Programación III
TEMA 6: HEAP
Práctica nº 6

1. Muestre las transformaciones que sufre una **Max HEAP** (inicialmente vacía) al realizar las siguientes operaciones (muestre las transformaciones en el arreglo que la representa y además bajo la forma de árbol):

    a. Insertar 50, 52, 41, 54, 46
    b. Eliminar uno a uno, los tres primeros elementos
    c. Insertar 45, 48, 55, 43
    d. Eliminar uno a uno, los tres primeros elementos

2. Considere la especificación de la clase MinHeap vista en la teoría

*por tratarse de una MinHeap, existe el método mínimo, en el caso de una MaxHeap sería más correcto tener un método llamado máximo.

- El constructor **MinHeap()** inicializa una Heap vacía con un tamaño por defecto (por ej. 100). Tenga en cuenta que Java no permite instanciar directamente un arreglo de genéricos, por tanto deberá realizar lo sgte. dentro del constructor:

```
datos = (T[]) new Comparable[100];
```

- El constructor **MinHeap(T [] datos)** inicializa una Heap vacía a partir de un arreglo vacío que se recibe como parámetro. De modo análogo lo hace el constructor que recibe una lista
- El método **eliminar(): T elimina** el tope y lo **retorna**.
- El método **agregar(T elem):boolean** agrega un elemento en la heap y devuelve true si la inserción fue exitosa y false en caso contrario.
- El método **esVacia():boolean** devuelve true si no hay elementos para extraer.
- El método **tope() : T retorna** el tope de la heap.
- El método **imprimir() : void** imprime el contenido de la heap.

a. A partir de la especificación anterior **implemente** la Clase MinHeap.
NOTA: Recordar que los arreglos en Java son Zero-Based, con lo cual la primera posición siempre es 0, y es conveniente **poner al primer elemento a partir de la posición 1 del arreglo**, dejando esa posición para hacer intercambios o simplemente no usarla.
También tener en cuenta que la estructura a utilizar internamente para almacenar los elementos, debe tener un mecanismo de comparación entre objetos para poder evaluar las claves, por lo que se recomienda utilizar una estructura que almacene Comparable<T>

b. ¿Qué cambios debería hacer sobre la definición y/o implementación anterior, si los elementos estuvieran ordenados con el criterio de Max Heap?

c. Implemente el método **T extract()** en la clase **MinHeap** para que no devuelva valores repetidos. Por ejemplo, si la Heap contiene el siguiente conjunto: 10, 10, 9, 8, 8, 7, 7, 7, 5, 4, 2, 2, la primera vez, el método extract() devuelve 10, la segunda vez devuelve 9, y las posteriores invocaciones devuelven 8, 7, 5, 4, 2.

3. Se desea implementar una clase Impresora que ofrezca la funcionalidad dada por la siguiente interfaz:

```
public interface DispositivoImpresion {
      // Almacena un nuevo documento en el dispositivo
      void nuevoDocumento(Documento d);

      // Imprime (saca por pantalla) el documento almacenado más corto
      //y lo elimina de memoria. Si no hay documentos en espera
      //devuelve false
      boolean imprimir();
}
```

La idea de imprimir el documento más corto en primer lugar es para evitar que un trabajo largo bloquee durante mucho tiempo al resto. Se necesita:

a. Modelar la clase Documento, que consiste simplemente en una cadena de texto (String)
b. Modelar la clase Impresora que implemente la interfaz DispositivoImpresion

4. Se cuenta con una cola de prioridades en donde se almacenan los **procesos** que se ejecutan en un sistema operativo multitarea. En un determinado momento uno de los procesos comienza a consumir muchos recursos, por lo que el sistema operativo decide bajarle la prioridad. Implemente un algoritmo que dada una ubicación en la MaxHeap, y considerando que el elemento en dicha ubicación acaba de decrementar su valor, permita reordenar los elementos de la MaxHeap.