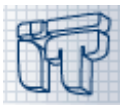


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

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)

Dadas las clases `LinkedList<E>` y `Node<E>` se pide:

```java
public class LinkedList<E> {
    private Node<E> first;
    private Node<E> last;

    public LinkedList() {
        first = null;
    }

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

    public Comparable max() { //todo }

}
```

```java
public class Node<E> {
    private E info;
    private Node<E> next;

    public Node(E info, Node<E> next) {
        this.info = info;
        this.next = next;
    }
    //setters and getters
}
```

Apartado 1 (2 puntos) método max

Implementa el método `public Comparable max()` de la clase `LinkedList` que devuelve como resultado la información del elemento de la lista con valor máximo.

- Cada vez que uses el valor de info tendrás que convertirlo a `Comparable` mediante un casting. Si no es posible captura la excepción `ClassCastException` para informar al usuario de que no se puede obtener el máximo si los objetos de la lista no son Comparables.
- Una vez que hayas realizado el primer paso, ya puedes comparar los elementos de la lista con el método `compareTo`

Ejemplo: si tenemos una lista con los `Strings` "`Alba`", "`Carlos`", "`Bea`", "`Vicente`" el máximo sería "`Vicente`" puesto que es el último por orden alfabético.

Ejercicio 2 (1 / 7 puntos)

Una empresa líder en el sector de la logística le ha pedido que desarrolle un programa para la gestión de un almacén. El almacén (`Store`) contiene un ArrayList llamado `currentOrders` con todos los pedidos realizados hasta la fecha y una estructura de tipo `BTree<Employee> employees` para almacenar todos los empleados (`Employee`) del almacén (`Store`).

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

```java
public class Employee {
    private int employeeID;
    // other attributes
    // constructors
    // getters and setters
}
```

```java
public class Store {
    ArrayList<Order> list;
    BTree<Employee> employees;
    // other attributes
    // constructors
    // getters and setters
}
```

```java
public interface BTree<E> {

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

    boolean isEmpty();

    E getInfo() throws BTreeException ;
    BTree<E> getLeft() throws BTreeException;
    BTree<E> getRight() throws BTreeException;

    //other methods
}
```

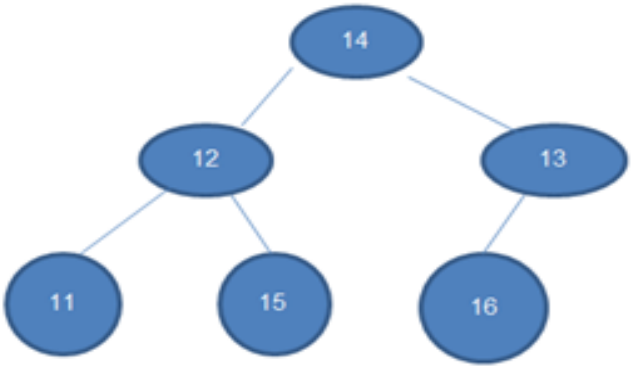
Apartado 2.1 (1 punto) método findOrder

Dadas la clase `Order` y la clase `Store` de la figura se pide implementar el método `public Order findOrder (int orderId) throws Exception` de la clase Store que dado el identificador de un producto nos diga si ese identificador corresponde o no a uno de los pedidos del almacén. En caso de que exista ese pedido en el almacén el método devuelve como resultado el pedido. En caso contrario lanza una Excepción.

Apartado 2.2 método countMultipleOf (2 puntos)

Se sabe que todos los empleados de un mismo departamento tienen identificadores (`employeeID`) múltiplos del mismo número. Dadas las clases `Store`, la clase `Employee` y la interfaz `BTree` de la figura se pide implementar el método recursivo `public int countMultipleOf(int number, BTree<Employee> tree)` de la clase `Store` que dado un árbol de empleados (`tree`) y un número de departamento (`number`) , indique el número de empleados que tiene dicho departamento. Para hacerlo deberá recorrer el árbol de empleados contando sólo aquellos nodos cuyo identificador sea múltiplo del número dado. Si no se hace recursivo no se tendrá en cuenta. (Nota: Recuerda que debes manejar las excepciones que lancen los métodos de BTree)

Por ejemplo, dado el almacén que cuyo árbol de empleados (`employees`) se muestra en la en la figura y el número 3 que caracteriza al departamento técnico, la llamada a `countMultipleOf(3, employees)` devolvería como resultado `2` puesto que hay dos empleados cuyo identificador es múltiplo de 3 (el empleado `12` y el `15`).



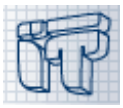
Ejercicio 3 Método selectionSort (2 / 7 Puntos).

Sabiendo que la ejecución de la clase `SelectionSortTest` que se muestra en la figura da como resultado:

```
Elements before sorting:
12 15 14 16 18 13 11 19 17

Elements after sorting:
19 18 17 16 15 14 13 12 11
```

Se pide implementar el método estático `selectionSort` de la clase `SelectionSortTest`.



```java
public class SelectionSortTest {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        numbers.add(12);
        numbers.add(15);
        numbers.add(14);
        numbers.add(16);
        numbers.add(18);
        numbers.add(13);
        numbers.add(11);
        numbers.add(19);
        numbers.add(17);

        System.out.println("Elements before sorting:");
        for (int i = 0; i < numbers.size(); i++) {
            System.out.print(numbers.get(i).toString() + " ");
        }
        selectionSort(numbers);
        System.out.println("\n\n" +"Elements after sorting:");
        for (int i = 0; i < numbers.size(); i++) {
            System.out.print(numbers.get(i).toString() + " ");
        }
    }
```

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

Ejercicio 1. (2 / 7 puntos)

Rúbrica:

- 0 si no tiene sentido
- 0,3 puntos manejo de excepciones
- 0,4 Realiza la conversión de info a Comparable todas las veces que se requiere tanto para maxInfo como para currentInfo
- 0,6 Recorre adecuadamente la lista
  - inicialización de current
  - condición del while
  - actualización de current
- 0,5 Actualiza el valor de maxInfo cuando es necesario
- 0,2 Devuelve resultado correcto

Solución

```
public Comparable max() {
    Comparable maxInfo = null;
    Node<E> current = first;
    if (isEmpty()) {
        maxInfo = null;
    } else {
        try {
            maxInfo = (Comparable) first.getInfo();
            while (current != null) {
                Comparable currentInfo = (Comparable) current.getInfo();
                if (currentInfo.compareTo(maxInfo) > 0) {
                    maxInfo = currentInfo;
                }
                current = current.getNext();
            }

        } catch (ClassCastException cce) {
            System.out.println("You cannot obtain max if info is not comparable");
        }
    }
    return maxInfo;
}
```

Ejercicio 2. (3 / 7 puntos)

Apartado 2.1 (1 punto)

Rúbrica:

- 0,25 si el bucle for es correcto con sus índices.(0,1 si pone length en vez de size())
- 0,5 condición correcta del bucle
  - 0,2 si recupera el Order de la lista con list.get(i) (0 si lo usa como si fuese un array accediendo con [])
  - 0,2 si recucpera el Id a partir del order con getOrderId().
  - 0,1 si la condición es correcta. (0 si usa compareTo en vez de ==)
- 0,25 si se retorna correctamente el resultado.

Solución

```
public Order findOrder(int orderID) {
    Order result = null;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i).getOrderID() == orderID) {
            return list.get(i);
        }
    }
    return result;
}
```

Apartado 2.2 (2 puntos)

Rúbrica:

- 0 si no se hace recursivo
- 0,25 inicialización y devolución del resultado correcto
- 0,25 manejo de excepciones (try/catch). Considerar correcto también si dentro del catch ponen un System.out.println
- 1 rama del if
  - 0,5 condición
  - 0,5 asignación del resultado
- 0,5 rama del else (asignación del resultado)

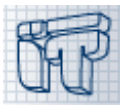
Solución:

```
public int countMultipleOf(int number, BTree<Employee> tree) {
    int result = 0;
    try {
        if (tree.isEmpty()) {
            result = 0;
        } else if (tree.getInfo().getEmployeeID() % number == 0) {
            result = 1 + countMultipleOf(number, tree.getLeft())
                       + countMultipleOf(number, tree.getRight());
        } else {
            result = countMultipleOf(number, tree.getLeft()) +
                     countMultipleOf(number, tree.getRight());
        }
    } catch (BTreeException bte) {
        bte.printStackTrace();
    }
    return result;
}
```

Ejercicio 3. (2 / 7 Puntos)

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 Declarar correctamente el método public static void SelectionSort(ArrayList<Integer> numeros).
  - 0,1 en total Si no se pone void y/o static.
  - Si pasa array como parámetro, como en las transparencias, en vez de usar ArrayList (ver criterio-1) -> 0,1 por todo el ejercicio si es correcto.



- 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 símbolo < en el orden adecuado. 0 si hace ordenación en otro orden.
    - 0,25 Llamada correcta a los elementos de la lista con get. 0 si accede como si fuese un array
- 0,25 actualización de m
- 0,25 llamada a swap . 0 si hay algún fallo en los parámetros.

Solución:

```
public static void selectionSort(ArrayList<Integer> numbers) {
    for (int i = 0; i < numbers.size(); i++) {
        int m = i;
        for (int j = i; j < numbers.size(); j++) {
            if (numbers.get(j) > numbers.get(m)) {
                m = j;
            }
        }
        int aux = numbers.get(i);
        numbers.set(i, numbers.get(m));
        numbers.set(m, aux);
    }
}
```