Список на основе узлов

Определение

```
package l29.ex1;

class Node {
   int data;    // Данные, хранимые в узле
   Node next;  // Ссылка на следующий узел в списке

   public Node(int data) {
       this.data = data;
       // По умолчанию следующий узел не задан
       this.next = null;
   }
}

class Main {
   public static void main(String[] args) {
       Node N1 = new Node(67);
       Node N2 = new Node(54);
       Node N3 = new Node(12);

       N1.next = N2;
       N2.next = N3;
   }
}
```

Список на основе узлов - это структура данных, где каждый элемент (узел) содержит данные и одну или несколько ссылок на следующие и/или предыдущие элементы в списке.

Один из самых распространенных типов - односвязный список, где каждый узел ссылается только на следующий узел.

В двусвязном списке каждый узел содержит ссылки как на следующий, так и на предыдущий узлы.

Двухсвязный список

```
package l29.ex2;


class DoubleNode {
   int data;
   DoubleNode prev;  // Ссылка на предыдущий узел
   DoubleNode next;  // Ссылка на следующий узел

   public DoubleNode(int data) {
       this.data = data;
       // По умолчанию предыдущий узел не задан
       this.prev = null;
       // По умолчанию следующий узел не задан
       this.next = null;
   }
}
```

Двусвязный список позволяет легко добавлять и удалять узлы как из начала, так и из конца списка, а также обеспечивает быстрый доступ к предшествующим элементам, что не всегда возможно в односвязных списках.

Двухсвязный список

```
package l29.ex2;


class DoublyLinkedList {
    DoubleNode head;  // Начало списка
    DoubleNode tail;  // Конец списка

    public DoublyLinkedList() {
        this.head = null;
        this.tail = null;
    }

    // Методы для добавления,
    // удаления и других операций...
}
```

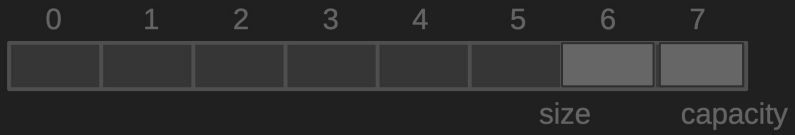
Двусвязный список позволяет легко добавлять и удалять узлы как из начала, так и из конца списка, а также обеспечивает быстрый доступ к предшествующим элементам, что не всегда возможно в односвязных списках.

Списки на основе узлов предоставляют гибкую и динамическую структуру данных, подходящую для множества сценариев использования, где требуется частое добавление и удаление элементов.

Односвязные и двусвязные списки - это два основных типа, каждый со своими преимуществами. Двусвязные списки особенно полезны, когда необходим быстрый доступ как к следующим, так и к предыдущим элементам списка.

ArrayList vs LinkedList

ArrayList и двусвязные списки (LinkedList) в Java оба реализуют интерфейс List, но имеют различные особенности и подходят для разных задач.



Подходит для ситуаций с частым доступом к элементам по индексу и редкими вставками/удалениями.

ArrayList основан на обычном массиве. Это позволяет ArrayList эффективно работать с индексами, так как доступ к элементу по индексу происходит за константное время O(1).

Вставка и удаление элементов в ArrayList, могут быть медленными (O(n)), так как может потребоваться сдвиг оставшейся части массива.

Расширение происходит полным копированием, т.е. при исчерпании свободных ячеек ArrayList для добавления новых эл-тов создает новый массив и копирует в него старые данные.

ArrayList может потреблять больше памяти, чем нужно для хранения элементов, из-за того, что размер массива увеличивается с некоторым запасом для минимизации количества операций изменения размера.

ArrayList vs LinkedList

Подходит для ситуаций когда где преобладают операции вставки и удаления.

LinkedList состоит из узлов, каждый из которых содержит данные и ссылки на предыдущий и следующий элементы в списке. Это обеспечивает эффективное добавление и удаление элементов, так как не требует сдвига элементов.

Вставка и удаление элементов в LinkedList в лучшем случае происходят за O(1), если у нас уже есть ссылка на узел, перед которым или после которого происходит операция. В противном случае, поиск позиции для вставки или удаления будет стоить **O(n) - в худшем случае.**

Доступ к элементам в LinkedList требует прохождения от начала или конца списка до нужного элемента, что делает операцию **доступа к элементу по индексу затратной по времени O(n).**

Каждый элемент в LinkedList требует дополнительной памяти для хранения ссылок на предыдущий и следующий элементы, что делает **LinkedList более затратным по памяти по сравнению с ArrayList для одних и тех же данных.**

ArrayList - использование

```java
package l29.arraylist;

import java.util.ArrayList;  // Импорт класса ArrayList

class ArrayListExample {
   public static void  main(String[] args) {
       // Создание экземпляра ArrayList для хранения строк
       ArrayList<String> fruits =  new ArrayList<>();

       // Добавление элементов в список
       fruits.add("Apple");
       fruits.add("Banana");
       fruits.add("Cherry");
       fruits.add("Date");
       // Доступ к элементу по индексу
       // (например, получение второго элемента, индекс 1)
       System.out.println(
           "Element at index 1: "  + fruits.get(1)
       ); // Выводит "Banana"

       // Изменение элемента (замена элемента на
       // индексе 2 на "Blackberry")
       fruits.set(2, "Blackberry");

       // Удаление элемента по индексу
       // (удаление элемента на индексе 3)
       fruits.remove(3);

       // Удаление элемента по объекту (удаление "Apple")
       fruits.remove("Apple");

       // Удаление элемента по объекту (удаление "Apple")
       fruits.remove("Apple");

       // Перебор элементов с использованием
       // улучшенного цикла for
       System.out.println("Current list:");
       for (String fruit : fruits) {
           // Выводит оставшиеся элементы списка
           System.out.println(fruit);
       }

       // Проверка наличия элемента в списке
       if (fruits.contains("Blackberry")) {
           System.out.println("Blackberry is in the list");
       }

       // Получение размера списка (количество элементов)
       System.out.println("Size of the list: "  +
fruits.size());

       // Очистка списка
       fruits.clear();
       System.out.println(
           "List cleared. Size of the list: "  + fruits.size()
       ); // Выводит 0
   }
}
```

LinkedList - использование

```java
package l29.linkedlist;

import java.util.LinkedList;

class LinkedListExample {
   public static void  main(String[] args) {
       // Создание экземпляра LinkedList
       LinkedList<String> names =  new LinkedList<>();

       // Добавление элементов
       names.add("Alice");
       names.add("Bob");
       // Добавление в конец списка
       names.addLast("Charlie");
       // Добавление в начало списка
       names.addFirst("Diana");

       // Доступ к элементам
       // Получение первого элемента
       String firstElement = names.getFirst();
       // Получение последнего элемента
       String lastElement = names.getLast();

       System.out.println(
           "First: " + firstElement + ", Last: " + lastElement
       );

       // Удаление элементов
       names.removeFirst();  // Удаление первого элемента
       names.removeLast();   // Удаление последнего элемента

       // Перебор элементов с использованием
       // улучшенного цикла for
       for (String name : names) {
           System.out.println(name);
       }

       // Добавление элемента по индексу
       names.add(1, "Eve");

       // Удаление элемента по индексу
       names.remove(1);

       // Проверка на наличие элемента
       if (names.contains("Alice")) {
           System.out.println("Alice is in the list");
       }

       // Получение размера списка
       int size = names.size();
       System.out.println("Size of the list: " + size);

       // Очистка списка
       names.clear();
   }
}
```

Домашнее задание

Задачи минимум:

- сделать домашку 28
- реализуйте ко вторнику свой собственный LinkedList (в примерах кода это файл ex2/Main.java)

The end