FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma, Desarrollo de Aplicaciones Web**
MÓDULO:	**Programación**

UNIDAD 6: RELACIONES ENTRE CLASES Y POLIMORFISMO

ANEXO I

AUTORES: **Fernando Rodríguez Alonso**
Sonia Pasamar Franco

1. ORDENACIÓN DE VECTORES DE OBJETOS

Los tipos primitivos y las clases envoltorio tienen predefinida la forma en la que se ordenan, por lo que para ordenar un vector compuesto por elementos de dichos tipos, no es necesario realizar ninguna tarea adicional. Simplemente se usa el método `sort` de la clase **`Arrays`** y el vector queda ordenado.

Pero para poder ordenar las componentes de un vector de objetos, es preciso indicar el criterio de ordenación. Esta tarea se puede realizar de varias formas: mediante la interfaz **`Comparable`** y mediante la interfaz **`Comparator`**.

1.1. INTERFAZ COMPARABLE

La interfaz **`Comparable`** debe implementarse en la clase en la que se quiere definir el criterio de ordenación. Al implementar esta interfaz será necesario implementar el método `compareTo`.

Además, es una interfaz genérica, por lo que se deberá especificar la clase que se va a ordenar.

En el siguiente ejemplo se muestra la implementación de esta interfaz en la clase **`Pelicula`** haciendo que la ordenación de objetos de dicha clase sea por su atributo `título` de forma ascendente.

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

    private String titulo;
    private int minutosDuracion;
    private boolean dobladaCastellano;
    private double puntuacion;

    public Pelicula(String titulo, int minutosDuracion,
                    boolean dobladaCastellano, double puntuacion) {
        this.titulo = titulo;
        this.minutosDuracion = minutosDuracion;
        this.dobladaCastellano = dobladaCastellano;
        this.puntuacion = puntuacion;
    }

    @Override
    public String toString() {
        return "Pelicula [titulo=" + titulo + ", minutosDuracion="
                + minutosDuracion + ", dobladaCastellano="
                + dobladaCastellano + ", puntuacion=" + puntuacion + "]";
    }

    @Override
    public int compareTo(Pelicula pelicula) {
        return this.titulo.compareTo(pelicula.titulo);
    }

}
```

Dado que la ordenación se realizará por un atributo de tipo cadena de caracteres, se aprovecha el método `compareTo` de la clase **`String`** para definir el criterio de ordenación. En este caso, la ordenación se realiza de forma ascendente por el título.

En caso de querer ordenar de forma descendente, dentro del método `compareTo` de la clase **`Pelicula`** se debería llamar al método `compareTo` de la clase **`String`** con los objetos al revés:

```
pelicula.titulo.compareTo(this.titulo)
```

En el siguiente ejemplo, se muestra la creación de un vector de películas y su posterior ordenación.

```
import java.util.Arrays;

public class PrincipalOrdenComparable {

    public static void main(String[] args) {

        Pelicula[] peliculas = new Pelicula[3];

        peliculas[0] = new Pelicula("Titanic", 194, true, 7.8);
        peliculas[1] = new Pelicula("Matrix", 136, true, 8.7);
        peliculas[2] = new Pelicula("Seven", 127, true, 8.6);

        System.out.println("*** Películas por orden de inserción ***");
        for(int i = 0; i < peliculas.length; i++ ) {
            System.out.println(peliculas[i].toString());
        }

        Arrays.sort(peliculas);

        System.out.println("*** Películas por orden según clase Película ***");
        for(int i = 0; i < peliculas.length; i++ ) {
            System.out.println(peliculas[i].toString());
        }

    }
}
```

1.2. INTERFAZ COMPARATOR

La interfaz **`Comparator`** también se utiliza para definir un criterio de ordenación, pero a diferencia de la interfaz **`Comparable`**, para una misma clase pueden definirse diferentes criterios de ordenación, y aplicarse uno u otro según convenga.

El siguiente ejemplo, parte de la clase **`Pelicula`**, que, en este caso, no implementa ninguna interfaz y contiene algunos *getters*, que serán necesarios para definir el criterio de ordenación.

```java
public class Pelicula{

    private String titulo;
    private int minutosDuracion;
    private boolean dobladaCastellano;
    private double puntuacion;

    public Pelicula(String titulo, int minutosDuracion,
                        boolean dobladaCastellano, double puntuacion) {
        this.titulo = titulo;
        this.minutosDuracion = minutosDuracion;
        this.dobladaCastellano = dobladaCastellano;
        this.puntuacion = puntuacion;
    }

    public int getMinutosDuracion() {
        return minutosDuracion;
    }

    public void setMinutosDuracion(int minutosDuracion) {
        this.minutosDuracion = minutosDuracion;
    }

    public double getPuntuacion() {
        return puntuacion;
    }

    public void setPuntuacion(double puntuacion) {
        this.puntuacion = puntuacion;
    }

    @Override
    public String toString() {
        return "Pelicula [titulo=" + titulo + ", minutosDuracion="
                    + minutosDuracion + ", dobladaCastellano="
                    + dobladaCastellano + ", puntuacion=" + puntuacion + "]";
    }

}
```

La interfaz **`Comparator`** se implementa en una clase nueva, donde se define un criterio de ordenación. Además, es una interfaz genérica, por lo que se especificará la clase que se va a ordenar.

En el siguiente ejemplo, se marca la ordenación por puntuación de forma ascendente.

```java
import java.util.Comparator;

public class OrdenPorPuntuacion implements Comparator<Pelicula>{

    @Override
    public int compare(Pelicula peli1, Pelicula peli2) {
        return (int)(peli1.getPuntuacion() -  peli2.getPuntuacion());
    }

}
```

Y en el siguiente ejemplo, se indica que la ordenación sea por duración de forma descendente.

```
import java.util.Comparator;

public class OrdenPorDuracion implements Comparator<Pelicula>{

    @Override
    public int compare(Pelicula peli1, Pelicula peli2) {
        return peli2.getMinutosDuracion() -  peli1.getMinutosDuracion();
    }

}
```

Finalmente, en el siguiente ejemplo, se muestra la creación de un vector de películas y su posterior ordenación, primero por un criterio y después por otro.

```
import java.util.Arrays;

public class PrincipalOrdenComparator {

    public static void main(String[] args) {

        Pelicula[] peliculas = new Pelicula[3];

        peliculas[0] = new Pelicula("Titanic", 194, true, 7.8);
        peliculas[1] = new Pelicula("Matrix", 136, true, 8.7);
        peliculas[2] = new Pelicula("Seven", 127, true, 8.6);

        System.out.println("*** Películas con orden de inserción ***");
        for(int i = 0; i < peliculas.length; i++ ) {
            System.out.println(peliculas[i].toString());
        }

        Arrays.sort(peliculas, new OrdenPorPuntuacion());

        System.out.println("*** Películas con orden según puntuación ***");
        for(int i = 0; i < peliculas.length; i++ ) {
            System.out.println(peliculas[i].toString());
        }

        Arrays.sort(peliculas, new OrdenPorDuracion());

        System.out.println("*** Películas con orden según duración ***");
        for(int i = 0; i < peliculas.length; i++ ) {
            System.out.println(peliculas[i].toString());
        }
    }
}
```