NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Segundo parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 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.

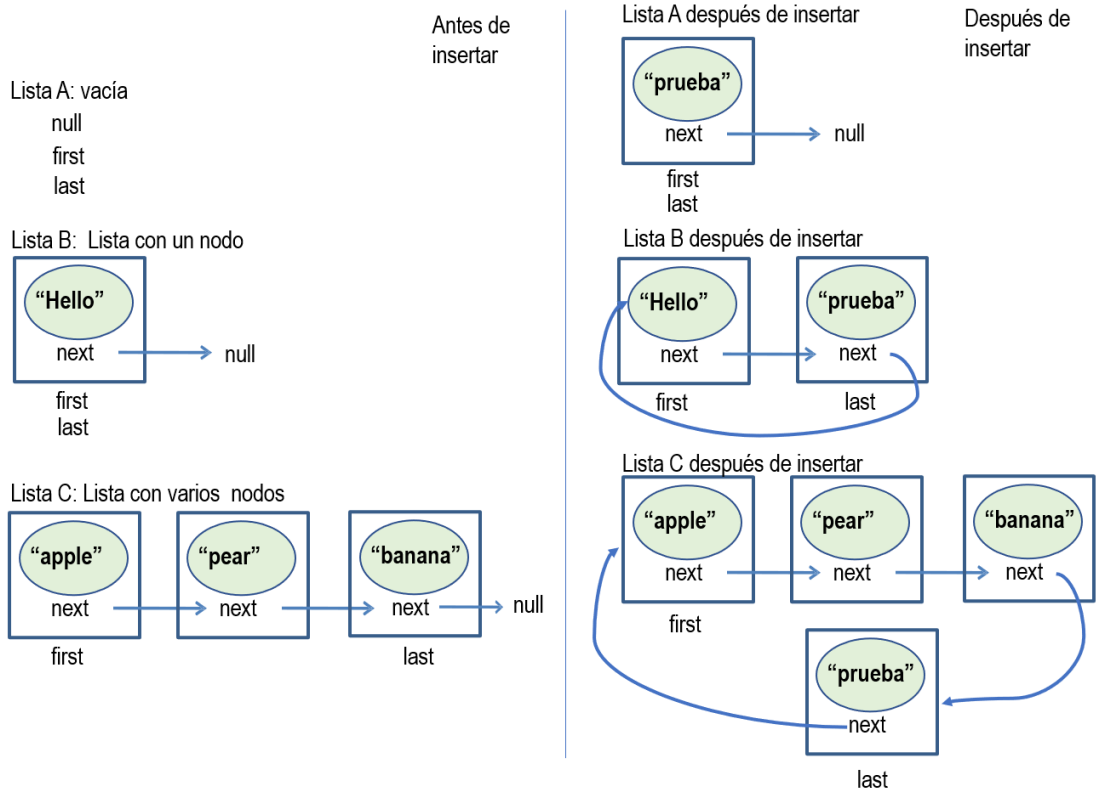
Ejercicio 1 (2 / 7 puntos)

Dadas las clases `LinkedList` y `Node` que aparecen en la figura se pide.

```java
public class LinkedList<E> {
    private Node<E> first;
    private Node<E> last;
    //constructors setters and getters
}
```

```java
public class Node<E> {
    private E info;
    private Node<E> next;
    //constructors setters and getters
}
```

Apartado 1. Método insertAndClose



Programa el método `insertAndClose(E info)` en la clase `LinkedList` que inserta un nuevo nodo en la lista conectándolo con el primer Nodo (`first`) y con el último (`last`) tal y como aparece en la figura para conseguir una lista circular. Una vez conectado el nodo nuevo este pasa a ser el último. (NOTA: considera los casos especiales en los que la lista esté vacía o sólo tenga un nodo. Puedes ver cómo queda la lista antes y después de llamar al método `insertAndClose(E info)` en la figura).

Apartado 2. Método print

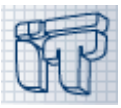
Programa el método print en la clase `LinkedList` que imprime los elementos de la lista separados por un espacio comenzando por el nodo que aparece como `first` en la figura. El resultado de ejecutar este método sobre la lista circular de 4 elementos que aparece en la figura sería:

```
apple pear banana prueba
```

Ejercicio 2. (3 / 7 puntos)

Dada un versión reducida de la clase `Provider` que programaste en el proyecto de gestión del Almacén y una clase `MainApp` que permite imprimir una lista ordenada de los proveedores se pide:

```
public class Provider implements Comparable {
    private String name;
    private int vat;
    //setters, getters and constructor
    public String toString() {
        return ("(" + name + ", " + vat + ")");
    }
    public int compareTo(Object other) {
      //todo
    }
}
```



```
public class MainApp {
    public static void main(String[] args) {
        Provider appleProvider = new Provider("Apple", 1111);
        Provider hpProvider = new Provider("HP", 2222);
        Provider dellProvider = new Provider ("Dell", 3333);
        Provider acerProvider1 = new Provider ("Acer", 4444);
        Provider acerProvider2 = new Provider("Acer", 5555);
        ArrayList<Provider> myProviders = new ArrayList<Provider>();
        myProviders.add(appleProvider);
        myProviders.add(hpProvider);
        myProviders.add(dellProvider);
        myProviders.add(acerProvider1);
        myProviders.add(acerProvider2);
        print(myProviders);
        selectionSort(myProviders);
        print(myProviders);
    }

    public static void swap(ArrayList<Provider> myProviders, int i, int j) {
        Provider tmp = myProviders.get(i);
        myProviders.set(i, myProviders.get(j));
        myProviders.set(j,  tmp);
    }

    public static void selectionSort(ArrayList<Provider> myProviders) {//todo}
```

Apartado 2.1. Método compareTo de la clase Provider.
Programa el método `compareTo` de la clase `Provider` que permite comparar dos proveedores alfabéticamente en función de su nombre (`name`) y a igualdad de nombre por su número de identificación fiscal (`vat`).

NOTA-1: Recuerda que la clase `String` también implementa la interfaz `Comparable` y por tanto, puede usar el método `compareTo`.
NOTA-2: Recuerda que para que un objeto de tipo `Object` pueda utilizar métodos de una clase que hereda de ella es necesario hacer un casting.

Apartado 2.2. Método selectionSort de la clase MainApp
Implementa el método `selectionSort` de la clase `MainApp` que permite ordenar a los Proveedores (`Provider`) de menor a mayor en función de su nombre (`name`) y su número de identificación fiscal (`vat`). (NOTA: para este apartado puedes asumir que los métodos `compareTo` de la clase `Provider` y `swap` de la clase `MainApp` ya existen y funcionan correctamente). En la siguiente figura se muestra el resultado de ejecutar el programa sobre el ArrayList de proveedores que aparece en la figura.

```
(Apple, 1111) (HP, 2222) (Dell, 3333) (Acer, 4444) (Acer, 5555)

(Acer, 4444) (Acer, 5555) (Apple, 1111) (Dell, 3333) (HP, 2222)
```

Ejercicio 3 (2 / 7 Puntos)

```java
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;
      }
      //setters, getters, toString
}
```

```java
public class LBSTree<E> implements BSTree<E> {
    private LBSNode<E> root;

    public LBSTree(Comparable key, E info) {
        root = new LBSNode<E>(key, info,
                              new LBSTree<E>(),
                              new LBSTree<E>());
    }
    //setters getters
}
```

Dado un árbol de búsqueda binaria `LBSTree` y la clase `LBSNode` que aparecen en la figura se pide:

Apartado 1. Método containsToStringInOrder de la clase LBSTree

Implementa el método `String containsToStringInOrder(String subString)`. Que recibe como parámetro un determinado prefijo (`subString`) y devuelve un String con el resultado de imprimir la información de todos los nodos cuya representación textual empiece por dicho prefijo ordenados alfabéticamente. Este método serviría por ejemplo para encontrar todas las palabras con una misma raíz en un diccionario.

NOTA-1: puedes asumir que todos los objetos de tipo E info tienen un método toString() para devolver su representación textual

NOTA-2: Puedes utilizar el método `boolean startsWith(String prefix)` de la clase `String` que devuelve `true` si el `String` comienza por el prefijo que recibe como parámetro y `false` en caso contrario).

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

Ejercicio 1. Solución (2 / 7 puntos)

Apartado 1. Método insertAndClose de la clase LinkedList (1 punto)

- 0 si no tiene sentido
- 0,2 creación del nodo
- 0,2 enlaza correctamente el nuevo al primero de la lista
- 0,2 enlaza el último al nuevo
- 0,2 controla el caso especial de last!=null
- 0,1 actualiza correctamente el valor de nuevo
- 0,1 actualiza correctamente el valor de last

```
public void insertAndClose(E info) {
    Node<E> nuevo = new Node<E>(info);
    nuevo.setNext(first);
    if(last!=null) {
        last.setNext(nuevo);
    }else {
        first = nuevo;
    }
    last = nuevo;
}
```

Apartado 2. Método print de la clase LinkedList (1 punto)

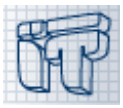
- 0 si no tiene sentido
- 0,4 Inicializar curent y recorrer la lista con current = current.getNext()
- 0,2 while y condiciones para salir del bucle al ser lista circular (current!=last)
- 0,2 Imprimir correctamente el contenido (0 si no llama a getInfo())
- 0,2 Tener en cuenta casos especiales (current!=null y last!=null)

```
public void print() {
    Node<E> current = first;
    while(current!=null && current!=last) {
        System.out.print(current.getInfo() + " ");
        current = current.getNext();
    }
    if(last!=null) {
        System.out.println(last);
    }
}
```

Ejercicio 2. Solución (3 / 7 puntos)

Apartado 2.1. Método compareTo de la clase Provider (1,5 puntos)

- 0 si no tiene sentido
- 0,25 devolver el valor de retorno
- 0,25 Hacer el casting de Object a Provider
- 0,25 llamada correcta a compareTo para el nombre
- 0,25 llamada correcta a < > para el vat
- 0,25 ramas name
- 0,25 ramas vat
    - 0,25 en todo el ejercicio si todo es correcto excepto que hace todo con < y > o todo con compareTo. 0 si tiene algún fallo adicional.



```
public int compareTo(Object other) {
    int result = -Integer.MIN_VALUE;
    Provider p = (Provider) other;
    if (this.name.compareTo(p.getName()) > 0) {
        result = 1;
    } else if (this.name.compareTo(p.getName()) < 0) {
        result = -1;
    } else {
        if (this.vat > p.getVat()) {
            result = 1;
        } else if (this.vat < p.getVat()) {
            result = -1;
        } else {
            result = 0;
        }
    }
    return result;
}
```

Apartado 2.2. Método selectionSort de la clase MainApp (1,5)

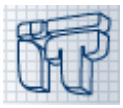
- 0 si no tiene sentido
    - 0,1 en total si han memorizado el método de las transparencias pero no lo adaptan al ejercicio.
- 0,25 Bucle externo (inicialización, condición y actualización)
- 0,2 inicialización de m
- 0,25 Bucle interno (inicialización, condición y actualización)
- 0,35 condición del if
    - 0,25 llamada correcta a compareTo con los índices y símbolo < en el orden adecuado
    - 0,1 llamada correcta a los elementos de la lista con get
- 0,2 actualización de m
- 0,25 llamada a swap . 0 si hay algún fallo.

```
public static void selectionSort(ArrayList<Provider> myProviders) {
    for (int i=0; i<myProviders.size(); i++) {
        int m = i;
        for (int j=i; j<myProviders.size(); j++) {
            if (myProviders.get(j).compareTo(myProviders.get(m))<0){
                m = j;
            }
        }
        swap(myProviders, i, m);
    }
}
```

Ejercicio 3. Solución (2 / 7 Puntos)

Apartado 1. Método containsToStringInOrder de la clase LBSTree (2 puntos)

- 0,25 inicialización y devolución de resultado
- 0,25 condición general getInfo()!=null
- 0,75 caso base
    - 0,5 condición
    - 0,25 devolución del resultado
- 0,75 casos recursivos
    - 0,25 condición de los casos recursivos
    - 0,5 devolución del resultado



```
public String containsToStringInOrder(String subString) {
    String resultado = "";
    if (getInfo() != null) {
        if (getLeft() != null) {
            resultado = resultado + getLeft().toStringInOrder();
        }
        if ((getInfo().toString()).startsWith(subString)) {
            resultado = resultado + getInfo().toString();
        }
        if (getRight() != null) {
            resultado = resultado + getRight().toStringInOrder();
        }
    }
    return resultado;
}
```