Descripción general

```
/** * La clase {@code Ordenamiento} proporciona métodos estáticos para ordenar * arreglos genéricos que implementan la interfaz {@link Comparable}. * <p> * Se incluyen implementaciones de los algoritmos de ordenamiento: burbuja, * selección, inserción, montículos (heap sort) y shell. *  * @author   */
```

La clase `Ordenamiento` implementa diferentes algoritmos de ordenamiento genéricos en Java. Cada método puede ordenar cualquier tipo de arreglo siempre que los elementos sean comparables, es decir, que implementen la interfaz `Comparable`. Además, incluye un método para imprimir arreglos y otro auxiliar para intercambiar elementos.

Métodos

1.burbuja(T[] arreglo)

```
* Ordena un arreglo utilizando el algoritmo de burbuja.     *     * @param <T> el tipo de elementos del arreglo, que debe implementar {@code Comparable}     * @param arreglo el arreglo a ordenar     */
```

Este método implementa el algoritmo **Bubble Sort (Ordenamiento Burbuja)**. Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Se repite el proceso varias veces hasta que el arreglo queda completamente ordenado.

Características:

- Comparaciones repetidas.
- Eficiencia baja en arreglos grandes ($O(n^2)$).

```java
//ordenamiento burbuja
public static <T extends Comparable<T>> void burbuja(T[] arreglo) {
    int n = arreglo.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arreglo[j].compareTo(arreglo[j + 1]) > 0) {
                intercambiar(arreglo, j, j + 1);
            }
        }
    }
}
```

2.seleccion(T[] arreglo

```
/**     * Ordena un arreglo utilizando el algoritmo de selección.     *     * @param <T>     el tipo de elementos del arreglo, que debe implementar {@code Comparable}     * @param arreglo el arreglo a ordenar     */
```

Este método usa el **algoritmo de selección directa**. En cada iteración se busca el elemento más pequeño del arreglo desordenado y se intercambia con el primero de la parte desordenada.

Ventajas:

- Sencillo de entender e implementar.

```java
//ordenamiento por seleccion
public static <T extends Comparable<T>> void seleccion(T[] arreglo) {
    int n = arreglo.length;
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arreglo[j].compareTo(arreglo[minIndex]) < 0) {
                minIndex = j;
            }
        }
        intercambiar(arreglo, i, j: minIndex);
    }
}
```

3.insercion(T[] arreglo)

```
/**     * Ordena un arreglo utilizando el algoritmo de inserción.     *     * @param <T>     el tipo de elementos del arreglo, que debe implementar {@code Comparable}     * @param arreglo el arreglo a ordenar     */
```

Implementa el **algoritmo de ordenamiento por inserción**, que es eficiente para arreglos pequeños o casi ordenados. Se recorre el arreglo y se insertan los elementos en su posición correspondiente como si se tratara de ordenar cartas en mano.

Eficiencia:

- Mejor rendimiento que burbuja o selección en algunos casos.
- En el peor caso es también $O(n^2)$, pero con buen rendimiento para arreglos parcialmente ordenados.

```java
//ordenamiento por insercion
public static <T extends Comparable<T>> void insercion(T[] arreglo) {
    int n = arreglo.length;
    for (int i = 1; i < n; i++) {
        T clave = arreglo[i];
        int j = i - 1;
        while (j >= 0 && arreglo[j].compareTo(o: clave) > 0) {
            arreglo[j + 1] = arreglo[j];
            j--;
        }
        arreglo[j + 1] = clave;
    }
}
```

4.monticulos(T[] arreglo)

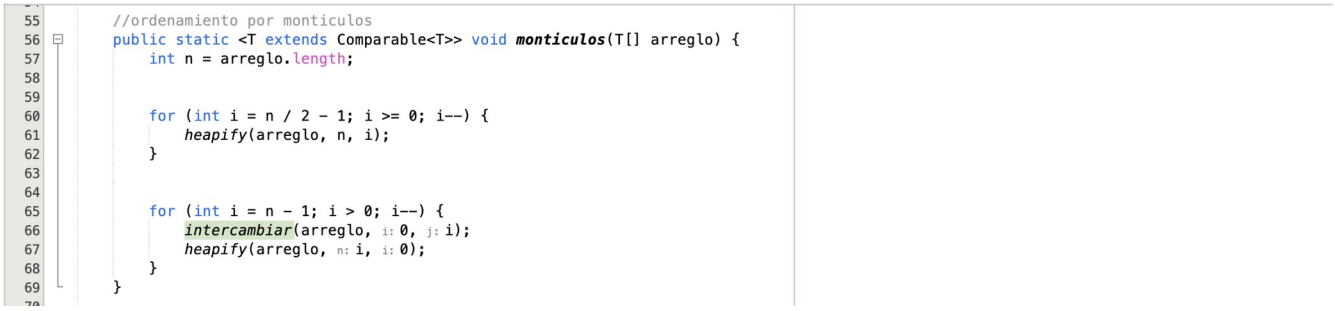
```
/**     * Ordena un arreglo utilizando el algoritmo de montículos (heap sort). * * @param <T>     el tipo de elementos del arreglo, que debe implementar {@code Comparable}     * @param arreglo el arreglo a ordenar     */
```

Este método utiliza el **Heap Sort (ordenamiento por montículos)**. Primero convierte el arreglo en un montículo (heap), y luego extrae el elemento máximo para colocarlo al final del arreglo, reconstruyendo el heap en cada iteración.

Ventajas:

•Eficiencia garantizada O(n log n).

•No requiere estructuras adicionales (es in-place).



```java
//ordenamiento por monticulos
public static <T extends Comparable<T>> void monticulos(T[] arreglo) {
    int n = arreglo.length;


    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arreglo, n, i);
    }


    for (int i = n - 1; i > 0; i--) {
        intercambiar(arreglo, i: 0, j: i);
        heapify(arreglo, n: i, i: 0);
    }
}
```

5.shell(T[] arreglo)

```
/**     * Ordena un arreglo utilizando el algoritmo de Shell.     *     * @param
<T>     el tipo de elementos del arreglo, que debe implementar {@code Comparable} *
@param arreglo el arreglo a ordenar     */
```

El **Shell Sort** es una mejora del ordenamiento por inserción. Compara y ordena elementos que están separados por una cierta distancia (`gap`). Este `gap` se reduce con el tiempo hasta llegar a 1.

Ventajas:

•Mejora considerable sobre el ordenamiento por inserción.

•Puede alcanzar una eficiencia cercana a O(n log n) dependiendo de los valores de `gap`.

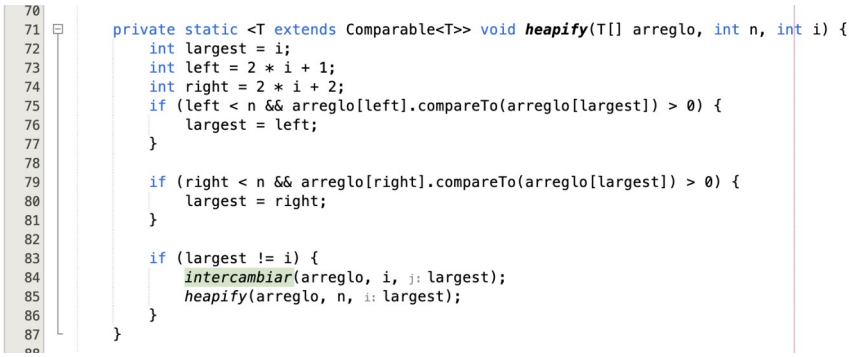
6.heapify(T[] arreglo, int n, int i)

```
/**     * Método auxiliar para mantener la propiedad del heap.     *     *
@param <T>     el tipo de elementos del arreglo, que debe implementar {@code
Comparable}     * @param arreglo el arreglo que representa el heap     *
@param n       el tamaño del heap     * @param i       el índice del elemento
a heapificar     */
```

Es un método auxiliar utilizado por `monticulos`. Su función es ajustar una subestructura del arreglo para que cumpla con las propiedades de un montículo máximo, asegurando que el elemento mayor esté en la raíz del subárbol.

7.intercambiar(T[] arreglo, int i, int j)

```
/**     * Intercambia dos elementos en el arreglo.     *     * @param <T>     el
tipo de elementos del arreglo     * @param arreglo el arreglo en el que se
```



```
intercambiarán los elementos     * @param i       el índice del primer elemento *
@param j       el índice del segundo elemento     */
```

Método privado que intercambia dos elementos dentro del arreglo. Se usa internamente en los diferentes algoritmos de ordenamiento para reorganizar los elementos.

8. imprimirArreglo(T[] arreglo)

```
/**     * Imprime los elementos del arreglo en la consola.     *     * @param <T>
el tipo de elementos del arreglo     * @param arreglo el arreglo a imprimir     */
```

Este método imprime el contenido del arreglo en consola. Recorre el arreglo usando un ciclo `for-each` y muestra los elementos separados por espacios.