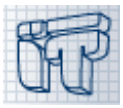


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

# Segundo parcial

## 2ª Parte: Problemas (7 puntos sobre 10)

Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 13 de mayo de 2021

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.

## Ejercicio 1 (2 / 7 puntos)

Codificar el método cuya firma es `public ArrayList<Integer> minMaxStack (LinkedStack<Integer> stack)` el cual recibe una pila enlazada de objetos `Integer` y devuelve en un objeto `ArrayList` el mayor y el menor de esa pila. En la primera posición del `ArrayList` debe devolver el menor, y en la segunda el mayor. Si la pila no contuviera elementos o fuera `null` se debe devolver `null`. La pila implementa la siguiente interfaz:

```
public interface Stack<E> {
    boolean isEmpty();
    int size();
    void push (E info);
    E pop();
    E top();
}
```

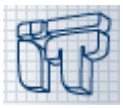
Ejemplo: Si la pila contiene los elementos {3, 5, 1, 5, 3, 2, 4}, el `ArrayList` a devolver debe contener los elementos {1, 5}.

**Nota:** No es necesario reconstruir el objeto `stack` para que tras el proceso quede con los elementos originales.

## Ejercicio 2 (3 / 7 Puntos)

Una empresa líder en el sector de la logística le ha pedido que desarrolle un programa para la gestión de un almacén. Se manejan, entre otras, estas clases (suponga que todas tienen implementadas los métodos `get` y `set` para los atributos):

- Persona (`Person`). Permite identificar al cliente del almacén. Para identificar a una persona únicamente necesitaremos su número de identificación (`id`), su nombre (`firstName`),



apellido (`lastName`) y el email de contacto (`email`). También se quiere premiar a las personas que hagan muchos pedidos mediante un sistema de puntos para proporcionar descuentos. Se necesita por tanto un atributo que indique si la persona es VIP (`vip`) y los puntos (`points`) y el descuento acumulado (`discount`). Por cada pedido que haga se le darán 100 puntos. Si llega a 1000 se convertirá en VIP, y a partir de ese momento por cada nuevo pedido tendrá un 10% de descuento hasta un máximo del 40%.

- Pedido (`Order`). Representa el documento que contiene cada uno de los pedidos que se realizan en el almacén. Este objeto entre otros atributos (no relevantes para este ejercicio) tiene el cliente que ha realizado el pedido (`client`).

- Gestor del almacén (`StoreManager`). Es el cerebro de la aplicación y contendrá toda la lógica del programa. Tendrá:

    - Una cola de pedidos (`LinkedQueue<E>`) que almacene los pedidos (`Order`) pendientes de procesar (`ordersToProcess`).
    - Un árbol binario de búsqueda (`LBSTree<E>`) de personas (`Person`) que permita almacenar los clientes del almacén (`storeCustomers`). La clave de búsqueda será el `ID` de las personas (`Person`).

**Apartado 1 (0,5 puntos)**

Tenemos la clase `Person` declarada así:

```
public class Person {
	private int id;
	private String firstName;
	private String lastName;
	private String email;
	private boolean VIP=false;
	private int points=0;
	private int discount=0;

	// getters y setters de los atributos ya implementados
}
```

Codifique en la clase `Person` el método `public void updateClientNewOrder()`. Dicho método actualizará el objeto `Person` cuando el cliente haga un nuevo pedido, es decir, deberá actualizar el estado si el cliente pasa a ser VIP y los puntos y porcentaje de descuento si procede.

**Apartado 2 (1,25 puntos)**

Codifique el método `public void updateClient(Comparable key)` en la clase `LBSTree` que actualice el cliente en el árbol de pedidos cuando se haga un nuevo pedido. Debe recorrer el árbol y cuando encuentre el cliente cuya clave sea el parámetro `key` actualizar su información relativa al estado de puntos y porcentaje de descuento. Tenemos las siguientes declaraciones:

```
public interface BSTree<E>
{
   public boolean isEmpty();
   public E getInfo();
   public Comparable getKey();
   public BSTree<E> getLeft();
   public BSTree<E> getRight();
   public void insert (Comparable key, E
info);
}
```

```
public class LBSTree<E> implements
BSTree<E> {

private LBSNode<E> root;

public LBSTree(Comparable key, E info) {
   this.root = new LBSNode<E>(key, info,
new LBSTree<E>(), new LBSTree<E>());
}

// Implementación del resto de métodos
//…
}
```

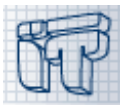
```
public class LBSNode<E> {
   private E info;
   private Comparable key;
   private BSTree<E> right;
   private BSTree<E> left;

   public LBSNode(Comparable key, E info, BSTree<E> left, BSTree<E> right) {
      this.info = info;
      this.key = key;

     this.right = right;
     this.left = left;
  }
// Implementación del resto de métodos
// …
}
```

**Apartado 3 (1,25 puntos)**

Codifique el método `public void updateClients()` en la clase `StoreManager`. Dicho método deberá recorrer la cola de pedidos a procesar (`ordersToProcess`) y para cada cliente de los pedidos debe actualizar en el árbol `storeCustomers` su información relativa al estado de puntos y porcentaje de descuento. Los clientes están ordenados por el árbol en función de su `ID`. La cola original debe quedar inalterada al finalizar el método. La cola implementa la siguiente interfaz:

```
public interface Queue<E> {
    boolean isEmpty();
    int size();
    void enqueue (E info);
    E dequeue();
    E front();
}
```

**Nota:** Para cada apartado si lo necesita puede usar la llamada a los métodos que se han solicitado en apartados anteriores, aunque no los haya resuelto.



## Ejercicio 3 (2 / 7 Puntos)

Dada la clase del siguiente código:

```
public class Point {

	private double x;
	private double y;

	public Point(double x, double y) {
		this.x = x;
		this.y = y;
	}

	public String toString() {
		return "(" + x + ", " + y + ")";
	}

	public double getX() {
		return x;
	}

	public double getY() {
		return y;
	}
}
```

Implemente el método `public static void selectionSortPoint (Point[] array)` (que no debe devolver nada y debe ser estático) para que ordene el array usando el método de ordenación por selección en orden **ascendente, teniendo en cuenta solamente la primera coordenada del punto.**

Ejemplo:

```
Elements before sorting:

(4.0, 5.0)
(3.0, 7.0)
(5.0, 4.0)
(6.0, 8.0)
(7.0, 1.0)

Elements after sorting:

(3.0, 7.0)
(4.0, 5.0)
(5.0, 4.0)
(6.0, 8.0)
(7.0, 1.0)
```

# SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

## Ejercicio 1 (2 / 7 puntos)

```
public ArrayList<Integer> minMaxStack (LinkedStack<Integer> stack){

		if (stack == null) return null;
		if (stack.isEmpty()) return null;

		ArrayList<Integer> result = new ArrayList<Integer>();

		int min= stack.pop();
		int max=min;
		int size = stack.size();

		for(int i=1; i<size;i++){

			Integer Elem = stack.pop();
			if (Elem < min) min=Elem;
			if (Elem > max) max=Elem;
		}

		result.add(min);
		result.add(max);

		return result;
	}
```

Rúbrica:

- 0 si no tiene sentido.
- 0,25 por tratar los casos de pila vacía o nula.
- 0,25 por inicializar el arrayList.
- 0,25 por recorrer correctamente la pila.
- 0,5 por desapilar correctamente.
- 0,25 por comparar correctamente el mayor y el menor.
- 0,25 por insertar correctamente el menor y el mayor en el arrayList.
- 0,25 por devolver el ArrayList.
- Los errores significativos están sometidos a penalizaciones.

# Ejercicio 2 (3 puntos)

## Apartado 1 (0,5 puntos)

```
public void updateClientNewOrder(){
		points=points+100;

		VIP =(points >1000);

		if (VIP && discount < 30)
			discount = discount+10;

	}
```

Rúbrica:

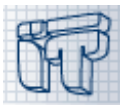
- 0 si no tiene sentido.
- 0,15 por incrementar los puntos.
- 0,15 por actualizar el atributo VIP.
- 0,2 por actualizar el atributo discount con la condición correcta.
- Los errores significativos están sometidos a penalizaciones.

## Apartado 2 (1,25 puntos)

```
public void updateClient(Comparable key) {

		if (!this.isEmpty()) {
			if (this.root.getKey().compareTo(key) > 0) {
				// lower key
				this.getLeft().updateClient(key);
			} else if (this.root.getKey().compareTo(key) < 0) {
				// greater key
				this.getRight().updateClient(key);
			} else {
				// equal keys
				Person p = (Person) this.root.getInfo();
				p.updateClientNewOrder();
			}
		} // if reaching an empty subtree -> key not found
	}
```

Rúbrica:

- 0 si no tiene sentido.
- 0,25 por comprobar si el árbol está vacío.



- 0,4 por cada comparación correcta y llamada recursiva (izquierda y derecha). Si no usa compareTo máximo 0,1 en cada llamada.
- 0,2 en el caso de encontrarlo por el casting y la llamada al método updateClientNewOrder().
- Los errores significativos están sometidos a penalizaciones.

### Apartado 3 (1,25 puntos)

```
public void updateClients(){

		for(int i=0;i<ordersToProcess.size();i++){
			Order o = (Order) ordersToProcess.dequeue();
			Person p = (Person) o.getClient();

			storeCustomers.updateClient(p.getId());
			ordersToProcess.enqueue(o);
		}
	}
```

Rúbrica:

- 0 si no tiene sentido.
- 0,25 por recorrer correctamente la cola.
- 0,25 por desencolar y el casting correcto del pedido.
- 0,25 por el casting a la clase Person.
- 0,25 por la llamada correcta a updateClient.
- 0,25 por volver a encolar el elemento.
- Los errores significativos están sometidos a penalizaciones.

## Ejercicio 3. Solución (2 / 7 puntos)

```
public static void selectionSortPoint ( Point[] array )
        {

          for ( int j=0; j < array.length-1; j++ )
          {

            int min = j;
            for ( int k=j+1; k < array.length; k++ )
              if ( array[k].getX() < array[min].getX() ) min = k;

            Point temp = array[j];
            array[j] = array[min];
            array[min] = temp;
          }

        }
```

Rúbrica:

- 0 si no tiene sentido.
- 0,2: Declaración correcta del método.
  - Penalizar 0,1 si no ponen void y/o static o sino ponen el argumento o si se equivocan en el tipo del array
- 0,4: Primer bucle for
  - Penalizar 0,1 si ponen size() en lugar de `length`
  - Si los límites no están bien definidos, entonces 0
- 0,4: Segundo bucle for
  - Penalizar 0,1 si ponen size() en lugar de `length`
  - Si los límites no están bien definidos, entonces 0
- 0,6. Condicional if
  - Penalizar 0,3 si ponen en el if la ordenación descendente
  - Penalizar 0,2 si ponen mal los índices
- 0,4: Líneas dentro del if
  - Penalizar 0.1 por la no declaración de la variable de tipo Point
- Los errores significativos están sometidos a penalizaciones.