NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Segundo parcial

2ª Parte: Problemas (7 puntos sobre 10)

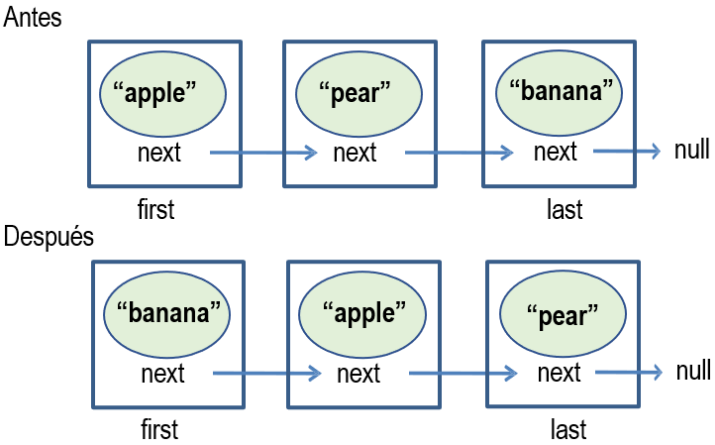
Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 14 de Mayo de 2021

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)

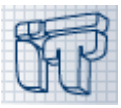
Dada la lista enlazada que aparece en la figura programa un método `moveLastToFirst` que permita extraer el último nodo de una lista enlazada cualquiera y colocarlo en la primera posición.



En la figura puedes ver cómo quedaría la lista antes y después de llamar a dicho método.
NOTA: Recuerda revisar los casos conflictivos como por ejemplo cuando la lista está vacía o tiene un único elemento.

Ejercicio 2 (3 / 7 puntos)

La aplicación que gestiona el almacén que has creado durante el proyecto se ha ampliado creando una nueva clase para representar los pedidos ya facturados (`InvoicedOrder`). Esta clase además de los atributos de la clase Order contiene un atributo adicional con el importe de la factura (invoice). Dadas las clases `Order` e `InvoicedOrder` Se pide:



```java
public class Order {
    private int orderID;
    private Person customer;
    private Person employee;
    // other attributes
    // constructors
    // setters and getters
    // other methods
}
```

```java
public class InvoicedOrder extends Order
                           implements Comparable{
    private double invoice;
    //getters and setters

    public int compareTo(Object other) {//todo }

}
```

Apartado 1. Método compareTo de la clase InvoicedOrder

Programa el método `compareTo` de la clase `InvoicedOrder` que permite comparar dos pedidos facturados en función del importe de su factura (`invoice`).

NOTA: Recuerda que para que un objeto de tipo Object pueda utilizar métodos de una clase que hereda de ella es necesario hacer un casting. Puedes asumir que todos los métodos que se mencionan en la figura están implementados salvo el que se pide en este apartado

Apartado 2. Método selectionSort de la clase StoreManager

```java
public class StoreManager {
    ArrayList<InvoicedOrder> list;
    //other attributes
    //constructors
    //setters and getters

    public  void swap(ArrayList<InvoicedOrder> list, int i, int j) {
        InvoicedOrder aux = list.get(i); // is equivalent to aux = a[i]; when using arrays
        list.set(i, list.get(j)); // is equivalent to a[i]=a[j]; when using arrays
        list.set(j, aux); // is equivalent to a[j]=aux; when using arrays
    }

    public  void selectionSort() { //todo}
}
```

Dada la clase StoreManager, Implementa el método selectionSort() que permita ordenar los pedidos de mayor a menor en función del importe de su factura (`invoice`) .

NOTA: Puedes asumir que existe el método `swap` (`ArrayList<InvoicedOrder> int i, int j`) que permite intercambiar los elementos en las posiciones i y j dentro del ArrayList de pedidos facturados. Puedes ver el código del método swap implementado en la clase `StoreManager`

Ejercicio 3 (2 / 7 Puntos)

Cuando tanto la clave como la información que almacenamos en un árbol son objetos comparables podemos escoger si ordenamos el árbol con respecto a la clave o con respecto a la información que contiene. Dada la interfaz `BSTree` y las clases `LBSTree` y `LBSNode` se pide implementar el método recursivo `public BSTree<Comparable> exchangeInfoAndKey(BSTree<Comparable> otherTree)` de la clase `LBSTree` que

crea y devuelve un nuevo árbol con el mismo contenido que el árbol original pero ordenado atendiendo al contenido de info en vez de utilizar el contenido de key.
Nota: puedes asumir que tanto Key como como Info almacenan objetos de tipo Comparable para que puedas usar uno u otro como clave.

Por ejemplo, dado un árbol que almacena los diferentes cargos de una empresa utilizando el nombre del cargo y el despacho que ocupa. La figura 1 representaría el árbol ordenado atendiendo al nombre del cargo (usa como clave el nombre). Si llamamos al método exchangeInfoAndKey() sobre el árbol de la figura-1 devolvería como resultado el árbol de la figura 2 que muestra la misma información ordenada según el número de despacho (es decir usando como clave el número de despacho).
NOTA: Puedes suponer que todos los métodos de las clases LBSTree y LBSNode están implementados (excepto el que se pide en este apartado).

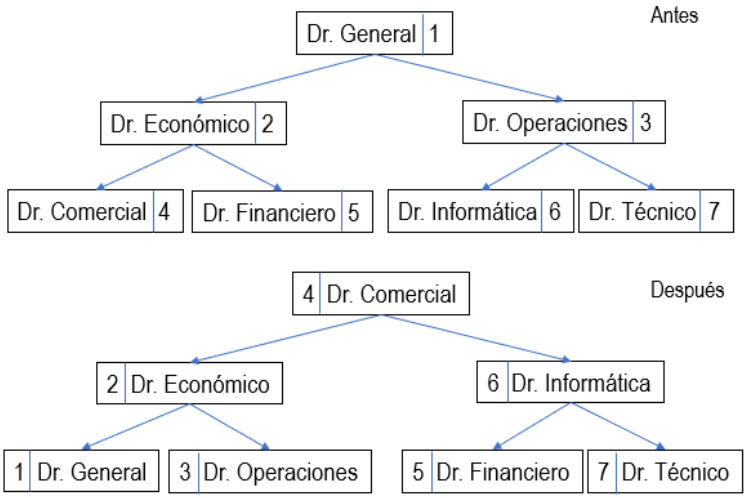
```
public interface BSTree<E> {
      boolean isEmpty();
      E getInfo();
      Comparable getKey();
      BSTree<E> getLeft();
      BSTree<E> getRight();
      String toStringPreOrder();
      String toStringInOrder();
      String toStringPostOrder();
      String toString(); // pre-order
      void insert(Comparable key, E info);
      BSTree<E> search(Comparable key);
      void exchangeInfoAndKey(LBSTree<Comparable> otherTree);
    }
```

```
public class LBSNode<E>{
      private E info;
      private Comparable key;
      private BSTree<E> right;
      private BSTree<E> left;
      //constructors
      //getters and setters
}
```

```
public class LBSTree<E> implements BSTree<E> {
    private LBSNode<E> root;
    //constructors getters and setters

    public BSTree<Comparable> exchangeInfoAndKey() {
        return exchangeInfoAndKey(new LBSTree<Comparable>());
    }

    public BSTree<Comparable> exchangeInfoAndKey(BSTree<Comparable> otherTree) {//todo }
```


Antes
Dr. General 1
Dr. Económico 2
Dr. Operaciones 3
Dr. Comercial 4
Dr. Financiero 5
Dr. Informática 6
Dr. Técnico 7
Después
4 Dr. Comercial
2 Dr. Económico
6 Dr. Informática
1 Dr. General
3 Dr. Operaciones
5 Dr. Financiero
7 Dr. Técnico

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

Ejercicio 1. Solución (2 / 7 puntos)

Rúbrica

- 0 si no tiene sentido
- 0,75 recorrer el nodo hasta el penúltimo
  - 0,25 inicializar el recorrido
  - 0,25 actualización
  - 0,25 condición correcta de parada y current !=null (considerar correcto si la condición de current!=null está fuera)
- 1,25 actualización correcta de los enlaces. 0 si no lo hacen en el orden correcto y la lista queda inconsistente.
  - 0,25 condición last !=null y current !=null (considerar correcto si la condición de current!=null está fuera)
  - 0,25 enlazar last con el primero
  - 0,25 actualizar el valor de first
  - 0,25 actualizar el valor de last
  - 0,25 enlazar úlimo nodo a null

Solución

```
public void moveLastToFirst() {
    Node<E> current = first;
    while (current != null && current.getNext() != last) {
        current = current.getNext();
    }
    if (last != null && current!=null) {
        last.setNext(first);
        first = last;
        last = current;
        current.setNext(null);
    }
}
```

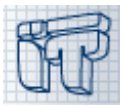
Ejercicio 2. Solución (3 / 7 puntos)

Apartado 2.1. Método compareTo de la clase InvoicedOrder (1,25)

Rúbrica

- 0,25 declarar y devolver el valor de retorno
- 0,25 sentencia que incluye casting de Object a InvoicedOrder
- 0,25 caso 0
- 0,25 caso 1
- 0,25 caso-1

Solución



```
public int compareTo(Object other) {//todo
    int result;
    //convert other to InvoicedOrder
    InvoicedOrder otherInvoicedOrder = (InvoicedOrder)other;
    if(this.getInvoice()== otherInvoicedOrder.getInvoice()) {
        result = 0;
    }else if(this.getInvoice() < otherInvoicedOrder.getInvoice()) {
        result = -1;
    }else {
        result = 1;
    }
    return result;
}
```

Apartado 2.2 Método selectionSort (1,75)

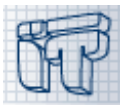
Rúbrica

- 0 si no tiene sentido
  - 0,1 en total si han memorizado el método de las transparencias pero no lo adaptan al ejercicio.
- 0,25 Bucle externo (inicialización, condición y actualización)
- 0,25 inicialización de m
- 0,25 Bucle interno (inicialización, condición y actualización)
- 0,5 condición del if
  - 0,25 Llamada correcta a compareTo con los índices y símobolo < en el orden adecuado
  - 0,25 Llamada correcta a los elementos de la lista con get
- 0,25 actualización de m
- 0,25 llamada a swap . 0 si hay algún fallo.

Solución

```
public void selectionSort() {

    for (int i = 0; i < list.size(); i++) {
        int m = i;
        for (int j = i; j < list.size(); j++) {
            if (list.get(j).compareTo(list.get(m)) < 0) {
                m = j;
            }
        }
        swap(list, i, m);
    }
}
```



Ejercicio 3. Solución (2 / 7 Puntos)

Rúbrica

- 0 si no tiene sentido
- 0,25 declarar y devolver el tipo de retorno. 0,1 si lo declaran con LBSTree en vez de como BBSTree o si no ponen comparable.
- 0,25 manejo de ClassCastException 0 si la lanzan porque se da la signatura en el enunciado.
- 0,25 sentencia para hacer casting de info a Comparable.
- 0,5 caso base (condición de parada + caso base)
- 0,25 condiciones de los casos recursivos (left and right)
- 0,5 casos recursivos. (left and right)

Solución

```
public BSTree<Comparable> exchangeInfoAndKey(BSTree<Comparable> otherTree) {// todo }

    BSTree<Comparable> result = otherTree;
    try {
        Comparable info = (Comparable) getInfo();
        if (root != null) {
            otherTree.insert(info, getKey());
            if (getLeft() != null) {
                result = getLeft().exchangeInfoAndKey(otherTree);
            }
            if (getRight() != null) {
                result = getRight().exchangeInfoAndKey(otherTree);
            }
        }

    } catch (ClassCastException cce) {
        System.out.println("You cannot use info as a key because info is not comparable");
    }
    return result;
}
```