СПИСЪЦИ

ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

- Преглед
- Garbage Collector
- Операции със списъци
- Примери

СВЪРЗАНИ СПИСЪЦИ: ЗАДАЧА

Основна задача на много програми:

Съхраняване и търсене на информация

Представяне: Array, List, Tree ...

Arrays:

```
int[] a = new int[n];
```

Броят на елементите n често е ясен едва по време на изпълнението

Но: веднъж определен n не може да бъде променян

Проблеми с Arrays:

→ n голям: прахосване на памет

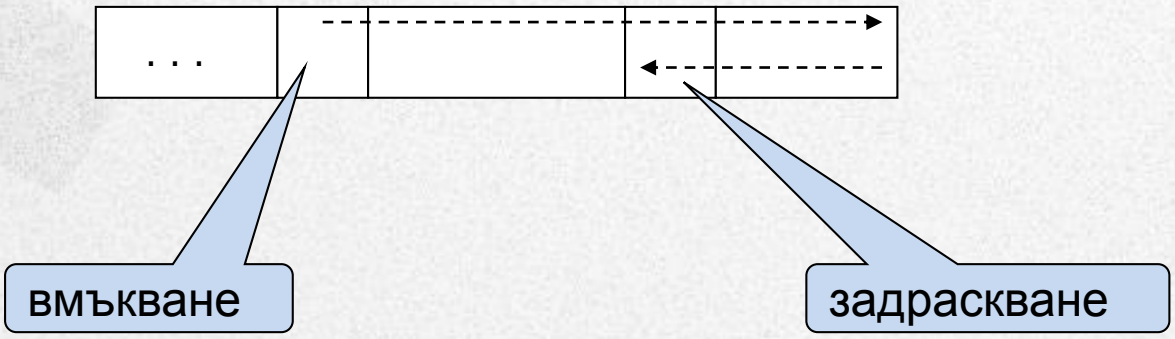
→ n малък: при определени условия проблеми в run-time

СВЪРЗАНИ СПИСЪЦИ: ПРЕДИМСТВА

Свързани списъци: много гъвкави структури от данни

- произволен брой елементи
- ефективно вмъкване, задраскване на елементи

Array:



РАЗЛИКИ: ARRAYS – ARRAY LISTS

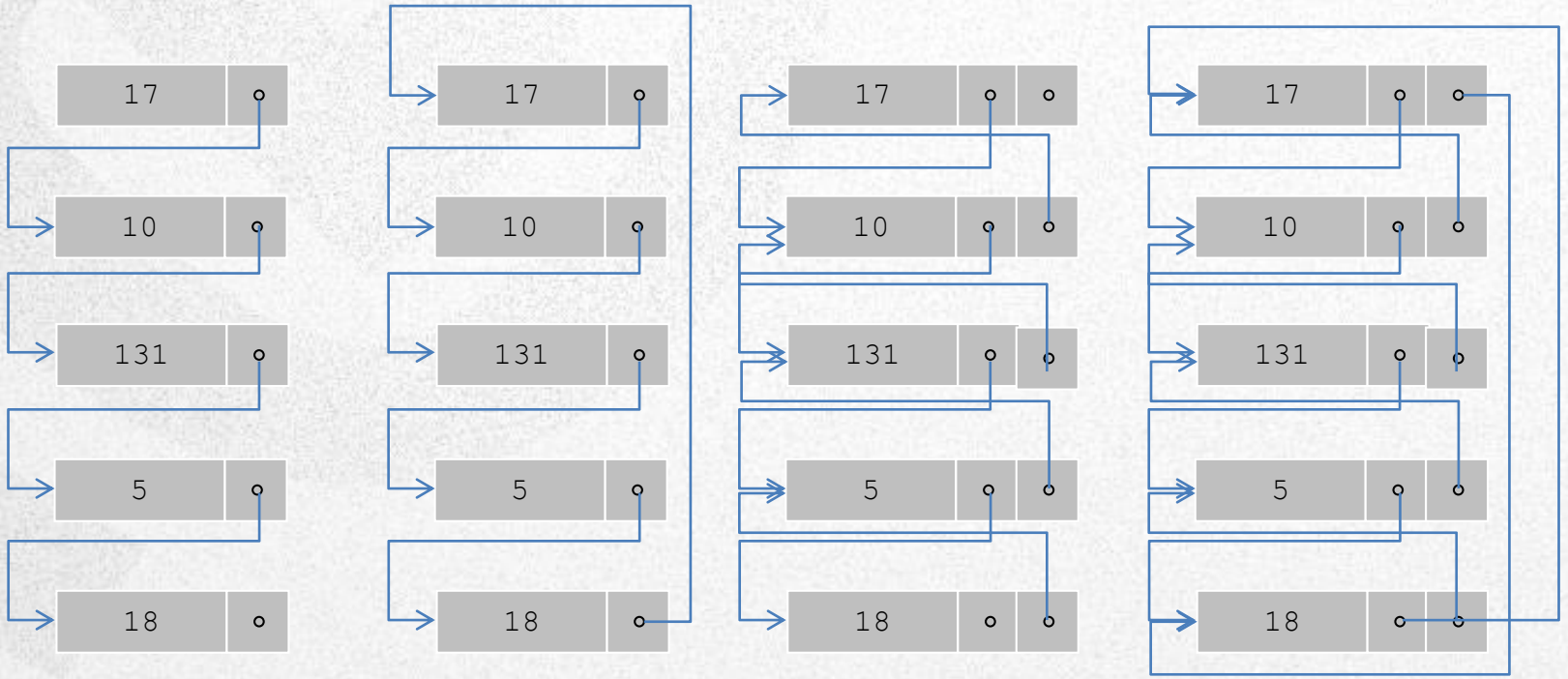
- Array:
  - Фиксирана дължина
  - Елементи от специфициран тип
- Array lists
  - Масив от обектни референции с променлива дължина
  - Може динамично да нараства или намалява
  - Създават се с начален размер
  - Когато се надхвърли размера масивът се разширява автоматично
  - Когато се изтриват обекти, масивът може да се намали

ПРИМЕР ЗА ARRAYLIST

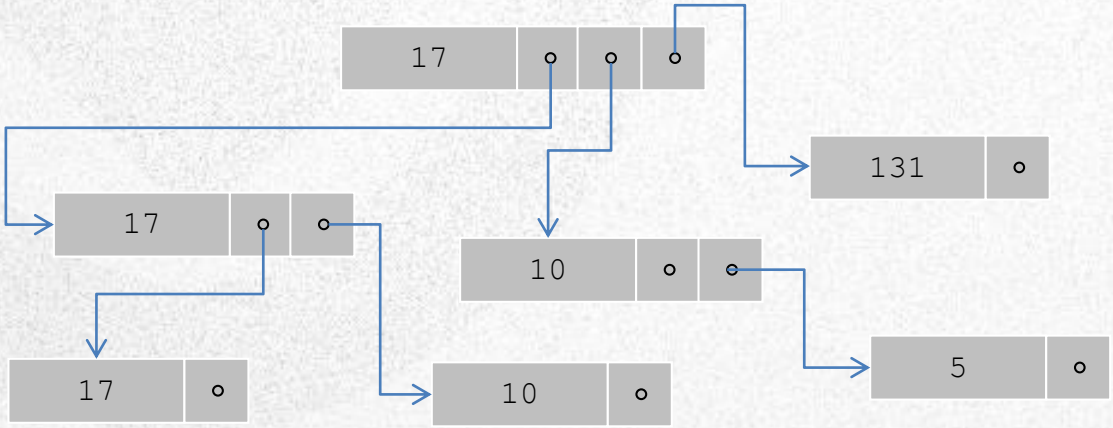
```
// Demonstrate ArrayList.
import java.util.*;
class ArrayListDemo {
        public static void main(String args[]) {
// create an array list
                ArrayList al = new ArrayList();
                System.out.println("Initial size of al: " + al.size());
// add elements to the array list
                al.add("C");
                al.add("A");
                al.add("E");
                al.add("B");
                al.add("D");
                al.add("F");
                al.add(1, "A2");
                System.out.println("Size of al after additions: " + al.size());
// display the array list
                System.out.println("Contents of al: " + al);
// Remove elements from the array list
                al.remove("F");
                al.remove(2);
                System.out.println("Size of al after deletions: " + al.size());
System.out.println("Contents of al: " + al);
}
}
```

```
Initial size of al: 0
Size of al after additions: 7
Contents of al: [C, A2, A, E, B, D, F]
Size of al after deletions: 5
Contents of al: [C, A2, E, B, D]
```

СВЪРЗАНИ И ДВОЙНО СВЪРЗАНИ СПИСЪЦИ



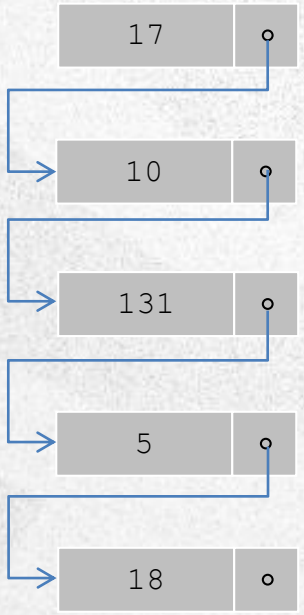
ДЪРВО



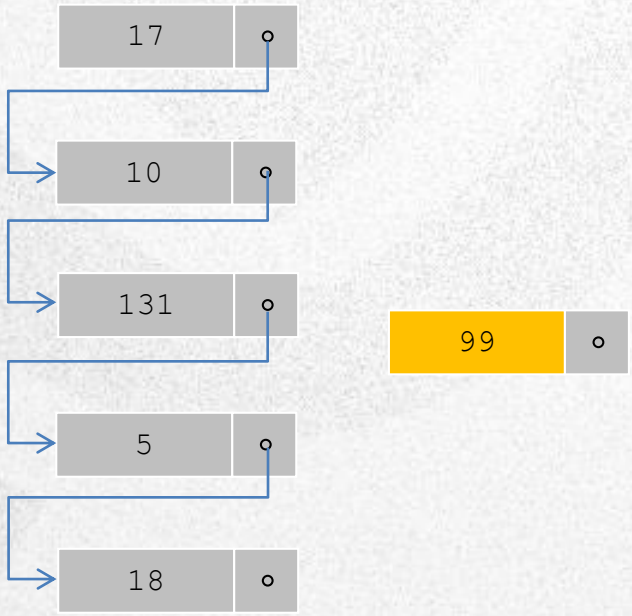
ЕДНОКРАТНО СВЪРЗАНИ СПИСЪЦИ

- Клетка (елемент):
  - Съдържание
  - Остатък (списък): Указател към клетка
- Прикачване на нов елемент
  - Заявяване памет за нова клетка (динамично в run-time)
  - Изграждане указател за тази клетка
- Други оператори
  - Добавяне на нов елемент на произволно място
    - Няма преместване на данни
  - Изтриване на произволен елемент
    - само логически задраскан
    - физически още в паметта:
      - Garbage Collector (Java System)

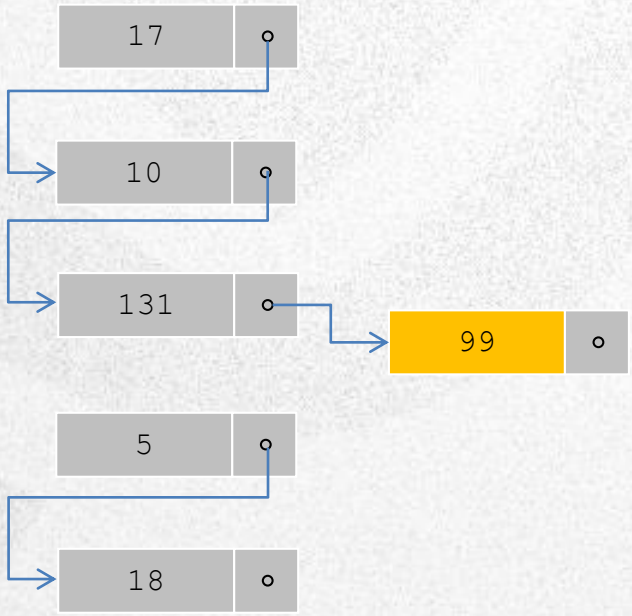
ДОБАВЯНЕ НА НОВ ЕЛЕМЕНТ



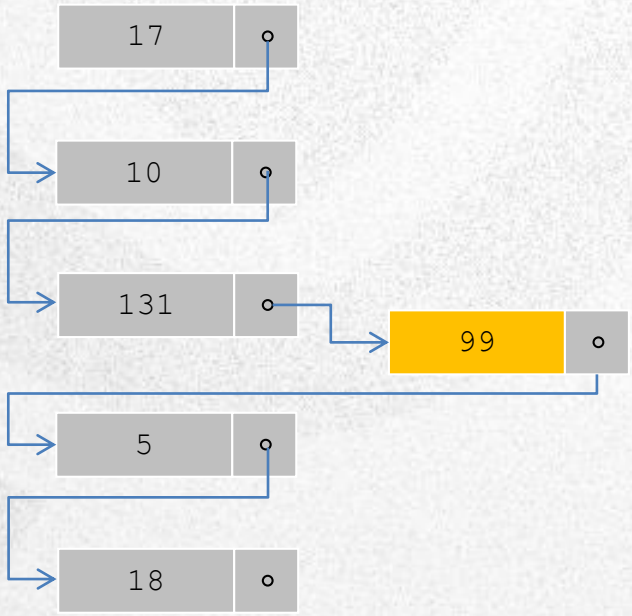
ДОБАВЯНЕ НА НОВ ЕЛЕМЕНТ



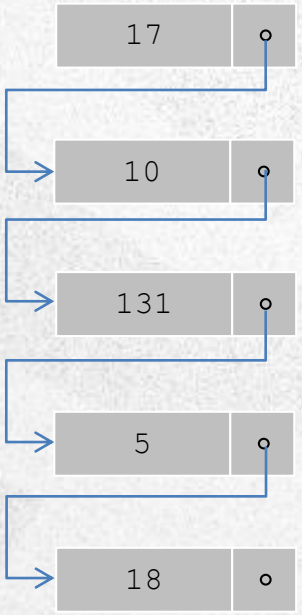
ДОБАВЯНЕ НА НОВ ЕЛЕМЕНТ



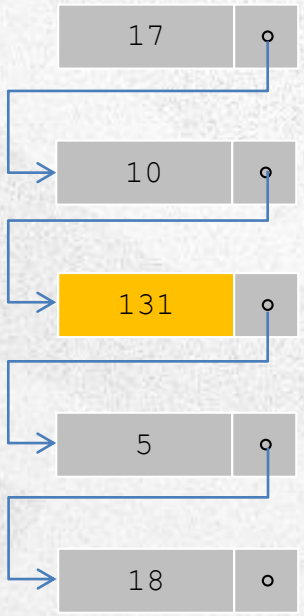
ДОБАВЯНЕ НА НОВ ЕЛЕМЕНТ



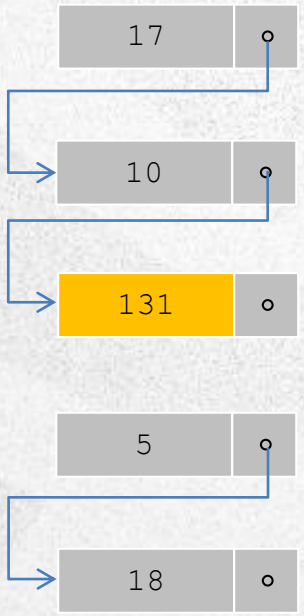
ИЗТРИВАНЕ НА ЕЛЕМЕНТ



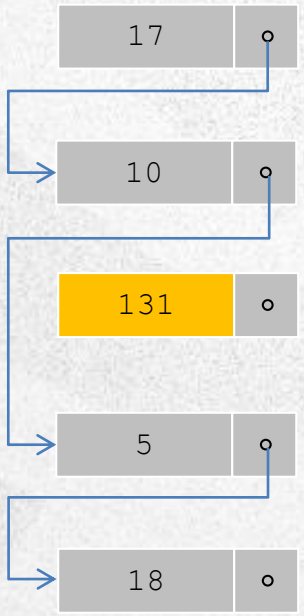
ИЗТРИВАНЕ НА ЕЛЕМЕНТ



ИЗТРИВАНЕ НА ЕЛЕМЕНТ



ИЗТРИВАНЕ НА ЕЛЕМЕНТ



GARBAGE COLLECTOR

Автоматично освобождаване на вече ненужната памет посредством Java-run-time-system *)

*) Клетки памет, които не са достъпни чрез променливите на програмата (директно или посредством указатели)

→ Програмистите сами трябва да се грижат за освобождаването
- C++, Pascal, Modula-2
  (там: явни оператори за освобождаване на памет в програмата)

- Техниката произлиза от функционалните езици: Lisp

GARBAGE COLLECTOR

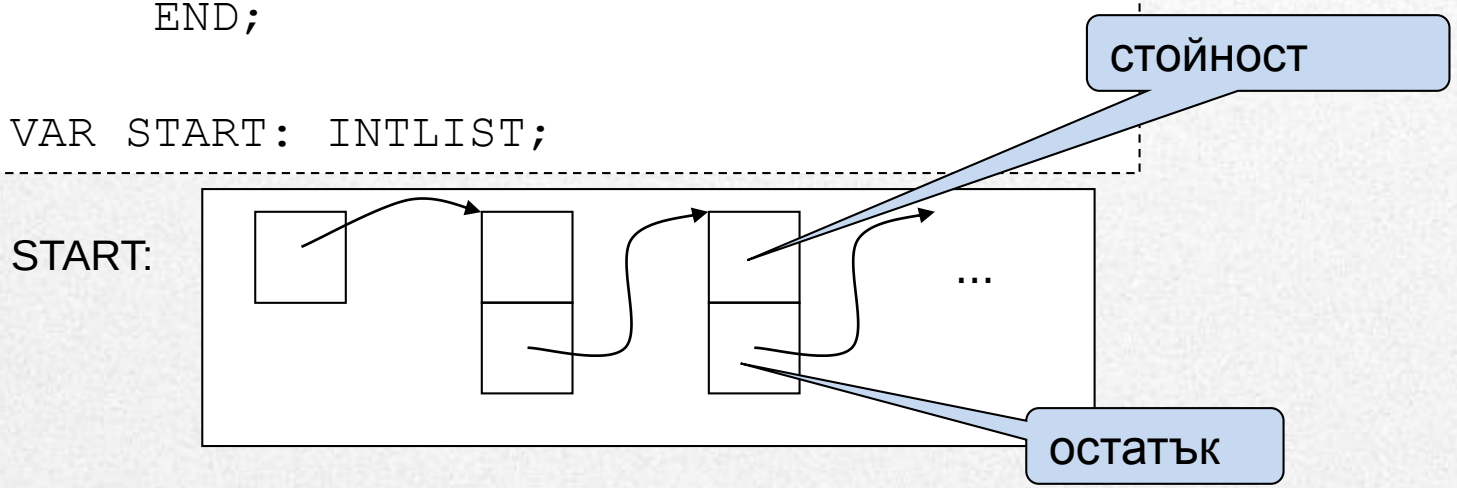
- Различава:
  - Живи обекти – съществуват референции към тях
  - Мъртви обекти (отпадък)– не съществуват
- Модел
  - Използва брояч на референции към обектите (count)
  - Обект X се обърне към обект Y (count++)
  - Обект X освобождава референция към обект Y (count--)
- Активиране
  - Няма възможност за явно (липсва delete)
  - Опасност за изчерпване на пространството, резервирано за програмата
  - Отнема време (неефективен run-time)
- Възможности за дереференциране
  - Промяна на референция към друг обект
  - Референция null
  - Напускане на метод – локалните променливи престават да съществуват

РЕАЛИЗАЦИЯ

Нататък: само еднократно свързани линейни списъци

- Pascal (C, C++): със структури данни
  - Records (Struct) + Pointer typ

```
TYPE INTLIST = @ CELL;
   CELL =
     RECORD
       VALUE: INTEGER;
       REST:  INTLIST
     END;

VAR START: INTLIST;
```



- Java: с класове

ОСНОВЕН ПРИНЦИП

- Възможността за създаване на такива структури се базирана на това, че:
  - Обекти от един клас T могат да съдържат променливи от тип T
  - m реално е рефенция към един обект от тип T
  - Когато създаваме T обект, той може да съдържа референция към друг обект от тип T, който от своя страна съдържа референция към трети обект от тип T и т.н.
  - Тази последователност може да бъде завършена, когато последния обект съдържа референция null

```
class T {
        private T m;
        …
}
```

РЕАЛИЗАЦИЯ В JAVA

IntList.java

```
class IntList  {

    private int value ;
    private IntList rest ;

    public IntList (int v, IntList next) {
        value = v;
        rest = next;
    }
    public int getValue () { return value; }

    public void setValue (int val) {
        value = val;
    }

    public IntList getRest () {
        return rest;
    }
    ...
}
```

Рекурсивна дефиниция:
тип на променливата= дефинирания клас

Както Pascal-Record
(една клетка)

СЪЗДАВАНЕ НА СПИСЪК С КЛАСА 'INTLIST'

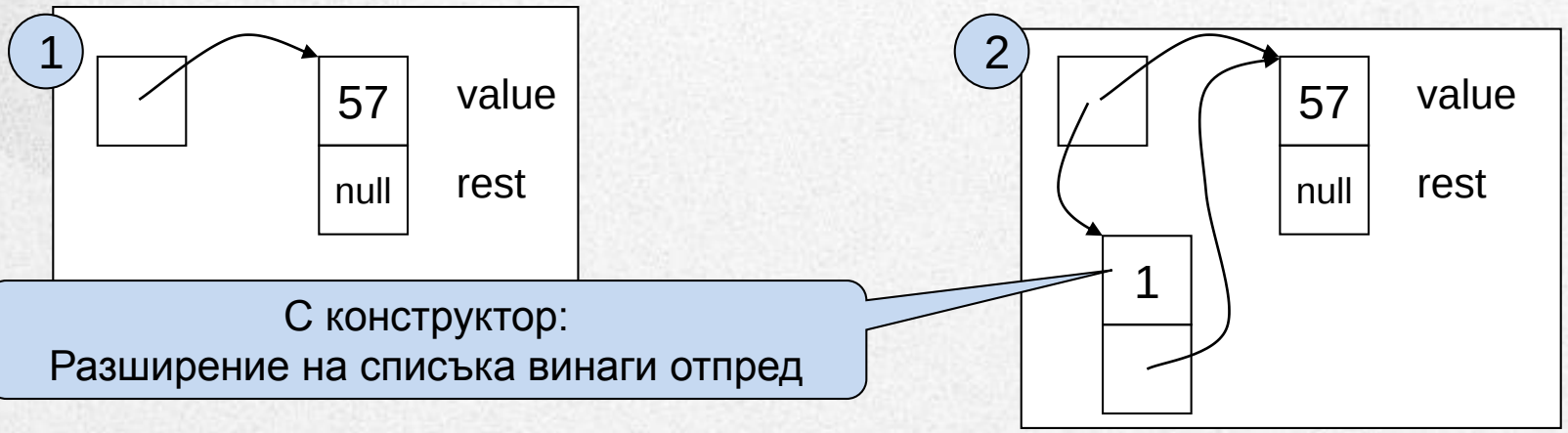
List.java
IntList.java

```
class IntList   {
   private int value ;
   private IntList rest ;
   public IntList (int v, IntList next) {
        value = v;
        rest = next;
   }
```

null Object:
празен указател

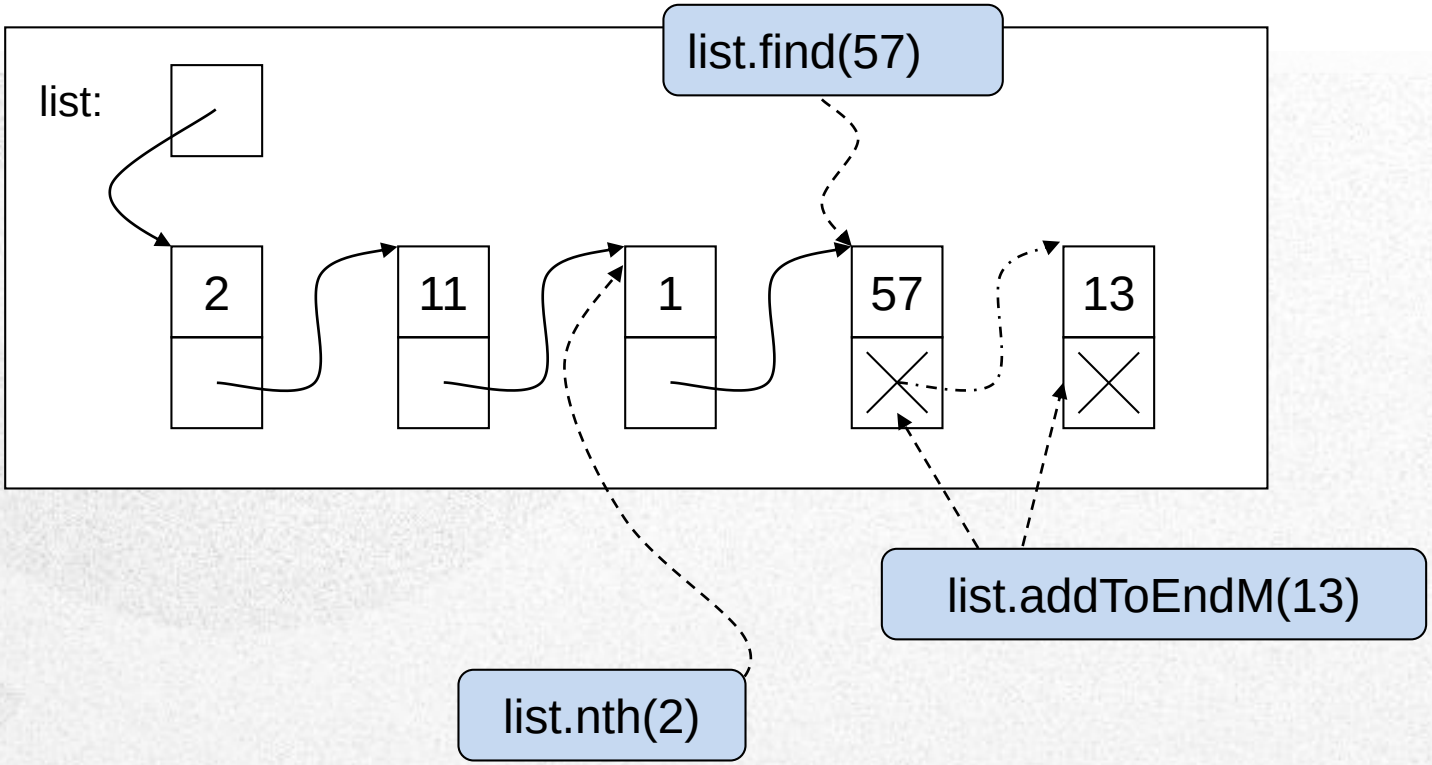
в: List.java:

```
1 Intlist list = new IntList(57, null);

2 list = new IntList(1, list);
```



С конструктор:
Разширение на списъка винаги отпред

СПИСЪЧНИ ФУНКЦИИ: ПРЕГЛЕД



list.length() ----------> 4

list.addInorderM(12) --> само за сортирани списъци

РЕАЛИЗАЦИЯ НА СПИСЪЧНИ ФУНКЦИИ

Списъци като структури данни

- рекурсивна структура данни
  (глава следвана от остатък)
  → рекурсивен алгоритъм естествено

- итеративна структура данни
  (последователност от елементи)
  → също итеративен алгоритъм

Задача: самостоятелна итеративна реализация

ВАЖНИ ФУНКЦИИ ЗА СПИСЪЦИ (1)

IntList.java

```
class IntList   {
   private int value ;
   private IntList rest ;
   public IntList (int v, IntList next)…

   public int length () …
}
```

- Дължина на списъка

```
public int length ()    {
    if (rest == null)
      return 1;
    else
      return 1 + rest.length();
 }
```

ВАЖНИ ФУНКЦИИ ЗА СПИСЪЦИ (2)

- Намери клетка със съдържание 'key'

IntList.java

```
public IntList find (int key)    {
    if (value == key)
        return this;
    else if (rest == null)
        return null;
    else
        return rest.find(key);
}
```

Двоично търсене невъзможно
→ O(n) = ½ n

ВАЖНИ ФУНКЦИИ ЗА СПИСЪЦИ (3)

IntList.java

- Намери указател към n-тата клетка

```
public IntList nth (int n)    {
    if (n == 0)
      return this;
    else if (rest == null)
      return null;
    else
      return rest.nth(n-1);
}
```

ВАЖНИ ФУНКЦИИ ЗА СПИСЪЦИ (4)

- Прикачи нова клетка в края на списъка

IntList.java

```
public void addToEndM (int val)    {
    if (rest != null)
        // a cell in the middle
      rest.addToEndM(val);
    else // the last cell
      rest = new IntList(val, null);
}
```

Свързване на списък на края с нова клетка

ВАЖНИ ФУНКЦИИ ЗА СПИСЪЦИ (5)

- Добави елемент в съответствие с големината в един сортиран списък → изграждане на сортиран списък: само с `addInOrderM()`

IntList.java

```
public IntList addInorderM (int n)    {
    if (n < value)
        return new IntList (n, this);
    else   if (n == value)
        return this;
    else   if (rest == null)    {
        rest = new IntList (n, null);
        return this;
    }
    else   {
        rest = rest.addInorderM(n);
        return this;
    }
}
```

n най-малко число:
Добавя отпред

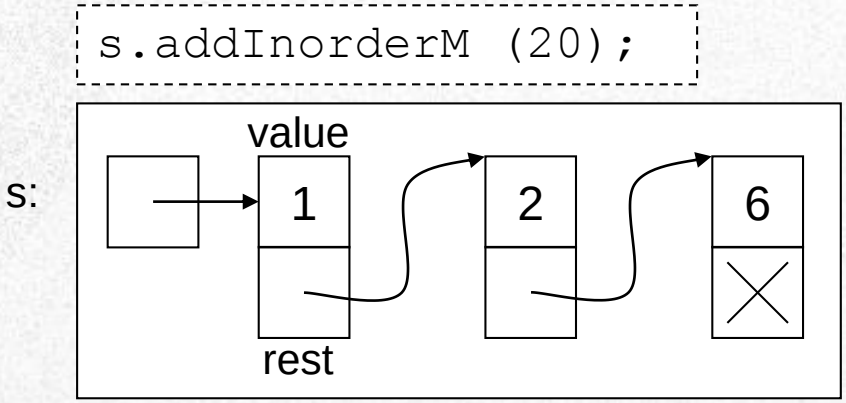
n вече наличен:
Недобавя

n най-голямото число:
Добавя отзад

в други случаи:
Рекурсивно добавя в средата

ДОРАБОТВАНЕ НА РЕАЛИЗАЦИЯ НА СПИСЪЦИ

- Преработване:
  - `IntList.java`
  - `List.java`
- Разбиране:
  - `рисуване на картини`



(3. случай: ново най-голямо число)

- **Задача:** **`самостоятелна реализация на итеративно решение`**

ОЩЕ ВЕДНЪЖ 'STACK': ТОЗИ ПЪТ НЕОГРАНИЧЕН

```
class Stack1   {

   private class Cell   {
       Object cont;          // content
       Cell next;            // pointer
   }

   // pointer to the top cell
   private Cell top;

   public Stack1()   {
       top = null;
   }
   public boolean isempty()   {
       return top == null;
   }
}
```

Локален клас (множество данни)

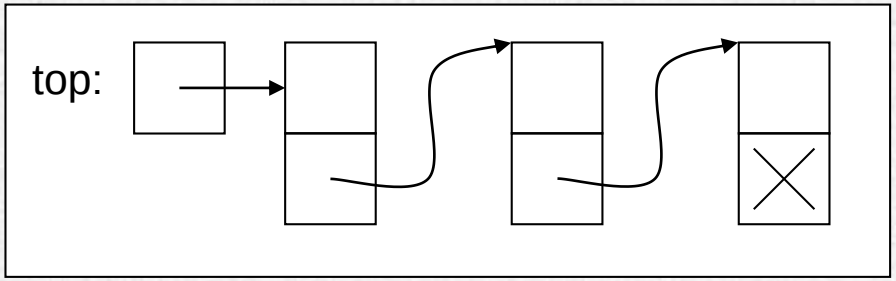
Stack1.java

КЛАС 'STACK': ПРЕДСТАВЯНЕ ДАННИ ЧРЕЗ ПРОМЕНЛИВА 'TOP'

```
class Stack   {

   private class Cell   {
       Object cont;          // content
       Cell next;            // pointer
   }

   // pointer to top cell
   private Cell top;

   ...
}
```

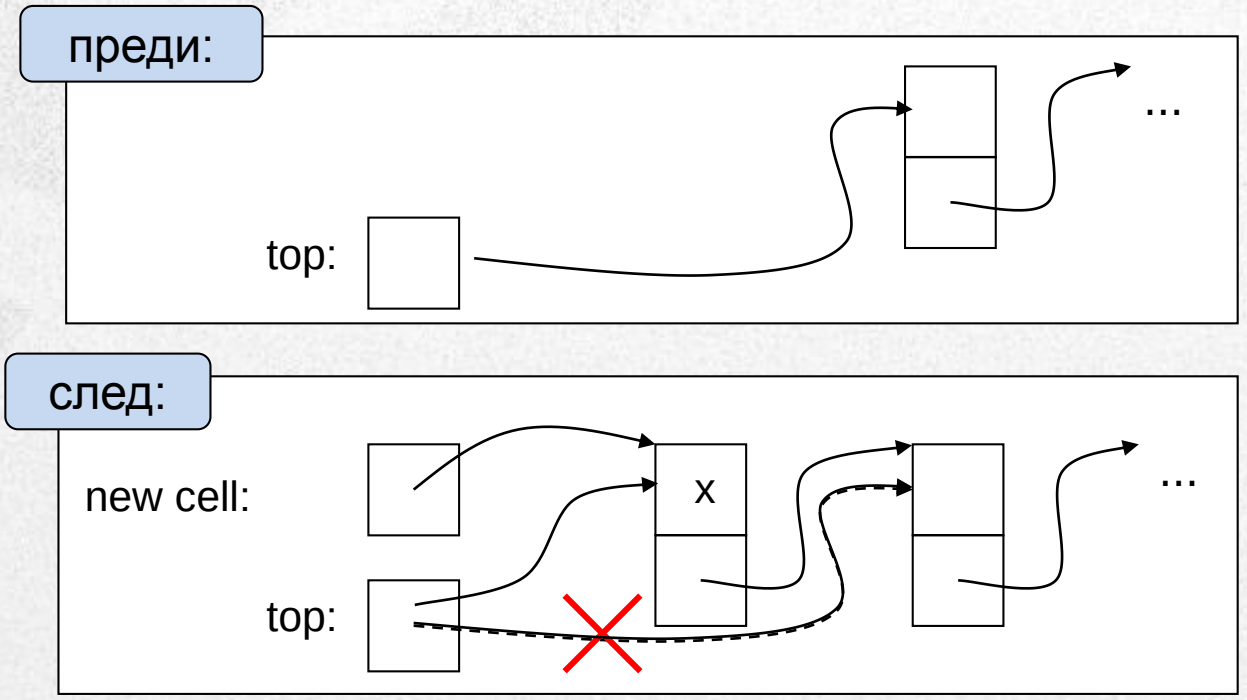
локален клас

Данни на стека, начало на списъка



'PUSH': УДЪЛЖАВАНЕ СПИСЪК ОТПРЕД

```
public void push (Object x)    {
    Cell newCell = new Cell();
    newCell.cont = x;
    newCell.next = top;
    top          = newCell;
}
```



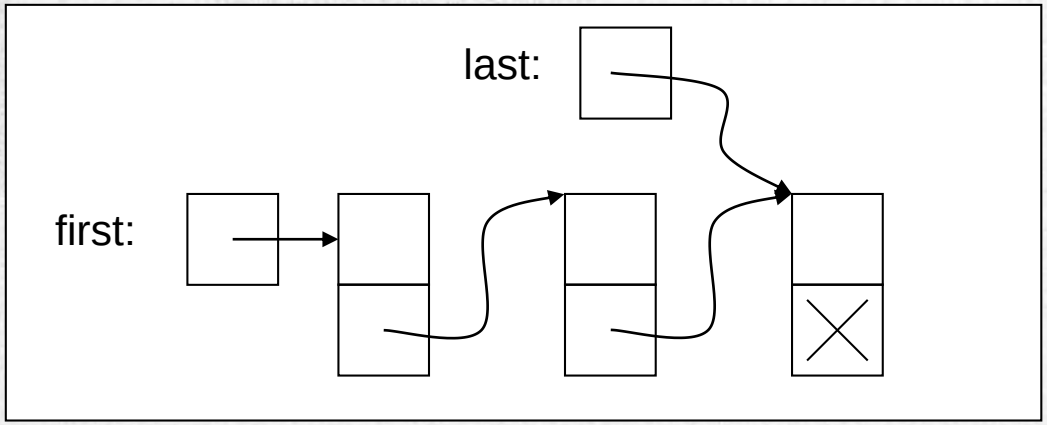
КЛАС 'QUEUE' (ОПАШКА)

Принцип: четене отпред, добавяне отзад - FIFO

```
class Queue   {

   private class Cell   {
        Object cont;   //content
        Cell next;     //pointer
   }
```

```
   // 2 pointers: first / last Cell

   private Cell first, last;
```

Идея?

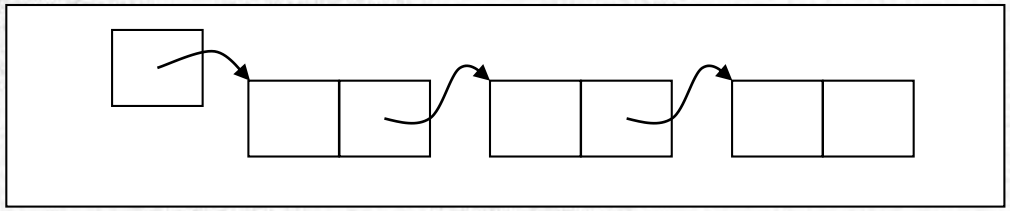


FIFO: First in, First out (Queue)
LIFO: Last in, First out (Stack)

Къде въвеждаме?
Къде задраскваме?

ОЦЕНКА: СВЪРЗВАНЕ В JAVA

- Свързани структури:
  - проблем на структурирането на данни



- Pascal, C, C++ ect.:
  - Record-Typ + Pointer Typ

- Java
  - Смесване на обектно-ориентиране и структуриране на данни:
  - Създаване на обекти се използва за генериране на указатели
  - Клас: реализиране на ADT
  - Основание: няма друг път, понеже преносимост е основна цел на Java (Pointer: проблемен за преносимостта – управление на адреси)

ДЕМОНСТРАЦИЯ НА СПИСЪК

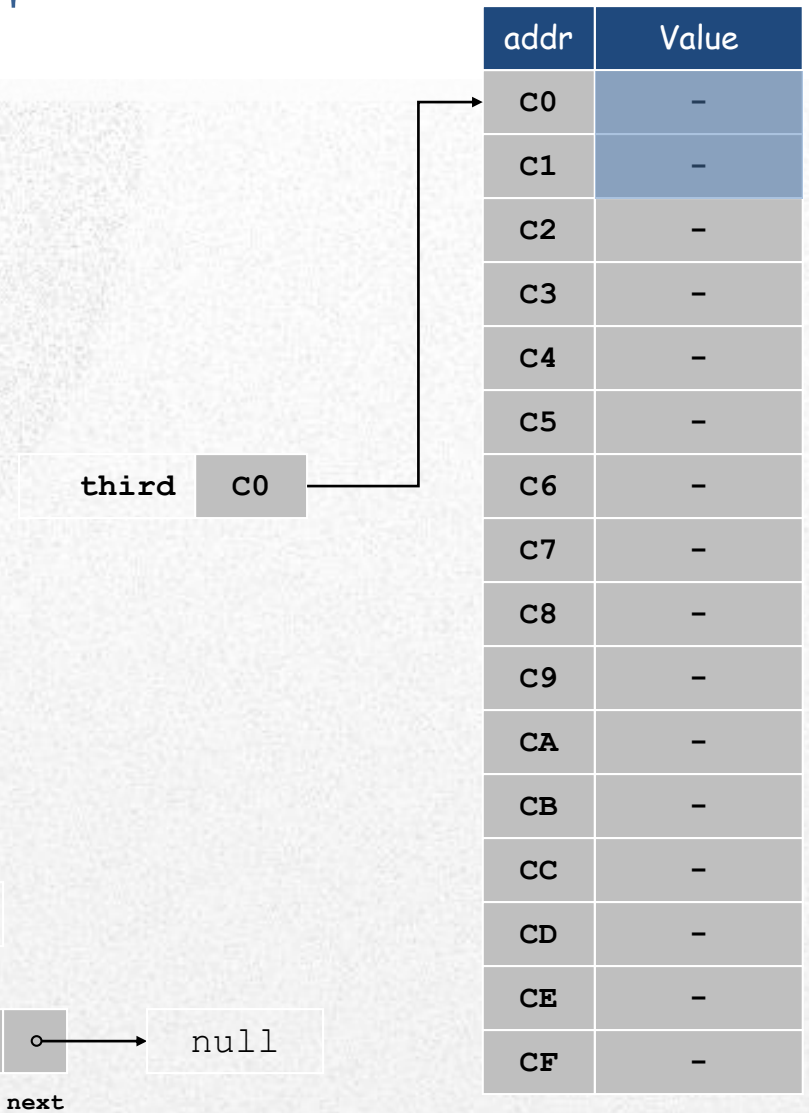
```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

main memory

addr	Value
C0	-
C1	-
C2	-
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-
CA	-
CB	-
CC	-
CD	-
CE	-
CF	-

ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```



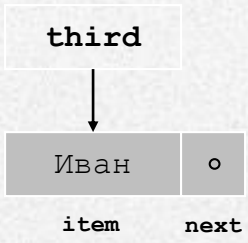
ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

main memory

addr	Value
C0	"Иван"
C1	-
C2	-
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-
CA	-
CB	-
CC	-
CD	-
CE	-
CF	-

third | C0



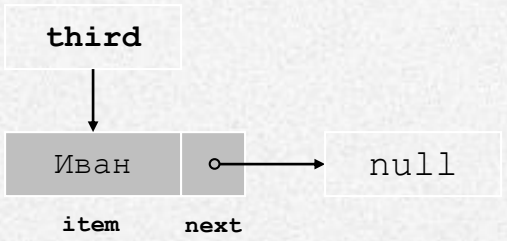
ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

main memory

addr	Value
C0	"Иван"
C1	null
C2	-
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-
CA	-
CB	-
CC	-
CD	-
CE	-
CF	-

third | C0



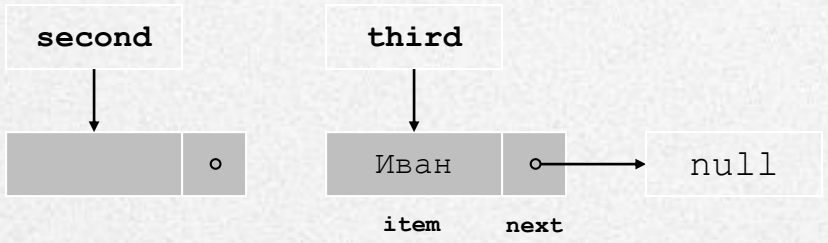
ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

main memory

addr	Value
C0	"Иван"
C1	null
C2	-
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-
CA	-
CB	-
CC	-
CD	-
CE	-
CF	-

second	CA
third	C0



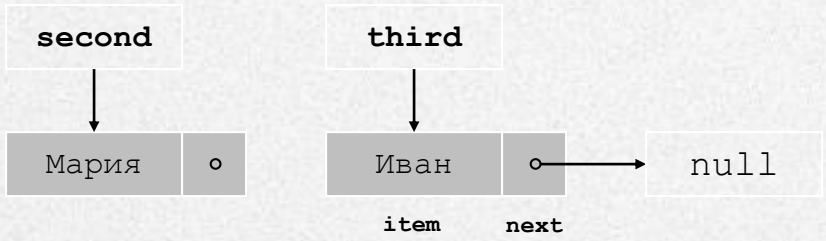
ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

main memory

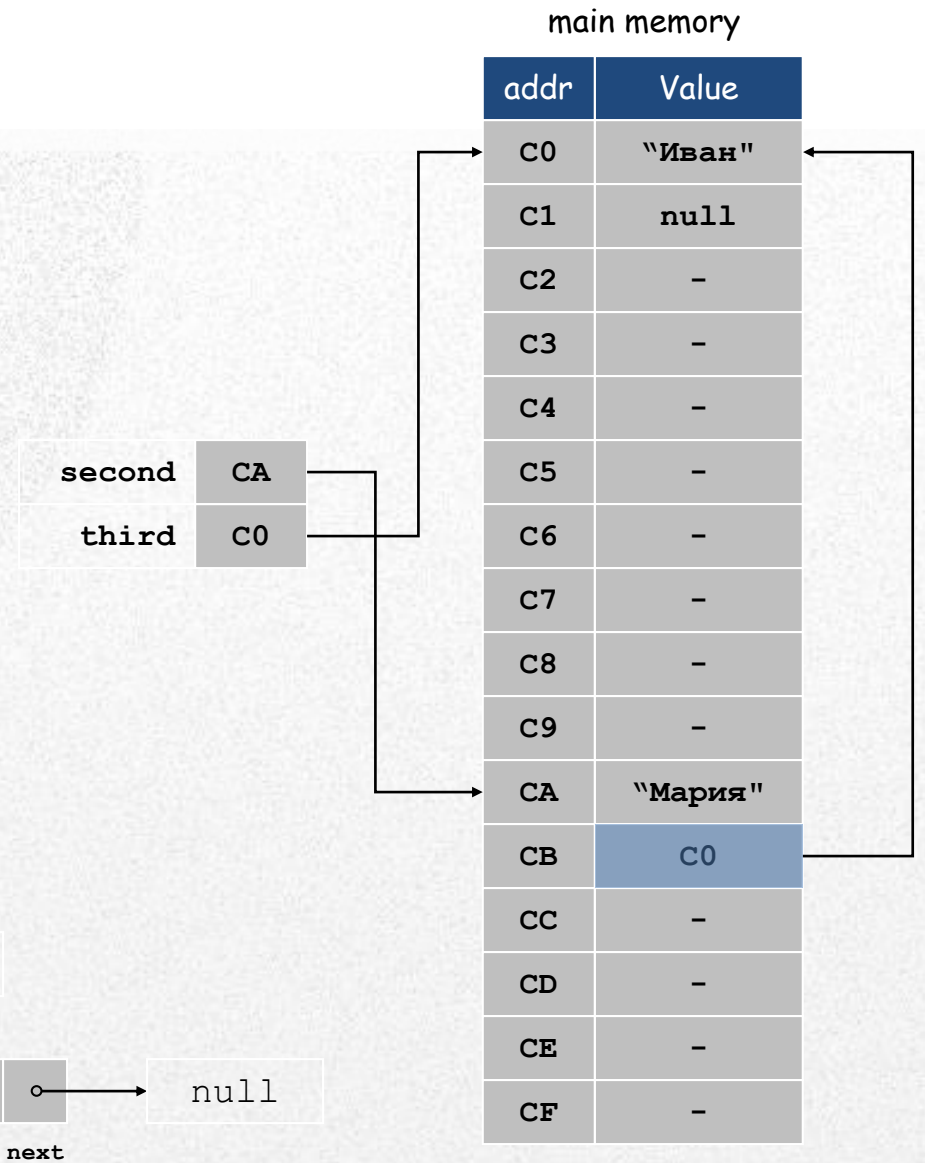
addr	Value
C0	"Иван"
C1	null
C2	-
C3	-
C4	-
C5	-
C6	-
C7	-
C8	-
C9	-
CA	"Мария"
CB	-
CC	-
CD	-
CE	-
CF	-

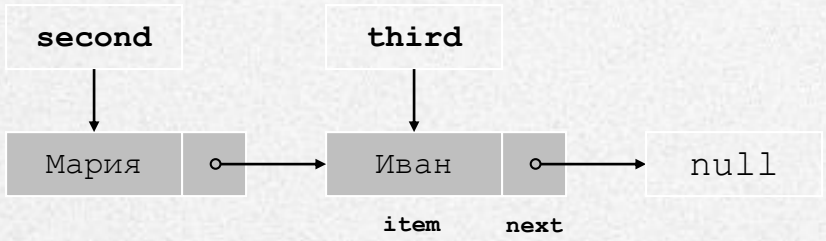
second	CA
third	C0



ДЕМОНСТРАЦИЯ НА СПИСЪК

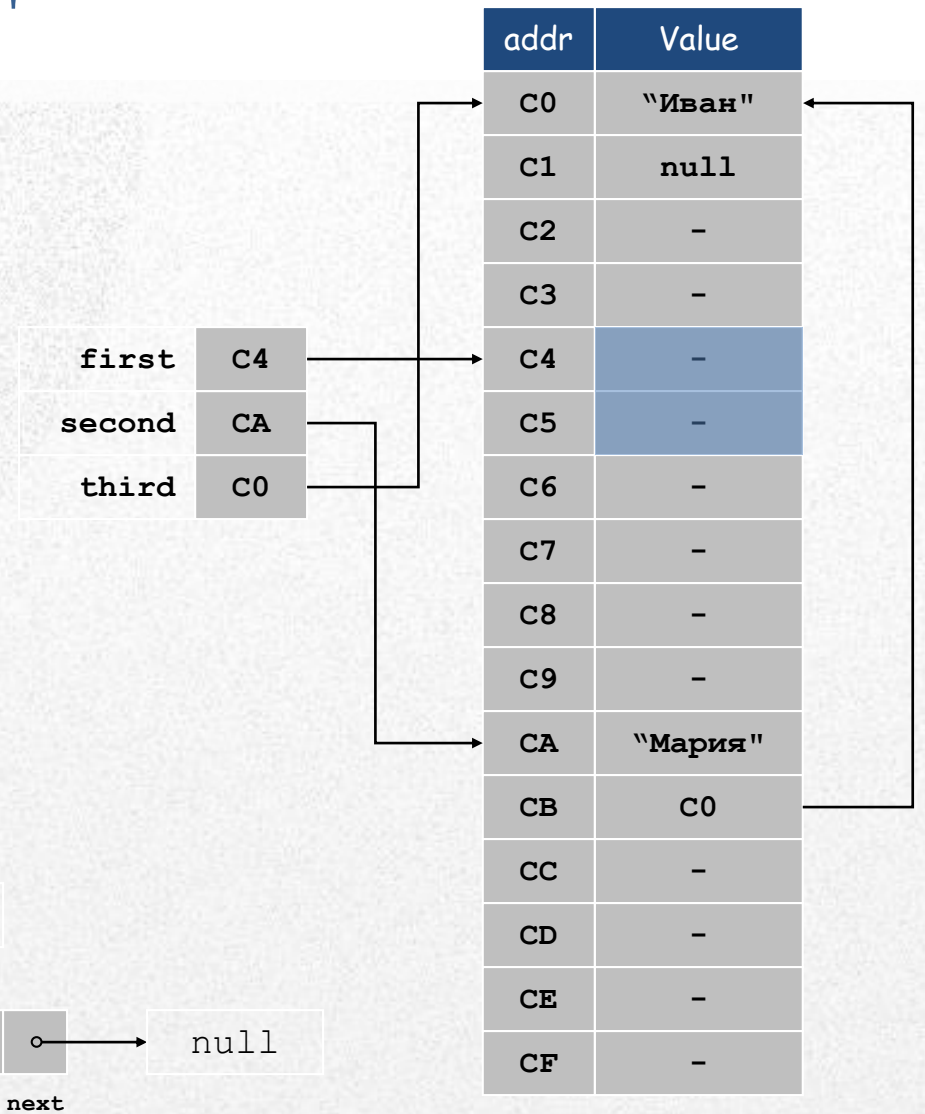
```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";


Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

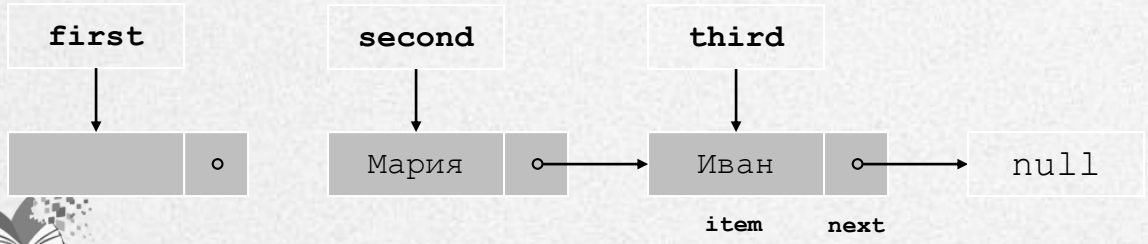




ДЕМОНСТРАЦИЯ НА СПИСЪК

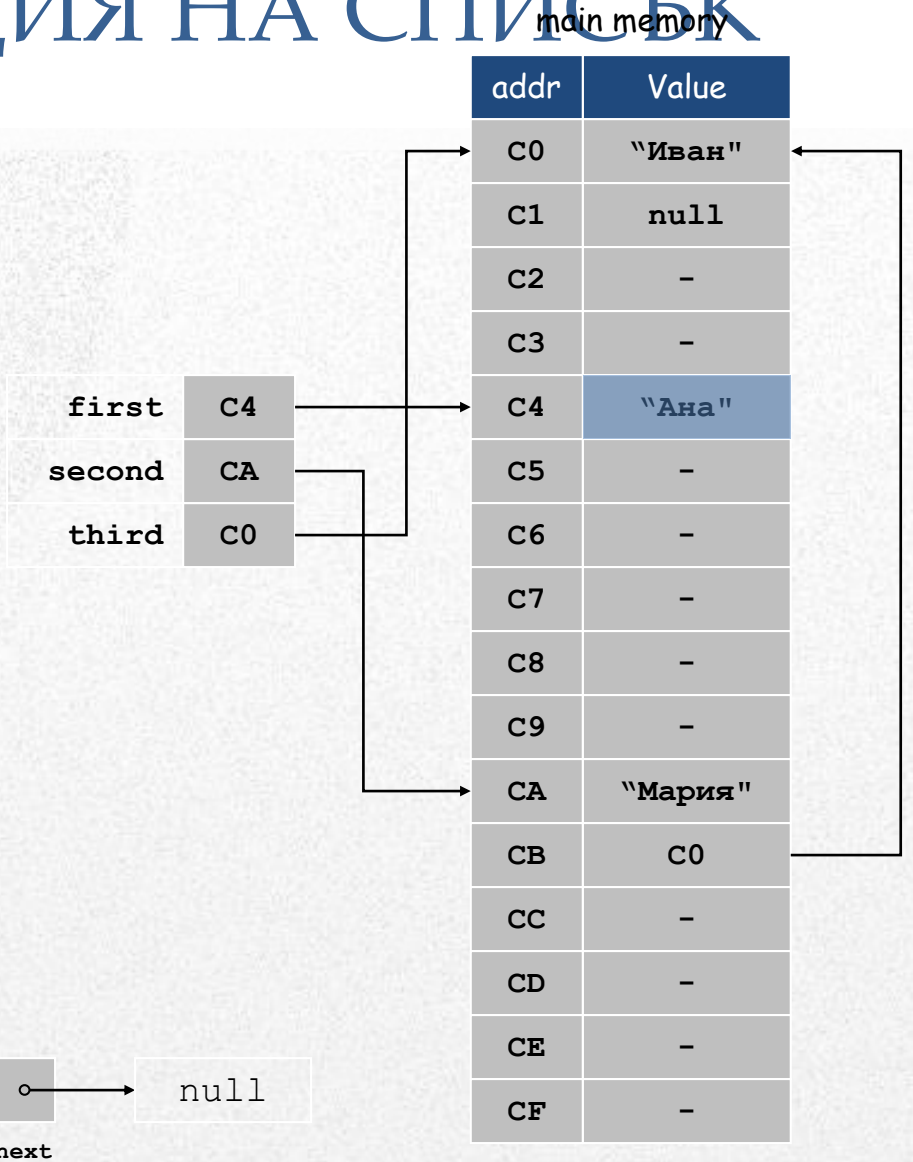
```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

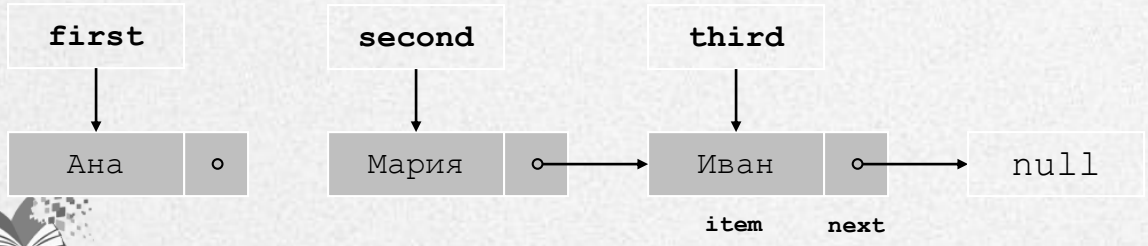




ДЕМОНСТРАЦИЯ НА СПИСЪК

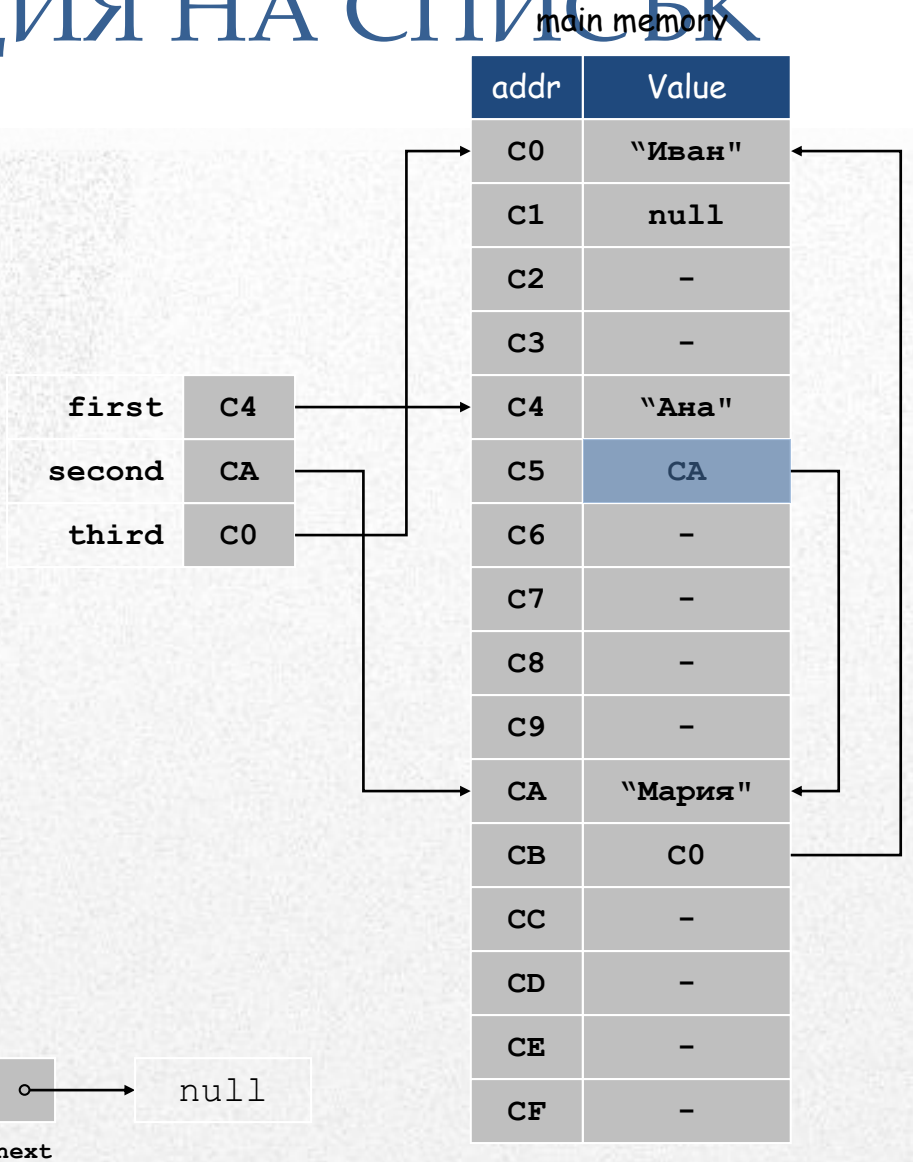
```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

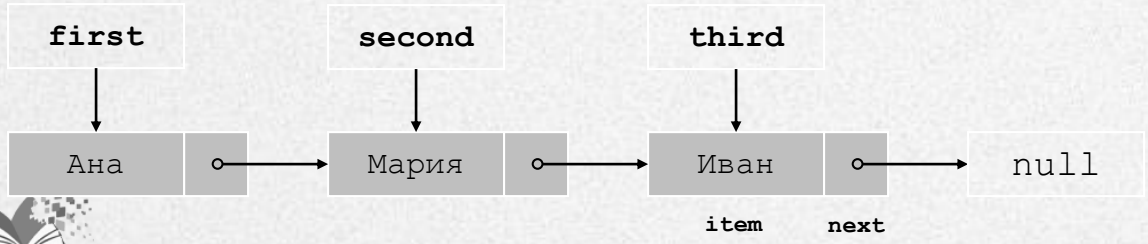




ДЕМОНСТРАЦИЯ НА СПИСЪК

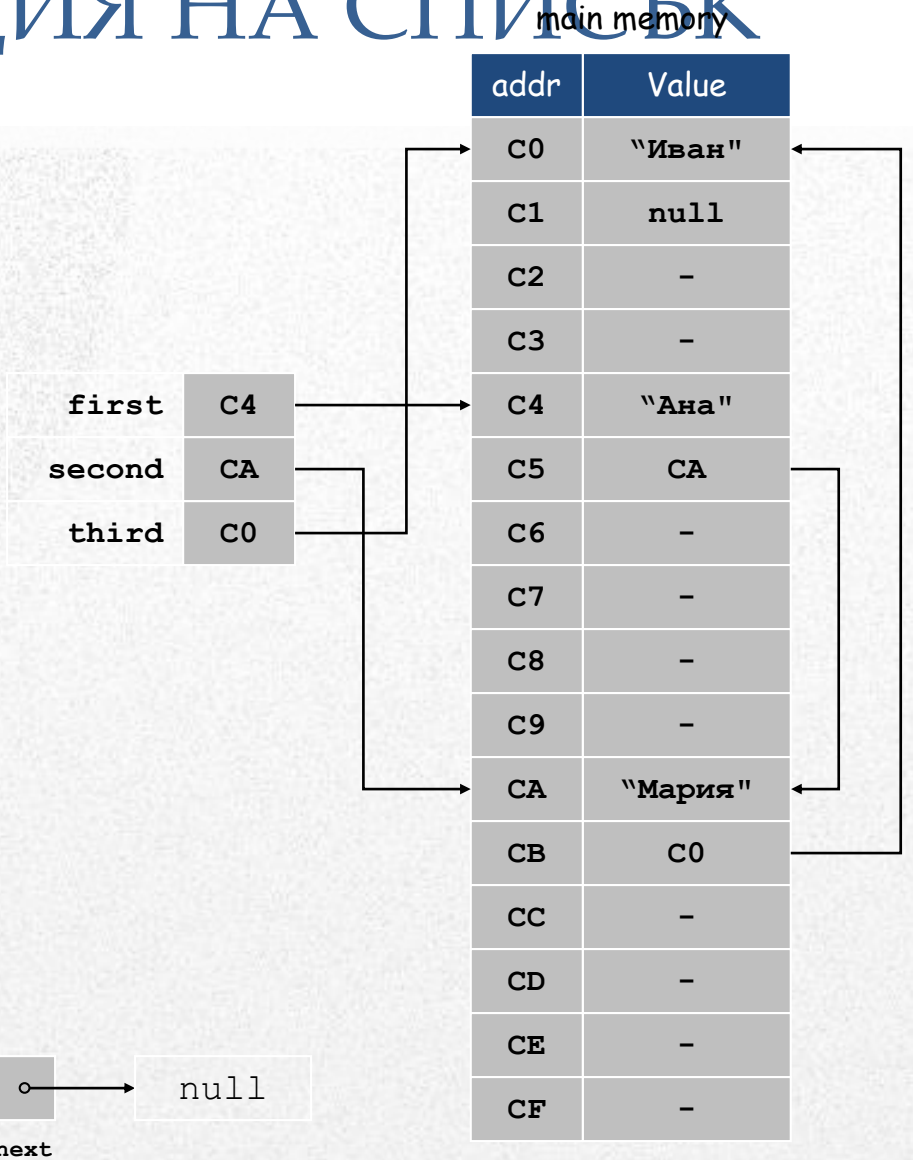
```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```

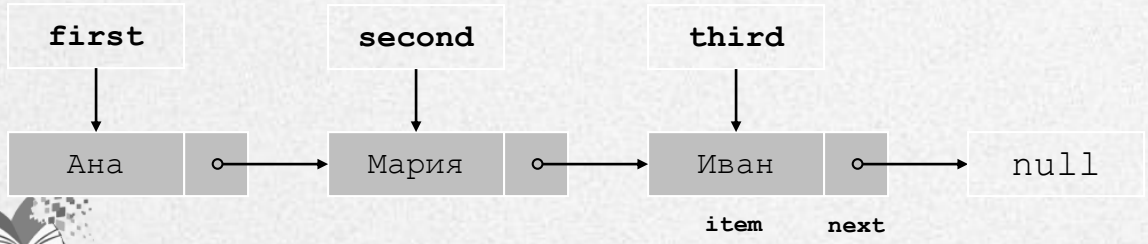




ДЕМОНСТРАЦИЯ НА СПИСЪК

```
Node third  = new Node();
third.item  = "Иван";
third.next  = null;

Node second = new Node();
second.item = "Мария";
second.next = third;

Node first  = new Node();
first.item  = "Ана";
first.next  = second;
```





БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “СПИСЪЦИ”

Distributed @learning center