



Lesson 9

08.01.2024

```
public class Ex1 {  
    int x = 3;  
  
    public static void main(String[] args) {  
        new Ex1().go1();  
    }  
  
    void go1() {  
        int x;  
        go2(++x);  
    }  
  
    void go2(int y) {  
        int x = ++y;  
        System.out.println(x);  
    }  
}
```

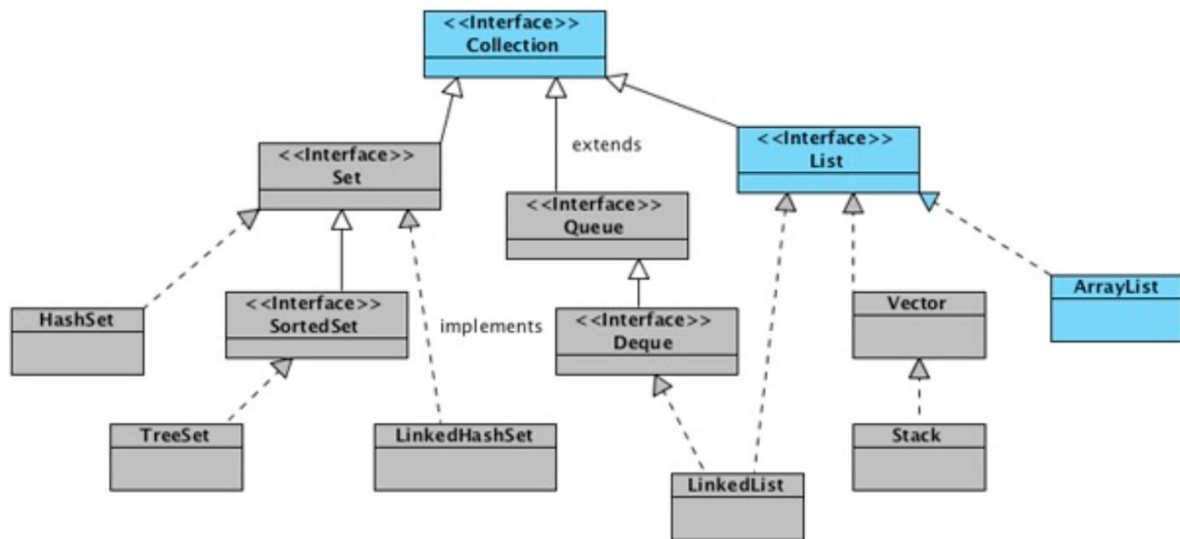
```
public class Ex2 {  
    public static void main(String[] args) {  
        byte[][] bytes = {{1,2,3,4,5}, {1,3,3,4,5,6}};  
        System.out.println(bytes[1].length + " " + bytes.length);  
    }  
}
```

```
public class Ex3 {  
    public static void main(String[] args) {  
        Days d1 = Days.TH;  
        Days d2 = Days.M;  
  
        for (Days d : Days.values()) {  
            if (d.equals(Days.F)) break;  
            d2 = d;  
        }  
        System.out.println((d1 == d2) ? "same" : "not same");  
    }  
    enum Days {M, T, W, TH, F, SA, S}  
}
```

```
public class Ex4 {  
    public static void main(String[] args) {  
        String s = "Hello";  
        String t = new String(s);  
  
        if ("Hello".equals(s)) System.out.println(1);  
        if (t == s) System.out.println(2);  
        if (t.equals(s)) System.out.println(3);  
        if ("Hello" == s) System.out.println(4);  
        if ("Hello" == t) System.out.println(5);  
    }  
}
```

```
public class Ex5 {  
    Ex5() {}  
    Ex5(Ex5 ex) {ex5 = ex;}  
    Ex5 ex5;  
  
    public static void main(String[] args) {  
        Ex5 ex5_1 = new Ex5();  
        Ex5 ex5_2 = new Ex5(ex5_1);        ex5_2.go();  
        Ex5 ex5_3 = ex5_2.ex5;             ex5_3.go();  
        Ex5 ex5_4 = ex5_1.ex5;             ex5_4.go();  
    }  
  
    void go() { System.out.println("hi");}  
}
```

ArrayList



ArrayList – реалізує інтерфейс `List`. Як відомо, Java масиви мають фіксовану довжину, і після того як масив створений, він не може зростати або зменшуватися. `ArrayList` може змінювати свій розмір під час виконання програми, не обов'язково вказувати розмірність під час створення об'єкта. Елементи `ArrayList` можуть бути абсолютно будь-яких типів навіть `null`.

Створення об'єкту

```
ArrayList<String> list = new ArrayList<String>();
```

Щойно створений об'єкт `list` містить властивості **`elementData`** і **`size`**.

```
elementData = (E[]) new Object[10];
```

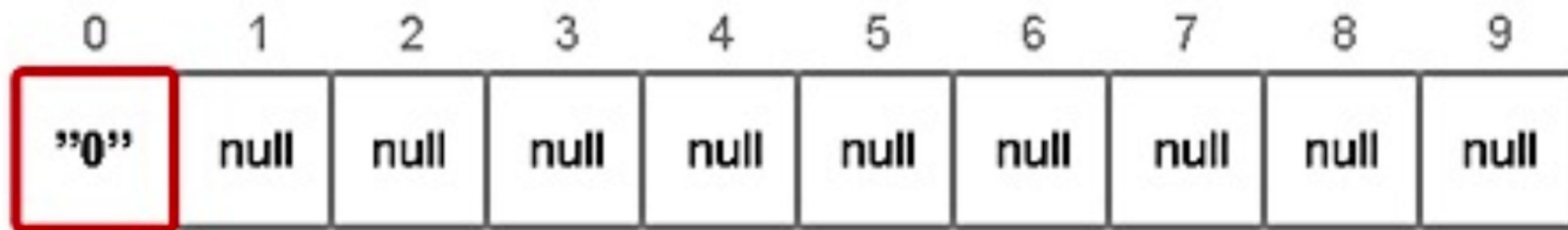
0	1	2	3	4	5	6	7	8	9
null	null	null	null	null	null	null	null	null	null

Ви можете використовувати конструктор **`ArrayList(capacity)`** та вказати свою початкову ємність списку.



Додавання елементів

```
list.add("0");
```



1) перевіряється, чи достатньо місця у масиві для вставки нового елемента;

```
ensureCapacity(size + 1);
```

2) додається елемент в кінець (відповідно до значення size) масиву

```
elementData[size++] = element;
```

ensureCapacity(minCapacity)

Якщо місця в масиві замало, нова ємність розраховується за формулою $(oldCapacity * 3) / 2 + 1$. Другий момент це копіювання елементів. Воно здійснюється за допомогою нативного методу `System.arraycopy()`, який написаний не на Java

```
// newCapacity - новое значение емкости
```

```
elementData = (E[])new Object[newCapacity];
```

```
// oldData - временное хранилище текущего массива с данными
```

```
System.arraycopy(oldData, 0, elementData, 0, size);
```



Додавання до «середини» списку

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"10"	"11"	"12"	"13"	"14"	null

```
list.add(5, "100");
```

1) перевіряється, чи достатньо місця у масиві для вставки нового елемента;

```
ensureCapacity(size+1);
```

2) готується місце нового елемента за допомогою System.arraycopy();

```
System.arraycopy(elementData, index, elementData, index + 1, size - index);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
"0"	"1"	"2"	"3"	"4"	"5"	"5"	"6"	"7"	"8"	"9"	"10"	"11"	"12"	"13"	"14"

3) перезаписується значення у елемента із зазначеним індексом.

```
elementData[index] = element;  
size++;
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
"0"	"1"	"2"	"3"	"4"	"100"	"5"	"6"	"7"	"8"	"9"	"10"	"11"	"12"	"13"	"14"

Як можна здогадатися, у випадках, коли відбувається вставка елемента за індексом i при цьому у вашому масиві немає вільних місць, виклик `System.arraycopy()` трапиться двічі: перший в `ensureCapacity()`, другий в самому методі `add(index, value)`, що явно позначиться на швидкості всієї операції додавання

Видалення елементів

- За індексом **remove (index)**
- За значенням **remove (value)**

```
list.remove(5);
```

1) Спочатку визначається яка кількість елементів треба скопіювати

```
int numMoved = size - index - 1;
```


2) Потім копіюємо елементи за допомогою System.arraycopy()

```
System.arraycopy(elementData, index + 1, elementData, index, numMoved);
```

3) Зменшуємо розмір масиву та забуваємо про останній елемент

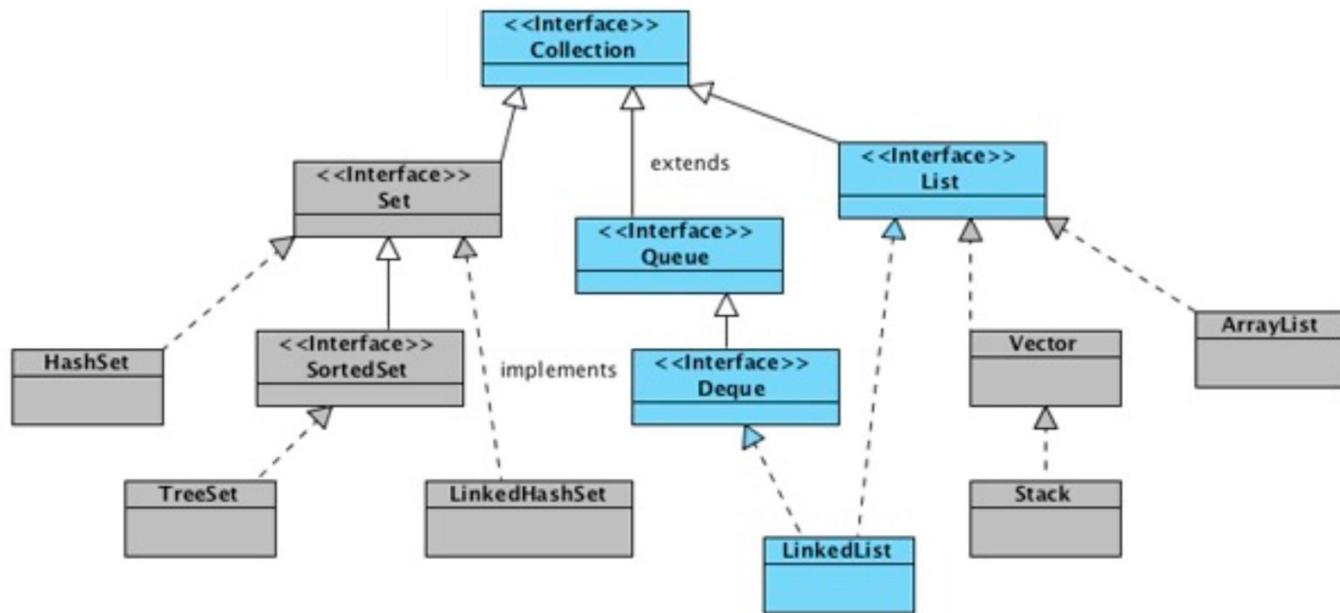
```
elementData[--size] = null; // Let gc do its work
```

При видаленні за значенням, у циклі проглядаються всі елементи списку, до тих пір поки не буде знайдено відповідність. Видалений буде лише перший знайдений елемент.

- 
- Швидкий доступ до елементів за індексом за час $O(1)$;
 - Доступ до елементів за значенням за лінійний час $O(n)$;
 - Повільний, коли вставляються та видаляються елементи із «середини» списку;
 - Дозволяє зберігати будь-які значення в тому числі і null;
 - Не синхронізовано



LinkedList



LinkedList - реалізує інтерфейс List. Є представником двонаправленого списку, де кожен елемент структури містить показники попередній і наступний елементи. Ітератор підтримує обхід по обидва боки. Реалізує методи отримання, видалення та вставки в початок, середину та кінець списку. Дозволяє додавати будь-які елементи у тому числі null



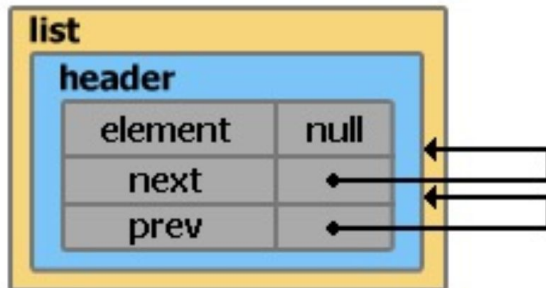
Створення об'єкту

```
List<String> list = new LinkedList<String>();
```

Щойно створений об'єкт `list` містить властивості **header** і **size**.

header - псевдо-елемент списку. Його значення завжди дорівнює `null`, а властивості **next** і **prev** завжди вказують перший і останній елемент списку відповідно. Так як на даний момент список ще порожній, властивості **next** і **prev** вказують на себе (тобто на елемент `header`). Розмір списку **size** дорівнює 0.

```
header.next = header.prev = header;
```





Додавання елементів

```
list.add("0");
```

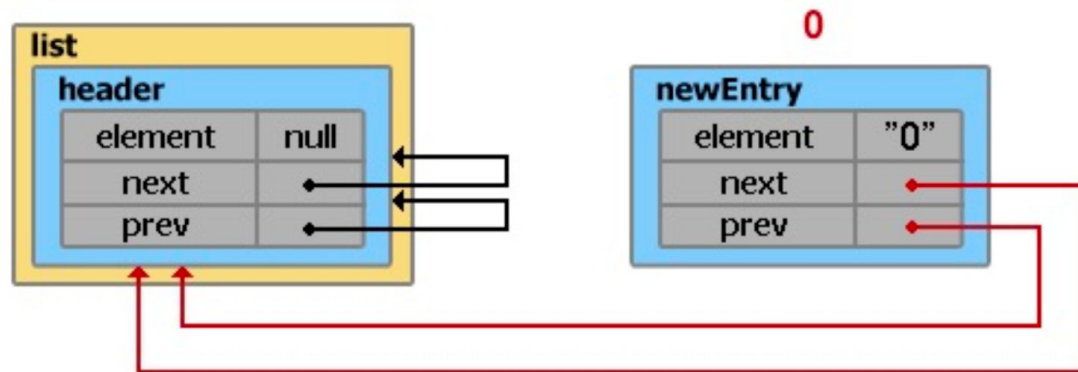
Щоразу при додаванні нового елемента, по суті виконується два кроки:

1) створюється новий екземпляр класу Entry

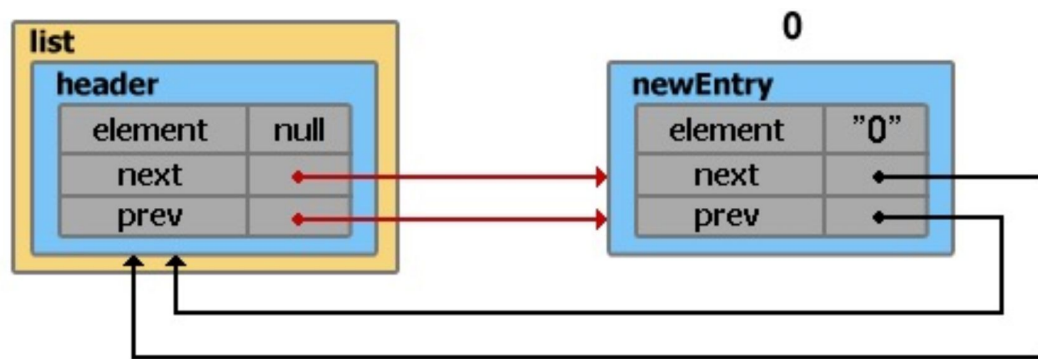
```
private static class Entry<E>
{
    E element;
    Entry<E> next;
    Entry<E> prev;

    Entry(E element, Entry<E> next, Entry<E> prev)
    {
        this.element = element;
        this.next = next;
        this.prev = prev;
    }
}
```

```
Entry newEntry = new Entry("0", header, header.prev);
```



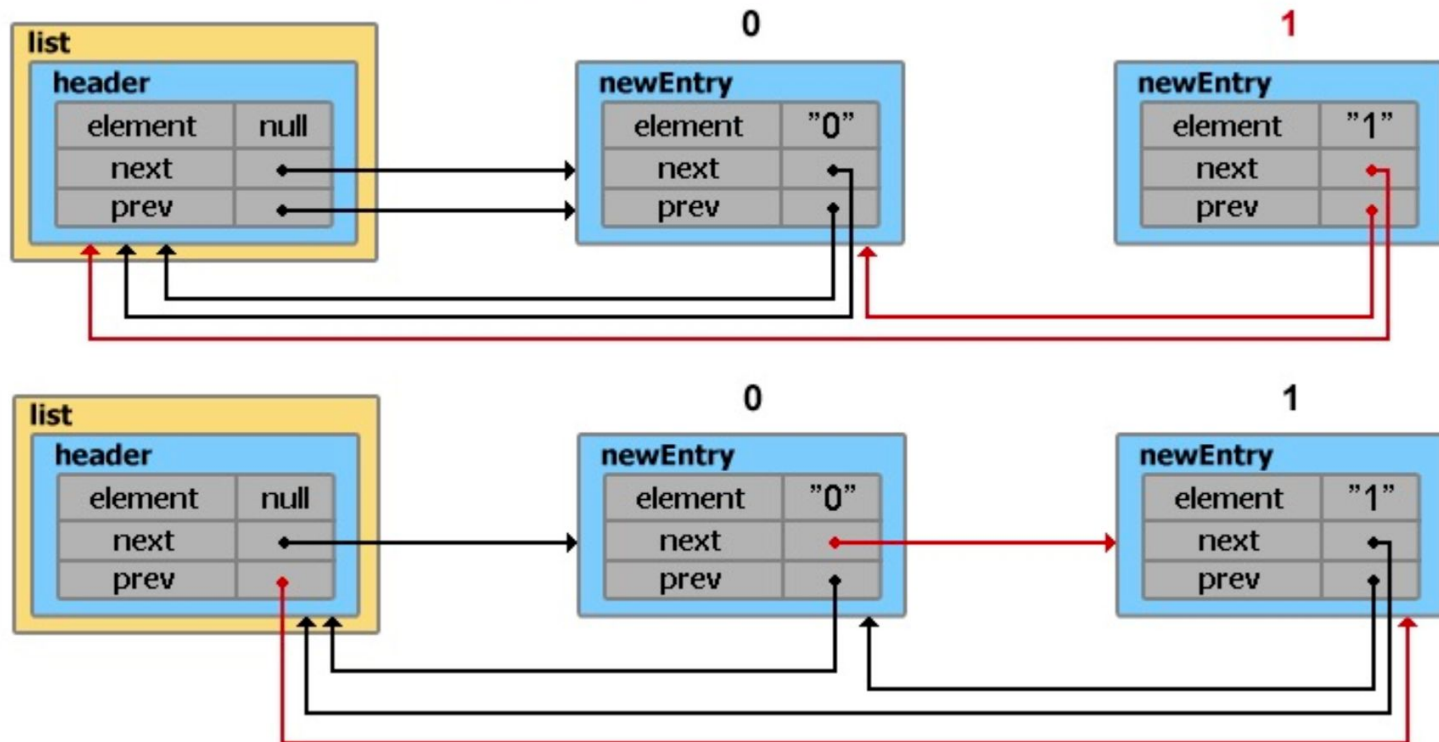
2) перевизначаються покажчики на попередній та наступний елемент



```
list.add("1");
```

// header.prev указывает на элемент с индексом 0

```
Entry newEntry = new Entry("1", header, header.prev);
```

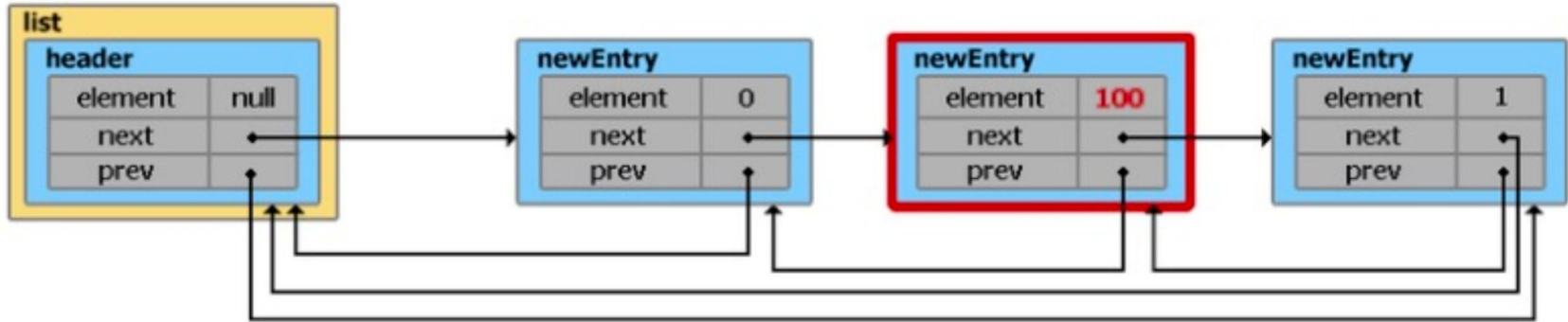


Видалення елементів

- З початку або кінця списку за допомогою `removeFirst()`, `removeLast()` за час $O(1)$;
- За індексом `remove(index)` і за значенням `remove(value)` за час $O(n)$.

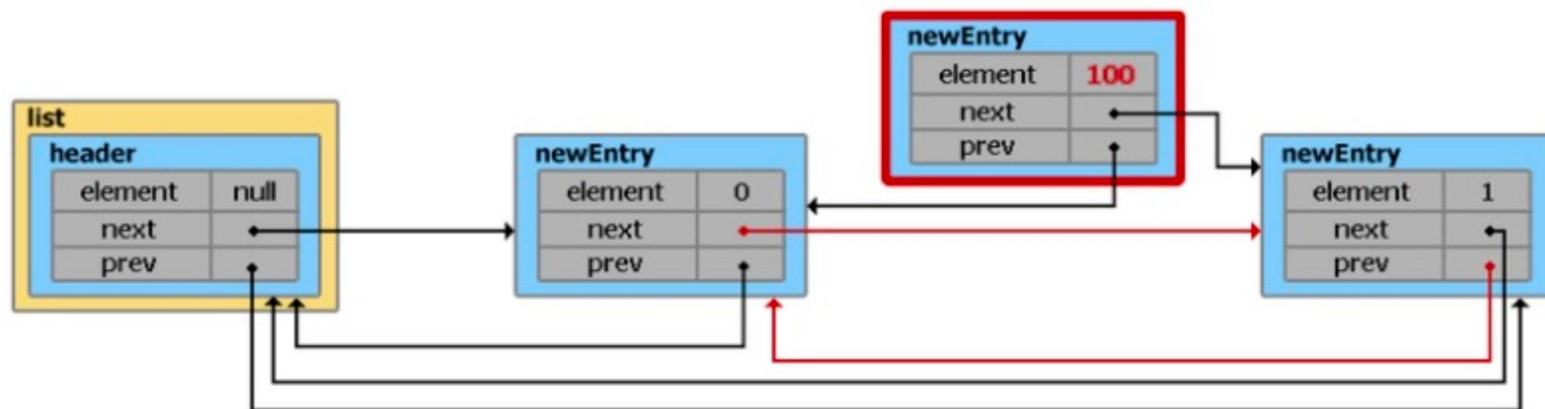
```
list.remove("100");
```

1) пошук першого елемента із відповідним значенням



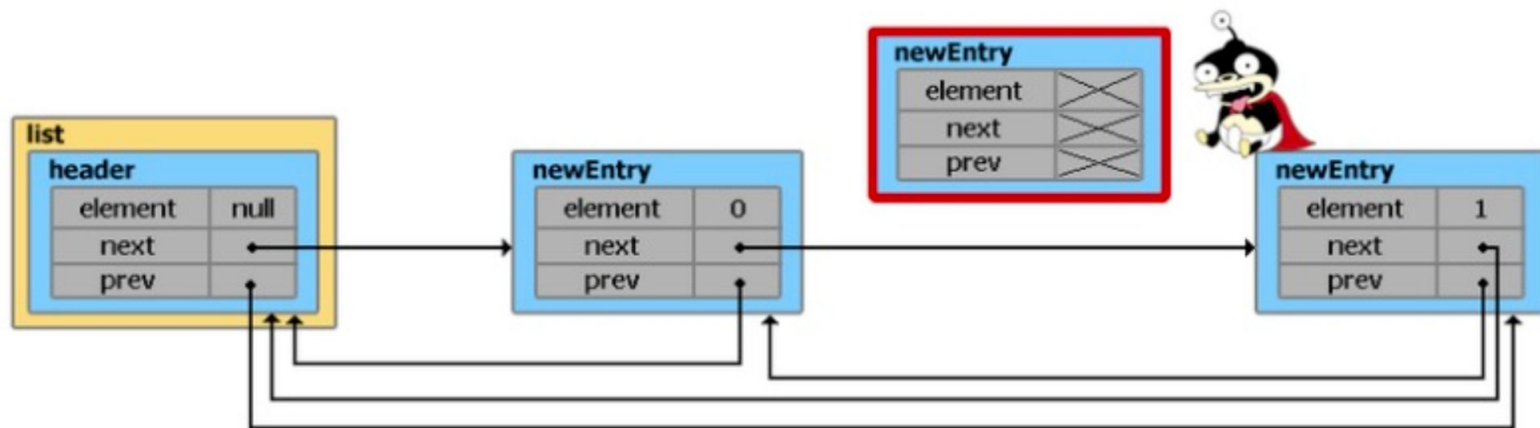
2) перевизначаються показники на попередній та наступний елемент


```
// Значение удаляемого элемента сохраняется  
// для того чтобы в конце метода вернуть его  
E result = e.element;  
e.prev.next = e.next;  
e.next.prev = e.prev;
```



3) видалення вказівників на інші елементи та передання забуттю самого елемента

```
e.next = e.prev = null;  
e.element = null;  
size--;
```

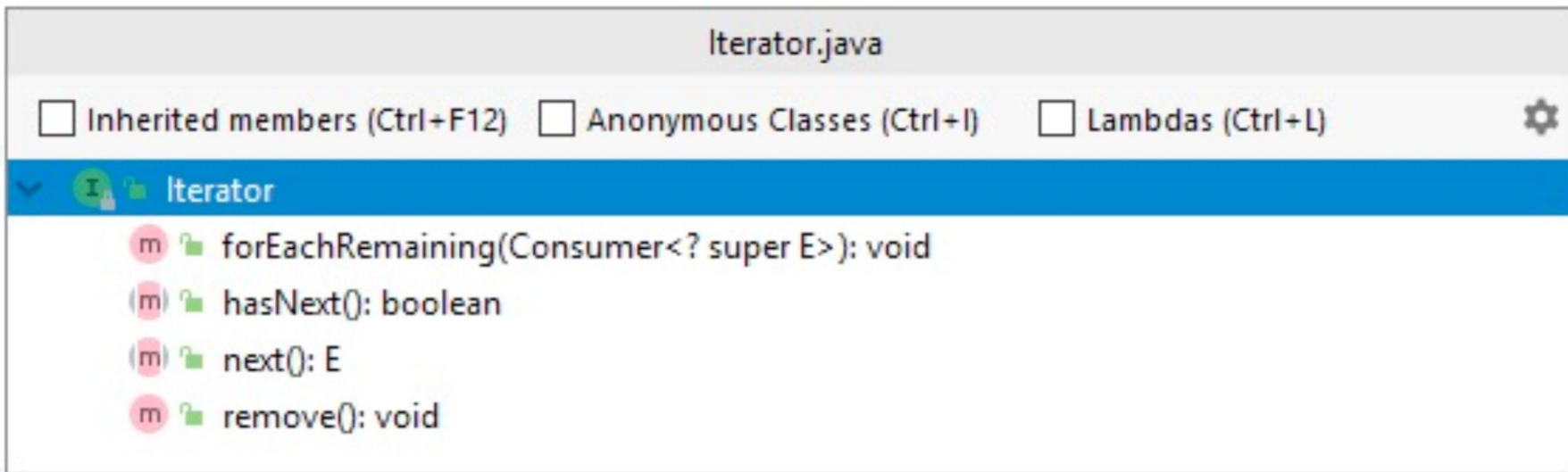


- 
- З LinkedList можна організувати стек, чергу, або подвійну чергу, з часом доступу $O(1)$;
 - На вставку та видалення з середини списку, отримання елемента за індексом або значенням буде потрібно лінійний час $O(n)$. Однак, на додавання і видалення з середини списку, використовуючи `ListIterator.add()` і `ListIterator.remove()`, буде потрібно $O(1)$;
 - Дозволяє додавати будь-які значення навіть `null`. Для зберігання примітивних типів використовує відповідні класи-обгортки;
 - Не синхронізовано



Iterator

Ітератор здатний по чергово обійти всі елементи колекції. При цьому він дозволяє це зробити без вникання у внутрішню структуру та влаштування колекції.



Методи, які має імплементувати Iterator:

boolean hasNext() — якщо в об'єкті, що ітерується (поки що це Collection) залишилися ще значення — метод поверне true, якщо значення скінчилися false.

E next () - Повертає наступний елемент колекції (об'єкта). Якщо елементів більше немає (не було перевірки hasNext()), а ми викликали next(), досягнувши кінця колекції), метод кине NoSuchElementException.

void remove() — видалить елемент, який був отриманий востаннє методом next(). Метод може кинути: UnsupportedOperationException, якщо даний ітератор не підтримує метод remove() (у випадку з read-only колекціями, наприклад) IllegalStateException, якщо метод next() ще не був викликаний, або якщо remove() вже був викликаний після останнього виклику next ()