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**. Impleméntese el siguiente comando de usuario para añadirlo a las funcionalidades del sistema de ficheros de la práctica.

touch touch - cambia el tamaño de un fichero

SINTAXIS

`touch path [-n num]`

Cambia el tamaño del fichero que indique el parámetro posicional `path`. Si el fichero no existía, lo crea sin contenido y con tamaño nulo. Si el fichero existía, lo **substituye** por uno de igual nombre y con el tamaño indicado por el parámetro de la opción -n. Su contenido será aleatorio.

OPCIONES

-n indica que el siguiente parámetro es el número de bytes del contenido del fichero

---

1. (1'5 puntos) ¿Qué cambios aplicaría sobre la implementación de un árbol binario para que se pueda conocer el número de nodos del árbol en tiempo constante? ¿qué operaciones habría que modificar para conseguirlo?

2. (1'5 puntos) Extienda la implementación de la interfaz `ListIF<T>` para incluir el método `boolean isPalindrome()`, que determine si la lista es palíndroma. Una lista se dice palíndroma si el resultado de recorrerla desde el primero al último es el mismo que el de recorrerla en sentido inverso. **Nota:** la comparación por igualdad de dos objetos `e1` y `e2` de tipo `T` se realiza como sigue: `e1.equals(e2)`

3. Queremos programar la clase de los polinomios con coeficientes enteros y de grado conocido, por ejemplo, $4x^3 - 3x^2 + 2x - 5$. Su interfaz sería:

PolynomialIF

```
// Representa un Polinomio con coeficientes enteros y de grado conocido
  public interface PolynomialIF {

    /* añade un término al polinomio cuyo coeficiente es coef y cuyo
     * grado es degree  [0'5 puntos] */
    public void addTerm (int degree, int coef);

    /* substituye el coeficiente del termino de grado degree por
     * el parámetro coef [0'5 puntos] */
    public void replaceTerm (int degree, int coef);

    /* devuelve el coeficiente del termino de grado degree en el
     * polinomio [0'5 puntos] */
    public int  getCoefficient (int degree);

    // devuelve la suma con el polinomio parámetro [1 punto]
    public PolynomialIF sum (PolynomialIF  poly);

    // evalua el polinomio en el punto x [1 punto]
    public float  eval (float x);
```

```
    /* calcula la derivada del polinomio, donde la derivada
     *    d(ax^n) = nax^{n-1}  [1 punto]  */
    public PolynomialIF derivative ();
}
```

a) (0'5 puntos) Detalle el constructor de una clase que implemente esta interfaz. Describa detalladamente cómo realizaría la representación interna de este tipo. Justifique su elección

b) (4'5 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz `PolynomialIF`. Se valorará que detalle los contratos de las operaciones (pre y postcondiciones) o que comente, al menos, las restricciones que deben aplicarse a los parámetros de entrada. (**Nota:** las puntuaciones asignadas a cada método se indican en la especificación de dicho método en la interfaz del tipo).

c) (0'5 punto) Calcule el coste asintótico temporal en el caso peor del método `eval (float x)` para la representación seleccionada.

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 ();
}
```