o2.

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

Cuando hemos ordenado por edad hemos realizado la comparación de una forma alternativa a los if, restando las dos edades. En este caso, como el sueldo es un atributo de tipo double no podemos hacerlo de esa forma ya que el valor obtenido al restar los dos sueldos es de tipo double y no podemos devolverlo ya que el método compare debe devolver un int.

Pero existe una forma de comparar dos números de tipo double sin utilizar instrucciones if. El método

Double.compare(double a, double b)

Este método compara dos números de tipo double y devuelve 0 si son iguales, un número entero < 0 si a es menor que b, un número entero > 0 si a es mayor que b.

Utilizando Double.compare, la ordenación del array empleados por sueldo la podemos escribir así:

```
Arrays.sort(a: empleados, new Comparator<Empleado>(){
    @Override
    public int compare(Empleado o1, Empleado o2) {
        return Double.compare(d1: o1.getSueldo(), d2: o2.getSueldo());
    }
});
```

En los ejemplos que hemos visto, las comparaciones que se hacen en los métodos compareTo y compare son para realizar la ordenación de menor a mayor.

Si queremos ordenar de mayor a menor tan solo hay que cambiar el orden en el que comparamos los objetos.

Por ejemplo, para indicar en el método compareTo de la clase Empleado que queremos ordenar alfabéticamente en orden inverso (de la Z a la A) lo escribimos así:

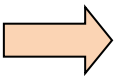
```
@Override
public int compareTo(Empleado o) {
    return o.nombre.compareToIgnoreCase(str: this.nombre);
}
```

Para indicar en el método compare dentro del Comparator que queremos ordenar por sueldo de mayor a menor:

```
Arrays.sort(a: empleados, new Comparator<Empleado>(){
    @Override
    public int compare(Empleado o1, Empleado o2) {
        return Double.compare(d1: o2.getSueldo(), d2: o1.getSueldo());
    }
});
```

Si realizamos la ordenación por sueldo utilizando instrucciones if, para ordenar de mayor a menor escribimos el código de esta forma:

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



Si en lugar de ser un array fuese un ArrayList lo que queremos ordenar, se hace todo de la misma forma. Solo hay que sustituir **Arrays**.sort() por **Collections**.sort().

Expresiones lambda

Con la incorporación de las expresiones lambda al lenguaje de Java, el código de los Comparator se puede escribir de forma mucho más resumida.

Por ejemplo, para ordenar por edad podemos indicarlo utilizando una expresión lambda:

```
Arrays.sort(a: empleados, (Empleado o1, Empleado o2) -> o1.getEdad() - o2.getEdad());
```

En la expresión lambda se indica el criterio de ordenación de los objetos, en este caso para cada pareja de objetos se debe realizar la resta de sus edades.

Para ordenar por sueldo con instrucciones if para comparar los sueldos se escribiría este código :

```
Arrays.sort(a: empleados, (Empleado o1, Empleado o2) -> {
    if (o1.getSueldo() > o2.getSueldo()) {
        return 1;
    } else if (o1.getSueldo() < o2.getSueldo()) {
        return -1;
    } else {
        return 0;
    }
});
```

Para ordenar por sueldo con el método Double.compare:

```
Arrays.sort(a: empleados, (Empleado o1, Empleado o2) -> Double.compare(d1: o1.getSueldo(), d2: o2.getSueldo()));
```