

Implementace abstraktních datových typů

- **Zásobník (Stack)**
- **Fronta (Queue)**
- **Obousměrná fronta (Deque)**
- **Vektor (Vector)**
- **Seznam (List)**

ADT Zásobník (Stack)

- Zásobník je datová struktura, do které se objekty vkládají a vybírají podle strategie **LIFO** (**L**ast-**I**n-**F**irst-**O**ut)
- Použití:
 - ukládání návratových adres podprogramů
 - ukládání změn v textu při operaci *Undo* v textových editorech
 - uchovávání historie adres ve Web browseru
 - zpracování výrazů v postfixových notacích
 - atd.

Metody pro práci se zásobníkem:

`push(o)` : Vkládá objekt **o** na vrchol zásobníku

Vstup: Objekt **Výstup:** Není

`pop()` : Odstraňuje objekt z vrcholu zásobníku. Pokud je zásobník prázdný – chyba

Vstup: Není **Výstup:** Object

`size()` : Vrací počet objektů v zásobníku

Vstup: Není **Výstup:** Integer

`isEmpty()` : Vrací *true* pokud je zásobník prázdný

Vstup: Není **Výstup:** Boolean

`top()` : Vrací objekt z vrcholu zásobníku bez jeho odstranění. Pokud je zásobník prázdný dojde k chybě.

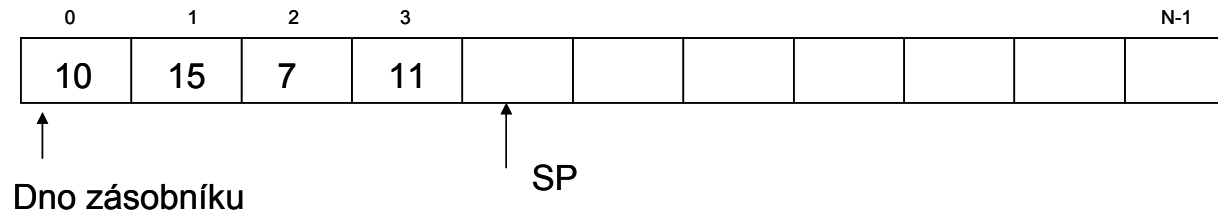
Vstup: Není **Výstup:** Object

- Př. Posloupnost operací se zásobníkem a jejich výsledek

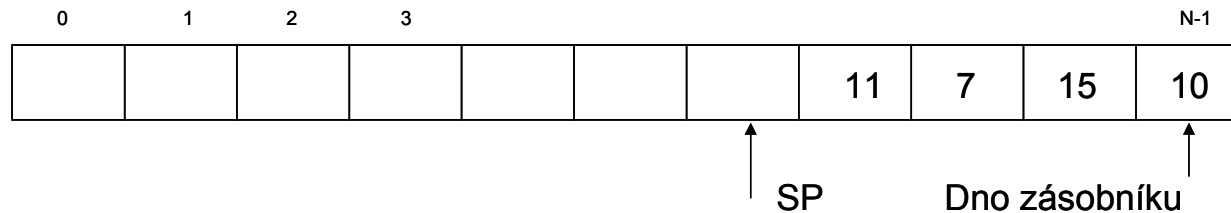
Operace	Výstup	S
push(5)	-	(5)
push(3)	-	(5,3)
pop()	3	(5)
push(7)	-	(5,7)
pop()	7	(5)
top()	5	(5)
pop()	5	()
pop()	chyba	()
isEmpty()	true	()
push(9)	-	(9)
push(7)	-	(9,7)

- **Implementace polem**

- přímý – ve směru rostoucích adres (vrchol zásobníku roste, dno zásobníku je prvek pole s indexem 0)



- Zpětný – ve směru klesajících adres (vrchol zásobníku klesá, dno zásobníku – prvek pole s indexem N-1)



SP – stack pointer, ukazuje na první volnou položku, (používá se i způsob kdy SP ukazuje na poslední obsazenou položku)

Příklad implementace zásobníku polem v Javě

```
public class StackA {
    private final int CAPACITY=100;
    private Object[] s; // pole pro ulozeni prvku zasobniku
    private int sz;
    private int sp;      //ukazatel zasobniku

    public StackA(){
        s=new Object[CAPACITY];
        sp=0;
        sz=0;
    }

    public boolean isEmpty(){
        if (sp==0)
            return(true);
        else return(false);
    }

    public int size(){
        return(sz);
    }
}
```

```
public Object top()throws StackEmptyException {
    if (sp==0)
        throw new StackEmptyException("Stack is empty");
    else return(s[sp-1]);
}

public Object pop() throws StackEmptyException {
    if (sp==0)
        throw new StackEmptyException("Stack is empty");
    else return(s[--sp]);
}

public void push(Object e) throws StackFullException{
    if (sp>=CAPACITY)
        throw new StackFullException("Stack is full");
    else s[sp++]=e;
}

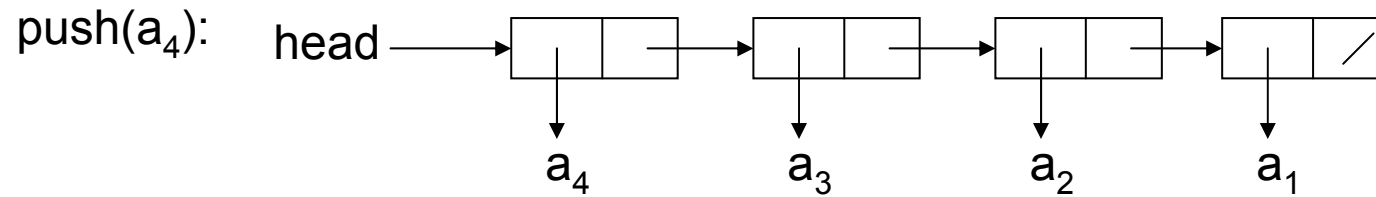
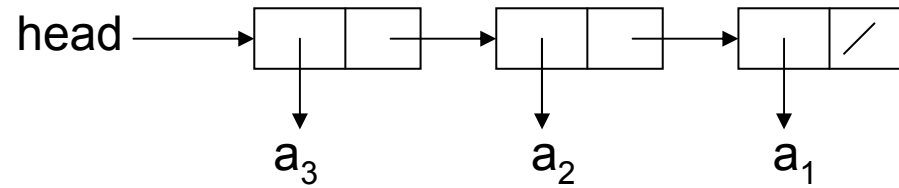
public class StackEmptyException extends RuntimeException {
    public StackEmptyException(String err) {
        super(err);
    }
}
```

```
public class StackFullException extends RuntimeException {
    public StackFullException(String err) {
        super(err);
    }
}

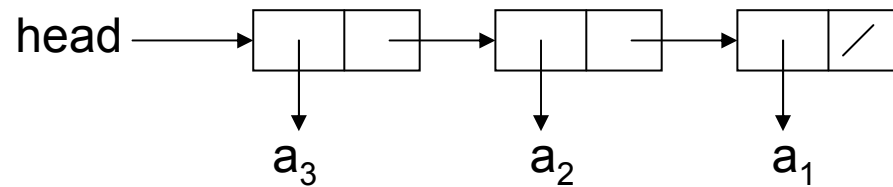
public static void main(String[] args) {
    StackA st=new StackA();
    int [] a={10, 12, 15, 21, 33};
    for (int i=0; i<a.length; i++)
        st.push(new Integer(a[i]));
    int n=st.size();
    System.out.println("Pocet prvku na zasobniku: "+n);
    while (!st.isEmpty())
        System.out.print(st.pop().toString()+" ");
    System.out.println("\n"+"Pocet prvku na zasobniku: "+st.size());
}
```


- **Implementace seznamem zřetězených prvků**

- prvky se vkládají na čelo seznamu a vybírají se z čela



pop() -> a_4 :



```
public class Node {
    private Object element;
    private Node    next;

    public Node() { element=null; next=null; }

    public Node(Object element, Node next){
        this.element=element;
        this.next=next;
    }

    public Object getElement(){ return (element); }

    public Node getNext() { return(next); }

    public void setElement(Object e){ element=e; }

    public void setNext(Node n){ next=n; }

}
```

Příklad implementace zásobníku seznamem v Javě

```
public class StackL {
    private Node sp; // ukazatel na vrchol zásobníku
    private int sz; // počet objektu v zásobníku

    public StackL() {
        sp=null;
        sz=0;
    }

    public boolean isEmpty() {
        if (sp==null)
            return true;
        else return false;
    }

    public int size(){ return(sz);} // vraci počet objektu v
    zásobníku
    public void push(Object o){
        sp=new Node(o,sp);
        sz++;
    }
}
```

```
public Object pop()throws StackEmptyException {
    if (sp==null)
        throw new StackEmptyException("Stack is empty");
    else {Object tmp=sp.getElement();
        sp=sp.getNext();
        sz--;
        return(tmp);
    }
}
```

```
public Object top() throws StackEmptyException {
    if (sp==null)
        throw new StackEmptyException("Stack is empty");
    else return(sp.getElement());
}
```

```
public class StackEmptyException extends RuntimeException {
    public StackEmptyException(String err) {
        super(err);
    }
}
```

```
public static void main(String[] args) {
    StackL st=new StackL();
    int [] a={10, 12, 15, 21, 33};
    for (int i=0; i<a.length; i++)
        st.push(new Integer(a[i]));
    int n=st.size();
    System.out.println("Pocet prvku na zasobniku: "+n);
    while (!st.isEmpty())
        System.out.print(st.pop().toString()+" ");
    System.out.println("\n"+"Pocet prvku na zasobniku:"+st.size());
}
}
```

Zásobník v Java Core API

- V Java Core Api je k dispozici třída `java.util.Stack` implementující zásobník (je odděděná od třídy `Vector`)
- Používá metody :

`boolean empty()` – vrací *true* pokud je zásobník prázdný

`Object peek()` – vrací objekt z vrcholu bez jeho odebrání ze zásobníku

`Object push(O)` – vkládá objekt **O** na vrchol zásobníku

`Object pop()` – vybírá objekt ze zásobníku

`int search(O)` – vrací pozici objektu **O** na zásobníku

```

import java.util.*;

public class PokusStack {

    public static void main(String[] args) {
        Stack st=new Stack();
        int [] a={10, 12, 15, 21, 33};
        for (int i=0; i<a.length; i++)
            st.push(new Integer(a[i]));
        int n=st.size();
        System.out.println("Pocet prvku na zasobniku: "+n);
        while (!st.empty())
            System.out.print(st.pop().toString()+" ");
        System.out.println("\n"+"Pocet prvku na zasobniku:"+
            st.size());
    }
}

```

```

Pocet prvku na zasobniku: 5
33 21 15 12 10
Pocet prvku na zasobniku: 0

```

ADT Fronta (Queue)

- Fronta je datová struktura, do které se objekty vkládají a ze které se vybírají podle strategie **FIFO** (**F**irst-**I**n-**F**irst-**O**ut)

Použití:

- systémy hromadné obsluhy (rezervace vstupenek apod.)
- simulace SHO
- na systémové úrovni - fronta požadavků na procesor popř. periferní zařízení

Metody pro práci s frontou

`enqueue(o)` : Vkládá objekt **o** na konec fronty

Vstup: Objekt **Výstup:** Není

`dequeue()` : Odstraňuje objekt z čela fronty Pokud je fronta prázdná – chyba

Vstup: Není **Výstup:** Object

`size()` : Vrací počet objektů ve frontě

Vstup: Není **Výstup:** Integer

`isEmpty()` : Vrací *true* pokud je fronta prázdná

Vstup: Není **Výstup:** Boolean

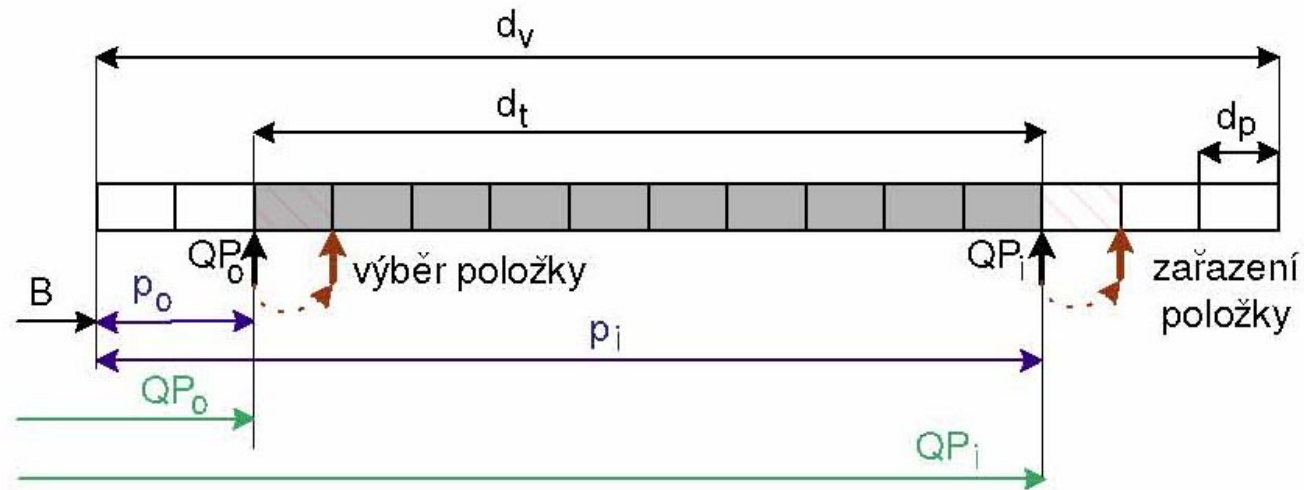
`front()` : Vrací objekt z čela fronty bez jeho odstranění. Pokud je fronta prázdná dojde k chybě.

Vstup: Není **Výstup:** Object

- Př. Posloupnost operací s frontou a jejich výsledek

Operace	Výstup	čelo ← Q ← tyl
enqueue(5)	-	(5)
enqueue(3)	-	(5,3)
dequeue()	5	(3)
enqueue(7)	-	(3,7)
dequeue()	3	(7)
front()	7	(7)
dequeue()	7	()
dequeue()	chyba	()
isEmpty()	true	()
enqueue(9)	-	(9)
enqueue(7)	-	(9,7)

- Implementace fronty polem prvků (tzv. cyklický buffer)



$$QP_i = B + (p_i + d_p) \bmod d_v$$

$$QP_0 = B + (p_0 + d_p) \bmod d_v$$

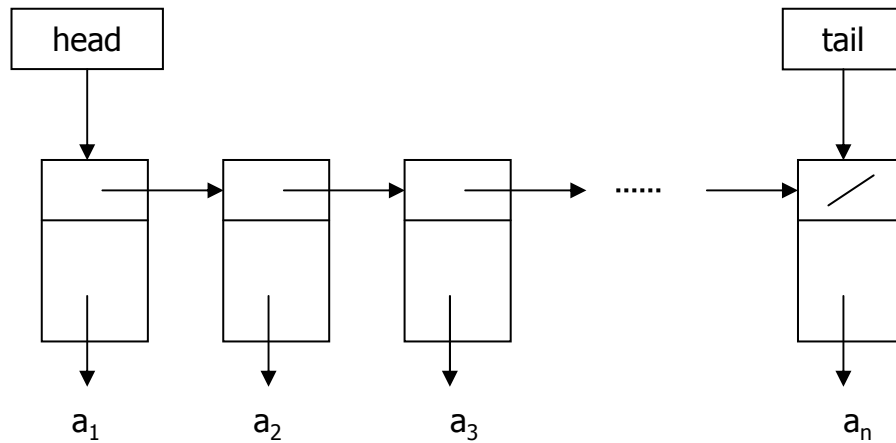
d_v – délka implementujícího vektoru

d_t – délka fronty

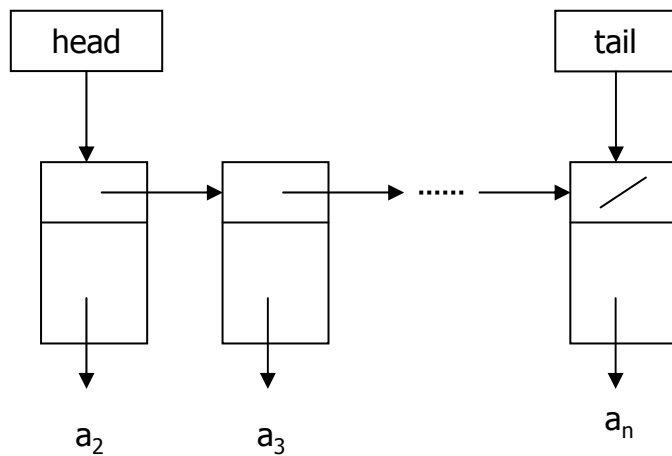
B – bazová adresa začátku fronty (index 0)

- **Implementace seznamem zřetěžených prvků**

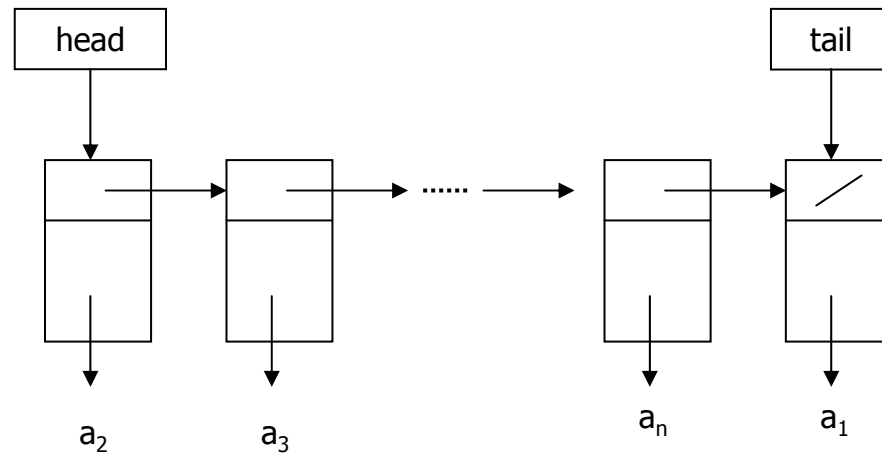
- prvky se vkládají na konec seznamu a vybírají se z čela



dequeue -> a_1 :



enqueue(a_1)



Fronta v Java Core API

- **Datový typ fronta není v Java Core API implementován, lze jej snadno implementovat pomocí třídy `java.util.LinkedList`**

Metody fronty	Odpovídající metody LinkedList
<code>size()</code>	<code>size()</code>
<code>isEmpty()</code>	<code>isEmpty()</code>
<code>front()</code>	<code>getFirst()</code>
<code>enqueue(o)</code>	<code>addLast(o)</code>
<code>dequeue()</code>	<code>removeFirst()</code>

ADT Obousměrná fronta (Double-Ended Queue)

- Obousměrná fronta je datová struktura podobná frontě, ale umožňuje vkládání a výběr prvků na obou koncích
- Použití:
 - lze ji použít v aplikacích kde používáme zároveň zásobník i frontu
 - pomocí obousměrné fronty lze implementovat některé další ADT

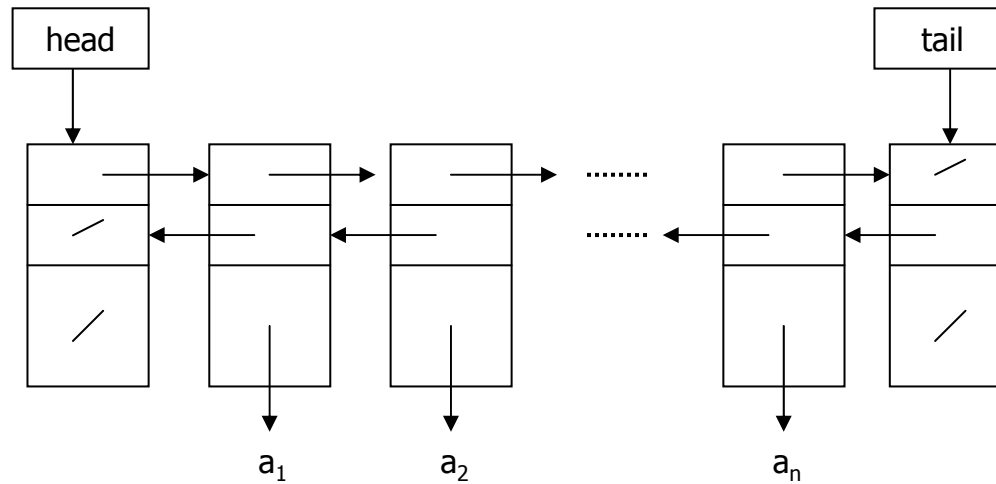
Metody pro práci s obousměrnou frontou

- `insertFirst(o)` : Vkládá objekt `o` na začátek fronty
Vstup: Objekt **Výstup:** Není
- `insertLast(o)` : Vkládá objekt `o` na konec fronty
Vstup: Objekt **Výstup:** Není
- `removeFirst()` : Odstraňuje objekt z čela fronty.
Pokud je fronta prázdná – chyba
Vstup: Není **Výstup:** Object
- `removeLast()` : Odstraňuje objekt z konce fronty.
Pokud je fronta prázdná – chyba
Vstup: Není **Výstup:** Object
- `size()` : Vrací počet objektů ve frontě
Vstup: Není **Výstup:** Integer
- `isEmpty()` : Vrací `true` pokud je fronta prázdná
Vstup: Není **Výstup:** Boolean
- `first()` : Vrací objekt z čela fronty.
Pokud je fronta prázdná – chyba
Vstup: Není **Výstup:** Object
- `last()` : Vrací objekt z konce fronty.
Pokud je fronta prázdná – chyba
Vstup: Není **Výstup:** Object

- Př. Posloupnost operací s obousměrnou frontou a jejich výsledek

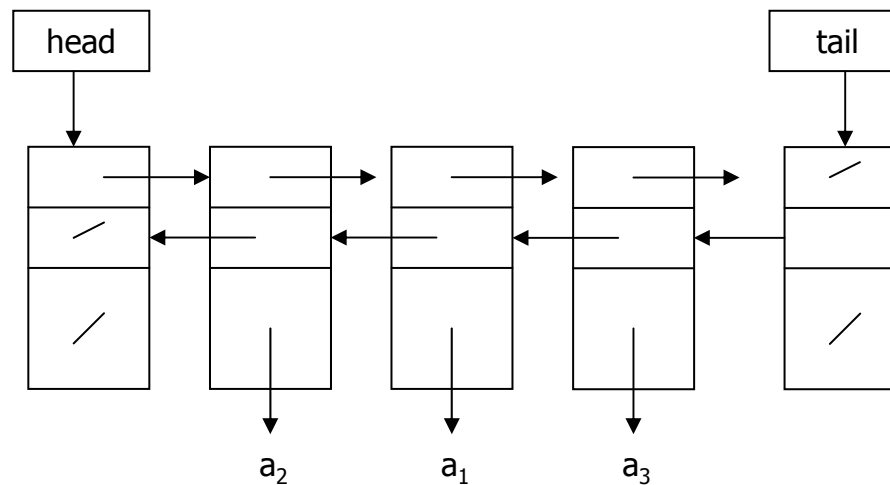
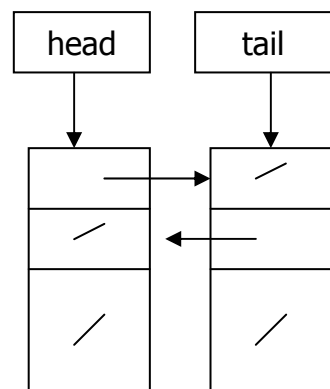
Operace	Výstup	DQ
insertFirst(3)	-	(3)
insertFirst(5)	-	(5,3)
removeFirst()	5	(3)
insertLast(7)	-	(3,7)
removeFirst()	3	(7)
removeLast()	7	(7)
removeFirst()	chyba	()
isEmpty()	true	()

- **Implementace seznamem obousměrně zřetězených prvků**



Prázdná fronta :

$\text{insertFirst}(a_1), \text{insertFirst}(a_2), \text{insertLast}(a_3) :$



Implementace zásobníku a fronty obousměrnou frontou

- Pokud máme implementovanou obousměrnou frontu, lze ji využít jako zásobník popř. jako obyčejnou frontu

Metody zásobníku	Odpovídající metody DQ
<code>size()</code>	<code>size()</code>
<code>isEmpty()</code>	<code>isEmpty()</code>
<code>top()</code>	<code>last()</code>
<code>push(o)</code>	<code>insertLast(o)</code>
<code>pop()</code>	<code>removeLast()</code>

Metody fronty	Odpovídající metody DQ
<code>size()</code>	<code>size()</code>
<code>isEmpty()</code>	<code>isEmpty()</code>
<code>front()</code>	<code>first()</code>
<code>enqueue(o)</code>	<code>insertLast(o)</code>
<code>dequeue()</code>	<code>removeFirst()</code>

ADT Vektor (Vector)

- Vektor je lineární posloupnost prvků V , která obsahuje n prvků. Každý prvek vektoru V je přístupný prostřednictvím indexu r (rank) v rozsahu $[0, n-1]$. Vektor připomíná datový typ pole, ale není to pole!!!

Metody pro práci s vektorem :

elemAtRank(r) : Vrací prvek vektoru V v pozici r , $r \in \langle 0, n-1 \rangle$ jinak chyba, n je počet prvků ve vektoru.

Vstup: Integer **Výstup**: Objekt

replaceAtRank(r, o) : Zamění prvek v pozici r prvkem o a vrátí původní prvek, $r \in \langle 0, n-1 \rangle$ jinak chyba.

Vstup: Integer r a Objekt o **Výstup**: Objekt

insertAtRank(r, o) : Vloží nový prvek o v pozici r , $r \in \langle 0, n \rangle$ jinak chyba.

Vstup: Integer r a Objekt o **Výstup**: Není

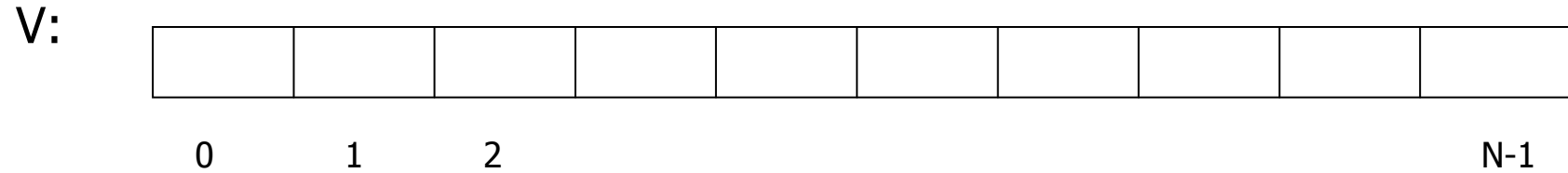
removeAtRank(r) : Odstraní a vrátí prvek v pozici r , $r \in \langle 0, n-1 \rangle$ jinak chyba.

Vstup: Integer r **Výstup**: Není

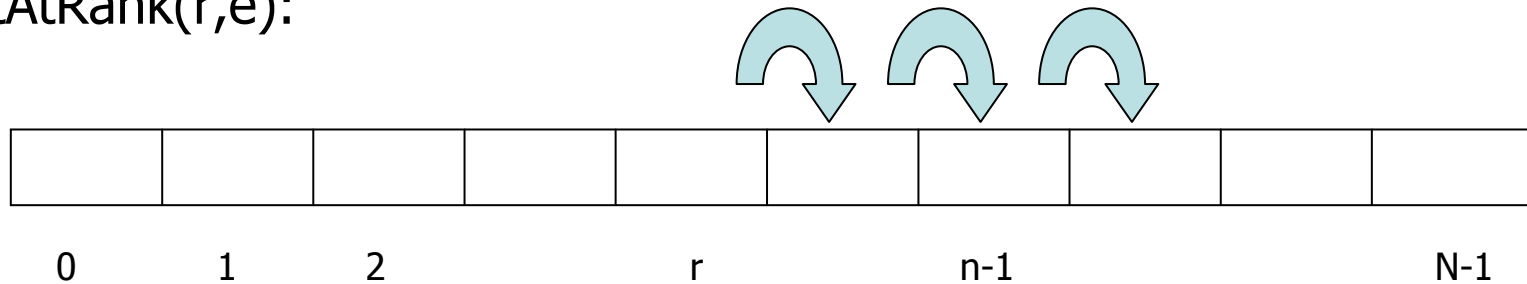
Př. Posloupnost operací s vektorem a jejich výsledek

Operace	Výstup	V
insertAtRank(0,7)	-	(7)
insertAtRank(0,4)	-	(4,7)
elemAtRank(1)	7	(4,7)
insertAtRank(2,2)	-	(4,7,2)
elemAtRank(3)	chyba	(4,7,2)
removeAtRank(1)	7	(4,2)
insertAtRank(1,5)	-	(4,5,2)
insertAtRank(1,3)	-	(4,3,5,2)
insertAtRank(4,9)	-	(4,3,5,2,9)
elemAtRank(2)	5	(4,3,5,2,9)

- Implementace vektoru polem pevné délky



insertAtRank(r,e):



1. Vytvoření místa pro nový prvek přesunem prvků
2. Vložení prvku **e** na pozici **r**

Implementace vektoru polem proměnné délky

```
public class ArrayVector {
    private Object[] a;           // Array storing the elements of the vector
    private int capacity = 16;    // Length of array a
    private int size = 0;         // Number of elements stored in the vector
    // Constructor
    public ArrayVector() { a = new Object[capacity]; } // O(1) time
    // Accessor methods
    public Object elemAtRank(int r) { return a[r]; } // O(1) time
    public int size() { return size; }
    public boolean isEmpty() { return size() == 0; } // O(1) time
    // Modifier methods
    public Object replaceAtRank(int r, Object e) { // O(1) time
        Object temp = a[r];
        a[r] = e;
        return temp;
    }
    public Object removeAtRank(int r) { // O(n) time
        Object temp = a[r];
        for (int i=r; i<size-1; i++) // Shift elements down
            a[i] = a[i+1];
        size--;
        return temp;
    }
}
```

```
public void insertAtRank(int r, Object e) {           // O(n) time
    if (size == capacity) {                          // An overflow
        capacity *= 2;
        Object[] b = new Object[capacity];
        for (int i=0; i<size; i++)
            b[i] = a[i];
        a = b;
    }
    for (int i=size-1; i>=r; i--)                    // Shift elements up
        a[i+1] = a[i];
    a[r] = e;
    size++;
}
}
```

ADT Seznam

- Seznam je lineární posloupnost prvků, které jsou propojeny ukazateli (pointery). Prvek se do seznamu vkládá na určitou pozici (obdoba indexu u vektoru).

Metody pro práci se seznamem :

first() : Vrací odkaz na první prvek seznamu S, je-li S prázdný nastává chyba.
Vstup: Není **Výstup:** Pozice (ukazatel na objekt)

last() : Vrací odkaz na poslední prvek seznamu S, je-li S prázdný nastává chyba.
Vstup: Není **Výstup:** Pozice (ukazatel na objekt)

before(p) : Vrací odkaz na prvek před prvkem na pozici **p**, chyba je-li p ukazatel na první prvek.
Vstup: Pozice **Výstup:** Pozice

after(p) : Vrací odkaz na prvek před prvkem na pozici **p**, chyba je-li p ukazatel na první prvek.
Vstup: Pozice **Výstup:** Pozice

- isFirst(p):** Vrací true pokud **p** ukazuje na první prvek
Vstup: Pozice **Výstup:** Boolean
- isLast(p):** Vrací true pokud **p** ukazuje na poslední prvek
Vstup: Pozice **Výstup:** Boolean
- replaceElement(p,o):** Zamění prvek v pozici **p** prvkem **o** a vrátí původní prvek
Vstup: Pozice **p** Objekt **o** **Výstup:** Objekt
- swapElements(p,q):** Zamění prvek v pozici **p** s prvkem v pozici **q**
Vstup: Pozice **p,q** **Výstup:** Není
- insertFirst(o):** Vloží prvek **o** na začátek seznamu
Vstup: Objekt **o** **Výstup:** Pozice – ukazatel na nově vložený prvek
- insertLast(o):** Vloží prvek **o** na konec seznamu
Vstup: Objekt **o** **Výstup:** Pozice – ukazatel na nově vložený prvek
- insertBefore(p,o):** Vloží prvek **o** před prvek na pozici **p**, chyba je-li **p** ukazatel na první prvek
Vstup: Pozice **p**, Objekt **o** **Výstup:** Pozice – ukazatel na nově vložený prvek

insertAfter(p,o): Vloží prvek **o** za prvek na pozici **p**, chyba je-li **p** ukazatel na první prvek

Vstup:Pozice **p**, Objekt **o** **Výstup:** Pozice ukazatel na nově vložený prvek

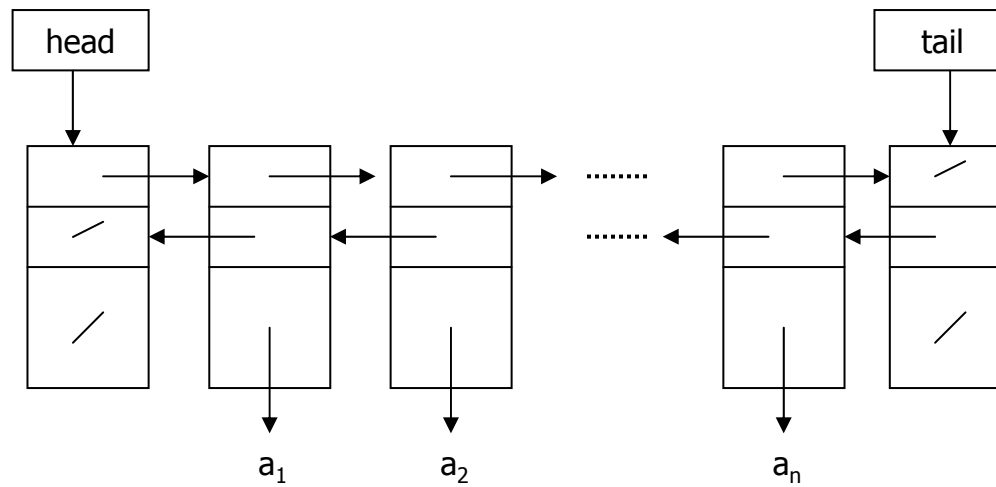
remove(p,o): Odstraní prvek v pozici **p** ze seznamu první prvek

Vstup:Pozice **p**, Objekt **o** **Výstup:** Pozice ukazatel na nově vložený prvek

Př. Posloupnost operací se seznamem a jejich výsledek

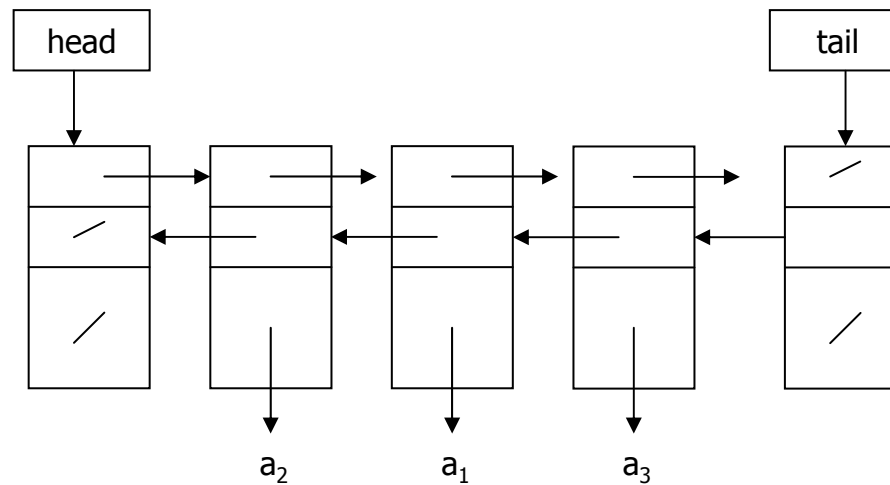
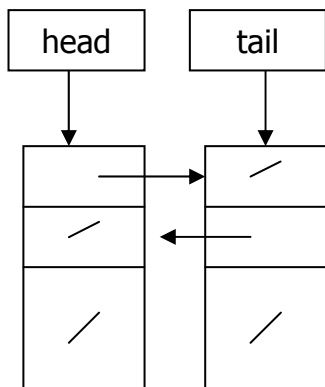
Operace	Výstup	V
insertFirst(8)	$p_1(8)$	(8)
insertAfter($p_1, 5$)	$p_1(5)$	(8,5)
insertBefore($p_2, 3$)	$p_3(3)$	(8,3,5)
insertFirst(9)	$p_4(9)$	(9,8,3,5)
before(p_3)	$p_1(8)$	(9,8,3,5)
last()	$p_2(8)$	(9,8,3,5)
remove(p_4)	9	(8,3,5)
swapElements(p_1, p_2)	-	(5,3,8)
replaceElement($p_3, 7$)	3	(5,7,8)
insertAfter(first(),2)	$p_5(2)$	(5,2,7,8)

- **Implementace seznamem obousměrně zřetěžených prvků**



Prázdný seznam :

$\text{insertFirst}(a_1), \text{insertFirst}(a_2), \text{insertLast}(a_3) :$



Metody třídy java.util.LinkedList

Method Summary	
void	add (int index, Object element) Inserts the specified element at the specified position in this list.
boolean	add (Object o) Appends the specified element to the end of this list.
boolean	addAll (Collection c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	addAll (int index, Collection c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	addFirst (Object o) Inserts the given element at the beginning of this list.
void	addLast (Object o) Appends the given element to the end of this list.
void	clear () Removes all of the elements from this list.

<code>Object</code>	<code>clone()</code> Returns a shallow copy of this <code>LinkedList</code> .
<code>boolean</code>	<code>contains(Object o)</code> Returns <code>true</code> if this list contains the specified element.
<code>Object</code>	<code>get(int index)</code> Returns the element at the specified position in this list.
<code>Object</code>	<code>getFirst()</code> Returns the first element in this list.
<code>Object</code>	<code>getLast()</code> Returns the last element in this list.
<code>int</code>	<code>indexOf(Object o)</code> Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
<code>int</code>	<code>lastIndexOf(Object o)</code> Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
<code>ListIterator</code>	<code>listIterator(int index)</code> Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
<code>Object</code>	<code>remove(int index)</code> Removes the element at the specified position in this list.
<code>boolean</code>	<code>remove(Object o)</code> Removes the first occurrence of the specified element in this list.
<code>Object</code>	<code>removeFirst()</code> Removes and returns the first element from this list.
<code>Object</code>	<code>removeLast()</code> Removes and returns the last element from this list.
<code>Object</code>	<code>set(int index, Object element)</code> Replaces the element at the specified position in this list with the specified element.
<code>int</code>	<code>size()</code> Returns the number of elements in this list.
<code>Object []</code>	<code>toArray()</code> Returns an array containing all of the elements in this list in the correct order.
<code>Object []</code>	<code>toArray(Object[] a)</code> Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.