PRÁCTICA A003.- LISTAS DOBLEMENTE ENLAZADAS

Se va a utilizar una lista doblemente enlazada, cuya estructura de datos estará definida por dos referencia a un nodo doble:

- **front** : referencia al primer nodo de la lista
- **last**: referencia al último nodo de la lista

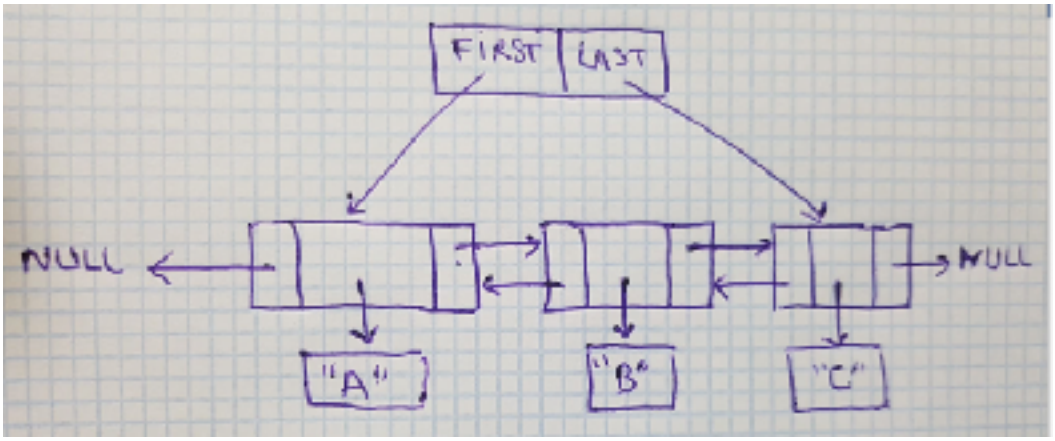
Los nodos de la lista doble son objetos de la clase DoubleNode<T>, que tienen 3 atributos:

- T **elem**: almacena el elemento.
- DoubleNode<T> **next**: referencia al siguiente nodo de la lista
- DoubleNode<T> **prev**: referencia al último nodo de la lista

ESTAS ESTRUCTURAS DE DATOS NO SE PUEDEN MODIFICAR NI AÑADIR NUEVOS ATRIBUTOS.

La referencia **prev** del primer nodo de la lista apuntará a null, ya que el primero no tiene anterior.

La referencia **next** del último nodo de la lista apuntará a null, ya que el último no tiene siguiente.



Si la lista es vacía, sus atributos front y last apuntarán a null.

VER PÁGINAS 27-30 DEL TEMA 6.- LISTAS

PASOS PARA LA REALIZACIÓN DE LA PRÁCTICA:

1. **Descargar el proyecto edi_a003_2020** de la página de la asignatura en agora.unileon.es.

2. **Importar** dicho proyecto en Eclipse : Import... **General**.....**Existing proyects into Workspace**... Select archive file... (indicar el archivo ZIP descargado)

3. El proyecto edi-a003-2020 contiene:

    A. UN fichero **DoubleList**, donde define el **interface** para la lista doble (define todas las operaciones a realizar con listas dobles) Este fichero **NO SE PUEDE MODIFICAR.**

    B. Un fichero DoubleLinkedListImpl, que se corresponde con la implementación del interface DoubleList, utilizando como estructuras de datos: LISTAS DOBLEMENTE ENLAZADAS.

    C. Un fichero DoubleLinkedListTests, que contiene métodos de tests en JUNIT4 para comprobar el buen funcionamiento de los métodos implementados en el fichero del apartado B.

Completar los ficheros de los apartados B y C, implementando los métodos definidos en el interface (así como el método toString() y añadiendo más tests para conseguir probar adecuadamente el buen funcionamiento del código implementado en el fichero DoubleLinkedListImpl. Los tests deben estar basados en asserciones que realicen las comprobaciones automáticamente (no deben tener System.out.println() que requieran consultar la consola para saber si funcionan los tests o no.

Se valorará la **cobertura total de los test implementados, por lo que debe usarse un plugin de Eclipse** para comprobar la cobertura de los test generados. Se buscará el 100% en la cobertura del fichero **DoubleLinkedListImpl**.

Se deberá entregar en agora.unileon.es la versión final de la práctica. Para ello habrá que exportar el proyecto edi-a003-2020 como zip (Export... General... Archive File).

IMPORTANTE: Utilizar JUNIT 4.

FECHA LIMITE DE ENTREGA: 19 de Abril de 2020 a las 23:59

public interface DoubleList<T> extends Iterable<T>{

Almacena una colección de objetos de tipo T, permitiendo elementos repetidos.
Ejemplo: (A B C A B D)

Excepciones

- No se permiten elementos **null**. Si a cualquier método que recibe un elemento se le pasa el valor null, lanzará una excepción **NullPointerException**.
- Los valores de parámetros **position** deben ser **mayores que cero y nunca negativos**. Si se recibe un valor negativo o cero se lanzará **IllegalArgumentException**.

Constructores

- Se definirá un constructor por defecto que inicialice la instancia como lista vacía.
- Se definirá otro constructor al que se le pueden pasar los elementos como parámetros y crea la lista con dichos elementos.

Método toString()

- Mostrará el contenido de la lista desde el primer elemento hasta el último, entre paréntesis: **(A B C D D D B)**

OPERACIONES DEL INTERFACE DoubleList :

1.- public boolean isEmpty(); Indica si esta lista está vacía

2.- public void clear(); Elimina todo el contenido de esta lista, es decir deja la lista vacía

3.- public void insertFirst(T elem);
Añade un elemento como primer elemento de la lista.
Si una lista l contiene (A B C) y hacemos l.insertFirst("C")
la lista quedará (C A B C)
@param elem el elemento a añadir
@throws NullPointerException si elem es null

4.- **public void insertLast(T elem);**
Añade un elemento como último elemento de la lista
Si una lista l contiene (A B C) y hacemos l.insertLast("C")
la lista quedará (A B C C)
@param elem el elemento a añadir
@throws NullPointerException si elem es null

5.- **public T removeFirst() throws EmptyCollectionException;**
Elimina y devuelve el primer elemento de la lista.
Si una lista l contiene (A B C) y hacemos l.removeFirst()
la lista quedará (B C) y devolverá A
@throws EmptyCollectionException si la lista es vacía

6.- public T removeLast() throws EmptyCollectionException;;

Elimina y devuelve el último elemento de la lista.
Si una lista l contiene (A B C) y hacemos l.removeLast()
la lista quedará (A B) y devolverá C
@throws EmptyCollectionException si la lista es vacía

7.- public void insertPos(T elem, int position);

Añade un elemento en la posición pasada como parámetro desplazando los elementos que estén a partir de esa posición.
Si una lista l contiene (A B C) y hacemos l.insertPos("C", 2)
la lista quedará (A C B C).
Si position>size() se insertará como último elemento.
@**param** elem el elemento a añadir
@**param** position la posición en la que añadirá el elemento

@throws NullPointerException si elem es null
@throws IllegalArgumentException si position <= 0

8.- **public void insertBefore(T elem, T target);**

Añade un elemento delante de la primera aparición del elemento pasado como 2º parámetro desplazando los elementos que estén a partir de ese elemento.
Si una lista l contiene (A B C B) l.insertPos("D", "B") ;
la lista quedará (A D B C B).
@param elem el elemento a añadir
@param target el elemento delante del cual insertará elem

@throws NullPointerException si elem o target son null
@throws NoSuchElementException si target no está en la lista

9.- **public T getElemPos(int position);**

Devuelve el elemento que está en position.
Si una lista l contiene (A B C D E):
* l.getElemPos(1) -> A
* l.getElemPos(3) -> C
* l.getElemPos(10) -> IllegalArgumentException

@param position posición a comprobar para devolver el elemento
@throws IllegalArgumentException si position no está entre 1 y size()

10.- public int getPosFirst(T elem);

Devuelve la posición de la primera aparición del elemento.
Si una lista l contiene (A B C B D A):
* l.getPosFirst("A") -> 1
* l.getPosFirst("B") -> 2
* l.getPosFirst("Z") -> NoSuchElementException

@param elem elemento a encontrar
@throws NoSuchElementException si elem no está en la lista.
@throws NullPointerException si elem es null

11.- public int getPosLast(T elem);

Devuelve la posición de la última aparición del elemento.
Si una lista l contiene (A B C B D A):

* l.getPosFirst("A") -> 6
* l.getPosFirst("B") -> 4
* l.getPosFirst("Z") -> NoSuchElementException

@param elem elemento a encontrar.

@throws NullPointerException si elem es null
@throws NoSuchElementException si elem no está en la lista.

12.- **public T removePos(int pos);**

Elimina y devuelve el elemento que está en position.

Si una lista l contiene (A B C B D A):

* l.removePos(1) -> "A", dejando la lista (B C B D A)
* l.removePos(3) -> "C", dejando la lista (A B B D A)
* l.removePos(10) -> IllegalArgumentException

@param position posición a comprobar para devolver el elemento.
@throws IllegalArgumentException si position no está entre 1 y size().

13.- public int removeAll(T elem);

Elimina todas las apariciones del elemento y
devuelve el número de instancias eliminadas.

Si una lista l contiene (A B C B D A B):

* l.removeAll("A") -> 2, dejando la lista (B C B D B)
* l.removeAll("B") -> 3, dejando la lista (A C D A)
* l.removeAll("Z") -> NoSuchElementException

@**param** elem elemento a eliminar.
@throws NullPointerException si elem es null
@throws NoSuchElementException si elem no está en la lista.

14.- public boolean contains(T elem);

Indica si el elemento está en esta lista.
Devuelve true si al menos existe una instancia del elemento dado en esta lista
(es decir, un elemento 'x' tal que x.equals(elemento)
y false en caso contrario.
@param elem elemento a buscar en esta lista
@throws NullPointerException el elemento indicado es null

15.- **public int size();**

Devuelve el número total de elementos en esta lista.
Ejemplo: Si una lista l contiene (A B C B D A B): l.size() -> 7

16.- **public String toStringReverse();**

Crea y devuelve un String con el contenido de la lista empezando por el final hasta el principio.
Si esta lista es vacía devuelve el toString() de la lista vacía -> ().
Ejemplo: Si una lista l contiene (A B C):
* l.toStringReverse() -> (C B A)
@**return** recorrido inverso de la lista (desde el final al principio)

17.- **public DoubleList<T> reverse();**

Crea una nueva lista inversa de esta lista.
Si esta lista es vacía devuelve la lista vacía.
Ejemplo:Si una lista l contiene (A B C):
* l.reverse().toString() -> (C B A)
@**return** lista inversa de esta lista

18.- **public int maxRepeated();**

Devuelve el número de veces que se repite el elemento con máximo número de repeticiones.
Ejemplo: Si una lista l contiene (A B C A A C A):
* l.maxRepeated() -> 4 // el elemento que más se repite es A,
y lo hace 4 veces
@**return** nº de repeticiones del elemento que más se repite

19.- **public boolean isEquals(DoubleList<T> other);**

Indica si esta lista es igual a la pasada como parámetro
(tienen los mismos elemento en el mismo orden).
Ejemplos:
* l1=(A B C) ; l2=(B C A) : l1.isEquals(l2) -> false
* l1=(A B C) ; l2=(A B C) : l1.isEquals(l2) -> true

@**param** other lista a comprobar si es igual a esta lista
@**return** true si son iguales, false en caso contrario

@**throws NullPointerException** other es null

20.- **public boolean containsAll(DoubleList<T> other);**

Indica si esta lista contiene todos los elementos de la lista pasada como parámetro.
Ejemplos:
* l1=(A B C D E) ; l2=(B C A) : l1.containsAll(l2) -> true
* l1=(A B C) ; l2=(A B D) : l1.containsAll(l2) -> false

@**param** other lista a comprobar
@**return** true si esta lista contiene todos los elementos de other,
false en caso contrario

@**throws NullPointerException** other es null

21.- **public boolean isSubList(DoubleList<T> other);**

Indica si other es sublista de esta lista.
Una lista vacía es sublista de cualquier lista.
Ejemplos:

* l1=(A B C D E) ; l2=(B C D) : l1.isSubList(l2) -> true
* l1=(A B C D E) ; l2=(A B D) : l1.isSubList(l2) -> false

@**param** other lista a comprobar
@**return** true si other es sublista de esta lista, false en caso contrario
@**throws NullPointerException** other es null

22.- **public String toStringFromUntil(int from, int until);**

Devuelve el toString de la sublista formada por los elementos situados entre las posiciones from hasta until incluidas.
Si **until** > **size**() se muestra hasta el final de la lista.
Si **from** > **size**() se muestra () (toString de la lista vacía)
Ejemplos:

* l1=(A B C D E) ; l1.toSringFromUntil(1,3) -> (A B C)
* l1=(A B C D E) ; l1.toSringFromUntil(3,10) -> (C D E)
* l1=(A B C D E) ;l1.toStringFromUntil(10,20) -> ()

@param from posición desde la que se empieza a considerar la lista (incluida)
@param until posición hasta la que se incluyen elementos (incluida)

@return String de la sublista formada por los elementos en el rango establecido por los dos parámetros.
@throws IllegalArgumentException si from o until son <=0 ; o si until < from

<u>ITERADORES A IMPLEMENTAR: Por cada uno de estos 4 iterados se deberá crear una clase que implemente el interface iterator</u>

23.- public Iterator<T> iterator();

Debido a que el interface extiende del interface iterable
(que tiene el método iterator())

Devuelve un iterador que recorre la lista en orden directo (desde front hasta last)
Por ejemplo, para una lista x con elementos (A B C D E)
el iterador creado con x.iterator()
devuelve en sucesivas llamadas a next(): A, B, C, D y E.

@**return** iterador para orden directo.

24.- public Iterator<T> reverseIterator();

Devuelve un iterador que recorre la lista en orden inverso.
Por ejemplo, para una lista x con elementos (A B C D E)
el iterador creado con x.reverseIterator()
devuelve en sucesivas llamadas a next(): E, D, C, B y A.

@**return** iterador para orden inverso.

25.- public Iterator<T> evenPositionsIterator();

Devuelve un iterador que recorre los elementos con posición par de la lista.
Por ejemplo, para una lista x con elementos (A B C D E)
el iterador creado con x.evenPositionIterator()
devuelve en sucesivas llamadas a next(): B y D.
@return iterador para recorrer elementos en posiciones pares.

26.- **public Iterator<T> progressIterator();**

Devuelve un iterador que recorre los elementos saltando progresivamente i posiciones de la lista.
Es decir, en cada llamada a next() **salta i posiciones**,
la i empieza en 1 y va incrementandose en cada llamada a next().
Por ejemplo,
para una lista x con elementos (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19)
el iterador creado con x.progressIterator()
devuelve en sucesivas llamadas a next(): 1, 2, 4, 7, 11 y 16.
@return iterador para recorrer ciertos elementos de la lista como se indica anteriormente.