Departamento de Lenguajes y Sistemas Informáticos

Material permitido: NINGUNO. **Duración:** 2 horas

Entregue este folio con sus datos consignados

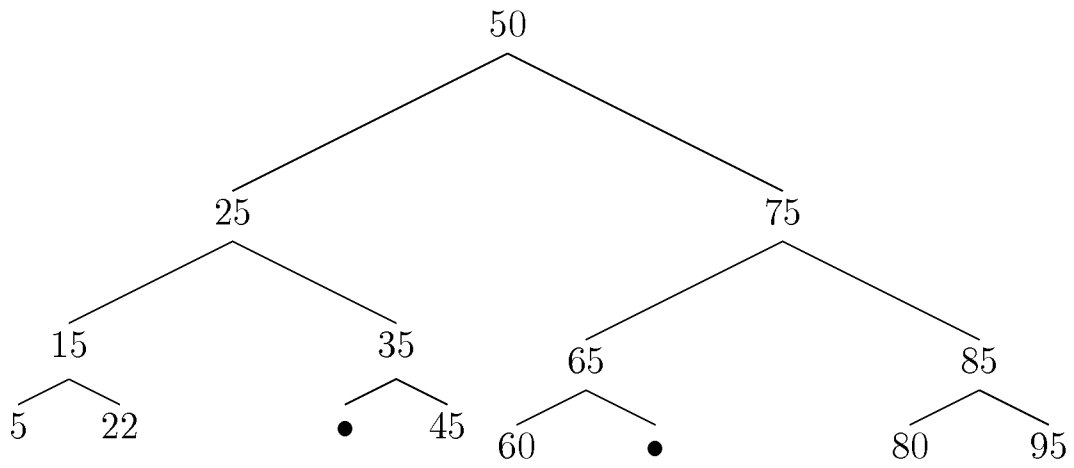
Alumno:	**Identificación:**
C. Asociado en que realizó la Práctica Obligatoria:	

P1 (1'5 puntos) **Práctica**. Implemente un método `ListIF<Integer> getSpaceFloor(TType type)` de la clase `Parking` que obtenga una lista con el número de plazas libres del tipo de vehículo especificado por parámetro **en cada planta** del parking.

---

1. (1 punto) Implementar en JAVA un método `void replace(T viejo, T nuevo)` dentro de la clase `StackIF<T>` consistente en asignar el valor del segundo argumento a los elementos de la pila cuyo valor sea el del primer argumento manteniendo el orden de la pila original.

2. Dado el siguiente árbol binario (los nodos marcados como • representan hojas vacías):



   Se pide:

   a) (0.25 puntos) Recorrido en preorden

   b) (0.25 puntos) Recorrido en inorden

   c) (0.25 puntos) Recorrido en postorden

   d) (0.25 puntos) Justificar razonadamente si se trata de un árbol AVL o no

3. Un multiconjunto o bolsa es una estructura que se diferencia de los conjuntos porque cada uno de sus elementos tiene asociada una multiplicidad que indica el número de repeticiones de dicho elemento en el multiconjunto. Suelen representarse mediante pares ordenados en los que la primera componente es el elemento y la segunda componente es su multiplicidad asociada. Por ejemplo, el multiconjunto $\{x, x, y, y, y, z\}$ se representa como $\{(x, 2), (y, 3), (z, 1)\}$. Las operaciones que pueden aplicarse sobre los multiconjuntos aparecen definidas en la siguiente interfaz:

   MultiSetIF

```
//Representa un multiconjunto de elementos
public interface MultiSetIF<T>{

/* Decide si el elemento e (@param) pertenece al
*  multiconjunto [0.25 puntos]*/
public boolean contains(T e);
```

```
    /* Añade el elemento e al multiconjunto (o aumenta multiplicidad
    * de e  en caso de que pertenezca al multiconjunto)[0.25 puntos]*/
    public void add(T e);

    /* Elimina una repetición del elemento e en el multiconjunto.
    * Elimina el elemento si la multiplicidad es 0.[0.25 puntos]*/
    public void removeOccurrence(T e);

    /* Multiplicidad del elemento e en el multiconjunto. Es 0 si
    *  no pertenece al mismo.[0.25 puntos]*/
    public int multiplicity(T e);

    /* Devuelve la unión con el multiconjunto M (@param): Añade los
    *  elementos no pertenecientes con la multiplicidad de M, y suma
    *  las multiplicidades para los elementos comunes [1 punto]*/
    public MultiSetIF<T> union(MultiSetIF<T> M);

    /* Devuelve la interseccion con el multiconjunto M (@param):
    *  El resultado contiene los elementos comunes entre los dos
    *  multiconjuntos. La multiplicidad de estos elementos es la
    *  menor entre los dos multiconjuntos (usar función mínimo
    *  Math.min(a,b) para calcularla) [1 punto]*/
    public MultiSetIF<T> intersection(MultiSetIF<T> M);

    /* Decide si un multiconjunto M (@param) es submulticonjunto.
    *  Debe cumplirse:
    *   1. El conjunto de elementos de M es un subconjunto de los
    *      elementos del multiconjunto
    *   2. La multiplicidad de cada elemento e en M es menor o igual
    *      que la de dicho elemento e en el multiconjunto [2 puntos].
    */
    public boolean isSubMultiSet(MultiSetIF<T> M);
}
```

a) (1 punto) Representación interna de este tipo de datos usando los TADs vistos en la asignatura. Justificar razonadamente la elección. (Pista: Puede ser conveniente crearse una clase que represente elementos de un multiconjunto)

b) (5 puntos) Basándose en la repuesta anterior, implementar todos los métodos de la interfaz `MultiSetIF<T>`

c) (0'5 puntos) Calcular el coste asintótico temporal en el caso peor del método `isSubMultiSet` en su implementación.

ListIF (Lista)

```
/* Representa una lista de elementos */
public interface ListIF<T>{
   /* Devuelve la cabeza de una lista*/
    *
    public T getFirst ();
    /* Devuelve: la lista excluyendo la
       cabeza. No modifica la estructura
       */
    public ListIF<T> getTail ();
   /* Inserta una elemento (modifica la
      estructura)
    * Devuelve: la lista modificada
    * @param elem El elemento que hay que
       añadir*/
    public ListIF<T> insert (T elem);
    /* Devuelve: cierto si la lista esta
       vacia */
    public boolean isEmpty ();
    /* Devuelve: cierto si la lista esta
       llena*/
    public boolean isFull();
    /* Devuelve: el numero de elementos
       de la lista*/
    public int getLength ();
    /* Devuelve: cierto si la lista
       contiene el elemento.
     * @param elem El elemento buscado */
    public boolean contains (T elem);
    /* Ordena la lista (modifica la lista)
     * @Devuelve: la lista ordenada
     * @param comparator El comparador de
        elementos*/
    public ListIF<T> sort
       (ComparatorIF<T> comparator);
    /*Devuelve: un iterador para la
       lista*/
    public IteratorIF<T> getIterator ();
}
```

StackIF (Pila)

```
/* Representa una pila de elementos */
public interface StackIF <T>{
    /* Devuelve:  la cima de la pila */
     public T getTop ();
    /* Incluye un elemento en la cima de
       la pila (modifica la estructura)
     * Devuelve: la pila incluyendo el
        elemento
     * @param elem Elemento que se quiere
        añadir */
     public StackIF<T> push (T elem);
    /* Elimina la cima de la pila
       (modifica la estructura)
     * Devuelve: la pila excluyendo la
        cabeza */
     public StackIF<T> pop ();
    /* Devuelve: cierto si la pila esta
       vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la pila esta
       llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos
       de la pila */
     public int getLength ();
    /* Devuelve: cierto si la pila
       contiene el elemento
     * @param elem Elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la pila*/
     public IteratorIF<T> getIterator ();
}
```

QueueIF (Cola)

```
/* Representa una cola de elementos */
public interface QueueIF <T>{
    /* Devuelve: la cabeza de la cola */
     public T getFirst ();
    /* Incluye un elemento al final de la
       cola (modifica la estructura)
     * Devuelve: la cola incluyendo el
        elemento
     * @param elem Elemento que se quiere
        añadir */
     public QueueIF<T> add (T elem);
    /* Elimina el principio de la cola
       (modifica la estructura)
     * Devuelve: la cola excluyendo la
        cabeza    */
     public QueueIF<T> remove ();
    /* Devuelve: cierto si la cola esta
       vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la cola esta
       llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos
       de la cola */
     public int getLength ();
    /* Devuelve: cierto si la cola
       contiene el elemento
     * @param elem elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la cola*/
     public IteratorIF<T> getIterator ();
}
```

TreeIF (Árbol general)

```
/* Representa un arbol general de
   elementos */
public interface TreeIF <T>{
    public int PREORDER  = 0;
    public int INORDER   = 1;
    public int POSTORDER = 2;
    public int BREADTH   = 3;
    /* Devuelve:  elemento raiz del arbol
       */
     public T getRoot ();
    /* Devuelve: lista de  hijos de un
       arbol.*/
     public ListIF <TreeIF <T>>
        getChildren ();
```

```
    /* Establece el elemento raiz.
     * @param elem Elemento que se quiere
        poner como raiz*/
     public void setRoot (T element);
    /* Inserta un subarbol como ultimo
       hijo
     * @param child el hijo a insertar*/
     public void addChild (TreeIF<T>
        child);
    /* Elimina el subarbol hijo en la
       posicion index-esima
     * @param index indice del subarbol
        comenzando en 0*/
     public void removeChild (int index);
    /* Devuelve: cierto si el arbol es un
       nodo hoja*/
     public boolean isLeaf ();
    /* Devuelve: cierto si el arbol es
       vacio*/
     public boolean isEmpty ();
    /* Devuelve: cierto si la lista
       contiene el elemento
     * @param elem Elemento buscado*/
     public boolean contains (T element);
    /* Devuelve: un iterador para la lista
     * @param traversalType el tipo de
        recorrido, que
     * sera PREORDER, POSTORDER o BREADTH
        */
     public IteratorIF<T> getIterator
        (int traversalType);
}
```

BTreeIF (Árbol Binario)

```
/* Representa un arbol binario de
   elementos */
public interface BTreeIF <T>{
  public int PREORDER  = 0;
  public int INORDER   = 1;
  public int POSTORDER = 2;
  public int LRBREADTH = 3;
  public int RLBREADTH = 4;
 /* Devuelve: el elemento raiz del arbol
    */
  public T getRoot ();
 /* Devuelve: el subarbol izquierdo o
    null si no existe */
  public BTreeIF <T> getLeftChild ();
 /* Devuelve: el subarbol derecho o null
    si no existe */
  public BTreeIF <T> getRightChild ();
 /* Establece el elemento raiz
  * @param elem Elemento para poner en la
     raiz */
  public void setRoot (T elem);
 /* Establece el subarbol izquierdo
  * @param tree el arbol para poner como
     hijo izquierdo */
  public void setLeftChild (BTreeIF <T>
     tree);
 /* Establece el subarbol derecho
  * @param tree el arbol para poner como
     hijo derecho */
  public void setRightChild (BTreeIF <T>
     tree);
 /* Borra el subarbol izquierdo */
  public void removeLeftChild ();
 /* Borra el subarbol derecho */
  public void removeRightChild ();
 /* Devuelve: cierto si el arbol es un
    nodo hoja*/
  public boolean isLeaf ();
 /* Devuelve: cierto si el arbol es vacio
    */
  public boolean isEmpty ();
 /* Devuelve: cierto si el arbol contiene
    el elemento
  * @param elem Elemento buscado */
  public boolean contains (T elem);
 /* Devuelve un iterador para la lista.
  * @param traversalType el tipo de
     recorrido que sera
    PREORDER, POSTORDER, INORDER,
       LRBREADTH o RLBREADTH */
 public IteratorIF<T> getIterator (int
    traversalType);
}
```

ComparatorIF

```
/* Representa un comparador entre
   elementos */
public interface ComparatorIF<T>{
  public static int LESS  = -1;
  public static int EQUAL = 0;
  public static int GREATER = 1;
 /* Devuelve: el orden de los elementos
  * Compara dos elementos para indicar si
     el primero es
  * menor, igual o mayor que el segundo
     elemento
  * @param e1 el primer elemento
  * @param e2 el segundo elemento   */
  public int compare (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    menor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isLess (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    igual que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isEqual (T e1, T e2);
 /* Devuelve: cierto si un elemento es
    mayor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento*/
  public boolean isGreater (T e1, T e2);
}
```

IteratorIF

```
/* Representa un iterador sobre una
   abstraccion de datos */
public interface IteratorIF<T>{
```

```
    /* Devuelve: el siguiente elemento de
       la iteracion */
     public T getNext ();
    /* Devuelve: cierto si existen mas
       elementos en el iterador */
     public boolean hasNext ();
    /* Restablece el iterador para volver
       a recorrer la estructura */
     public void reset ();
}
```