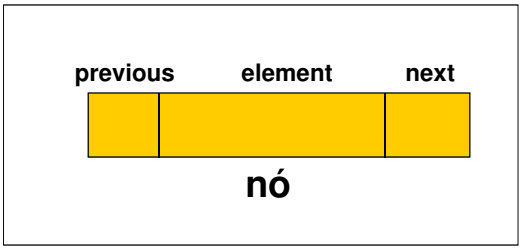
Listas Duplamente Encadeadas

Profs Prog2 e Lab2 Unisinos

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

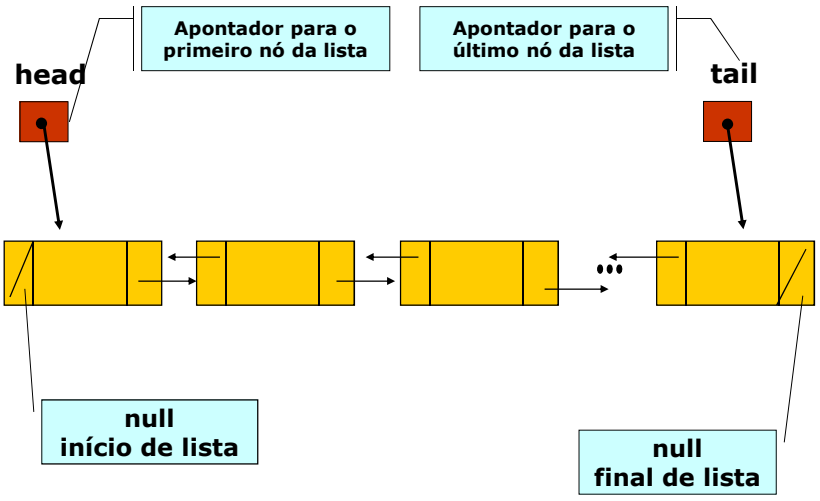
Listas Duplamente Encadeadas

- Cada nó possui dois ponteiros:



- Vantagem: simplifica certas operações e permite percorrer a lista nos dois sentidos.
- Desvantagem: gasta mais espaço do que a simplesmente encadeada (mais um ponteiro em cada nó) e pode tornar mais complexas certas operações.

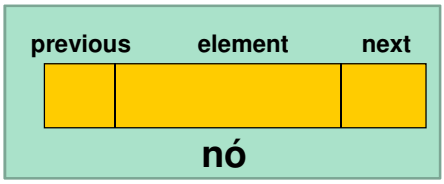
Lista encadeada com referência ao ultimo elemento da lista



classe DNode

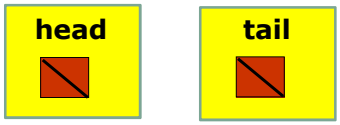
```
public class DNode<E> {
    private E element;
    private DNode<E> next;
    private DNode<E> previous;

public DNode(E element) {
   this (element, null, null);
}
public DNode(E element, DNode<E> next,
   DNode<E> previous) {
    super();
    this.element = element;
    this.next = next;
    this.previous = previous;
}
public E getElement() {
   return element;
}
public void setElement(E element) {
   this.element = element;
}
```

```
public DNode<E> getNext() {
   return next;
}
public void setNext(DNode<E> next) {
   this.next = next;
}
public DNode<E> getPrevious() {
   return previous;
}
public void setPrevious(DNode<E> previous)
{
   this.previous = previous;
}

} // end of class
```



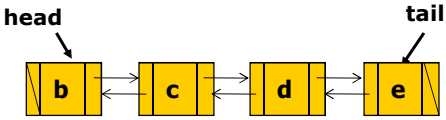
class DLinkedList (1)

```
public class DLinkedList<E> {
    protected DNode<E> head; //nodo cabeça da lista
    protected DNode<E> tail; //nodo cauda da lista
    protected long size; //número de nodos da lista

    public DLinkedList() {
        size = 0;
        head = tail = null;
    }

    public boolean isEmpty() {
       return head == null;
    }
```



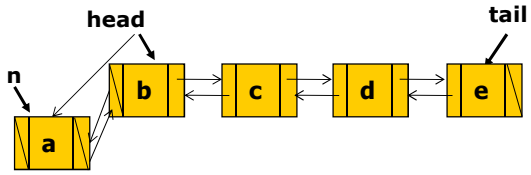
class DLinkedList (2)

```
public E getFirst() throws UnderflowException {
        if (isEmpty())
             throw new UnderflowException();
        return head.getElement();
}

public E getLast() throws UnderflowException {
        if (isEmpty())
             throw new UnderflowException();
        return tail.getElement();
}
```



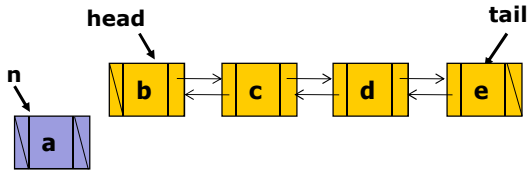
class DLinkedList (3)

```
public void addFirst(E insertItem) {
    DNode<E> n = new DNode<E>(insertItem);
    if (isEmpty()) {
      head = tail = n;
    } else {
       head.setPrevious(n);
       n.setNext(head);
       head = n;
    }
    size++;
}
```



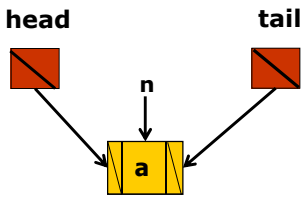
class DLinkedList (3)

```
public void addFirst(E insertItem) {
    DNode<E> n = new DNode<E>(insertItem);









}
```



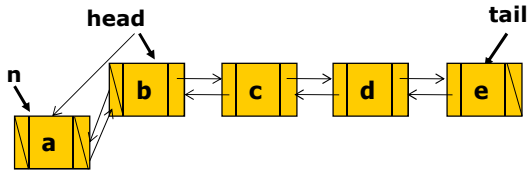
class DLinkedList (3)

```
public void addFirst(E insertItem) {
    DNode<E> n = new DNode<E>(insertItem);
    if (isEmpty()) {
      head = tail = n;
    }
}
```



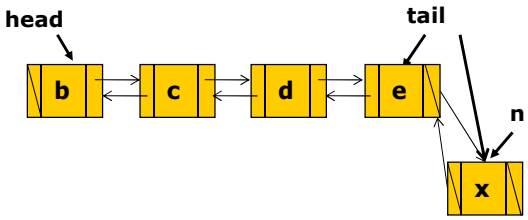
class DLinkedList (3)

```
public void addFirst(E insertItem) {
    DNode<E> n = new DNode<E>(insertItem);
    if (isEmpty()) {
      head = tail = n;
    }
    else {
        head.setPrevious(n);
        n.setNext(head);
        head = n;
    }
    size++;
}
```



class DLinkedList (4)

```
public void addLast(E insertItem) {
    DNode<E> n = new DNode<E>(insertItem);
    if (isEmpty()) {
      head = tail = n;
    }
    else {
        tail.setNext(n);
        n.setPrevious(tail);
        tail = n;
    }
    size++;
}
```

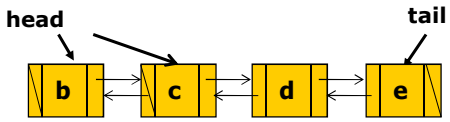


class DLinkedList (5)

```
public E removeFirst() throws UnderflowException {
    if (isEmpty()) {
      throw new UnderflowException();
    }
    E removedItem = head.getElement();
    if (head == tail) {
      head = tail = null;
    }


   return removedItem;
}
```

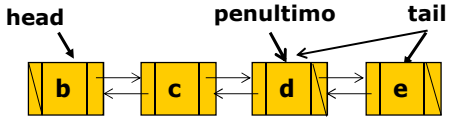
class DLinkedList (5)

```
public E removeFirst() throws UnderflowException {
    if (isEmpty()) {
      throw new UnderflowException();
    }
    E removedItem = head.getElement();
    if (head == tail) {
      head = tail = null;
    }
    else {
        head = head.getNext();
        head.setPrevious(null);
    }
    size--;
    return removedItem;
}
```



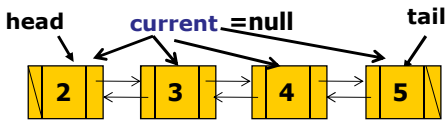
class DLinkedList (6)

```
public E removeLast() throws UnderflowException {
    if (isEmpty()) {
      throw new UnderflowException();
    }
    E removedItem = tail.getElement();
    if (head == tail) {
      head = tail = null;
    }
    else {
        DNode<E> penultimo = tail.getPrevious();
        tail = penultimo;
        tail.setNext(null);
    }
    size--;
    return removedItem;
}
```



class DLinkedList (7)

```
public String toString() {
        String str = "\nExibindo a lista: ";
        DNode<E> current = head;
        while (current != null) {
                str += current.getElement() + " - ";
                current = current.getNext();
        }
        return str;
}
```



Testando a lista

```
public static void main(String args[]) {
        DLinkedList<Integer> list = new DLinkedList<Integer>();
        list.addLast(2);
        list.addLast(4);
        list.addLast(6);
        list.addLast(1);
        list.addLast(8);
        list.addLast(9);
        System.out.println(list.toString());

        try {
                list.removeFirst();
        } catch (UnderflowException e) {
                e.printStackTrace();
        }
        System.out.println(list.toString());

}
```

Exercício 1

- Na classe DLinkedList (implementação de lista **duplamente** encadeada), implementar um método com a seguinte assinatura:
  - "`private DNode<E> find(E key)`".
- Este método deve buscar na lista o nó de chave key e retorná-lo.

Exercício 2

- Na classe DLinkedList (implementação de lista **duplamente** encadeada), implementar um método com a seguinte assinatura:
  - "`public boolean addBefore (E el, E key)`".
- Este método deve inserir na lista um nó com chave *el* na posição anterior ao nó que contenha a chave *key*.