



Programación Java

► Introducción al lenguaje Java

// Práctica integradora

Objetivo

El objetivo de esta guía práctica, además de aplicar los conocimientos teóricos que hemos estudiado, es investigar (Google mediante) cómo realizar ciertas tareas que no vimos, y poner todo junto a trabajar..

Sabemos que puedes hacerlo, sólo ¡¡ HAZLO !!



Ejercicio

Crearemos una clase que nos permitirá crear dinámicamente instancias de diferentes objetos. Una factoría genérica. Para esto, será necesario investigar algunas cuestiones. Usaremos las clases Properties, Class y alguna más.

1. Crear una interface llamada `Sorter<T>`, con el método abstracto `public void sort(T arr[], Comparator<T> c)`.
2. Crear una clase llamada `QuickSortSorterImple` que implemente la interface (en T) anterior y sobrescriba el método `sort` (dejarlo vacío por el momento).

3. Buscar en Int

ernet alguna implementación del algoritmo de ordenamiento QuickSort (por ejemplo, usar las palabras: QuickSort.java) y utilizarlo para programar el método sort del punto anterior, haciendo las modificaciones que sean necesarias.

4. Idem 2 y 3, pero con el algoritmo de ordenamiento HeapSort, creando la clase HeapSortSorterImpl. 5. Idem 2 y 3, pero con el algoritmo de la burbuja, creando la clase BubbleSortSorterImpl.



6. Crear un archivo de texto llamado MiFactory.properties en la carpeta del proyecto. El contenido del archivo debe ser: `sorter=paquete.QuickSortSorterImpl` (sin comillas ni nada más que lo resaltado en amarillo).
7. Crear la clase MiFactory con el método public static Object getInstance(String objName).
- Usando la clase Properties (investigar) leer el contenido de la propiedad “sorter” del archivo creado en (6), que en este caso será la cadena: “paquete.QuickSortSorterImpl”.
 - Usando la clase Class (investigar) crear dinámicamente un objeto del tipo obtenido en (8) y retornarlo.
8. Crear un programa, instanciar un Sorter usando Factory, ordenar un array de enteros y otro de cadenas.
9. Modificar la implementación del objeto sorter en MiFactory.properties, por alguna de las otras implementaciones disponibles (bubble o heap).
10. Probar que todo funciona correctamente.



11. Crear una clase llamada Time, con dos métodos: start y stop, que respectivamente guardan en variables de instancia la hora actual expresada en milisegundos (investigar cómo hacerlo).
12. Agregar a la clase Timer el método elapsedTime que retorna el tiempo transcurrido entre el tiempo de inicio (start) y el de fin (stop).
13. Crear un array de 100 mil valores enteros, ordenado de mayor a menor.
14. Ordenarlo con BubbleSort y verificar cuánto tiempo demoró el proceso. 15. Cambiar la implementación (modificando en MiFactory.properties) por Heap o Quick sort y verificar cuánto demora el proceso con un algoritmo más performante.
- 16. En grupo: sacar conclusiones.**