Лекция 9. Односвязные списки

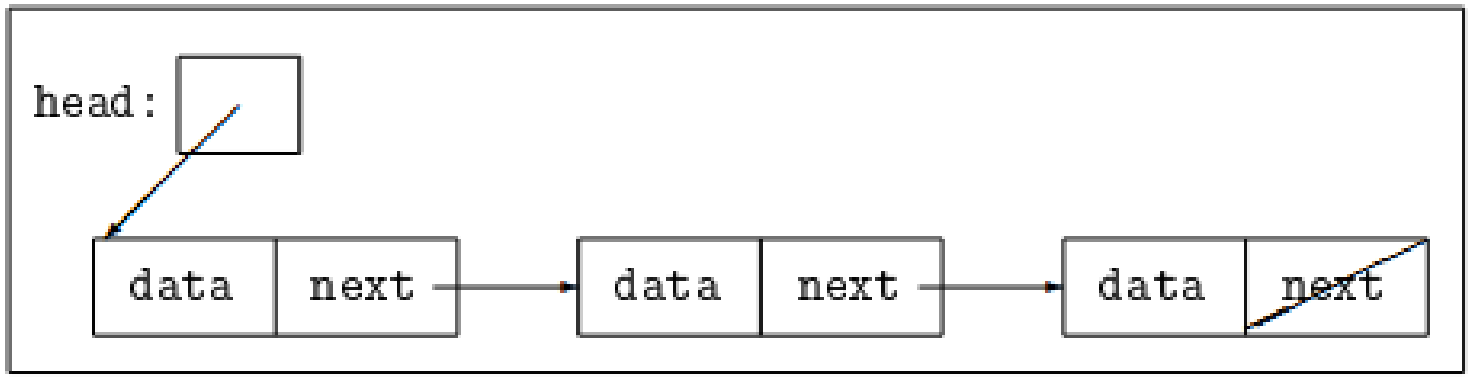
Списки

- **Список (list)** – это структура данных, представляющая собой упорядоченный набор значений, в котором значения могут повторяться
- Значения в списке называют **элементами** списка

- В курсе мы рассмотрим:
  - Списки на массиве
  - Связанные списки
    - Односвязные
    - Двусвязные

Односвязный список

- **Односвязный список** представляет собой ссылку на первый элемент списка
- Эту ссылку называют **головой списка** (**head**)
- Первый элемент списка хранит ссылку на второй элемент, второй элемент – на третий элемент и т.д.
- У последнего элемента ссылка на следующий элемент – пустая (равна null)



Односвязный список

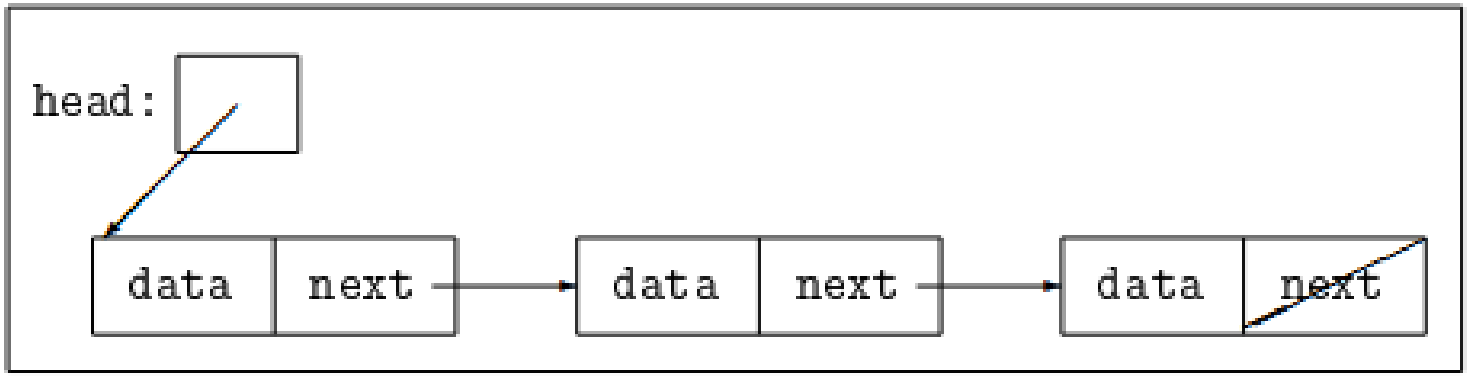
- Элемент списка можно представить таким классом
- ```
  class ListItem<T> {
      private T data;
      private ListItem<T> next;
      // часть кода опущена
  }
  ```

Тут опущены геттеры, сеттеры и конструкторы

data – полезные данные в узле, next – ссылка на некоторый элемент

- Имея ссылку на первый элемент, мы имеем ссылку на весь список

Элементы связанного списка еще называют узлами


  ```

Односвязный список

- class ListItem<T> {
  private T data;
  private ListItem<T> next;

  public ListItem() {}
  public ListItem(T data) { this.data = data; }
  public ListItem(T data, ListItem<T> next) {
  this.data = data;
  this.next = next;
  }

  public T getData() { return data; }
  public void setData(T data) { this.data = data; }
  public ListItem<T> getNext() { return next; }
  public void setNext(ListItem<T> next) { this.next = next; }
  }

Чтобы влезало в экран, тела методов записаны в одну строку, так делать не надо

Пустой конструктор только для наглядности примеров в следующих слайдах

Односвязный список

```
class SinglyLinkedList<T> {
    private ListItem<T> head;
    private int count;

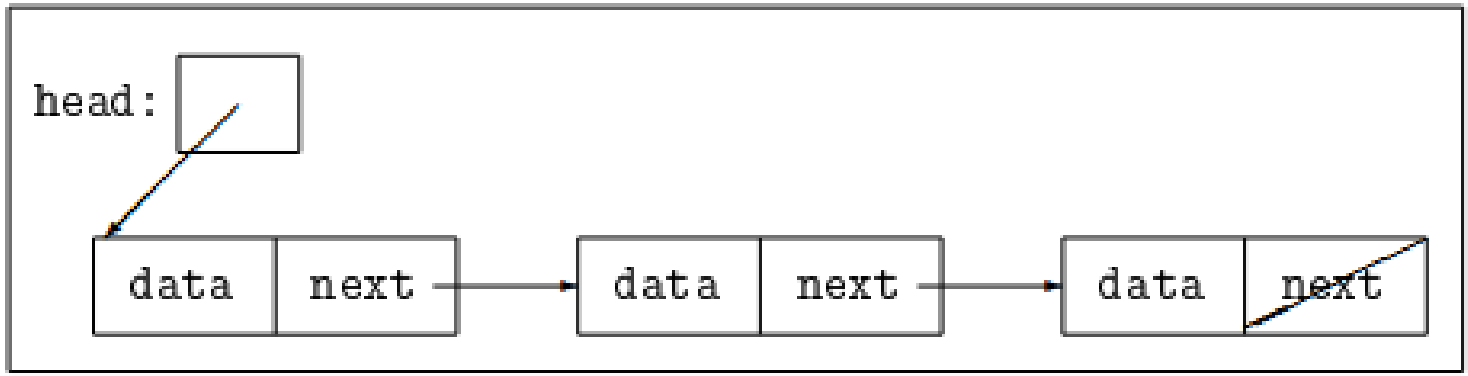
    public SinglyLinkedList() {}

    // различные методы вставки, удаления и т.д.
}
```

- Тут еще храним в поле **count** длину списка, чтобы не вычислять ее каждый раз заново

Односвязный список

- Этот список называется **односвязным**, потому что каждый узел хранит одну ссылку на следующий элемент
- Бывают **двусвязные списки**, когда каждый элемент ещё хранит ссылку на предыдущий элемент



Зачем нужны связанные списки?

- У массива длина ограничена, а список можно расширять сколько угодно
- У связанного списка лучше временные оценки по вставке/удалению из середины и начала, чем у массива и списка на массиве
- В отличие от списка на массиве не требуется сплошной кусок памяти

Вставка в начало списка

- Допустим, у нас есть переменная **head**, которая указывает на начало списка (первый элемент)

- Алгоритм вставки в начало списка:
  - Создать новый элемент списка, пусть на него ссылается ссылка **p**
  - Заполнить ему поле **data**
  - Присвоить полю **next** текущее значение **head**
  - Присвоить **head** значение **p**

Вставка в начало списка

- ListItem<T> p = new ListItem<>(3, head);
  head = p;

Удаление первого элемента

- Допустим, у нас есть переменная **head**, которая указывает на начало списка (первый элемент)

- Алгоритм удаления из списка:
  - В переменную **p** сохранить значение **head.getNext()**
  - (Для языков, где можно управлять памятью) Освободить память из-под переменной **head**
  - **head** присвоить **p**

Удаление первого элемента

- Предположим, что надо было бы освобождать память:
- ListItem<Integer> p = head.getNext();
  free(head); // псевдокоманда освобождения памяти
  head = p;

- А на Java все проще:
- head = head.getNext();

Проход по списку

- Удобно использовать цикл for:
- ```
  for (ListItem<Integer> p = head; p != null; p = p.getNext()) {
      System.out.println(p.getData());
  }
  ```
  ```

Задачи

- Допустим, у нас есть ссылка **p** на некоторый элемент списка (не первый и не последний)
- Как вставить элемент после этого элемента **p**?
- Как удалить элемент, который идет после **p**?

Проход по списку двумя ссылками

- Частая задача – вставить элемент перед элементом, для которого выполняется некоторое условие
- Для этого нам понадобится ссылка на предыдущий элемент от нужного нам
- Удобнее всего для этой цели сделать такой проход с двумя счетчиками:
- ```
  for (ListItem<Integer> p = head, prev = null;
        p != null;
        prev = p, p = p.getNext()) {
     System.out.println(p.getData());
  }
  ```
  ```

Задача List

- См. файл