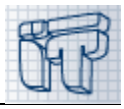


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 09 mayo 2019

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.
- Utiliza el espacio de los recuadros en blanco para responder a bolígrafo a cada uno de los apartados de los problemas (no usar lápiz para las respuestas finales).

NOTA: No está permitido crear más atributos ni métodos de los que figuran en el enunciado (no son necesarios).

Problema 1 (Listas / pilas / colas /o colas dobles)

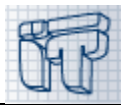
Dadas las clases Node y MyBasicLinkedList. Asumiendo que todos los métodos están implementados correctamente.

```
public class Node<E> {
      private E info;
      private Node<E> next;
      public Node(E info, Node<E> next) {
            this.info = info;
            this.next = next;
      }
      public Node(E info) {
            this(info, null);
      }
      public Node() {
            this(null, null);
      }
      public E getInfo(){…}
      public Node<E> getNext(){…}
      public void setInfo(E info){…}
      public void setNext(Node<E>next)
{…}
}
```

```
public class MyBasicLinkedList<E> {
      private Node<E> first;
      public MyBasicLinkedList() {
            this.first = null;
      }
      public Node<E> getFirst(){…}
      public void setFirst(Node<E>
first){…}
      public boolean isEmpty(){…}
      public void insert(E info){…}
      public int size(){…}
      public void print(){…}

      void moveToFront(){
           /*Programar*/
      }
}
```

Apartado 1:

Programe el método `void moveToFront()` de la clase `MyBasicLinkedList`, cuyo objetivo es mover el último elemento de la lista al principio de la misma.



Apartado 1 (1,5 puntos)

Apartado 2:

Implemente un método main que haga lo siguiente: (1) Crea una lista, (2) inserta los elementos de la lista en este orden: `["Google", "Facebook", "Instagram", "Whatsapp"]`. (3) Imprime el contenido de la lista, (4) invoca al método `moveToFront` creado en el apartado anterior y (5) vuelve a imprimir la lista. El resultado debería ser el siguiente:

```
Lista inicial: Whatsapp, Instagram, Facebook, Google
Lista final: Google, Whatsapp, Instagram, Facebook
```

Apartado 2 (1 puntos)

Problema 2 (Árboles)

Dada la interfaz `BTree<E>` y las clases `LBNode<E>` y `LBTree<E>`. Asumiendo que todos los métodos están implementados correctamente y que la clase `LBTree` tiene un constructor `LBTree(E info)`.

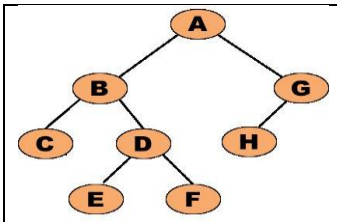
```
public interface BTree<E> {

   static final int LEFT = 0;
   static final int RIGHT = 1;

   public boolean isEmpty();
   public E getInfo();
   public BTree<E> getLeft();
   public BTree<E> getRight();
   public void insert(BTree<E> tree, int side);
   public BTree<E> extract(int side);
   public String toStringPreOrder();
   public String toStringInOrder();
   public String toStringPostOrder();
   public String toString();
   public int size();
   public int height();
   public boolean equals(BTree<E> tree);
   public boolean find(BTree<E> tree);
}
```

```
public class LBNode<E> {

   private E info;
   private BTree<E> left;
   private BTree<E> right;

   public LBNode(E info, BTree<E> left, BTree<E> right) {
       this.info = info;
       this.left = left;
       this.right = right;
   }
   public E getInfo() { return info;   }
   public void setInfo(E info) {this.info = info;}
   public BTree<E> getLeft() {return left;}
   public void setLeft(BTree<E> left) {this.left = left;}
   public BTree<E> getRight() {return right;}
   void setRight(BTree<E> right) {
       this.right = right;
   }
}
```

```
public class LBTree<E> implements BTree<E> { ... }
```

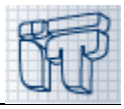
Apartado 1:

Implemente el método **recursivo** `int nodosEnNivel(BTree<Character> árbol, int nivel)` que, **dado un árbol y un nivel**, nos diga cuántos nodos hay en ese nivel del árbol.



Nota: El método obligatoriamente debe aceptar por parámetros un árbol y un nivel.

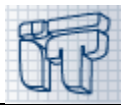
Ejemplo: Dado el siguiente árbol y el número de nivel 3, nos devuelve que hay 3 nodos (C,D y H)



```
Apartado 1 (1,25 puntos)
private static int nodosEnNivel(BTree<Character> árbol, int nivel){
```

Apartado 2:

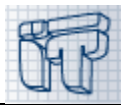
Implemente un método `main` que haga lo siguiente: (1) Crea un árbol con los nodos de los dos primeros niveles [`A, B, G`], (2) realiza los procesos de inserción necesarios para construir el subárbol formado por los tres nodos indicados. (3) Imprima el árbol en post-orden, (4) imprima el número de nodos del árbol en el nivel 2. Puedes asumir que el método `main` y el método `nodosEnNivel` implementado en el apartado anterior están dentro de la misma clase.



Apartado 2 (1,25 puntos)

Problema 3 (Algoritmos de ordenación y búsqueda)

Los Vengadores que quedan han conseguido reunir las seis piedras del infinito en un `ArrayList`. Sin embargo, deben estar ordenadas de una forma especial para que funcionen y puedan completar su misión. Las piedras deben ordenarse con un `InsertionSort` que compare su nombre y las ponga de mayor a menor (orden alfabéticamente inverso). El siguiente código muestra la creación del `ArrayList`, cómo se han insertado las piedras y en qué orden deberían aparecer.



```java
import java.util.ArrayList;

public class InsertionSort {
  public static void main(String[] args) {
    ArrayList<String> a = new
ArrayList<String>();
    a.add("Mente");
    a.add("Alma");
    a.add("Realidad");
    a.add("Tiempo");
    a.add("Espacio");
    a.add("Poder");

    insertionSort(a);

    System.out.println(a);
  }
}
```

Nota: Para implementar el algoritmo puede usar:

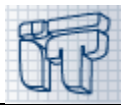
- el método `int compareTo(…)` de la clase `String`
- métodos `int Size()`, `E get(int i)`, `void set(int i, E info)` de la clase `ArrayList`.

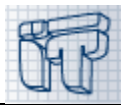
Ejemplo: Tras ejecutar el código el resultado debería ser:

`[Tiempo, Realidad, Poder, Mente, Espacio, Alma]`

Implementa el método `insertionSort` (que no debe devolver nada y debe ser estático) para que haga lo requerido.

Apartado 1 (2 puntos)





Criterios de corrección

Problema 1. Apartado 1 (1, 5 puntos)

- (0,5) Caso básico, lista con "0" o "1" elemento.
  - (0,25) Si solo se tiene en cuenta uno de los dos
- (0,25) Asignar correctamente las variables auxiliares y recorrer la lista hasta el último elemento.
- (0,75) Asignar correctamente las variables para la inserción del primer elemento en la lista.
  - (0,25) Último nuevo elemento a null.
  - (0,25) Último al primero para su inserción.
  - (0,25) Mover la variable first al primer elemento insertado

Problema 1. Apartado 2 (1 punto)

- (0,25) Creación de la lista.
- (0,25) Inserción de los elementos en la lista.
- (0,25) Llamar correctamente al método print().
- (0,25) Llamar correctamente al método moveToFront().
- (-0,1) si declaran mal el método main

Problema 2. Apartado 1 (1,25 puntos)

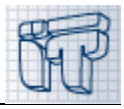
- (0,25) Caso base-1 correcto (árbol vacío)
  - 0 si tiene cualquier error.
- (0,25) Caso base-2 correcto (nivel 1)
  - 0 si tiene cualquier error.
- (0,75) Caso recursivo correcto
  - (-0,25) si solo hace llamada al hijo izquierdo o el derecho.
  - (-0,25) si no devuelve ningún resultado.
  - 0 si no se hace la llamada recursiva con nivel -1

Problema 2. Apartado 2 (1,25 puntos)

- (0,25) Creación de los árboles individuales (0 si presenta cualquier error)
- (0,25) Inserción en el orden correcto (0 si presenta cualquier error)
- (0,25) Llamada al método imprimir en post-orden. (0 si presenta cualquier error)
- (0,5) Llamar al método nodosEnNivel
- (-0,1) si declaran mal el método main

Problema 3. Apartado 1 (2 puntos)

- (0,2) Declaración correcta del método.
  - Penalizar si no ponen `void` y/o `static` (-0.1)
  - Penalizar si no ponen el argumento o si se equivocan en el tipo del `ArrayList` (-0.1)
- (0,4) Primer bucle for.
  - Penalizar si ponen `length` en lugar de `size()` (-0.1)
  - Si los límites no están bien definidos no asignar puntos.
- (0.2) Declaración de tmp y j=i
  - Penalizar si ponen `a[i]` siendo un `ArrayList` (-0.1)
- (0.5) Declaración del bucle while
  - Penalizar el mal uso o no uso del compareTo (-0.1)
  - Penalizar si ponen menor que 0 en lugar de mayor (en la comparación del compareTo) (-0.2)
- (0.5) Líneas dentro del bucle while
  - Penalizar el mal uso de set (-0.15)
  - Penalizar el mal uso de get (-0.15)
  - Penalizar el no decremento de j (-0.2)
- (0.2) Asignación de tmp
  - No puntuar si utilizan mal el set.



Solución:

Problema 1. Apartado 1 (1,5 puntos). Versión-1.

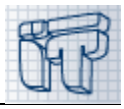
```
public void moveToFront(){
  Node<E> aux = first;
  if(first != null && first.getNext()!= null){
    while(aux.getNext().getNext()!= null){
      aux = aux.getNext();
    }
    Node<E> nodoAMover = aux.getNext();
    aux.setNext(null);
    nodoAMover.setNext(first);
    first=nodoAMover;
  }
}
```

Problema 1. Apartado 1 (1,5 puntos) Versión -2

```
public void moveToFront(){
       if(first == null || first.getNext() == null)
               return;

       Node<E> secLast = null;
       Node<E> last = first;

       while (last.getNext() != null)
       {
              secLast = last;
              last = last.getNext();
       }
       secLast.setNext(null);
       last.setNext(first);
      first = last;
}
```

Problema 1. Apartado 2 (1 punto)

```
public class MyBasicLinkedListTest {
  public static void main(String args[]) {
    <String> miLista = new MyBasicLinkedList<String>();
    // Añado diferentes nodos con información:
    miLista.insert(new String("Google"));
    miLista.insert(new String("Facebook"));
    miLista.insert(new String("Instagram"));
    miLista.insert(new String("Whatsapp"));
    miLista.print();
    miLista.moveToFront();
    miLista.print();
  }
}
```



Problema 2. Apartado 1 (1,25 puntos)

```
public static int nodosEnNivel(BTree<Character> arbol, int nivel){
  int resultado = 0;
  if(arbol.isEmpty()){
    resultado = 0;
  }else if (nivel == 1){
    resultado = 1;
  }else {
    resultado = nodosEnNivel(arbol.getLeft(), nivel-1) +
                nodosEnNivel(arbol.getRight(), nivel-1);
  }
  return resultado;
}
```

Problema 2. Apartado 2 (1,25 puntos)

```
public static void main(String[] args){
  BTree<Character> miArbolA = new LBTree<Character>('A');
  BTree<Character> miArbolB = new LBTree<Character>('B');
  BTree<Character> miArbolG = new LBTree<Character>('G');
  miArbolA.insert(miArbolB, BTree.LEFT);
  miArbolA.insert(miArbolG, BTree.RIGHT);
  System.out.println(miArbolA.toStringPostOrder());
  System.out.println("nivel-2:" + nodosEnNivel(miArbolA, 2));
}
```

Problema 3. Apartado 1 (2 puntos)

```
public static void insertionSort(ArrayList<String> a) {
      for(int i=0; i<a.size(); i++) {
            String tmp = a.get(i);
            int j=i;
            while(j>0 && tmp.compareTo(a.get(j-1))>0) {
                  a.set(j, a.get(j-1));
                  j--;
            }
            a.set(j, tmp);
      }
}
```