ESTRUCTURAS DE DATOS

Curso 2025/26

PRÁCTICA 4

Cadenas de nodos enlazados

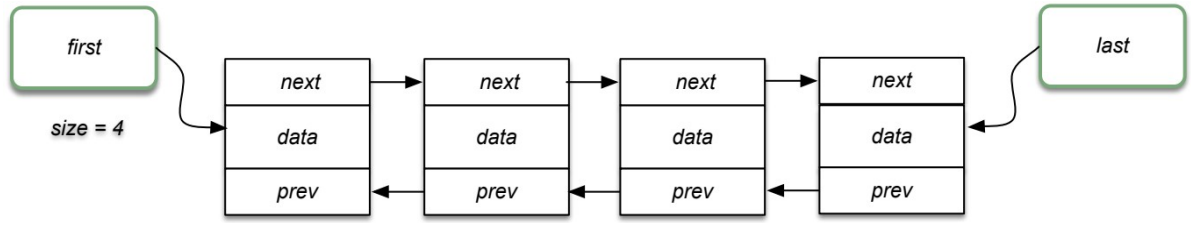
Instrucciones

- *Se debe completar en dos sesiones.*
- *Práctica individual.*
- *Lee el enunciado completo antes de comenzar.*
- *Se habilitará una tarea para que entregues el código desarrollado.*
- *La práctica será APTA si se superan todos los test de validación proporcionados.*

Tenemos una clase **`EDDoubleLinkedList<T>`** que implementa sólo parcialmente la interfaz **`List<T>`** de la *Java Collection Framework*. En ella, se han almacenado los datos de la lista usando una cadena no circular de nodos doblemente enlazados. La definición de su parte privada eta formada por:

- **`Node`**: clase privada que representa un nodo con doble enlace
- **`first`**: referencia al primer nodo de la cadena
- **`last`**: referencia al último nodo de la cadena.
- **`size`**: número de nodos en la cadena.

El dibujo representa gráficamente cómo se implenta la lista.



Ejercicio 1

Añade un método **`reverse`** a la clase que invierta el orden de los elementos de la lista. Es decir, que cambie el orden de todos sus elementos de forma que los que estén al principio acaben al final y viceversa. La implementación debe modificar los enlaces de los nodos existentes y no debe usar estructuras de datos auxiliares.

```
public void reverse()
```

Ejemplo:

- [5, 6, 3, 5, 3, 9, 1, 0] ⇒ [0, 1, 9, 3, 5, 3, 6, 5]

Ejercicio 2

Añade un método **shuffle** a la clase. El método tomará otra lista como parámetro y la mezclará con la lista local **this**, modificándola. De tal forma que el primer elemento de **lista** se insertará entre el primero y el segundo de la lista local, el segundo elemento de **lista** entre el segundo y el tercero de la lista local, y así sucesivamente. Si la lista local fuese demasiado corta se insertarían los elementos sobrantes de **lista** al final de la lista local. En el caso de que **lista** sea **null** se lanzará la excepción **NullPointerException**. Implementa el ejercicio sin usar estructuras de datos auxilares.

```
public void shuffle(List<T> lista)
```

Ejemplos:

- this: [9, 8, 7, 6]; lista: [2, 4, 5]; ⇒ this: [9, 2, 8, 4, 7, 5, 6]
- this: [99, 100]; lista: [2, 4, 5]; ⇒ this: [99, 2, 100, 4, 5]

Ejercicio 3

Añade un método **prune(inicio, fin)** a la clase que "pode" la lista de acuerdo con el valor de los índices **inicio** y **fin**. Así, dados dos valores enteros representando dos índices de la lista, el método dejará en la lista los elementos comprendidos entre el elemento que ocupa la posición **inicio** (inclusive) y el elemento que ocupa la posición **fin** sin incluirlo (exclusive). Si **inicio** es menor que 0, se considerará el valor 0. Por otro lado, si **fin** es mayor que el tamaño de la lista se considerará **size**. Si **fin** es menor o igual que **inicio**, entonces la lista quedará vacía. En el caso de que ambos índices sean menores que 0 o ambos mayores o iguales al tamaño de la lista, el método lanzará la excepción **IndexOutOfBoundsException**. El método devolverá **true** si la lista ha sido modificada y **false** en caso contrario. Implementa el ejercicio sin usar estructuras de datos auxilares.

```
public boolean prune(int inicio, int fin)
```

Ejemplos:

- this: [9, 8, 7, 6]; inicio:-1, fin: 4; ⇒ this: [9, 8, 7, 6], false
- this: [99, 100]; inicio: 1, fin: 0; ⇒ this: [ ], true
- this: [9, 8, 7, 6]; inicio: 0, fin: 2; ⇒ this: [9, 8], true
- this: [99, 100, 1, 4, 10, 18]; inicio: 2, fin: 5; ⇒ this: [1, 4, 10], true
- this: [10, 4, 10, 8]; inicio: 0, fin: 0; ⇒ this: [ ], true
- this: [10, 4, 10, 8]; inicio: 0, fin: 1; ⇒ this: [10], true
- this: [10, 4, 10, 8]; inicio: 1, fin: 2; ⇒ this: [4], true
- this: [10, 4, 10, 8]; inicio: 3, fin: 1; ⇒ this: [ ], true

Ejercicio 4

Implementa un método que borre de la lista todos los los nodos con el valor que se le pasa como párametro. Devolverá el número de nodos borrados.

```
public int removeAllElements (T elem)
```

Ejemplos:

- this: [9, 8, 3], elem = 3 ⇒ this: [9, 8], devuelve 1
- this: [9, 8, 3], elem = 5 ⇒ this: [9, 8, 3], devuelve 0
- this: [9, 9, 8, 3, 9, 4, 5], elem = 9 ⇒ this: [8, 3, 4, 5], devuelve 3

Ejercicio 5

Implementa un método que borra de la lista los nodos adyacentes con elementos repetidos, dejando un solo nodo con ese valor. Devuelve el número de nodos borrados.

```
public int removeAdjacentDuplicates ()
```

Ejemplos:

- this: [9, 8, 3] ⇒ this: [9, 8, 3], devuelve 0
- this: [2, 1, 0, 0, 3, 0, 1, 1, 1, 0] ⇒ this:[2, 1, 0, 3, 0, 1, 0], devuelve 3

Ejercicio 6

Implementa el método **`retainAll`**:

```
public boolean retainAll(List<T> otra)
```

NOTA: La clase **`EDDoubleLinkedList`** que se te proporciona tan sólo implementa **los** métodos **`size`**, **`isEmpty`**, **`clear`**, **`toArray`**, **`toString`** y un constructor de copia. Si necesitas alguno de los otros métodos, deberás implementarlos.