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 (3 / 7 puntos)

Un almacén de productos dispone de una aplicación programada en Java que incorpora las clases *Product* y *StoreManager*. La clase *Product*, mostrada a continuación, dispone de un identificador *Integer* como código de producto, junto con los métodos constructor, *getID* y *toString* para manejarlo.

```
class Product {
  private Integer id;
  public Product(Integer id)  { this.id = id; }
  public Integer getId()      { return this.id;}
  public String toString()    { return this.id + ""; }
}
```

Por otro lado, la clase *StoreManager* incorpora un atributo -*ordersToProcess*- (pedidosAProcesar) implementado como una estructura *Deque* (cola doblemente enlazada), que permite la gestión de diferentes maneras de los pedidos que deben ser procesados. La estructura de implementación de esta *Deque* (*DLDeque*) se corresponde con el siguiente código:

```
interface Deque<E> {
  void insertFirst(E element);
  void insertLast(E element);
  E removeFirst();
  E removeLast();
  int size();
}
```

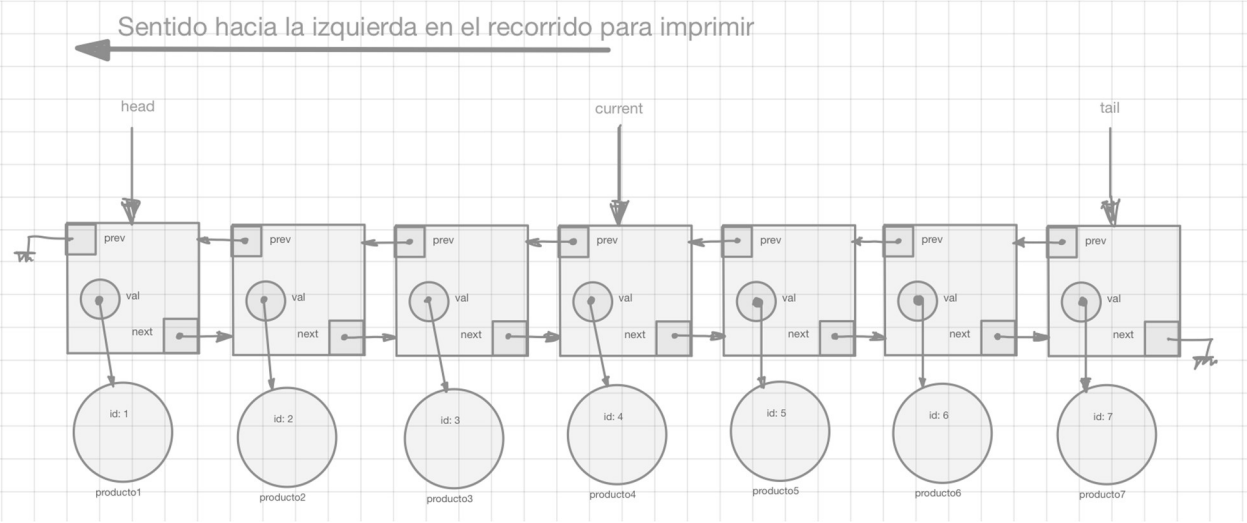
```java
class DNode<E> {

  DNode next, prev;
  E val;

  public DNode() { ... }
  public DNode(E val, DNode first, DNode last) { ... }
  public DNode getNext() { ... }
  public void setNext(DNode next) { ... }
  public DNode getPrev() { ... }
  public void setPrev(DNode prev) { ... }
  public E getVal() { ... }
  public void setVal(E val) { ... }

}
```

```java
public class DLDeque<E> implements Deque<E> {
  DNode<E> head, tail;
  int size;

  DNode<E> current;

  public DLDeque() { ... }
  public void insertFirst(E element) { ... }
  public void insertLast(E element) { ... }
  public E removeFirst() { ... }
  public E removeLast() { ... }
  public int size() { ... }
}
```

Observa cómo la clase *DLDeque*, además de *head* y *tail*, incorpora una referencia a objeto *DNode*, llamada -*current*-, que se va a utilizar en el procesado de los pedidos de la cola.

Implementa el método: *public void printInverseFromCurrent(),* de la clase *DLDeque,*que imprime por pantalla todos los identificadores de los productos de la Deque, teniendo en cuenta que se hace un recorrido desde -current- hacia la izquierda, terminando en el elemento anterior a -current-.

El resultado mostrado por pantalla para la siguiente cola de productos sería: 4 - 3 - 2 - 1 - 7 - 6 – 5



Se ha realizado un recorrido desde *current* hacia la izquierda, terminando en el elemento anterior a *current*, mostrando todos los identificadores de los productos.

Ejercicio 2 (2 / 7 puntos)

Dado un árbol binario de búsqueda *LBSTree*, que implementa la siguiente interfaz:

```
interface BSTree<E>
{
    boolean isEmpty();
    E getInfo();
    Comparable getKey();
    BSTree<E> getLeft();
    BSTree<E> getRight();
    String toStringPreOrder();
    String toStringInOrder();
    String toStringPostOrder();
    String toString();
    void insert(Comparable key, E info);
    BSTree<E> search(Comparable key);
}
```

Programa el método *public String toStringLevel(int level)* en la clase *LBSTree*, que devuelva una cadena con los nodos que se encuentran en el nivel -*level*-. Ten en cuenta que la raíz se encuentra en el nivel 0.

Nota: Ten en cuenta que el método *public String toStringLevel(int level)* no está en la interfaz *BSTree* por lo que cuando llames a este método deberás aplicar el casting correspondiente.

Ejercicio 3 (2 / 7 puntos)

Dada la siguiente clase *ArrayTree*, que representa un árbol binario de búsqueda, con implementación basada en array, donde la raíz está en la posición 1 (la posición 0 del array no se utiliza) e inicializado con los siguientes valores:

```
public class ArrayTree{
    private int [] elements = {0, 20, 10, 30, 5, 12, 25, 35};
    ...
}
```

Implementa el algoritmo de búsqueda binaria (de manera recursiva) con el método *public boolean search(int element)* en la clase *ArrayTree.* Devuelve -true- en el caso de que -element- sea encontrado en el árbol, -false- en otro caso.

SOLUCIONES

Ejercicio 1 (3 / 7 puntos)

Rúbrica:

- [0,5 puntos] - Inicialización correcta de referencia auxiliar (back) para no cambiar -current-
- [0,5 puntos] - Condición de parada correcta en el while
- [1 punto] - Avance correcto de referencia auxiliar: llegar a -head- vs -no estar en head-
- [1 punto] - Impresión correcta del último de los nodos.

Solución:

Nota: -prepareListToExam()- no se pedía en el enunciado. Es un método de apoyo implementado por el profesor para la comprobación del ejercicio.

```
public void printInverseFromCurrent() {
    prepareListToExam();

    DNode back = current;
    while (back != current.getNext()) {
      System.out.print(back.getVal() + " - ");
      if ( back != head ) {
        back = back.getPrev();
      }
      else
      {
        back = tail;
      }
    }
    // Por la condición del while falta imprimir el último.
    System.out.println(back.getVal());

}
```

Ejercicio 2 (2 / 7 puntos)

Rúbrica:

- [0,5 puntos] - Apoyo correcto en el método recursivo.
- [0,5 puntos] - Sólo añades al string aquél nodo que pertenezca al nivel.
- [0,5 puntos] - Recorrido correcto RID, IRD o IDR. Indiferente uno de los tres.
- [0,5 puntos] - Condiciones correctas de noVacio y getLeft y getRight != null.

Solución:

```
private String toStringLevel(int level, int currentLevel){

        String treeStr = "";
        LBSTree tree = null;

        if(!this.isEmpty())
        {
                // Raíz
                if (level == currentLevel)
                    treeStr = this.getInfo().toString() + " - ";

                // Izquierda
                if(this.getLeft() != null)
                {
                        tree = (LBSTree)this.getLeft();
                        treeStr = treeStr + tree.toStringLevel(level,
                        currentLevel + 1);
                }

                // Derecha
                if(this.getRight() != null)
                {
                        tree = (LBSTree)this.getRight();
                        treeStr = treeStr + tree.toStringLevel(level,
                        currentLevel + 1);
                }
        }
        return treeStr;


}

public String toStringLevel(int level){
    return toStringLevel(level, 0);

}
```

Ejercicio 3 (2 / 7 puntos)

Rúbrica:

- [0,4 puntos] - Elección correcta del método de apoyo recursivo.
- [0,4 puntos] - Condición de parada correcta.
- [0,4 puntos] - true elemento encontrado / false si no encontrado.
- [0,4 puntos] - Navegar correctamente por la derecha.
- [0,4 puntos] - Navegar correctamente por la izquierda.

Solución:

```
public class ArrayTree{

    private int [] elements = {0, 20, 10, 30, 5, 12, 25, 35};

    private boolean search(int element, int pos){
        if (pos >= elements.length)
            return false;
        else if (element == elements[pos])
            return true;
        else if (element > elements[pos]) // Derecha
            return search(element, (2*pos)+1);
        else if (element < elements[pos]) // Izquierda
            return search(element, 2*pos);
        else
            return false;

    }

    public boolean search(int element){
        return search(element, 1);
    }

    public static void main(String args[]){
        ArrayTree at = new ArrayTree();

        System.out.println("20: " + at.search(20));
        System.out.println("10: " + at.search(10));
        System.out.println("30: " + at.search(30));
        System.out.println("5: " + at.search(5));
        System.out.println("12: " + at.search(12));
        System.out.println("25: " + at.search(25));
        System.out.println("35: " + at.search(35));

        System.out.println("85: " + at.search(85));

    }


}
```