



Programación Java

► Introducción al lenguaje Java

//Práctica integradora

Objetivo

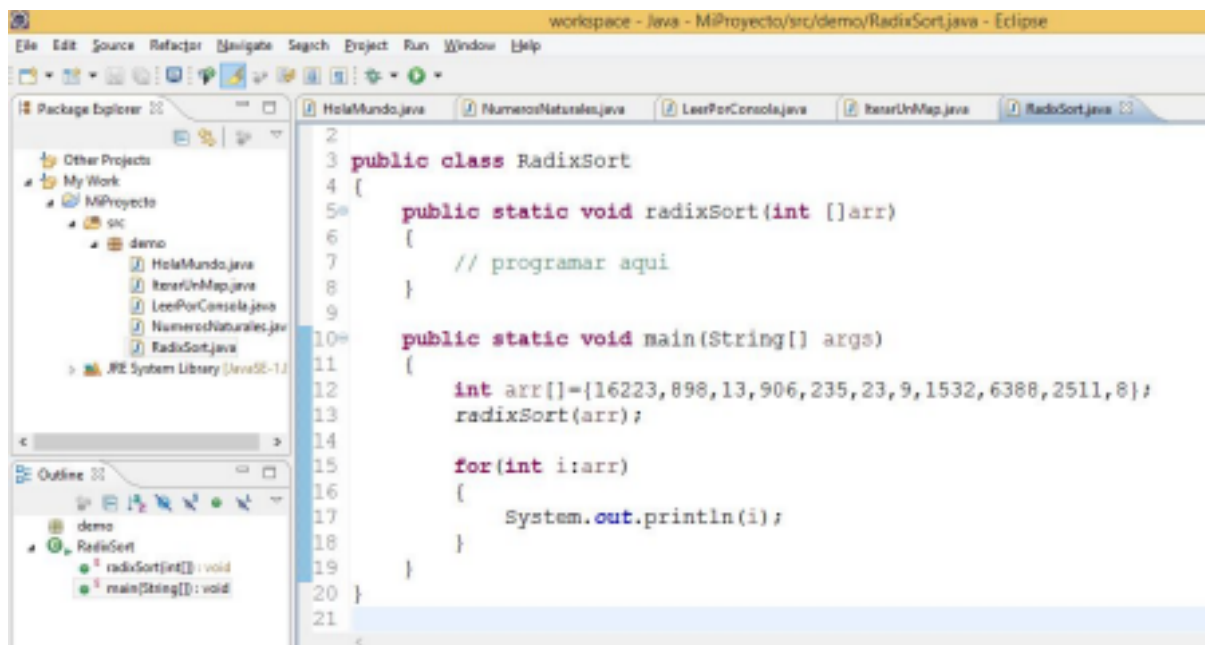
El objetivo de esta guía práctica es que podamos afianzar y profundizar los conceptos sobre colecciones y estructura de datos. Para esto vamos a plantear un único ejercicio; una vez que lo resolvamos ya no tendremos dudas sobre cómo se programa en Java.

¿Preparados?



Ordenamiento de raíz

Desarrollar el método radixSort, que ordena un array de enteros mediante dicho algoritmo de ordenamiento: “ordenamiento de raíz”.



```
2
3 public class RadixSort
4 {
5     public static void radixSort(int []arr)
6     {
7         // programar aqui
8     }
9
10    public static void main(String[] args)
11    {
12        int arr[]={16223,898,13,906,235,23,9,1532,6388,2511,8};
13        radixSort(arr);
14
15        for(int i:arr)
16        {
17            System.out.println(i);
18        }
19    }
20 }
21
```



Como este algoritmo es relativamente complejo, luego de analizarlo dividiremos la tarea en problemas bien definidos y más acotados, de modo tal que cada uno sea, en sí mismo, un pequeño desafío.

Entendiendo el algoritmo

Radix Sort es un algoritmo diseñado para ordenar arrays de cadenas de caracteres, pues su lógica se basa en clasificar los elementos del array según cuál sea el carácter que tienen en la última posición, luego en la ante última, en la penúltima, y así sucesivamente hasta llegar a clasificarlos según cuál sea el primer carácter de cada elemento. Es importante conocer este funcionamiento, porque si vamos a ordenar un array de enteros, primero tendremos que convertirlo en un array de cadenas. Por



supuesto, podemos usar Radix Sort para ordenar arrays de cualquier tipo de dato, pero para evitar complicaciones que en este momento serían irrelevantes, trabajaremos pensando en arrays de enteros: `int[]`.

Dicho esto, el algoritmo consiste en los siguientes pasos:

1. Convertir el array que vamos a ordenar, de `int[]` a `String[]`.

Por ejemplo, si vamos a ordenar `int iArr[] = {4,28,132,3,61,5}`, debemos convertirlo en `String sArr[] = {"4","28","132","3","61","5"}`.

2. Todos los elementos del `String[]` deben tener la misma longitud. Para esto los completaremos con 0 (ceros) a la izquierda (coincidiendo con la cantidad de dígitos del número más grande). Según nuestro ejemplo, el array `sArr` debería quedar así: `{"004", "028", "132", "003", "061", "005"}`.

3. Creamos 10 listas inicialmente vacías. `L0` (será la lista para colocar los números que terminan con 0, En `L1` colocaremos los números que terminan con 1, y genéricamente hablando, `Li` será la lista donde colocaremos aquellos números

cuyo último dígito es `i`.

4. Recorremos el `String[]`; por cada elemento verificamos cuál es su último dígito, lo removemos del array y lo agregamos a la lista que corresponda.
5. Recorremos las listas en orden: `L0`, `L1`, ..., `L9`, y regeneramos el `String[]` con los elementos de `L0`, luego los de `L1` y así sucesivamente.





6. Volvemos a (4), pero ahora trabajaremos con el anteúltimo dígito. Es decir L0 será la lista de los elementos que tienen 0 en su anteúltimo dígito, L1 tendrá aquellos elementos que tienen 1 en su anteúltimo lugar, etcétera.
7. En cada iteración evaluamos y clasificamos los elementos del array según cuál sea su último, anteúltimo, antepenúltimo, (...) y primer dígito. Luego de esto el array quedará ordenado.

Ejemplo: `int[] a = {3, 673, 106, 45, 2,23 }.`

Convertimos y completamos con ceros a la izquierda:

`String[] s = {003, 673, 106, 045, 002,023 }` (por claridad, omití poner las comillas). Recorremos clasificando en listas según cuál sea el último dígito.

`L2 = {002}`

`L3 = {003,673,023}`

`L5 = {045}`

`L6 = {106}`

Rearmamos el array:

`String[] s = {002,003,673,023,045,106}.`

Repetimos considerando el anteúltimo dígito.

`L0 = {002,003,106}`

`L2 = {023}`

`L4 = {045}`

`L7 = {673}`

Rearmamos el array:





```
String[] s = {002,003,106,023,045,673}
```

Repetimos considerando el antepenúltimo dígito (que en este caso es el primero). L0 = {002,003,023,045}

L1 = {106}

L6 = {673}

Rearmamos y el array quedó ordenado.

```
String[] s = {003,004,023,045,106,673}.
```

Desarrollo del algoritmo

Como comentamos más arriba, programar este algoritmo de ordenamiento podría resultar algo complejo para programadores con poca experiencia. Sin embargo, teniendo en cuenta cada uno de los pasos del algoritmo, podemos dividir este problema complejo en varios problemas más pequeños y fáciles de resolver.

1. Desarrollar, en el orden en que aparecen, las funciones de la clase `StringUtil.java`.
2. Discutir (en grupo) y diseñar una estructura de datos que permita dar soporte al algoritmo Radix Sort.
3. Desarrollar el método `radixSort` según el algoritmo planteado.