Задача:  
Реализовать любую коллекцию на выбор из LinkedList

**Алгоритм:**

**1. Создаю класс Node, описывающий элементы будущего LInkedList.**

\*Тип данных оперируемый - generic T, для возможности работы с любыми типами данных.

\*Создаю ссылки на следующий/предыдущий элементы next/previous

**\***Переопределяю equals()

**2. Описываю класc doubleLinkedList, описывающий будущую коллекцию из элементов Node<T>.**

\*Создаю стартовые элементы: head & tail**.**

**3. Описываю метод для добавления нового элемента( в конец):** \*В зависимости от позиции элемента обновляю ссылки next/previous

\*Обрабатываю случай, когда коллекция пуста.

**4. Описываю метод для удаления элемента по значению**

\*Нахожу элемент по значению (циклом for) и переопределяю ссылки предыдущего/следующего элементов, исключая из цепочки элементов удаляемый объект.

\*Также обрабатываю случаи, когда остается 1 элемент, либо список оказывается пуст.

**5. Добавляю методы для нахождения элемента, измерения длины, добавления элемента на первую позицию, вывода элементов, проверки на пустоту и очистки списка.**

**6. Создаю экземпляр описанной коллекции и провожу тестовые операции.**

**\*Описаны были как односвязный список, так и двусвязный**

**Листинг кода представлен только для двусвязного списка.**

**Листинг кода:**

**Node:**

public class Node<T>{  
 T data; // Добавляемые данные  
 Node<T> next; // Ссылка на следующий элемент  
 Node<T> previous; // Для двусвязного списка ссылка на пред.элемент  
  
 public Node(T data){  
 this.data = data;  
 this.next = null;  
 this.previous = null;  
  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (this == object) return true;  
 if (object == null || getClass() != object.getClass()) return false;  
 Node<?> node = (Node<?>) object;  
 return Objects.*equals*(data, node.data) && Objects.*equals*(next, node.next) && Objects.*equals*(previous, node.previous);  
 }  
}

doubleLinkedList:

public class doubleLinkedList<T> {  
 private Node<T> head;  
 private Node<T> tail;  
  
 public doubleLinkedList(){  
 // При создании список пуст  
 this.head = null;  
 this.tail = null;  
 }  
  
 /\* Добавление нового элемента:  
 1- если первый: меняю сслку для next; previous остается null  
 2- если последний: previous - меняется, next = null  
 3- если в середину: меняются и next и previous  
 \*/  
 public void append(T data){  
 Node<T> newNode = new Node<>(data);  
 Node<T> current = tail;  
  
 if (head == null){  
 head = newNode;  
 tail = newNode;  
 return;  
 }  
 current.next = newNode;  
 newNode.previous = current;  
 tail = newNode;  
 }  
 // Поочередно вывожу все элементы  
 public void printList(){  
 if (head == null){  
 System.*out*.println("List is empty!");  
 return;  
 }  
 Node<T> current = head;  
 System.*out*.print("[");  
 while (current.next != null){  
 System.*out*.print(current.data + ",");  
 current = current.next;  
 }  
 System.*out*.println(current.data + "]");  
 }  
  
 public void delete(T data){  
 Node<T> current = head;  
  
 while (current != null && current.data != data) {  
 current = current.next;  
 }  
  
 if (current == null) {  
 // Элемент не найден  
 return;  
 }  
  
 if (current == head){  
 head = current.next;  
 if (head != null){  
 head.previous = null;  
 } else {  
 tail = null;  
 }  
 } else if (current == tail){  
 tail = current.previous.next = null ;  
 }else {  
 if (current.previous != null) {  
 current.previous.next = current.next;  
 }  
 if (current.next != null) {  
 current.next.previous = current.previous;  
 }  
 }  
 }  
  
 public void addFirst(T data){  
 Node<T> newNode = new Node(data);  
 newNode.next = head;  
 head.previous = newNode;  
 head = newNode;  
  
 }  
  
 public boolean find(T data){  
 Node<T> current = head;  
  
 while (current != null){  
 if (current.data == data)  
 return true;  
 current = current.next;  
 } return false;  
 }  
  
 public int size(){  
 Node<T> current = head;  
 int count = 0;  
  
 while(current != null){  
 count++;  
 current = current.next;  
 } return count;  
 }  
  
 public boolean isEmpty(){  
 return head == null;  
 }  
  
 public void clear(){  
 head = null;  
 }  
}

Проверка корректности работы списков:

public class Main {  
 public static void main(String[] args){  
 LinkedList<Integer> list = new LinkedList<>();  
 list.append(5);  
 list.append(3);  
 list.append(10);  
  
 list.printList();  
 list.addFirst(0);  
  
 System.*out*.println("Size: " + list.size());  
 System.*out*.println("Is empty: " + list.isEmpty());  
 System.*out*.println("Node \"10\"" + list.find(10));  
  
 list.delete(0);  
 list.printList();  
 list.clear();  
 list.printList();  
  
 System.*out*.println("--------------");  
 doubleLinkedList<Integer> dlist = new doubleLinkedList<>();  
 dlist.append(5);  
 dlist.append(3);  
 dlist.append(7);  
 dlist.append(6);  
 dlist.append(10);  
  
 dlist.printList();  
 dlist.addFirst(0);  
  
 System.*out*.println("Size: " + dlist.size());  
 System.*out*.println("Is empty: " + dlist.isEmpty());  
 System.*out*.println("Node \"10\"" + dlist.find(10));  
  
 dlist.delete(5);  
 dlist.printList();  
 dlist.clear();  
 dlist.printList();  
 }  
}

Результат:

[5,3,10]

Size: 4

Is empty: false

Node "10"10

[5,3,10]

List is empty!

--------------

[5,3,7,6,10]

Size: 6

Is empty: false

Node "10"lab3.Node@34c45dca

[0,3,7,6,10]

List is empty!