Estructuras de datos y algoritmos
Laboratorio 1-1: Algoritmos de ordenación y Búsqueda

Objetivos:
- Experimentar con los algoritmos de ordenación y búsqueda vistas en clase
- Utilizar el debugger para un mejor entendimiento de los pasos principales de los algoritmos
- Modificar los algoritmos para obtener unos algoritmos genéricos o independientes del tipo de información y criterio de ordenación

Proyecto
- *ordenacionGenerica*

1. Ordenar la secuencia (8, 6, 9, 5, 4, 10, 1, 3) utilizando los siguientes métodos y describe el estado posterior a la ejecución del paso principal de cada algoritmo. Utiliza el debugger para entender el funcionamiento de los algoritmos.

a. InsertionSort: ordenación por inserción
b. SelectionSort: ordenación por selección

2. Implementar un programa que crea un array de cadena de caracteres (`String`), inicialmente desordenado, y ordena los valores del array en orden **ascendente**, utilizando alguno de los algoritmos vistos en clase.

La clase `String` implementa el interfaz `java.lang.Comparable`, el cuál provee el siguiente método para la comparación de dos cadenas de caracteres:

```
public int compareTo(String s)
```

Ejemplo: Si tenemos dos objetos s1 y s2 de la clase `String`, la siguiente llamada

`s1.compareTo(s2)` {
- devuelve **un valor negativo**, si *s1* es menor que *s2*
- devuelve **0**, si *s1* es igual a *s2*
- devuelve **un valor positivo**, si *s1* es mayor que *s2*

3. Implementar un programa que define un array de cadena de caracteres (`String`), inicialmente desordenado, y ordena los valores del array en orden **descendente**, utilizando alguno de los algoritmos vistos en clase.

4. La clase `Persona` representa la información de una persona describiendo el nombre y la edad. La forma natural de comparar objetos (el criterio por defecto) en Java es implementando el interfaz `Comparable.` Se pide:
a. Implementar un programa que crea un array de `Persona`s y muestra en pantalla los datos de las personas del array ordenados, en orden ascendente, por el nombre de las personas.
b. Que modificarías del apartado anterior, para que los datos de las personas del array se muestren ordenados, en orden ascendente, por la edad de las personas? Impleméntalo, comentando el código anterior.
c. Que modificarías del apartado anterior, para que los datos de las personas del array se muestren ordenados, en orden ascendente, por el nombre de las personas y en caso de que coincida por la edad de las personas?
d. Utilizando el interfaz `Comparable`, es posible implementar un programa que tenga 3 arrays de `Persona`s y se pueda ordenar cada array utilizando un criterio de ordenación distinto para cada uno de ellos, por ejemplo los anteriores? Por qué?

5. Utilizando la clase `Persona` del ejercicio anterior, se quiere implementar un programa que crea 3 arrays de `Persona`s, en las cuales se almacena la información de las mismas personas, y ordena cada una de ellas con un criterio distinto de ordenación:

a. Ordenar por el nombre de la persona, en orden descendente.
b. Ordenar por la edad de la persona, en orden ascendente.
c. Ordenar primero por el nombre de las personas y en caso de que coincida por la edad, ambos en orden ascendente.

Nota: La clase `Persona` implementa el siguiente criterio de comparación por defecto (i.e. `Comparable`): una persona es menor que otra si el nombre del primero es menor que el segundo en orden alfabético. Para implementar los demás criterios implementa el interfaz `Comparator`.

6. Se dispone de dos arrays *Maestro* y *Esclavo*, del mismo tipo y número de elementos. Se deben imprimir dos columnas adyacentes. Se ordena el vector *Maestro*, pero siempre que un elemento *Maestro* se mueva, el elemento correspondiente de *Esclavo* debe moverse también; es decir, cualquier cosa que se haga a *Maestro[i]* debe hacerse a *Esclavo[i]*. Después de realizar la ordenación, se imprimen de nuevo los arrays. Escribir un programa que realice esta tarea.

Ayuda1 : Funcionamiento del *debugger*

1. Introducir un *breakpoint* para que el *debugger* pare en esa línea de código
2. Ejecutar el proyecto con el *debugger*
3. Step Into (F7): Para entrar dentro de la ejecución de un método o un bloque de código
4. Step over (F8): Para ejecutar sentencia por sentencia, pero sin entrar en la ejecución de los métodos
5. Step out (Shfit+F7): Para salir de la ejecución de un método
6. Ventana Stack: se ve la pila de llamadas producidas a métodos
7. Ventana Data: se ve el contenido de las variables locales del método seleccionado en la ventana Stack
8. Ventana Smart Data: se ve el contenido de las variables utilizadas en la sentencia que se va a ejecutar
9. Ventana Watch: se pueden añadir las expresiones que queremos analizar