


|                                                                                   |                                                                   |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------|
|  | ESCUELA DE INGENIERÍAS T.I.C                                      |
|                                                                                   | <b>Workshop - Sort - Arrays &amp; Collections</b><br>Julio 6-2018 |

En el siguiente taller se presentara la forma como el lenguaje de programación Java permite utilizar las rutinas implementadas para ordenamiento de datos.

Las librerías utilizadas y que deben ser importadas en el programa que necesita ordenar datos son las siguientes:

- **import** java.util.Arrays;
- **import** java.util.Collections;

La primera librería permite ordenar datos almacenados en arreglos estáticos. La segunda librería permite ordenar colecciones de datos almacenados en las clases LinkedList, Array List entre otras.

#### 1. Ordenamiento de datos en Arreglos estáticos.

- Ordenamiento ascendente
- Ordenamiento de un sub-array
- Ordenamiento descendente.
- Ordenamiento de arreglos con datos de tipo String

```
import java.util.Arrays;
import java.util.Collections;

/*
 * ARRAYS SORTS
 */

public class SortJava
{
    public static void main(String[] args)
    {
        /* We can also sort in ASCENDING order.
         * Our arr contains 8 elements
         * -----
         */
        int[] arr1 = {13, 7, 6, 45, 21, 9, 101, 102};

        Arrays.sort(arr1);
        System.out.println("Modified arr[] : ASC : " + Arrays.toString(arr1));

        /* We can also use sort() to sort a SUBARRAY of arr[]
         * Sort subarray from index 1 to 4, i.e.,
         * Our arr contains 8 elements
         * -----
         * only sort subarray {7, 6, 45, 21} and
         * keep other elements as it is.
         */
        int[] arr2 = {13, 7, 6, 45, 21, 9, 101, 102};

        Arrays.sort(arr2, 1, 5);
        System.out.println("Modified arr[] : SUB : " + Arrays.toString(arr2));
    }
}
```

```

/* We can also sort in DESCENDING order.
 * Our arr contains 8 elements
 * -----
 * Note that we have Integer here instead of
 * int[] as Collections.reverseOrder doesn't
 * work for primitive types.
 */
Integer[] arr3 = {13, 7, 6, 45, 21, 9, 101, 102};

Arrays.sort(arr3, Collections.reverseOrder());
System.out.println("Modified arr[] : DSC : " + Arrays.toString(arr3));

/* We can also sort strings in alphabetical order.
 * -----
 */
String arr[] = {"practice.geeksforgeeks.org", "quiz.geeksforgeeks.org",
               "code.geeksforgeeks.org"};

/* Sorts arr[] in ascending order
 */
Arrays.sort(arr);
System.out.println("Modified arr[] : ASC : " + Arrays.toString(arr));

/* Sorts arr[] in descending order
 */
Arrays.sort(arr, Collections.reverseOrder());
System.out.println("Modified arr[] : DSC : " + Arrays.toString(arr));

}
}

```

## 2. Ordenamientos de datos en Colecciones.

En este ejemplo se va utilizar las colecciones ArrayList y LinkedList donde se almacenaran objetos de tipo Students.

La clase Students implementará la interface "Comparable", esto indica que dependiendo de la forma como se desea ordenar la colección se tiene que reescribir el método "compareTo".

Primera versión del método "compareTo".

```

@Override
/*
 * Ordena por un solo criterio (Nombre)
 * Retorna this < o : valor negativo
 * Retorna this > o : valor positivo
 * Retorna this == o : valor 0
 * return (comparación) : Ascendente
 * return -1 * (comparación) : Descendente
 */

public int compareTo(Students o) { return getNombre().compareTo(o.getNombre()); }

```

Con esta versión del método "compareTo" se ordenan los estudiantes por el nombre, si existen varios estudiantes con el mismo nombre y diferente apellido, los apellidos no son tenidos en cuenta para el ordenamiento.

Segunda versión del método “comparableTo”.

```
/*
 * Ordena por dos criterios (Nombre y luego Apellidos)
 * Retorna this < o : valor negativo
 * Retorna this > o : valor positivo
 * Retorna this == o : valor 0
 * return (comparación) : Ascendente
 * return -1 * (comparación) : Descendente
 */

public int compareTo(Students o)
{ if (getNombre().compareTo(o.getNombre())==0)
    return getApellidos().compareTo(o.getApellidos());
  else
    return getNombre().compareTo(o.getNombre());
}
```

Esta segunda versión del método “comparableTo”, ordena los estudiantes por el nombre y también por apellidos.

Tercera versión del método “comparableTo”.

```
/*
 * Ordena por tres criterios (Edad, Nombre y luego Apellidos)
 * Retorna this < o : valor negativo
 * Retorna this > o : valor positivo
 * Retorna this == o : valor 0
 * return (comparación) : Ascendente
 * return -1 * (comparación) : Descendente
 */

public int compareTo(Students o)
{ if (Integer.compare(getEdad(), o.getEdad()) == 0)
    if (getNombre().compareTo(o.getNombre())==0)
        return getApellidos().compareTo(o.getApellidos());
    else
        return getNombre().compareTo(o.getNombre());
  else
    return Integer.compare(getEdad(), o.getEdad());
}
```

En esta última versión, ordena los estudiantes primero por edad y luego por nombre y finalmente por apellidos.

### Clase Students

A continuación se presenta la clase “Students”, reescriba el método “comparableTo” de las tres formas explicadas anteriormente y realice la pruebas necesarias para verificar los resultados.

```
public class Students implements Comparable<Students>
{
    // Atributos
    private String nombre;
    private String apellidos;
    private int edad;

    // Constructor
    public Students(String nom, String ape, int edad)
    {
        this.nombre = nom; this.apellidos = ape; this.edad = edad;
    }

    public String getNombre() { return nombre; }
    public String getApellidos() { return apellidos; }
    public int getEdad() { return edad; }

    public int comparableTo() { return 0; } // reescribir este método.
}
```

### Clase SortCollections

En esta clase se instancian varios objetos de tipo “Students” y se almacenan en cualquiera de las dos colecciones “ArrayList, LinkedList”, realice pruebas con estas colecciones.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;

public class SortCollections
{
    public static void main(String[] args)
    {
        // LinkedList<Students> lista = new LinkedList<Students>();
        ArrayList<Students> lista = new ArrayList<Students>();

        lista.add(new Students("Barbara", "Lopez Zambrano",30));
        lista.add(new Students("Xuan", "Gomez Amador",15));
        lista.add(new Students("Barbara", "Lopez Aldna",28));
        lista.add(new Students("Xuan", "Martinez Lopez",45));
        lista.add(new Students("Xuan", "Alcantara Alape",39));
        lista.add(new Students("Alex", "Jimenez Suarez",12));
        lista.add(new Students("Kaka", "King Lucena",34));
        lista.add(new Students("Xuan", "Gomez Alfonso",45));
        lista.add(new Students("Logi", "Ciceron Figueroa",18));
        lista.add(new Students("Loki", "Urdaneta Hernandez",64));
        lista.add(new Students("Alexis", "Sabogal Pedraza",73));
        lista.add(new Students("Zuan", "Barbosa Bueno",25));
        lista.add(new Students("Xuan", "Ñañez Velasquez",10));

        Iterator<Students> iterador = lista.listIterator();

        System.out.println("Unsorted...");

        while (iterador.hasNext())
        {
            Students e = iterador.next();
            System.out.println(e.getEdad()+ "\t : " + e.getNombre() + "\t" + " : " +
                e.getApellidos());
        }
    }
}
```

```
Collections.sort(lista);

System.out.println("\n.....\n");

iterador = lista.listIterator();
System.out.println("sorted...");

while (iterador.hasNext())
{ Students e = iterador.next();
  System.out.println(e.getEdad()+ "\t : " + e.getNombre() + "\t" + " : " +
    e.getApellidos());
}
}
```