Реализация АТД список:динамические структуры

List class

структура одного элемента

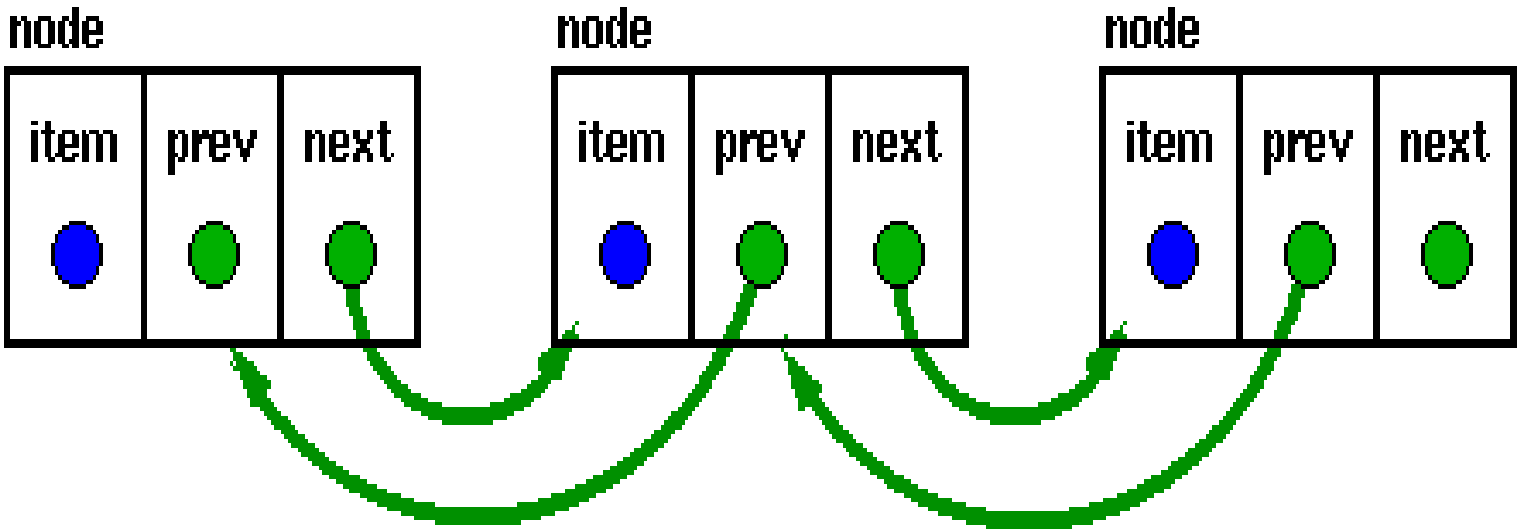
```
type
LIST = ↑celltype;
celltype = record
             element: eltype;
             next: LIST
           end;
position = ↑celltype;
```

Java_node

cursor

```
var
        Space: array [1..maxlength] of
         record
            element: eltype;
            next :integer
         end;
```

Двунаправленные списки



АТД «связный список» (linked list) является объектно - ориентированным расширением структуры данных связного списка.

Он описывает методы доступа и обновления, основанные на инкапсуляции узлов связного списка.

Данные абстрактные узлы-объекты называются позициями, так как содержат ссылки на «места» хранения элементов, независимые от особенностей реализации списка.

В данной схеме список рассматривается как контейнер элементов, хранящихся в определенных позициях, и эти позиции линейно упорядочены.

Позиция сама по себе является абстрактным типом данных, который поддерживает следующий метод:

element(): возвращает элемент, хранящийся в данной позиции.

Input: нет; Output: объект.

АТД «связный список». поддерживает следующие методы, выполняемые над списком S:

- **first(): возвращает позицию первого элемента списка S**
- **last(): возвращает позицию последнего элемента списка S**
- **isFirst (p) : возвращает логическое значение, показывающее, является ли данная позиция первой в списке.**
- **isLast(p) возвращает логическое значение, показывающее, является ли данная позиция последней в списке.**

- **before(p): возвращает позицию элемента S, который предшествует элементу позиции p,если p является первой позицией, выдается сообщение об ошибке.**
- **after(p): возвращает позицию элемента S, который следует элементом позиции p; если p является последней позицией, выдается сообщение об ошибке.**
- **replaceElement(p,e): замещает элемент в позиции p на e и возвращает элемент, который до этого был в позиции p.**

- **swapElements(p,q): меняет местами элементы в позициях p и q**
- **insertFirst(e): вставляет новый элемент e в S в качестве первого элемента списка.**
- **insertLast(e): вставляет новый элемент e в S качестве последнего элемента списка.**
- **insertBefore(p, e): вставляет новый элемент e в S перед позицией p если p является первой позицией, выдается сообщение об ошибке.**
- **insertAfter(p, e): вставляет новый элемент e в S после позиции p если p является последней позицией, выдается сообщение об ошибке**

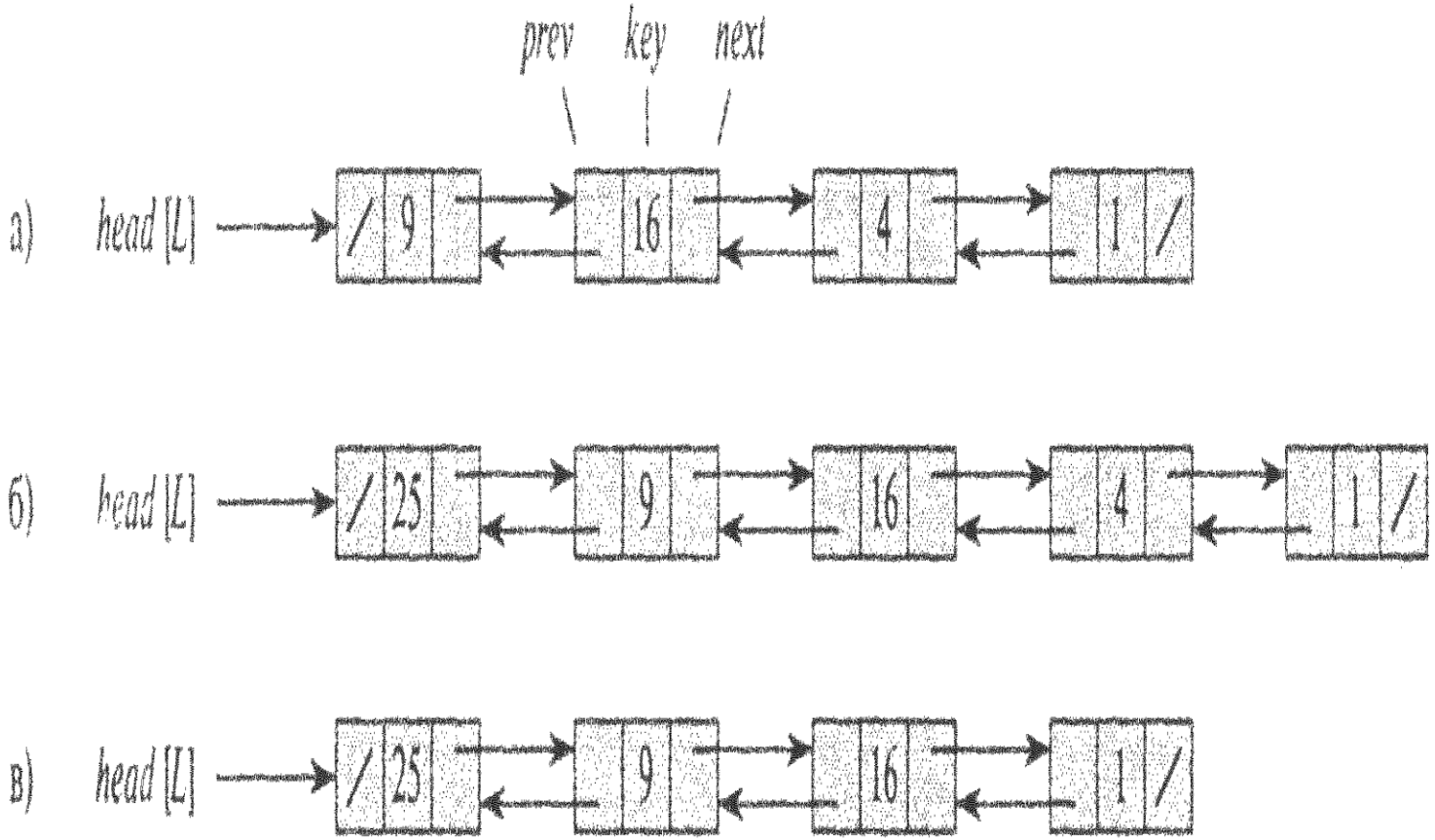
- **remove(p): удаляет из S элемент в позиции p.**

Алгоритм insertAfter(p,e):
//**Create a new node v**
v.setElement(e)
v.setPrev(p) // j связывает v с предшествующим узлом
v.setNext(p.getNext()) // связывает v с последующим узлом
(p.getNext ()).setPrev (v)//связывает ранее следовавший за *p* узел с v
p.setNext(v)// связывает p с новым последующим узлом v
return v //позиция элемента e

Двунаправленные списки

- интерфейс List

Поиск в списке



Поиск в списке

Для заданного элемента списка x (x-указатель) указатель next [x] указывает на следующий элемент связанного списка.

Если next [x] = NIL, то у элемента x нет последующего, поэтому он является последним, т.е. хвостовым в списке.

Аналогично prev [x]

key[x]- значение ключа, соответствующего x

Атрибут head [L] указывает на первый элемент списка.

Если head [L] = NIL, то список пуст.

List_Search(L, к)

1 x ← head[L]

2 while x ≠ nil и кеу[x] ≠ k

3 do x ← next[x]

4 return x

Временная сложность

T(n) = O(n)

Вставка в списке (на первую позицию)

List_Insert(L, x)

1 next[x] ← head[L]

2 if head[L] ≠ nil

3 then prev[head[L]] ← x

4 head[L] ← x

5 prev[x] ← nil

Временная сложность

T(n) = O(1)

List_Delete(L, x)

1 if prev[x] ≠ nil

2 then next[prev[x]] ← next[x]

3 else head[L] ← next[x]

4 if next[x] ≠ nil

5 then prev[next[x]] ← prev[x]

Временная сложность

T(n) = O(1)

Удаление с заданным ключом T(n) = O(n)

Коллекции

Коллекция LinkedList реализует связанный список.

В отличие от массива, который хранит объекты в последовательных ячейках памяти,

связанный список хранит объекты отдельно, но вместе со ссылками на следующее и предыдущее звенья последовательности.

Коллекции

В добавление ко всем имеющимся методам в LinkedList реализованы методы

void addFirst(Object ob)

void addLast(Object ob)

Object getFirst( )

Object getLast( )

Object removeFirst( )

Object removeLast ( )

Коллекции

```
/* пример: добавление и удаление элементов :  */
import java.util.*;
public class DemoLinkedList {
 public static void main(String[] args){
        LinkedList aL = new LinkedList();
        for(int i = 10; i <= 20; i++)
                aL.add("" + i);
```

Коллекции

```
Iterator it = aL.iterator();
while(it.hasNext())
 System.out.print(it.next() + " -> ");

ListIterator list = aL.listIterator();
      Object o = list.next();
```

Коллекции

```
System.out.println("\n" + list.nextIndex()
                        + "-й индекс");
    //удаление элемента с текущим индексом
list.remove();
    while(list.hasNext()) //переход к
                        // последнему индексу
     Object o = list.next();
    while(list.hasPrevious())
              /*вывод в обратном порядке */
    System.out.print(list.previous() + "  ");
```

Коллекции

```
//методы, характерные для LinkedList
  aL.removeFirst();
  aL.removeLast();
  aL.removeLast();
  aL.addFirst("FIRST");
  aL.addLast("LAST");
  System.out.println("\n" + aL);

}
}
```

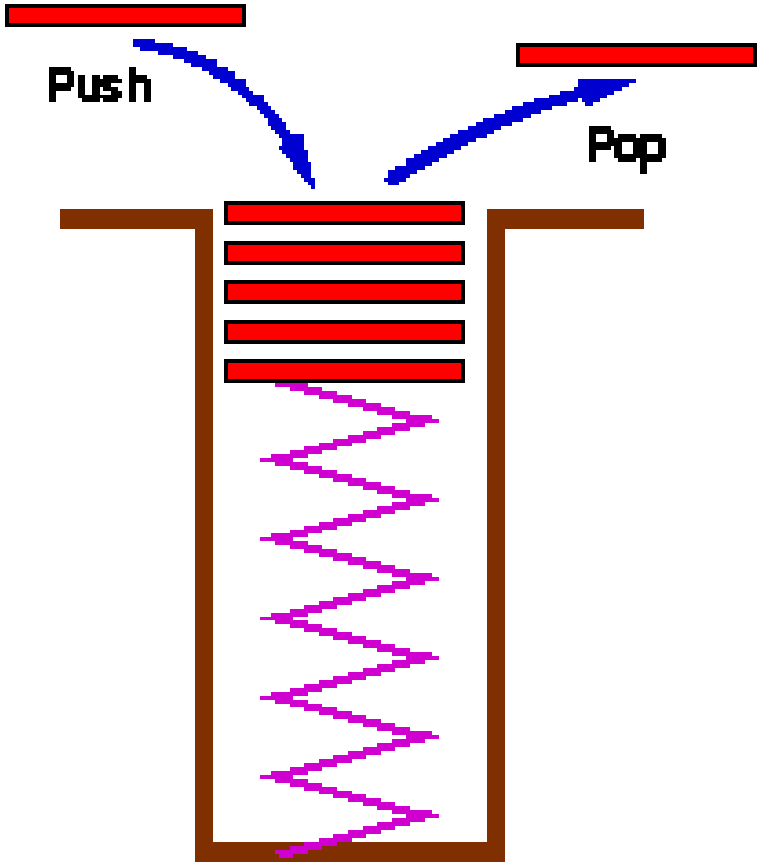
1. **Реализация списков с помощью массивов расточительна в отношении компьютерной памяти.**
2. **Реализация с помощью указателей использует столько памяти, сколько необходимо для хранения текущего списка, но требует дополнительную память для указателя каждой ячейки.**
3. **Таким образом, в разных ситуациях по критерию используемой памяти могут быть выгодны разные реализации.**

Стеки (stack) (1946 Алан Тьюринг)

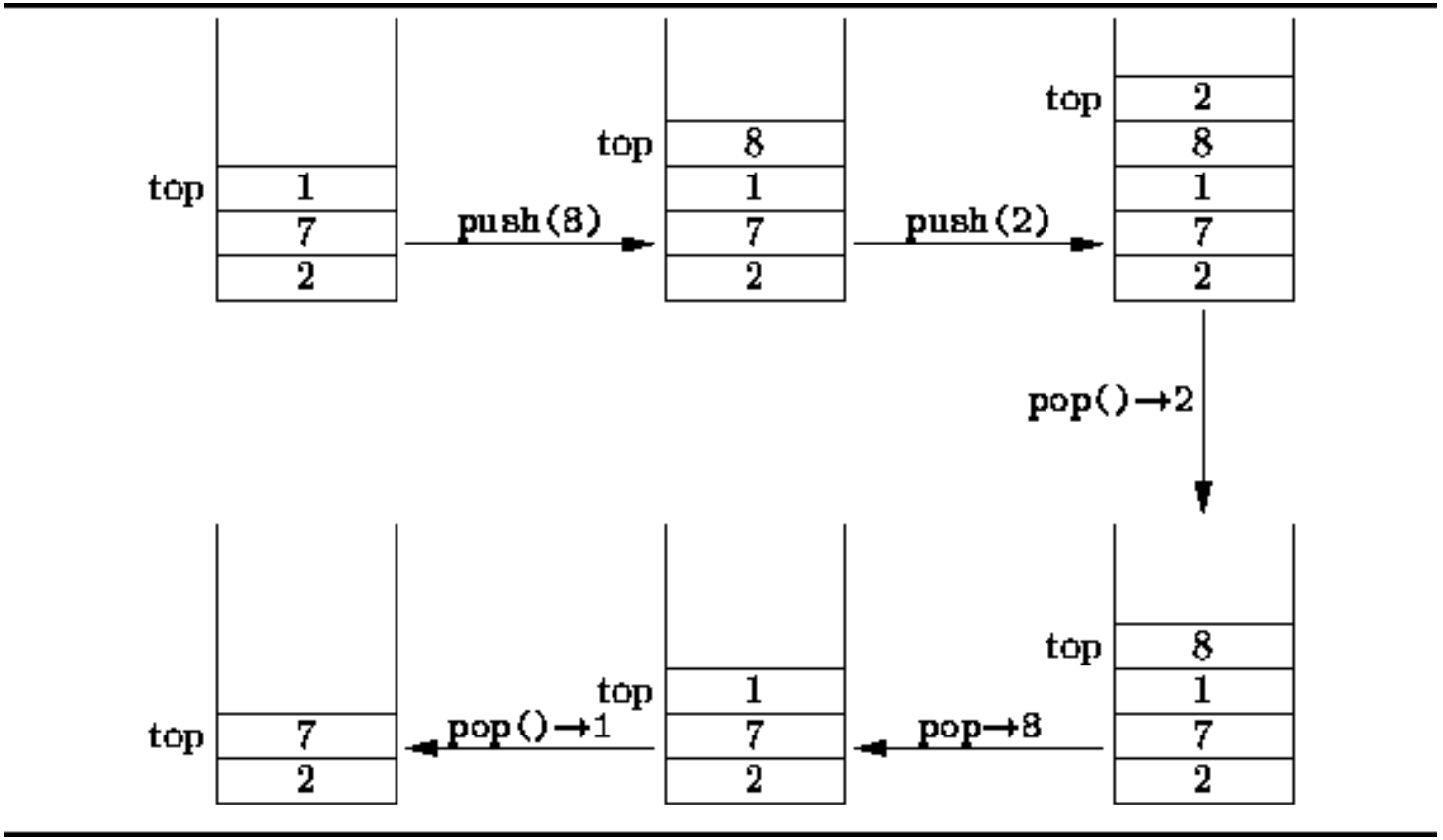
Стек **— это специальный тип списка, в котором все вставки и удаления выполняются в начале , называемом вершиной (top).**

Стеки также иногда называют "магазинами" или список с дисциплиной LIFO (last-in-first-out).

Стеки (stack)



Стеки (stack)


top
1
7
2
push(8)
top
8
1
7
2
push(2)
top
2
8
1
7
2
pop()→2
top
8
1
7
2
pop→8
top
1
7
2
pop()→1
top
7
2

АТД стек поддерживает следующие основные методы:

- **push (o): помещает объект o в стек.**

Input: объект; Output: нет.

- **pop (): удаляет объект из стека и возвращает новый верхний объект стека; если стек пуст, выдается сообщение об ошибке.**

Input: нет; Output: объект.

Дополнительнее методы:

- **size ():** возвращает число объектов в стеке.

Input: нет; **Output**: целое число.

- **isEmpty ():** возвращает логическое значение, подтверждающее, что стек пуст.

Input: нет; **Output**: логическое значение.

- **top ():** возвращает верхний объект в стеке, не удаляя его; если стек пуст, выдается сообщение об ошибке.

Input: нет; **Output**: объект.

Линейная реализация:

Algorithm size():

return t +1

Algorithm IsEmpty():

return (t < 0)

Линейная реализация:

Algorithm top():

if isEmpty() then

вызов StackEmptyException

return S[t]

Algorithm push(o):

if size() = N then

вызов StackFullException

t ← t + 1

S[t] ← o

Линейная реализация:

Algorithm pop():

if isEmpty() then

вызов StackEmptyException

e ← S[t)

S[t] ← null

t ← t - 1

return e

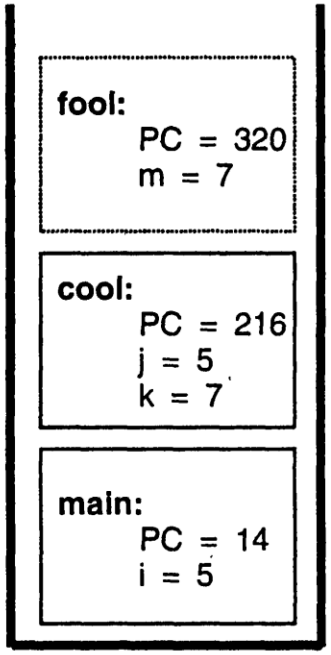
A simple Java implementation of the stack

Stack linear **Class_Stack** **Stack dynamic**

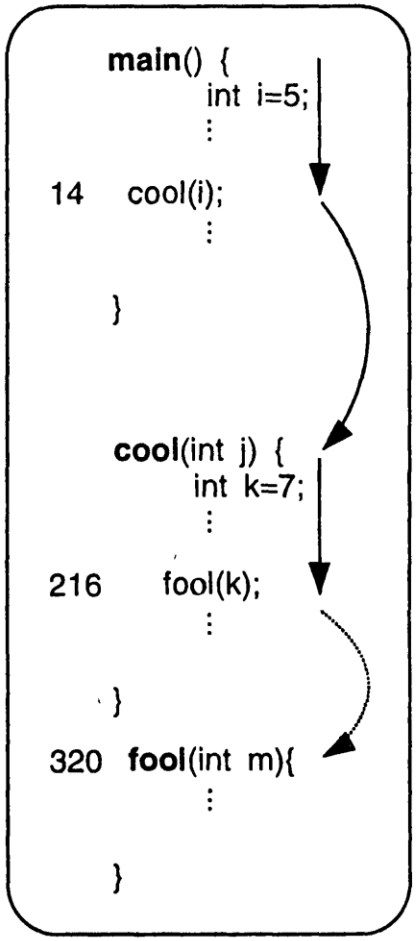
Использование стека

1. **Вложенные вызовы подпрограмм**
2. **Размещение локальных переменных в блоках**
3. **Преобразование арифметических выражений в польскую запись**
4. **Аппаратные стеки**

Реализация Стеков



Java-стек



Java-программа

Польская запись:

1-2/3*4+5

ссылка

В микропроцессорах семейства Intel, как и в большинстве современных процессорных архитектур, поддерживается аппаратный стек.

Аппаратный стек расположен в ОЗУ, указатель стека содержится в паре специальных регистров - SS:SP, доступных для программиста.

Очередь – специальный тип списка-(queue), в котором вставка элементов осуществляется с одного конца (rear/back), а удаление с другого (front).

Очереди также называют "списками типа FIFO" (first-in-first-out)

АТД «очередь» поддерживает два следующих основных метода:

- **enqueue (o): помещает объект o в конец очереди.**

Input: объект; Output: нет.

- **dequeue (): производит удаление и возвращает объект из начала очереди; если очередь пуста, выдается сообщение об ошибке.**

Input: нет; Output: объект.

дополнительные методы АТД «очередь» :

- **size (): возвращает число объектов в очереди.**

Input: нет; Output: целое число.

- **isEmpty (): возвращает логическое значение, подтверждающее, что очередь пуста.**

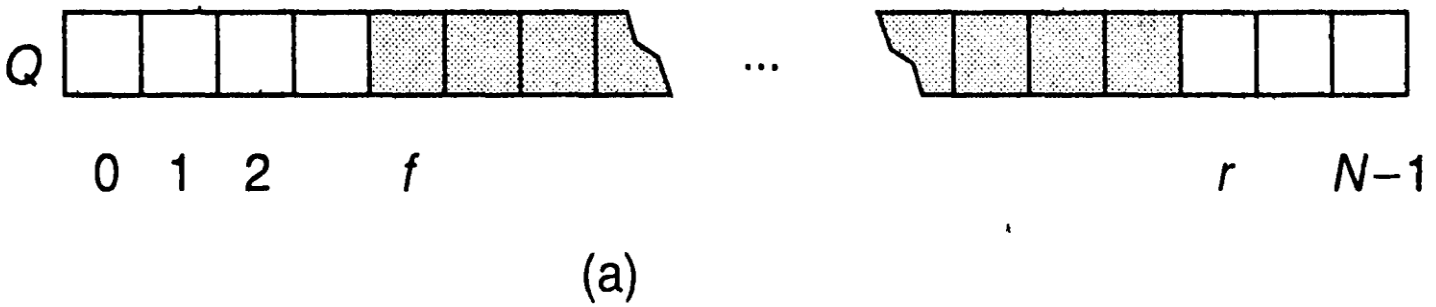
Input: нет; Output: логическое значение.

- **front (): возвращает первый объект в очереди, не удаляя его; если очередь пуста, выдается сообщение об ошибке.**

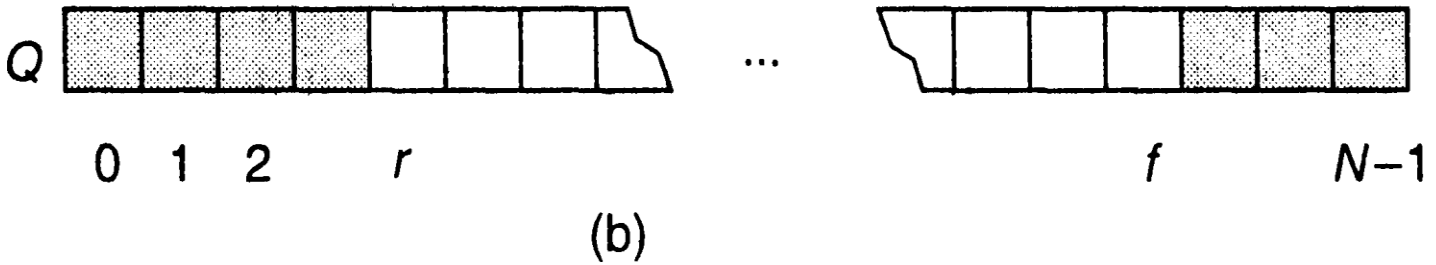
Input: нет; Output: объект.

Интерфейс

Реализация очередей - линейное представление



(a)



(b)

DEQUEUE

Java_implementation

Java.util. QUEUE

Примеры очередей

1. **Буфер клавиатуры**
2. **Очереди в многозадачных ОС**
3. **Очереди сообщений для параллельно выполняемых задач**
4. **Очереди в имитационном моделировании**

Деки(Deque - double ended queue)

Дек – структура данных с двусторонним доступом, хранящая упорядоченное множество значений, для которой определены следующие операции:

Деки

АТД «дек» поддерживает следующие базовые методы:

- **insertFirst (e): помещает новый элемент e в начало D.**
- **Input: объект; Output: нет.**
- **insertLast (e): помещает новый элемент e в конец D.**
- **Input: объект; Output: нет.**

Деки

- **removeFirst (): удаляет и возвращает первый элемент D, если D пуст, выдается сообщение об ошибке.**

Input: нет; Output: объект.

- **removeLast (): удаляет и возвращает последний элемент D, если D пуст, выдается сообщение об ошибке.**

Input: нет; Output: объект.

Деки

дополнительные методы:

- **first(): возвращает первый элемент D, если D пуст, выдается сообщение об ошибке.**

Input: нет; Output: объект.

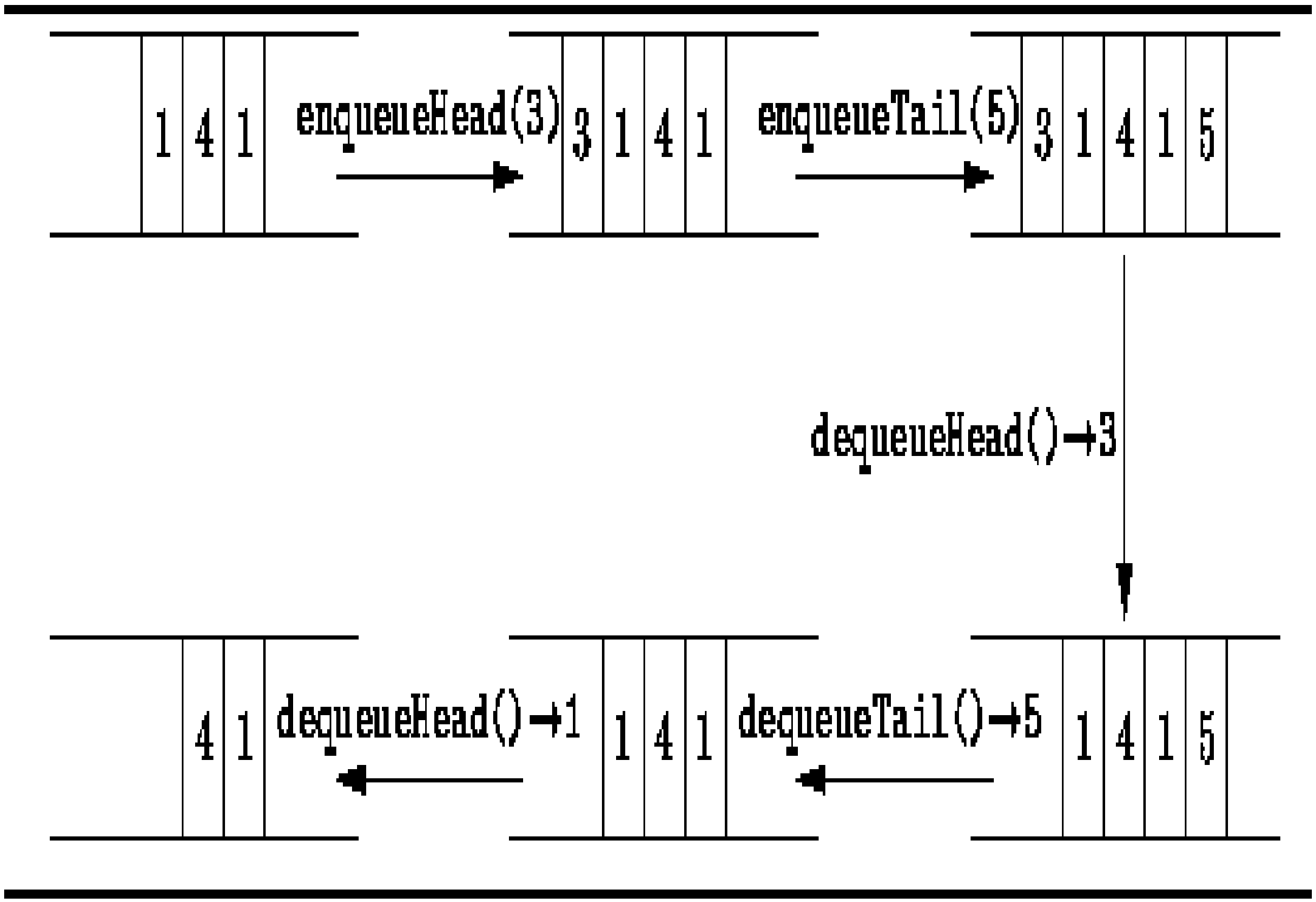
- **last(): возвращает последний элемент D, если D пуст, выдается сообщение об ошибке.**

Input: нет; Output: объект,

Деки

- **size (): возвращает число элементов D.**
- **Input: нет; Output: целое число.**
- **isEmpty (): определяет, является ли D пустым.**
- **Input: нет; Output: логическое значение.**

Деки


1 4 1
enqueueHead(3)
3 1 4 1
enqueueTail(5)
3 1 4 1 5
dequeueHead()→3
4 1
dequeueHead()→1
1 4 1
dequeueTail()→5
1 4 1 5

Деки

```
public interface Deque extends Container {
    Object getHead ();
    Object getTail ();
    void enqueueHead (Object object);
    void enqueueTail (Object object);
    Object dequeueHead ( );
    Object dequeueTail ();
}
```

Деки

Класс Deque

Object dequeue()

Object dequeueBack()

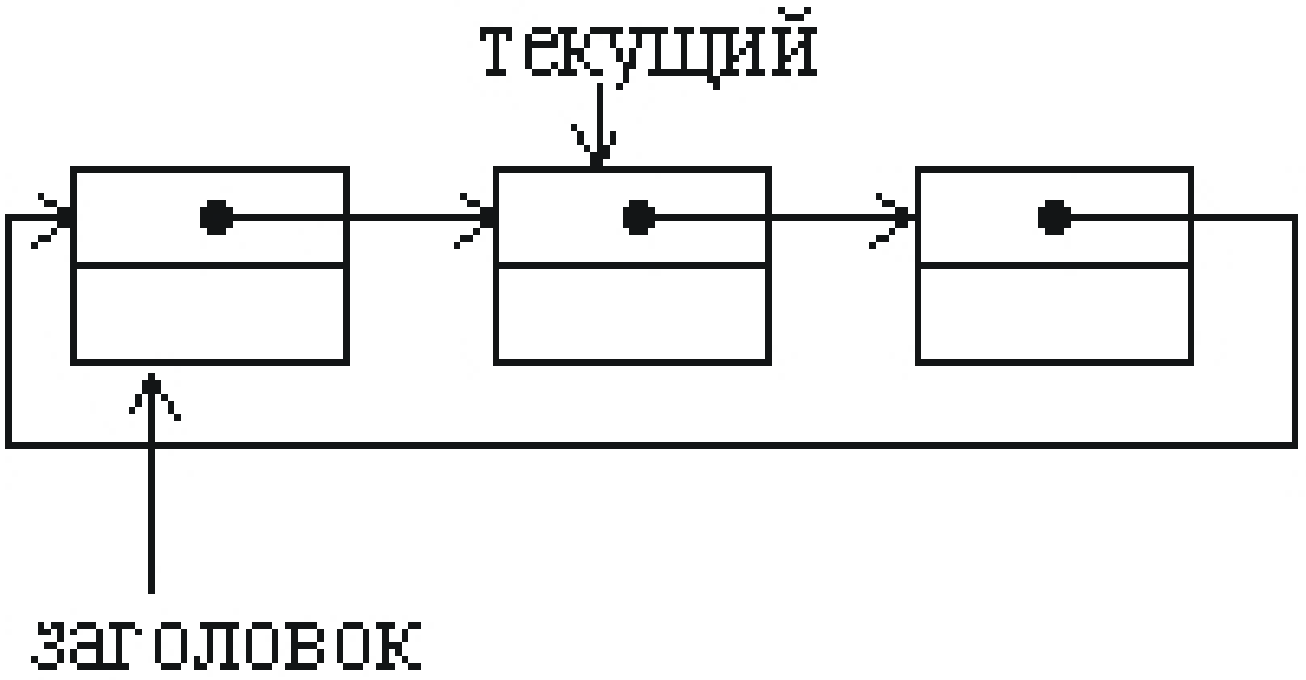
void enqueueFront(Object o)

void enqueue(Object o)

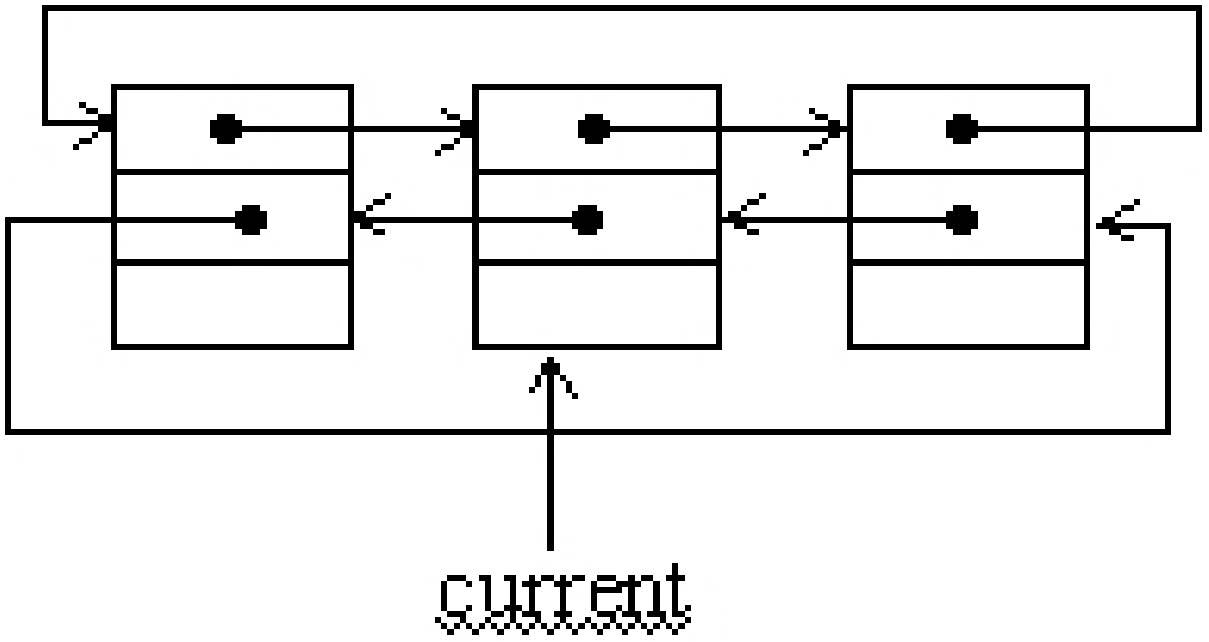
Object peekBack()

Object peekFront()

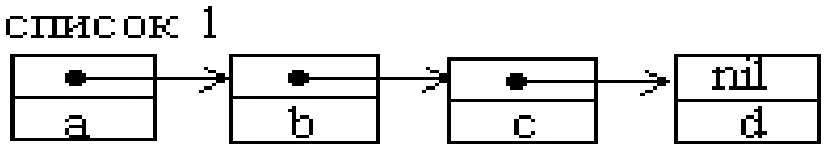
Циклические списки

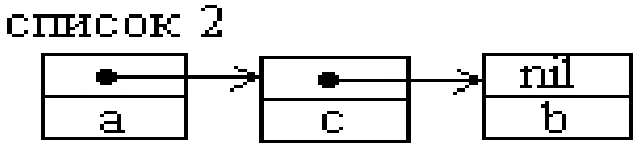


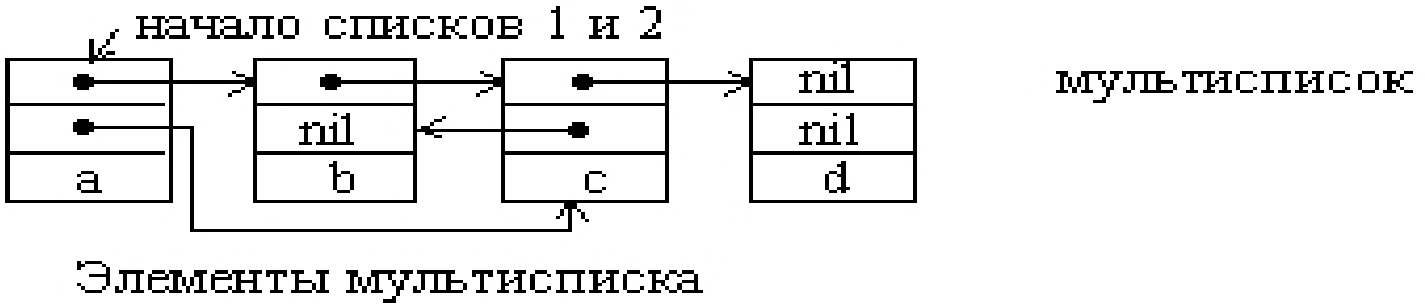
Циклические списки



Мультисписки







A - множество элементов списка 1

B - множество элементов списка 2

C – множество элементов мультисписка ($C = A \cup B$)