Семинар №3

Связный список

---

1. **Цели семинара №3:**
    - Научиться писать структуру связного списка
    - Понимать принципы работы со связными списками
    - Уметь модифицировать связный список

    По итогам семинара №3 слушатель должен **знать**:
    - Внутреннюю структуру связного списка
    - Различные типы связных список
    - Особенности работы со структурой связного списка

    По итогам семинара №3 слушатель должен **уметь**:
    - Писать различные реализации связных списков
    - Уметь оперировать с элементами связного списка для его модификации

Формат проведения семинара

1. Ученики разбиваются на группы по 5 человек в сессионные комнаты зум
2. Один ученик расшаривает экран
3. Обсуждают задание и как его лучше выполнить
4. Ученик, расшаривший экран пишет код, другие ученики участвуют в обсуждении и подсказывают что нужно еще добавить
5. После выполнения первого задания, ученики скидывают архивы с работой для проверки

---

2. **План Содержание:**

Этап урока	Тайминг, минуты	Формат
Вопросы по лекции, подготовка к семинару	10	Преподаватель спрашивает, есть ли вопросы по пройденному материалу и отвечает на них

Реализуем односвязный список	5	Реализуем простой односвязный список. Пишем только структуру, никаких методов не требуется
Реализуем функционал добавления и удаления данных в начало списка	10	Реализуем метод добавления новых элементов в начало списка и удаление первого элемента связного списка
Реализуем алгоритм поиска элемента в связном списке (наличие или отсутствие элемента в списке)	10	Реализуем метод поиска элемента в односвязном списке для проверки наличия элемента внутри списка
Реализуем алгоритм добавления и удаления последнего элемента из связного списка	10	Реализуем метод добавления новых элементов в конец списка и удаление последнего элемента связного списка
Преобразуем односвязный список в двухсвязный список	5	Расширяем структуру связного списка до двухсвязного
Модифицируем методы добавления и удаления элементов из конца списка	10	Обновляем методы согласно новой структуре
Реализуем функцию сортировки для связного списка	15	Добавляем метод сортировки для связного списка. Можно использовать любой алгоритм, что мы использовали на предыдущем семинаре, но с точки зрения работы связного списка лучше ориентировать студентов на пузырьковую сортировку
Длительность:	**75**	

3. Блок 1.

Задание:
Реализуем простой односвязный список. Пишем только структуру, никаких методов не требуется. Напоминаем студентам структур связного списка и что односвязный список представляет собой последовательность элементов, каждый из которых ссылается на следующий элемент цепочки

Пример решения:

```
public class List {
    private Node head;

    private class Node {
        private Node next;
        private int value;
    }
}
```

Блок 2.

Задание:
Реализуем метод добавления новых элементов в начало списка и удаление первого элемента связного списка. Односвязный список всегда имеет ссылку на первый элемент последовательности, потому именно с реализации методов для первого элемента последовательности стоит начать

Пример решения:

```
public class List {
    private Node head;

    public void addFirst(int value){
        Node node = new Node();
        node.value = value;
        if (head != null) {
            node.next = head;
        }
        head = node;
    }

    public void removeFirst(){
        if (head != null){
            head = head.next;
        }
    }

    private class Node {
        private Node next;
        private int value;
    }
}
```

Блок 3.

Задание:
Реализуем метод поиска элемента в односвязном списке для проверки наличия элемента внутри списка. Для корректной работы со связным список необходимо понимать, как именно можно обходить все значения внутри связного списка. Для нашего примера проще всего будет

написать метод поиска значения в связном списке и возвращения из метода информации о наличии искомого внутри списка.

Пример решения:

```
public boolean contains(int value){
    Node node = head;
    while (node != null){
        if (node.value == value){
            return true;
        }
        node = node.next;
    }
    return false;
}
```

Блок 4.

Задание:

Реализуем метод добавления новых элементов в конец списка и удаление последнего элемента связного списка. Теперь, когда мы понимаем, как можно искать значения внутри связного списка, мы можем сделать методы добавления и удаления элементов в конец нашего односвязного списка.

Пример решения:

```
public void addLast(int value) {
    Node node = new Node();
    node.value = value;
    if (head == null) {
        head = node;
    } else {
        Node last = head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = node;
    }
}

public void removeLast() {
    if (head != null) {
        Node node = head;
        while (node.next != null) {
            if (node.next.next == null) {
                node.next = null;
                return;
            }
            node = node.next;
        }
        head = null;
    }
}
```

Возможные проблемы:

Стоит обратить внимание на кейс удаления последнего элемента. В частности, на пограничные условия, когда в списке всего 1 элемент и это head. Тогда удаление последнего элемента должно удалять ссылку в head.

Блок 5.

Задание:

Расширяем структуру связного списка до двухсвязного. Мы научились работать с односвязным список, теперь можно ближе познакомиться со структурой двухсвязного списка и особенностей его внутреннего строения. Стоит напомнить студентам, что двухсвязный список представляет из себя цепочку элементов, которые умеют ссылаться не только на следующий элемент последовательности, но и на предыдущий. Стоит предупредить студентов, что вносить корректировки в уже готовые методы на текущий момент не стоит, их модификацией мы займемся позднее

Пример решения:

```
public class List {
    private Node head;
    private Node tail;

    private class Node {
        private Node next;

        private Node prev;

        private int value;
    }
}
```

Блок 6.

Задание:

Обновляем методы согласно новой структуре. Появилась дополнительная переменная, которую необходимо отслеживать во всех операциях. Так же благодаря ссылке на последний элемент списка операции работы с концом стали проще и их стоит заменить на логику аналогичную работе с началом списка

Пример решения:

```
public void addFirst(int value) {
    Node node = new Node();
    node.value = value;
    if (head != null) {
        node.next = head;
        head.prev = node;
    } else {
        tail = node;
    }
    head = node;
}

public void removeFirst() {
    if (head != null && head.next != null) {
        head.next.prev = null;
        head = head.next;
    } else  {
        head = null;
        tail = null;
    }
}

public void addLast(int value) {
    Node node = new Node();
    node.value = value;
    if (tail != null) {
        node.prev = tail;
        tail.next = node;
    } else {
        head = node;
    }
    tail = node;
}

public void removeLast() {
    if (tail != null && tail.prev != null) {
        tail.prev.next = null;
        tail = tail.prev;
    } else  {
        head = null;
        tail = null;
    }
}
```

Возможные проблемы:
При работе с элементами стоит не забывать про элементы, которые ссылаются на ноду. В частности, при работе с первым и последним элементом всегда нужно так же обновлять ссылки на втором и предпоследнем элементах соответсвенно, а не только менять значения данных в head и tail.

Блок 7.

Задание:
Добавляем метод сортировки для связного списка. Можно использовать любой алгоритм, что мы использовали на предыдущем семинаре, но с точки зрения работы

связного списка лучше ориентировать студентов на пузырьковую сортировку, т.к. она взаимодействует с соседними элементами, а не только по индексам, как делают все остальные сортировки.

Пример решения:

```
public void sort() {
    boolean needSort;
    do {
        needSort = false;
        Node node = head;
        while (node != null && node.next != null){
            if (node.value > node.next.value){
                Node before = node.prev;
                Node after = node.next.next;
                Node current = node;
                Node next = node.next;

                current.next = after;
                current.prev = next;
                next.next = current;
                next.prev = before;
                if (before != null){
                    before.next = next;
                } else {
                    head = next;
                }
                if (after != null) {
                    after.prev = current;
                } else  {
                    tail = current;
                }

                needSort = true;
            }
            node = node.next;
        }
    } while (needSort);
}
```

Возможные проблемы:
В отличии от сортировки массива, при обмене двух соседних позиций в связном списке, так же не обходимо обновить ссылки на соседних от них элементах, либо обновить ссылки на head и tail. Стоит обратить особое внимание, что в ситуации, когда необходим обмен, работать придется не с 2, а с 4 нодами.

4. Итоги

На данном семинаре студенты научились проектировать такую структуру данных, как связный список. Познакомились с двумя основными его реализациями – односвязный и двухсвязный списки. Научились искать элементы, а также удалять и добавлять их в связный список. Реализовали один из алгоритмов сортировки, применимый к связным спискам.

5. Домашнее задание

Необходимо реализовать метод разворота связного списка (двухсвязного или односвязного на выбор).

Пример решения:

```
//Решение для двухсвязного списка
public void revert() {
    Node node = head;
    while (node != null) {
        Node temp = node.next;
        node.next = node.prev;
        node.prev = temp;
        node = temp;
    }
    Node temp = head;
    head = tail;
    tail = temp;
}
```

```
//Решение для односвязного списка
public void revert() {
    if (head != null && head.next != null) {
        revert(head, head.next);
    }
}

private void revert(Node current, Node next) {
    if (next.next != null) {
        revert(next, next.next);
    } else {
        head = next;
    }

    next.next = current;
    current.next = null;
}
```