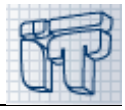


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Segundo parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 10 de mayo de 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.

Ejercicio 1 (2 / 7 puntos)

Se requiere implementar el método `public double calculateMean(double[] numbers)` que calcule de manera recursiva la media aritmética de los números que recibe como parámetro en el array `numbers`. Si el array `numbers` está vacío se debe devolver 0.

Ejercicio 2 (2,5 / 7 puntos)

Ejercicio 2. Apartado 1 (1,5 puntos)

Con la ayuda de las clases proporcionadas: `Node<E>` y `Stack<E>`, implemente la clase `LinkedStack<E>` completa (atributos, constructores, métodos, ...).

```
public class Node<E>{
  private E info;
  private Node<E> next;
  public Node(){
    info = null; next = null;}
  public Node(E info){
    setInfo(info);}
  public Node(E info, Node<E> next){
    setInfo(info); setNext(next);
  }
  public E getInfo(){ return info; }

  public void setInfo(E info){
        this.info = info;
  }
  public Node<E> getNext(){ return next;}

  public void setNext(Node<E> next){
        this.next = next;
  }
}
```

```
public interface Stack<E> {
  //Devuelve true si la pila está vacía;
  //false en caso contrario.
  boolean isEmpty();

  //Devuelve el tamaño o número de
  //elementos de la pila.
  int size();

  //Devuelve la información del primer
  //nodo de la pila. Si la pila está
  //vacía, devuelve null.
  E top();

 //Inserta un nuevo nodo en la pila.
  void push(E info);

  //Realiza la operación de extracción
  //de la pila y devuelve la información
  //del nodo extraído. Si la pila está
  //vacía, devuelve null.
        E pop();
}
```

Ejercicio 2. Apartado 2 (1 punto)

Utilizando la clase `LinkedStack<E>`, se quiere programar una clase con un método main que realice varias operaciones sobre una pila. Para ello, se definen tres instrucciones: "`write`", "`read`" y "`size`".

- Instrucción "`write`": almacena en la pila el dato de tipo String proporcionado después de la instrucción.
  Ejemplo: "`write hello`" almacena el String "hello" en la pila.
- Operación "`read`": lee el último String almacenado en la pila, lo muestra en pantalla y lo elimina de la pila.
- Instrucción "`size`": muestra en pantalla el tamaño actual de la pila.

El programa debe recibir estas instrucciones en tiempo de ejecución a través del teclado. El ejemplo de la derecha muestra una ejecución correcta del programa. En él se comprueba que el tamaño inicial de la pila es 0. Posteriormente se almacenan dos datos tipo String: "hello" y "bye" y se vuelve a comprobar el tamaño de la pila, que ahora es 2. Finalmente se ejecuta dos veces la instrucción de lectura y la pila vuelve a quedar vacía.

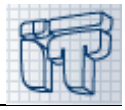
```
> size
0
> write
hello
> write
bye
> size
2
```

Complete el código proporcionado para alcanzar la funcionalidad explicada.

```
public class Program {
	public static void main(String[] args) throws IOException {

		//------------>Completar código<------------


		boolean end = false;
		BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

		while (!end) {
			System.out.print("> ");
			String command = input.readLine();
			if (command != null) {
				String[] parts = command.trim().split("\\s");

				//------------>Completar código<------------


			}
		}
	}
}
```

Ejercicio 3 (2,5 / 7 puntos)

Se requiere implementar la clase `FamilyTree` que modela un árbol genealógico donde el sub-árbol izquierdo almacena la madre de una persona y el sub-árbol derecho almacena el padre. La clase `FamilyTree` es un árbol binario cuyos nodos almacenan objetos de la clase `Person` como se puede observar a continuación.
Se proporciona la implementación de un árbol enlazado binario en la clase `LBTree<E>` que implementa la interfaz `BTree<E>` y de la clase `Person`.



```java
public class LBTree<E> implements
BTree<E> {

/… Métodos de BTree<E>
implementados …/

}

public interface BTree<E> {
   static final int LEFT = 0;
   static final int RIGHT = 1;

   boolean isEmpty();
   E getInfo() ;
   BTree<E> getLeft() ;
   BTree<E> getRight();
   void insert(BTree<E> tree,
          int side);
   BTree<E> extract(int side);
   int size();
   int height();
   boolean equals(BTree<E> tree);
   boolean find(BTree<E> tree);
} // BTree

public class Person {
   private String name;
   private String firstSurname;
   private String secondSurname;
   private String dni;
   private int age;

   public Person() {
   }

/… Métodos de acceso de los
atributos …/
} // Person
```

```java
public class FamilyTree extends LBTree<Person> {
   public FamilyTree(Person origin) {
       super(origin);
   }
   /**
    * Inserta la madre sobre la persona con DNI dniOfChild
    * @param dniOfChild DNI de la persona a la que insertar
    *                   la madre.
    * @param mother Madre a insertar sobre la persona con DNI
    *               dniOfChild
    */
   public void insertMother(String dniOfChild,
                            Person mother){
       // Completar apartado 1
   }
   private boolean insertMother( BTree<Person> tree,
                                 String dniOfChild,
                                 Person mother) {
       // Completar apartado 1
   }

   /**
    * Encuentra la mayor edad de todo el árbol
    * @return Devuelve la edad del ancestro con mayor edad o
    *          Integer.MIN_VALE si el árbol está vacío
    */
   public int ageOldestAncestor() {
       // Completar apartado 2
   }
   private int ageOldestAncestor( BTree<Person> tree) {
       // Completar apartado 2
   }
} // FamilyTree
```

Ejercicio 3. Apartado 1 (1 punto)

Se solicita la implementación del método `insertMother(String dniOfTheChild, Person mother)` que busca la persona con DNI `dniOfTheChild` y, si la encuentra, inserta como su madre el objeto `mother` que se pasa por parámetro.

Nota: se sugiere el uso de un método auxiliar como se indica en el esqueleto proporcionado pero la solución es libre.

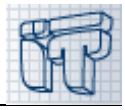
Ejercicio 3. Apartado 2 (1,5 puntos)

Se solicita la implementación del método `int ageOldestAncestor()` que devuelve la edad de la persona con mayor edad de todo el árbol genealógico. En caso de que el árbol esté vacío se debe devolver Integer.MIN_VALUE.

Nota: se sugiere el uso de un método auxiliar como se indica en el esqueleto proporcionado pero la solución es libre.

Ejercicio 1. Rúbrica.

Criterios de corrección



Apartado 1: 2 puntos

- Condición de salida: 0,6.
- Suma del array de números recursiva: 0,8.
- Cálculo de la media: 0,4
- Devolución 0 en caso de array vacío: 0,2

Ejercicio 1. Soluciones

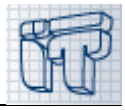
```
public class ScientificCalculator {

   public ScientificCalculator() {
   }

   public double calculateMean(double[] numbers) {
        double mean = 0;

        if (numbers.length > 0) {
            double sum = add(numbers, numbers.length - 1);
            mean = sum / numbers.length;
        }

        return mean;
   }

   private double add(double[] numbers, int position) {
        if (position < 0) {
            return 0;
        }

        return numbers[position] + add(numbers, --position);
   }

}
```

Ejercicio 2. Solución

Solución (2,5 puntos)

Apartado 1 (1,5 puntos)

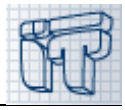
```
public class LinkedStack<E> implements Stack<E> {
```



```
	private Node<E> top;
	private int size;

	public LinkedStack(){
	}

	public boolean isEmpty(){
		return (top == null);
	}

	public int size(){
		return size;
	}

	public E top(){
		if(isEmpty()){
			return null;
		}
		return top.getInfo();
	}

	public void push(E info){
		Node<E> n = new Node<E>(info, top);
		top = n;
		size++;
	}

	public E pop(){
		E info;
		if(isEmpty()){
			info = null;
		}else{
			info = top.getInfo();
			top = top.getNext();
			size--;
		}
		return info;
	}
}
```

Apartado 2 (1 punto)

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Program{
	public static void main(String[] args) throws IOException{

		LinkedStack<String> stack = new LinkedStack<String>();
		boolean end = false;
		BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

		while(!end){
			System.out.print("> ");
```



```
                        String command = input.readLine();
                        if(command != null){
                                String[] parts = command.trim().split("\\s");
                                if(parts.length == 1 && parts[0].equals("read")){
                                        System.out.println(stack.pop());
                                }else if(parts.length == 1 && parts[0].equals("size")){
                                        System.out.println(stack.size());
                                }else if(parts.length == 2 && parts[0].equals("write")){
                                        stack.push(parts[1]);
                                }else{
                                        System.err.print("Command not valid. End.");
                                        end = true;
                                }
                        }
                }
        }
}
```

Ejercicio 2. Rúbrica

Apartado 1:

Total: 1,5 puntos.

- Implementa interfaz: 0,1 puntos.
- Definición de atributos: 0,2 puntos.
- Definición de constructor: 0,1 puntos.
- Método isEmpty(): 0,1 puntos
- Método size(): 0,1 puntos
- Método top(): 0,1 puntos
- Método push(E info): 0,4 puntos.
- Método pop(): 0,4 puntos.

Apartado 2:

Total: 1 punto.

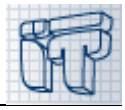
- Inicialz. de la pila: 0,15 puntos.
- Operación write: 0,35 puntos.
- Operación read: 0,25 puntos
- Operación size: 0,25 puntos

Ejercicio 3. Rúbrica.

Criterios de corrección

Apartado 1: 1 punto

- Condición de salida: 0,1.
- Comparación dni: 0,2.
  - Uso de = en lugar de equals: -0,1
  - Otro error: no usar getInfo u otro: -0,1
- Creación subárbol: 0,1
- Inserción del nuevo subárbol: 0,2

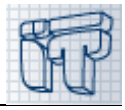


- Si inserta en subárbol derecho: -0,1
- Llamada recursiva: 0,4

Apartado 2: 1,5 puntos
- Condición de salida: 0,2.
- Devolver MIN_VALUE si árbol vacío: 0,1.
- Obtener mayor edad del nodo actual: 0,2.
- Llamada recursiva a ambos sub-árboles: 0,6.
- Determinación edad mayor de los valores: 0,4.

Ejercicio 3. Soluciones

```
public class FamilyTree extends LBTree<Person> {

   public FamilyTree(Person origin) {
       super(origin);
   }

   /**
    * Inserta la madre sobre la persona con DNI dniOfChild
    * @param dniOfChild DNI de la persona a la que insertar la madre
    * @param mother Madre a insertar sobre la persona con DNI
dniOfChild
    */
   public void insertMother(String dniOfChild, Person mother) {
       insertMother(this, dniOfChild, mother);
   }

   private boolean insertMother(BTree<Person> tree, String dniOfChild,
Person mother) {
       if (!tree.isEmpty()) {
           if (tree.getInfo().getDni().equals(dniOfChild)) {
               tree.insert(new LBTree<Person>(mother), BTree.LEFT);
               return true;
           }else {
               return insertMother(getLeft(), dniOfChild, mother)
                 || insertMother(getRight(), dniOfChild, mother);
           }
       }
   }

   /**
    * Find the age of the oldest ancestor
    * @return Devuelve la edad del ancestro con mayor edad o
Integer.MIN_VALE si el árbol está vacío
    */
   public int ageOldestAncestor() {
       int maxLeftAge = ageOldestAncestor(getLeft());
       int maxRightAge = ageOldestAncestor(getRight());
```



```
        if (maxLeftAge > maxRightAge) {
            return maxLeftAge;
        }
        return maxRightAge;
    }

    private int ageOldestAncestor(BTree<Person> tree) {
        int maxAge = Integer.MIN_VALUE;

        if (!tree.isEmpty()) {
            maxAge = tree.getInfo().getAge();
            int maxLeft = ageOldestAncestor(tree.getLeft());
            int maxRight = ageOldestAncestor(tree.getRight());
            if (maxLeft > maxAge) {
                maxAge = maxLeft;
            }
            if (maxRight > maxAge) {
                maxAge = maxRight;
            }
        }

        return maxAge;
    }
}
```