



UNIDAD 9:

INTERFACES

CLASE 3: COMPARABLE Y COMPARADOR

INTERFACES IMPLEMENTADAS EN JAVA

Dentro de Java podemos encontrar algunas interfaces creadas por programadores para el uso general.

- Comparable
- Comparator

INTERFAZ COMPARABLE

- Establece un criterio de comparación natural, o por defecto, entre los objetos de una clase.
- Consta de un único método abstracto:
`int compareTo(Object ob);`

INTERFAZ COMPARABLE

Para comparar dos objetos de una clase se escribirá **obj1.compareTo(obj2)** que devolverá un entero en función de:

- `obj1.compareTo(obj2) < 0` si `obj1` es menor
- `obj1.compareTo(obj2) > 0` si `obj1` es mayor
- `obj1.compareTo(obj2) = 0` si ambos son iguales

INTERFAZ COMPARABLE

```
public class Persona{  
    private int edad;  
    double altura;  
    String nombre;  
    double peso;  
    String dni;
```

```
@Override  
public int compareTo(Object o) {  
    Persona otraP = (Persona) o;  
    if(edad == otraP.edad) {  
        if(nombre.equals(otraP.nombre)) {  
            return 0;  
        }else if(nombre.compareTo(otraP.nombre)<0) {  
            return -1;  
        }else if(nombre.compareTo(otraP.nombre)>0) {  
            return 1;  
        }  
    }else if(edad < otraP.edad) {  
        return -1;  
    }else if(edad > otraP.edad) {  
        return 1;  
    }  
    return 0;  
}
```

INTERFAZ COMPARABLE

```
Persona persona;
persona = new Persona(21,1.61,"Ana",64.6,"12345678K");
Persona persona2;
persona2 = new Persona(21,1.64,"Irene",70,"87654321K");
System.out.println(persona.compareTo(persona2));
```

-1

INTERFAZ COMPARABLE

La clase Arrays implementa el método sort() que ordena los elementos del array de manera “natural”

```
int tabla[] = {3,6,8,10,0,1,4,7};  
Arrays.sort(tabla);  
System.out.println(Arrays.toString(tabla));
```

[0, 1, 3, 4, 6, 7, 8, 10]

INTERFAZ COMPARABLE

Si implementamos Comparable en nuestras clases, podremos hacer uso de estos métodos

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                    new Persona(21,1.64,"Irene",70,"87654321K"),  
                    new Persona(19,1.64,"Martin",70,"10293847J")};  
Arrays.sort(tabla);  
System.out.println(Arrays.deepToString(tabla));
```

```
[  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19,  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 21]
```

INTERFAZ COMPARABLE

deepToString() accede al método
toString() de cada objeto

```
System.out.println(Arrays.deepToString(tabla));
```

```
@Override public String toString(){
    return "DNI: "+dni+", NOMBRE: "+nombre+", EDAD: "+edad;
}
```

```
[
  DNI: 10293847J, NOMBRE: Martin, EDAD: 19,
  DNI: 12345678K, NOMBRE: Ana, EDAD: 21,
  DNI: 87654321K, NOMBRE: Irene, EDAD: 21]
```

INTERFAZ COMPARABLE

Junto al método compareTo, también deberíamos implementar el método equals

```
@Override  
public boolean equals(Object o) {  
    Persona otraP = (Persona) o;  
    if(edad == otraP.edad && nombre.equals(otraP.nombre)) {  
        return true;  
    }  
    return false;  
}
```

INTERFAZ COMPARATOR

Mientras que Comparable es utilizado por métodos predeterminados en Java, Comparator nos servirá para poder ordenar objetos de una clase siguiendo diferentes criterios

Ordenar Personas por altura, edad, alfabéticamente...

INTERFAZ COMPARATOR

Para utilizarla tendremos que importarla

```
import java.util.Comparator;

public class ComparadorEdadesPersonas implements Comparator{
    @Override
    public int compare(Object o1, Object o2) {
        Persona p1 = (Persona)o1;
        Persona p2 = (Persona)o2;
        int resul = p1.getEdad() - p2.getEdad();
        return resul;
    }
}
```

INTERFAZ COMPARATOR

```
import java.util.Comparator;

public class ComparadorNombrePersona implements Comparator{
    @Override
    public int compare(Object o1, Object o2) {
        Persona p1 = (Persona)o1;
        Persona p2 = (Persona)o2;
        if(p1.nombre.equals(p2.nombre)) {
            return 0;
        }else if(p1.nombre.compareTo(p2.nombre) < 0) {
            return -1;
        }else {
            return 1;
        }
    }
}
```

INTERFAZ COMPARATOR

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                   new Persona(20,1.64,"Irene",70,"87654321K"),  
                   new Persona(19,1.64,"Martin",70,"10293847J")};  
ComparadorEdadesPersonas cep = new ComparadorEdadesPersonas();  
Arrays.sort(tabla,cep); //tabla a ordenar, comparador a utilizar  
System.out.println(Arrays.deepToString(tabla));
```

```
[  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21]
```

INTERFAZ COMPARATOR

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                    new Persona(20,1.64,"Irene",70,"87654321K"),  
                    new Persona(19,1.64,"Martin",70,"10293847J")};  
ComparadorNombrePersona cep = new ComparadorNombrePersona();  
Arrays.sort(tabla,cep); //tabla a ordenar, comparador a utilizar  
System.out.println(Arrays.deepToString(tabla));
```

```
[  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19]
```

INTERFAZ COMPARATOR

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                    new Persona(20,1.64,"Irene",70,"87654321K"),  
                    new Persona(19,1.64,"Martin",70,"10293847J")};  
Arrays.sort(tabla,new ComparadorNombrePersona());  
//tabla a ordenar, comparador a utilizar  
System.out.println(Arrays.deepToString(tabla));
```

```
[  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19]
```

INTERFAZ COMPARATOR

Si solo vamos a utilizar el comparador 1 vez
podemos crearlo como clase anónima

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                    new Persona(20,1.64,"Irene",70,"87654321K"),  
                    new Persona(19,1.64,"Martin",70,"10293847J")};  
Arrays.sort(tabla,new Comparator() {  
    public int compare(Object o1, Object o2) {  
        Persona p1 = (Persona)o1;  
        Persona p2 = (Persona)o2;  
        int resul = p1.getEdad() - p2.getEdad();  
        return resul;  
    }  
});  
//tabla a ordenar, comparador a utilizar  
System.out.println(Arrays.deepToString(tabla));
```

INTERFAZ COMPARATOR

Si solo vamos a utilizar el comparador 1 vez
podemos crearlo como clase anónima

Sin embargo, es el método menos recomendado

INTERFAZ COMPARATOR

Para ordenar de manera inversa a como funciona otro comparador podremos utilizar el método reversed()

```
ComparadorNombrePersona cep = new ComparadorNombrePersona();
Comparator c = cep.reversed();
Arrays.sort(tabla, c); //tabla a ordenar, comparador a utilizar
System.out.println(Arrays.deepToString(tabla));
```

```
[  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21]
```

INTERFAZ COMPARATOR

También podemos realizar búsquedas binarias en Arrays de objetos

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),
                    new Persona(20,1.64,"Irene",70,"87654321K"),
                    new Persona(19,1.64,"Martin",70,"10293847")};
ComparadorEdadesPersonas cep = new ComparadorEdadesPersonas();
Arrays.sort(tabla,cep); //tabla a ordenar, comparador a utilizar
int indice = Arrays.binarySearch(tabla,
                                  new Persona(21,1.61,"Ana",64.6,"12345678K"));
System.out.println(Arrays.deepToString(tabla)+"\n"+indice);
```

INTERFAZ COMPARATOR

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),  
                   new Persona(20,1.64,"Irene",70,"87654321K"),  
                   new Persona(19,1.64,"Martin",70,"10293847J")};  
ComparadorEdadesPersonas cep = new ComparadorEdadesPersonas();  
Arrays.sort(tabla,cep); //tabla a ordenar, comparador a utilizar  
int indice = Arrays.binarySearch(tabla,  
                                   new Persona(21,1.61,"Ana",64.6,"12345678K"));  
System.out.println(Arrays.deepToString(tabla)+"\n"+indice);
```

```
DNI: 10293847J, NOMBRE: Martin, EDAD: 19,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 12345678K, NOMBRE: Ana, EDAD: 21]  
2
```

INTERFAZ COMPARATOR

```
Persona tabla[] = {new Persona(21,1.61,"Ana",64.6,"12345678K"),
                   new Persona(20,1.64,"Irene",70,"87654321K"),
                   new Persona(19,1.64,"Martin",70,"10293847J")};
ComparadorEdadesPersonas cep = new ComparadorEdadesPersonas();
Comparator c = cep.reversed();
Arrays.sort(tabla); //tabla a ordenar, comparador a utilizar
int indice = Arrays.binarySearch(tabla,
                                  new Persona(21,1.61,"Ana",64.6,"12345678K"),c);
System.out.println(Arrays.deepToString(tabla)+"\n"+indice);
```

```
DNI: 12345678K, NOMBRE: Ana, EDAD: 21,  
DNI: 87654321K, NOMBRE: Irene, EDAD: 20,  
DNI: 10293847J, NOMBRE: Martin, EDAD: 19]  
0
```

EJERCICIOS

Ejercicio 1:

Crea la clase **Carta** que tenga los atributos:

- **Palo** (Oros, Bastos, Espadas o Copas que puede ser enumerado)
- **Número (1 al 13 siendo J-11, C-12, R-13)**

También tendrá un **constructor** que recibirá todos los atributos así como métodos **get/set** para los dos y un método **toString** que muestre la información de la carta como : “Carta **Numero** del palo **Palo**”

Crea los métodos o las clases necesarias para ordenar una serie de cartas de menor número a mayor y los palos por orden alfabético (1 de Bastos, 12 de Bastos, 10 de Copas, 1 de Espadas...). También crea una que ordene exclusivamente por el número de mayor a menor.

Dentro de la clase Carta crea también un método estático que cree (con new) una nueva Carta con un palo y número aleatorio y lo devuelva.

EJERCICIOS

Ejercicio 1:

Después crea un main que pregunte por teclado al usuario cuantas cartas **diferentes** quiere generar mediante el método de la clase Carta. Almacena todas esas cartas en un array y pregunta al usuario qué método de ordenación quiere utilizar. Finalmente, muestra el array ordenado por el método elegido.

EJERCICIOS

Ejercicio 2:

Crea la clase **Alumno** que tenga los atributos:

- **Nombre**
- **Apellido** (sólo pondremos el primero)
- **Fecha de nacimiento**
- **Edad**
- **Nota media**
- **Total asignaturas matriculado**

También tendrá un **constructor** que recibirá todos los atributos así como métodos **get/set** para todos ellos y un método **toString** que muestre la información del alumno como : “*nombre apellido nacido en la fecha fecha_nac con edad edad, tiene una nota media de nota_media y actualmente está matriculado en total_asig asignaturas*”

EJERCICIOS

Ejercicio 2:

La clase **Alumno** además contará con una implementación del método `compareTo()` y `equals()` que tendrán en cuenta en orden de prioridad:

- 1º - Edad
- 2º - Fecha de nacimiento
- 3º - Apellido
- 4º - Nombre

También se crearán Comparadores de la clase `Comparator` que nos permitan ordenar a los alumnos por cada uno de los 4 datos señalados previamente en orden ascendente.

EJERCICIOS

Ejercicio 2:

En main, crea un array de Alumnos que contenga 5 alumnos con valores aleatorios elegidos por ti. Después crea un menú con las siguientes opciones:

- Mostrar la lista de alumnos sin ordenar
- Mostrar la lista de alumnos ordenada por edad (mayor a más joven)
- Mostrar la lista de alumnos ordenada por fecha de nacimiento (mayor a más joven)
- Mostrar la lista de alumnos ordenada por apellido (a-z)
- Mostrar la lista de alumnos ordenada por nombre (a-z)
- Comparar dos alumnos indicando sus índices en el array (habrá que introducir dos números que entren dentro de los límites del array)

EJERCICIOS

Ejercicio 3:

Crea la clase **Llamada** que tenga los atributos:

- **Número de teléfono del cliente** (9 cifras)
- **Número de teléfono al que se llamó** (9 cifras)
- **Booleano** que indique si la llamada es **entrante** (true) o **saliente** (false)
- **Fecha y hora** de inicio de la llamada (dd / mm / yyyy - hh : mm : ss)
- **Fecha y hora de fin de la llamada** (dd / mm / yyyy - hh : mm : ss)
- **Enumerado** que indique la **distancia entre ambos teléfonos** (**A** (menor a 1 km), **B** (menor a 100km), **C** (menor a 1000km), **D** (mayor a 1000km))
- También tendrá una **tabla que almacena el valor €/minuto en cada una de las zonas** que será el mismo para todas las instancias de **Llamada**.

Ejercicio 3:

Tendrá un **constructor** que recibirá todos los atributos así como métodos **get/set** para todos ellos y un método **toString** que muestre la información de la llamada como : “*El teléfono de la compañía **número del cliente** ha recibido/realizado (en función de **saliente/entrante**) una llamada del teléfono **número externo** en la fecha **fecha hora inicial (solo fecha!!)** desde la hora **fecha hora inicial (solo hora!!)** hasta la hora **fecha hora final (solo hora!!)** del mismo día/del día **fecha hora final** (solo fecha!!) (en función de si las dos fechas son iguales o distintas) y ha supuesto un coste de **calcularCoste€/minuto en función de la zona (si la llamada es saliente)**”*

Se recomienda crear un método que devuelva la duración de la llamada en minutos así como una que calcule el gasto total de la misma si fue saliente
Además, se establecerá un orden natural basado en los criterios:

- 1º - Número del cliente
- 2º - Fecha_hora_inicio

EJERCICIOS

Ejercicio 3:

Crea también un Comparator que ordene las llamadas por coste de menor a mayor.

En main, crea un array de Llamadas que contenga 10 llamadas con valores aleatorios elegidos por ti. Después crea un menú con las siguientes opciones:

- Mostrar la lista de llamadas tal y como están en el array
- Mostrar la lista de llamadas ordenadas por orden natural
- Mostrar la lista de llamadas ordenadas en orden inverso
- Mostrar la lista de llamas ordenadas por coste

Se listan valores para la tabla de la clase llamada:

A - 0,03€/m / B - 0,10€/m / C - 0,25€/m / D - 0,50€/m