ПОДАТОЧНИ СТРУКТУРИ И АНАЛИЗА НА АЛГОРИТМИ

ЧАС 3: ADT, НИЗИ, ПОВРЗАНИ ЛИСТИ

АУДИТОРИСКИ ВЕЖБИ

БОЈАНА ВЕЛИЧКОВСКА

НИЗИ

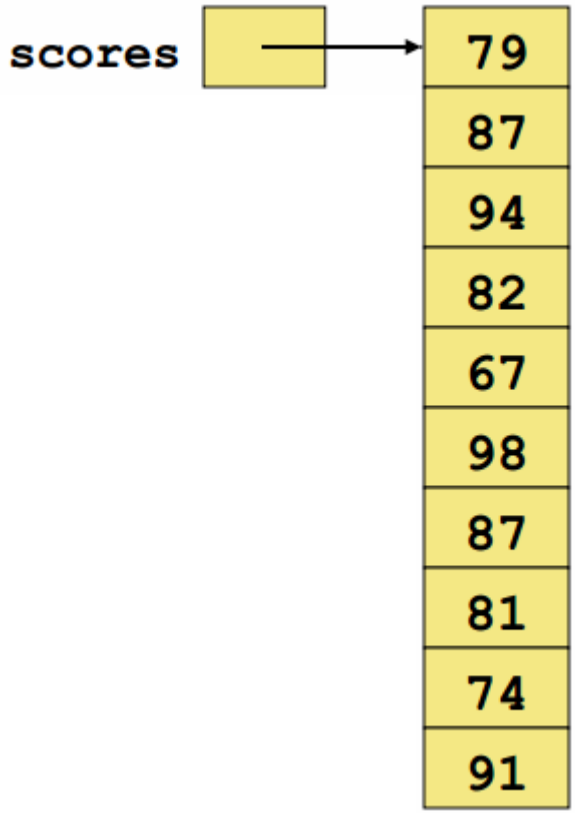
- Чуваат вредости од ист тип
- Во Java, низата е објект и мора да биде инстанцирана
- Декларирање:
  - int[ ] scores = new int [10]; //се полни со нули

  или
  - int [ ] scores = {79, 87, 94, 82, 67, 98, 87, 81, 74, 91};
- Итерирање:
  - for(int i = 0; i < scores.length; i++)

    System.out.println(scores[i]);
  - **for (int score : scores) //for each циклус**

    System.out.println (score);

  Се користи само кога треба да се процесираат сите елементи во низата



ОСНОВНИ ОПЕРАЦИИ СО НИЗИ

- Креирање празна низа
- Пребарување на вредност според индекс
- Пребарување на индекс според вредност
- Додавање на елемент
- Бришење на елемент

НИЗИ

- Откако е креирана, низата има фиксна големина

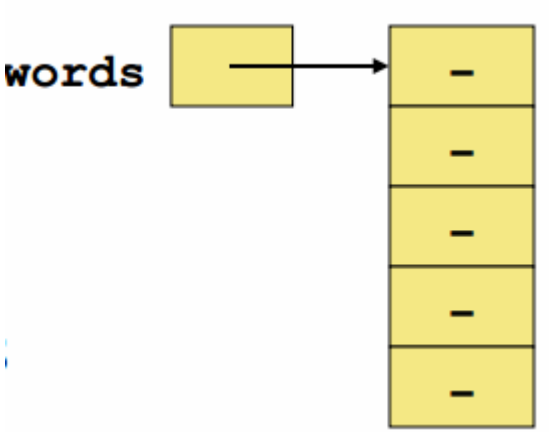
```
int[ ] scores = new int [10];
for(int i=0;i<20; i++)
    System.out.println(scores[i]);
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
```

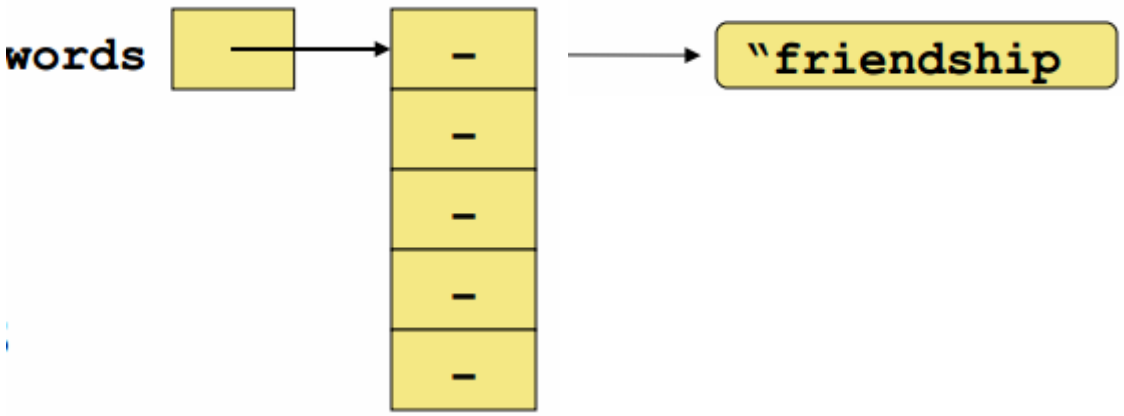
НИЗИ

- Низи од објекти

String[ ] words = new String[5];



words[0]="friendship";



КЛАСАТА ARRAYS

- Класата е дефинирана во пакетот java.util.Arrays
- Постојат веќе дефинирани функции за сортирање и пребарување на низи

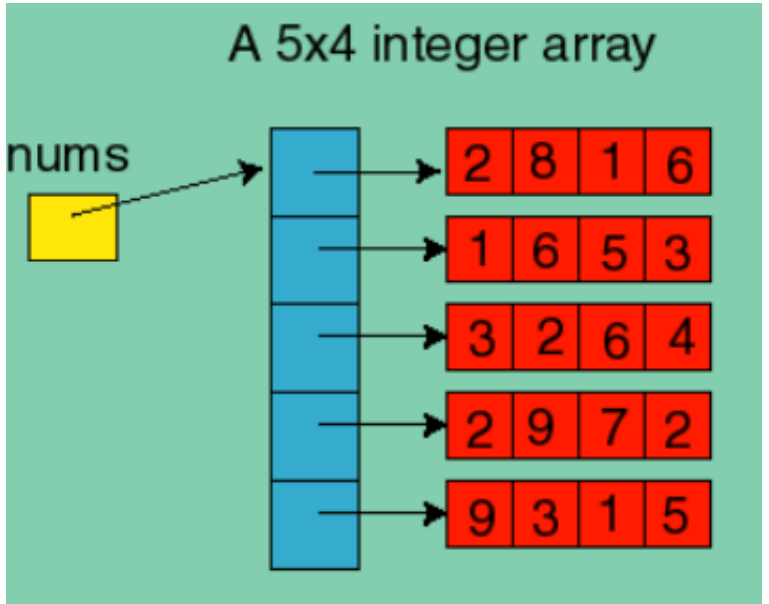
```
double[ ] nums = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
Arrays.sort(nums);   //ја сортира цела низа
Arrays.sort(nums,2,4);  //сортира само опсег
int index = Arrays.binarySearch(nums, 4.4) //го враќа индексот на елементот 4.4
Arrays.fill(nums,5); //ја полни цела низа со 5ки
Arrays.fill(nums,1,3,8); //елементите со опсег 1-3 ги поставува на 8
```

МАТРИЦИ

```
int [ ][ ] nums = new int[5][4];
```

или

```
int [ ][ ] nums = {{2,8,1,6},{1,6,5,3}, {3,3,6,4},{2,9,7,2},{9,3,1,5}};
```



НИЗИ (ПРЕДНОСТИ И НЕДОСТАТОЦИ)

- +
  - Брз, случаен пристап до елементи, константно време O(1)
  - Мемориски ефикасни (нема **overhead**)
- -
  - Големината мора да биде однапред позната и е фиксна (статичка)
  - Вметнувањето и бришењето на елементи е бавно (шифтирања)

ДИНАМИЧКИ НИЗИ (ПРЕДНОСТИ И НЕДОСТАТОЦИ)

- +
  - Брз, случаен пристап до елементи, константно време O(1)
  - Мемориски ефикасни (нема overhead)
- -
  - Вметнувањето и бришењето на елементи е бавно (шифтирања)
  - Класата ArrayList динамички расте, но како?
    - Временска ефикасност???
    - Скапо копирање на елементи

КЛАСАТА ARRAYLIST

- ArrayList е:
  - Класа во стандардната Java библиотека java.util.ArrayList
  - Може да чува објекти од секаков тип
  - Големината на низата може да се менува за време на извршување на програмата (за разлика од обичните низи)

КЛАСАТА ARRAYLIST

- Креирање на објект:
  - ArrayList list = new ArrayList();

  //Елементите се од типот Object, ако не е специфицирано поинаку
- Конструктори:
  - ArrayList()

  //Креира празна листа со почетен капацитет 10
  - ArrayList(int initialCapacity)

  // Креира празна листа со зададен капацитет

КЛАСАТА ARRAYLIST

- void add(int index, Object element)

//Го додава објектот element на позиција index

- boolean add(Object o)

//Го додава објектот o на крај на низата

- list.add(“something”);

  list.add(1,”sth else”);

КЛАСАТА ARRAYLIST

- int size() – колку елементи моментално има во низата
  - int howMany = list.size();
- Object set(int index, Object element)

//Го поставува објектот element на позиција index

  - list.set(index, "something else");
- Object get(int index)

//Го враќа елементот на позиција index

  - String thing = (String) list.get(index);

До елемент не се пристапува со [ ]

КЛАСАТА ARRAYLIST

```
public static void main( String[ ] args)
{
 ArrayList myInts = new ArrayList();
 System.out.println( "Size of myInts = " + myInts.size());
 for (int k = 0; k < 10; k++)
 myInts.add( 3 * k );
 myInts.set( 6, 44 );
 System.out.println( "Size of myInts = " + myInts.size());
 for (int k = 0; k < myInts.size(); k++)
 System.out.print( myInts.get( k ) + "," );
}
```

```
// izlez:
Size of myInts = 0
Size of myInts = 10
0, 3, 6, 9, 12, 15, 44, 21, 24, 27
```

КЛАСАТА ARRAYLIST

- Object remove(int index)

//Го отсранува елементот на позиција index

- void removeRange(int fromIndex, int toIndex)

//Ги отстранува сите елементи во специфираниот опсег

- void clear()

//Ги отстранува сите елементи

- Object clone()

//Враќа копија на низата

КЛАСАТА ARRAYLIST

- int indexOf(Object elem)

//Го враќа индексот на првото појавување на елементот elem

- int lastIndexOf(Object elem)

//Го враќа индексот на последното појавување на елементот elem

FOR-EACH ЦИКЛУС

```
import java.util.Date;
public class ForEach
{  public static void main(String[ ] args)
 {  ArrayList list = new ArrayList();
 list.add(new Date(1, 1, 1000));
 list.add(new Date(7, 4, 1776));
 list.add(new Date(9, 1, 2011));
for( Object i : list )
 System.out.println( i );  } }
```

```
// izlez:
1/1/1000
7/4/1776
9/1/2011
```

ARRAY VS. ARRAYLIST

- +
  - Големината на ArrayList може да се менува
  - Постојат голем број моќни, веќе дефинирани методи
  - Нема потреба да се грижиме за големината, постои метод
- -
  - Помалку ефикасни од низите
  - Не користат [ ] за пристап

КЛАСАТА ARRAYLIST

- Вака дефинираните објекти од типот ArrayList работат со елементи од типот Object
  - Мора да се употребува кастирање
  - Грешка при извршување ако мешаме типови на податоци

```
ArrayList list = new ArrayList();
 populateNumbers(list);
private static void populateNumbers(ArrayList list)
 {  list.add(new Integer(1));
    list.add(new Integer(2));
    list.add("hello");    }
```

Грешка при **извршување**:
Exception in thread "main"
java.lang.ClassCastException:
java.lang.String at
com.agiledeveloper.Test.main(Test.java:17)

КЛАСАТА ARRAYLIST

```
ArrayList <Integer> list = new ArrayList<Integer>();
populateNumbers(list);
private static void populateNumbers(ArrayList<Integer> list)
 {
 list.add(new Integer(1));
 list.add(new Integer(2));
 list.add("hello");
 }
```

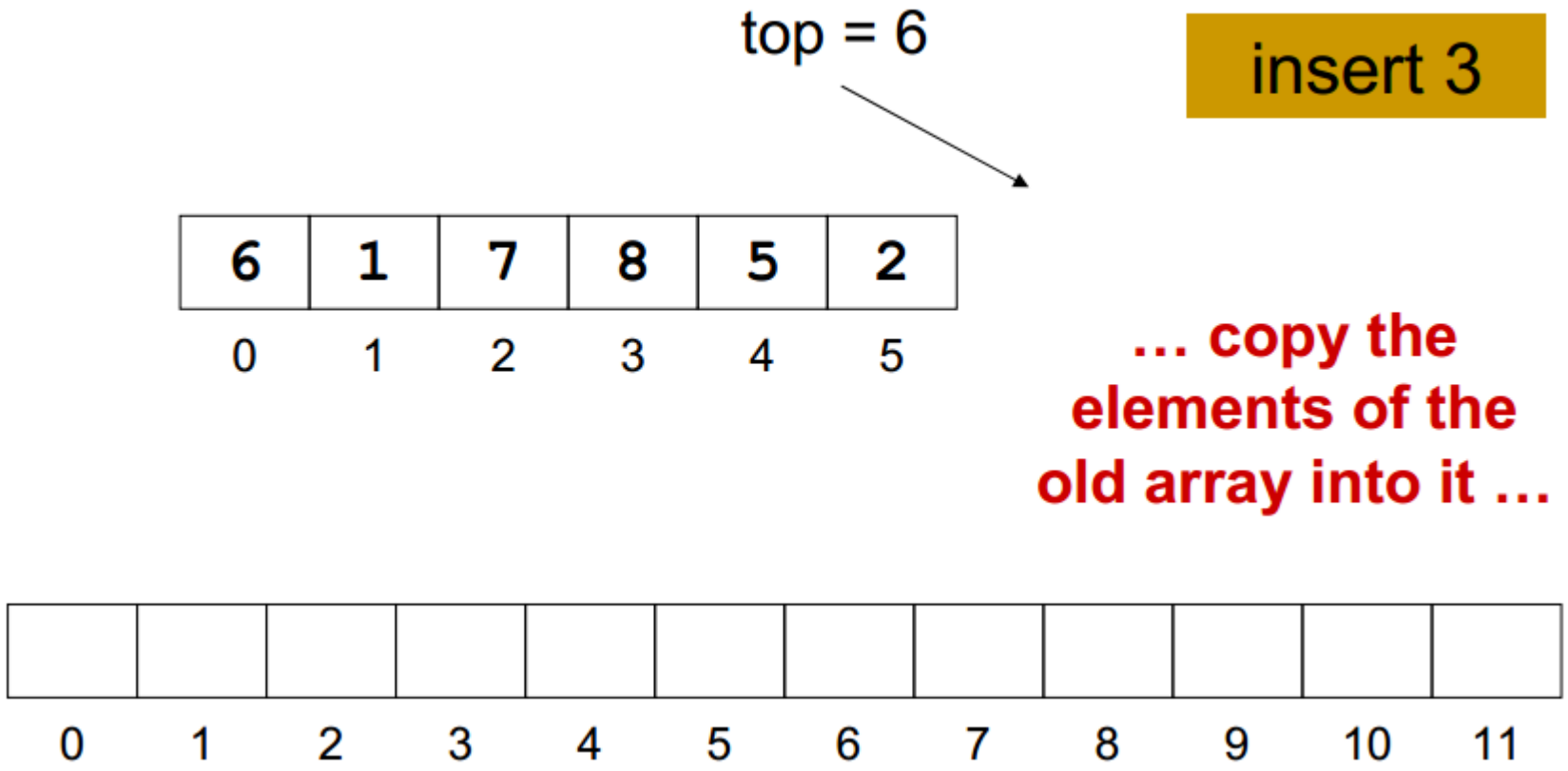
Грешка при **компајлирање**:

```
Test.java:26: cannot find symbol
symbol : method add(java.lang.String)
location: class java.util.ArrayList<java.lang.Integer>
 list.add("hello");
 ^
1 error
```

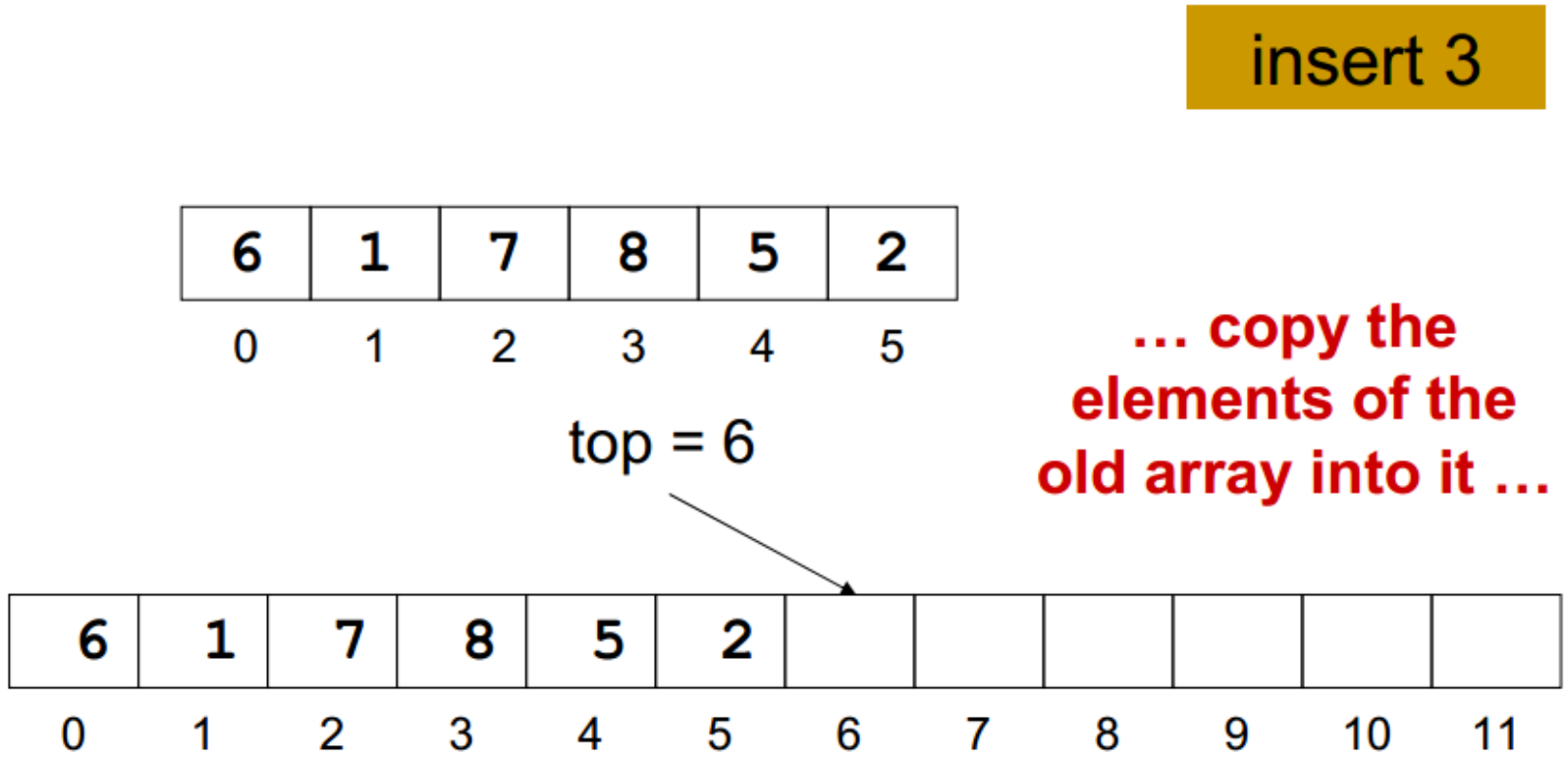
КЛАСАТА ARRAYLIST

```
ArrayList<String> band = new ArrayList<String>();
band.add ("Paul");
int location = band.indexOf ("Paul");
band.remove (location);
boolean success = band.remove ("Paul");
for (String str : band)
System.out.println (str);
```

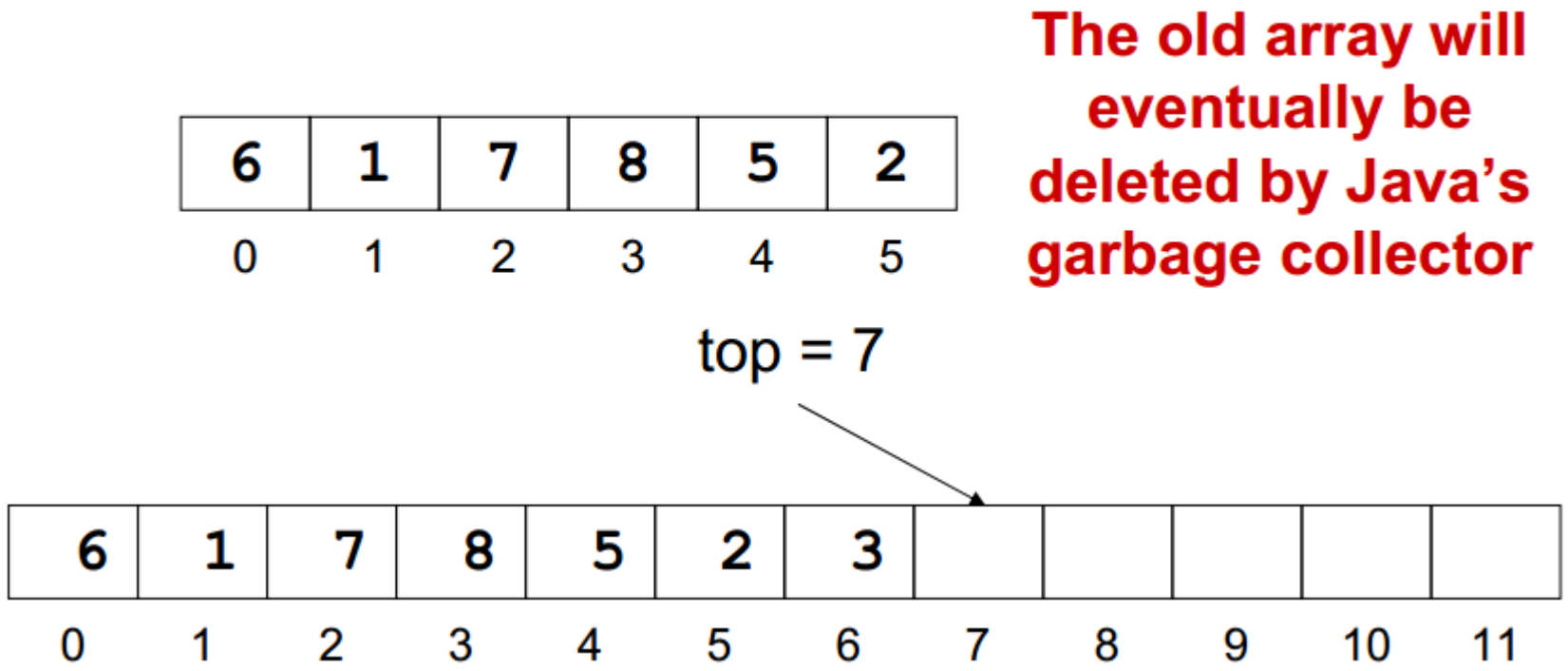
ДИНАМИЧКО РАСТЕЊЕ КАЈ ARRAYLIST



ДИНАМИЧКО РАСТЕЊЕ КАЈ ARRAYLIST



ДИНАМИЧКО РАСТЕЊЕ КАЈ ARRAYLIST



ЗАДАЧА 1

- Да се напише програма која собира два полиноми

$$f(x) = ax^3 + bx + c$$

$$g(x) = mx^4 + nx^3 + ox^2 + px + q$$

Полиномите ги сместуваме во низа на следниов начин:

f = {3, a, 1, b, 0, c}

g = {4, m, 3, n, 2, o, 1, p, 0, q}

rez = {4, m, 3, a+n, 2, o, 1, b+p, 0, c+q}

ЗАДАЧА 1

- Што доколку ги чуваме само коефициентите?
  - Ако имаме ваков полином:

$$f(x) = ax^{16} + bx + c$$

f = {a, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, b, c}

Vs.

f = {16, a, 1, b, 0, c}

ЗАДАЧА 1

```
package soberipolinomi;

import java.util.ArrayList;

import java.util.Scanner;


public class SoberiPolinomi {

    public static void main(String[] args) {

        /* креираме три генерички објекти од класата ArrayList */

        ArrayList<Integer> prvPolinom = new ArrayList();

        ArrayList<Integer> vtorPolinom = new ArrayList();

        ArrayList<Integer> rezultat;

        Scanner input = new Scanner(System.in);
```

ЗАДАЧА 1

```
System.out.println("Vnesuvajte gi eksponentite i koeficientite na prviot polinom");
        vnesiElementi(prvPolinom);
        pechatiPolinom(prvPolinom);
System.out.println("Vnesuvajte gi eksponentite i koeficientite na vtoriot polinom");
        vnesiElementi(vtorPolinom);
        pechatiPolinom(vtorPolinom);

        rezultat = soberi(prvPolinom,vtorPolinom);
        pechatiPolinom(rezultat);
    }
```

ЗАДАЧА 1

```
/* следниве методи се статички за да може да ги повикаме без да
 * креираме објект од класата
 */
static void pechatiPolinom(ArrayList al)
{
    for(int i=0;i<al.size();i+=2)
        System.out.print("("+al.get(i)+", "+al.get(i+1)+")");
    System.out.println();

}
```

ЗАДАЧА 1

```
static void vnesiElementi(ArrayList al)
   {  Scanner input = new Scanner(System.in);
      do     {
        System.out.println("Vnesete eksponent");
        al.add(input.nextInt());
        System.out.println("Vnesete koeficient");
        al.add(input.nextInt());
        if(al.get(al.size()-2)==0) break;    // ako eksponentot koj e vnesen e 0 se izleguva
        System.out.println("Ako zavrshivte so vnesuvanje vnesete 0 za kraj"); // пр. a^2+ b^1
        if(input.nextInt()==0) break;
      }while(true);
   }
```

ЗАДАЧА 1

```
static ArrayList soberi(ArrayList<Integer> p1,ArrayList<Integer> p2)
 {   ArrayList<Integer> rez=new ArrayList(2);
     int i=0,j=0;
     while(i<p1.size()&&j<p2.size())
     {   if(p1.get(i)==p2.get(j)) /* ако експонентите им се исти */
      {
            rez.add(p1.get(i));
             rez.add(p1.get(i+1)+p2.get(j+1));
             i+=2;
             j+=2;
        }
```

ЗАДАЧА 1

```
else
        if(p1.get(i)>p2.get(j))   //ako prviot eksponent e pogolem od vtoriot
        {     rez.add(p1.get(i));
        rez.add(p1.get(i+1));
        i+=2;
        }
    else   // ako vtoriot eksponent e pogolem od prviot
        {  rez.add(p2.get(j));
        rez.add(p2.get(j+1));
        j+=2;
        }
}
```

ЗАДАЧА 1

```
/*ако некоја од низите не е изминатa до крај */
    while(i<p1.size())
     {     rez.add(p1.get(i));
          rez.add(p1.get(i+1));
          i+=2;
     }
      while(j<p2.size())
     {    rez.add(p2.get(j));
          rez.add(p2.get(j+1));
          j+=2;
      }
     return rez;
  } }
```

ADT (ABSTRACT DATA TYPES)

- ADT – множество од објекти заедно со множество од операции кои се извршуваат врз нив
- Објектите како листи, дрва, магацини, графови, може да се разгледуваат како апстрактни податочни типови, исто како цели броеви, реални броеви, знакови, итн.
- Јава дозволува имплементација на ADT, со соодветно криење на имплементациските детали. Програмата кога сака да изврши операција со даден ADT единствено треба да повика определен метод.
- Не постои правило кои операции треба да бидат поддржани од некој ADT, тоа е одлука на дизајнерот

LIST ADT

- Општ облик на листа: A0, A1, …, An-1, големина n, големина 0 => празна листа
- Ai е следбеник на Ai-1 (i<n), а претходник на Ai+1 (i>0)
- Позиција на елементот Ai е i
- Операции: **printList, find, findKth, insert, remove** (оставено на програмерот)

ЕДНОСТАВНА ИМПЛЕМЕНТАЦИЈА НА ЛИСТИ

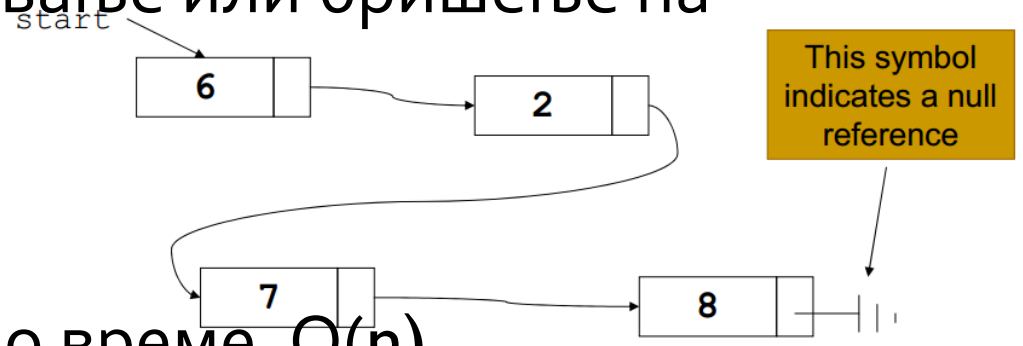
- Сите горенаведени операции може да бидат имплементирани со помош на низа.

```
int [ ] arr = new int[ 10 ];
...
// Later on we decide arr needs to be larger.
int [ ] newArr = new int[ arr.length * 2 ];
for( int i = 0; i < arr.length; i++ )
newArr[ i ] = arr[ i ];
arr = newArr;
```

- **printList** (линеарно време) , **findKth** (константно време), **insert, remove** (најдобро и најлошо време, последна позиција,прва позиција)
- Кога е подобро да се користи друга податочна структура – ако додавањата или отстранувањата на елемент почесто се случуваат на почеток и низ листата

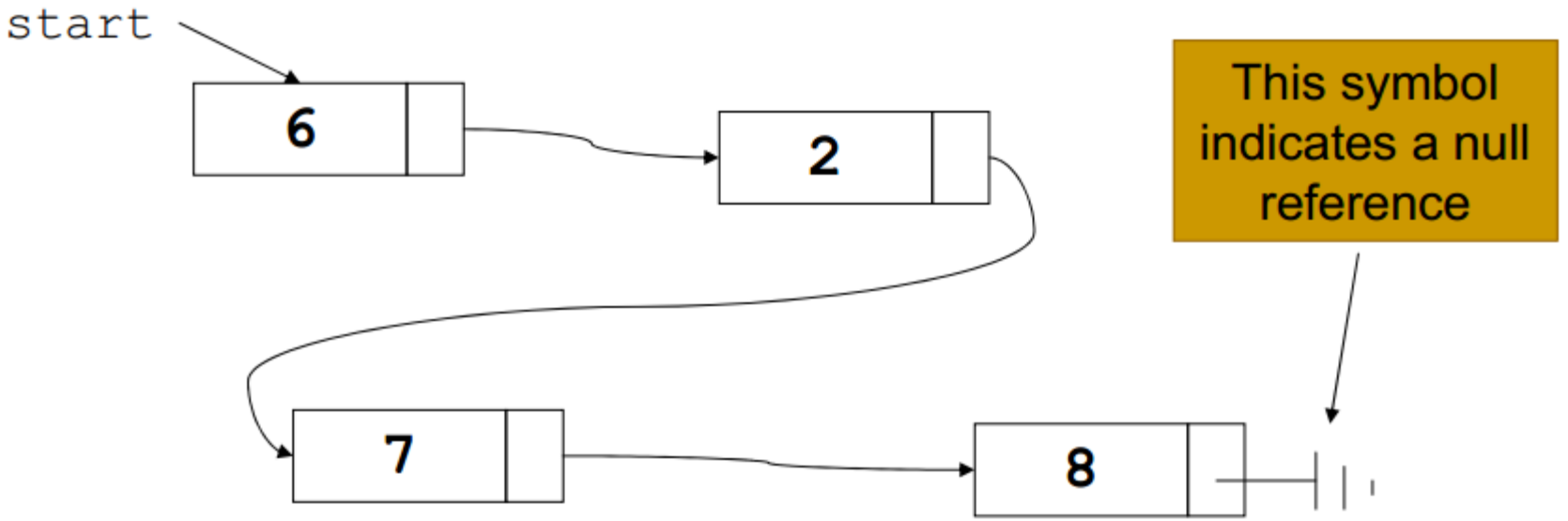
ПОВРЗАНИ ЛИСТИ (ПРЕДНОСТИ И НЕДОСТАТОЦИ)

- +
  - Динамички растат (не бараат скапи копирања на елементи)
  - Нема шифтирање на елементи при додавање или бришење на елемент
- -
  - Пристапот до елемент не е во константно време, O(n)
  - Мора да чуваат врски (линкови) за да го одржат логичкиот распоред на елементите (мемориски **overhead**)



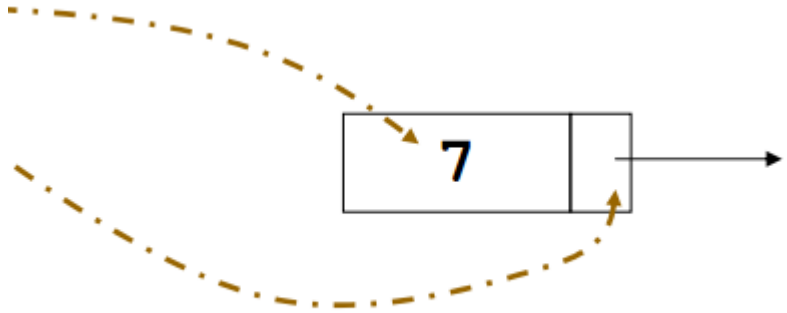
ЕДИНЕЧНО ПОВРЗАНИ ЛИСТИ

- Колекција од линеарно поврзани јазли
- Физичката локација на јазлите во меморија е случајна
- Редоследот е утврден од врските (линковите)



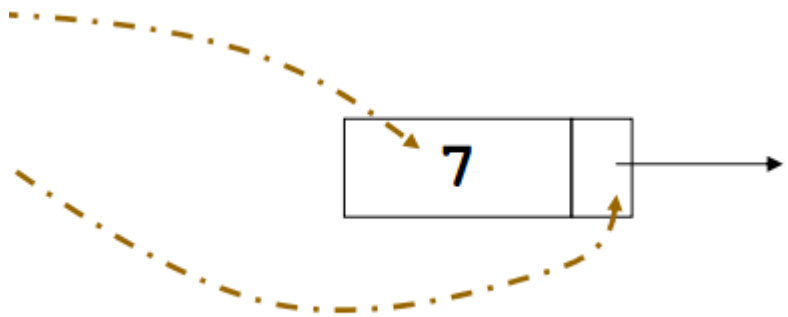
ЕДИНЕЧНО ПОВРЗАНИ ЛИСТИ

- Секој јазол содржи два елементи:
  - Податок
  - Покажувач (врска) кон наредниот јазол



ЕДИНЕЧНО ПОВРЗАНИ ЛИСТИ

```
class Node <E>
{
protected E data;
protected Node <E> next;
}
```

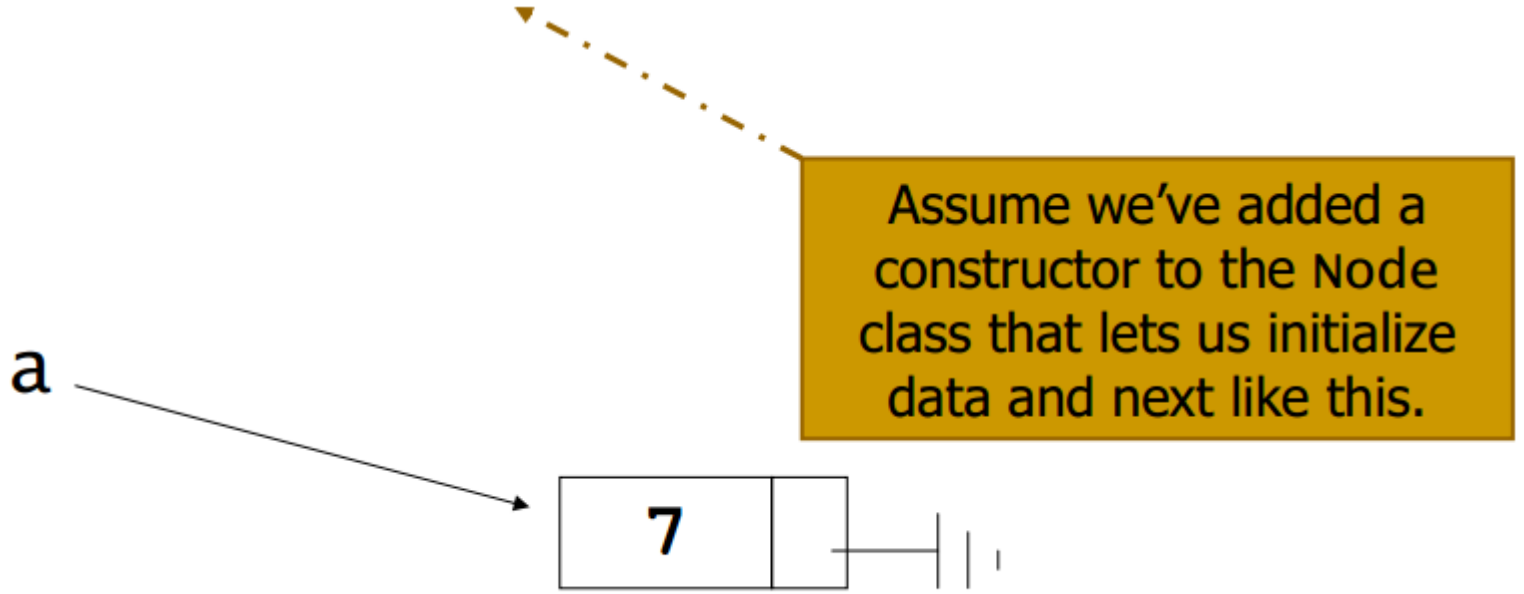


ЕДИНЕЧНО ПОВРЗАНИ ЛИСТИ (ЈАЗОЛ)

```
class Node <E> {
    protected E data;
    protected Node <E> next;
    public Node()
        { data = null; next = null; }
    public Node(E data, Node <E> next)
        { this.data = data; this.next = next; }

}
```

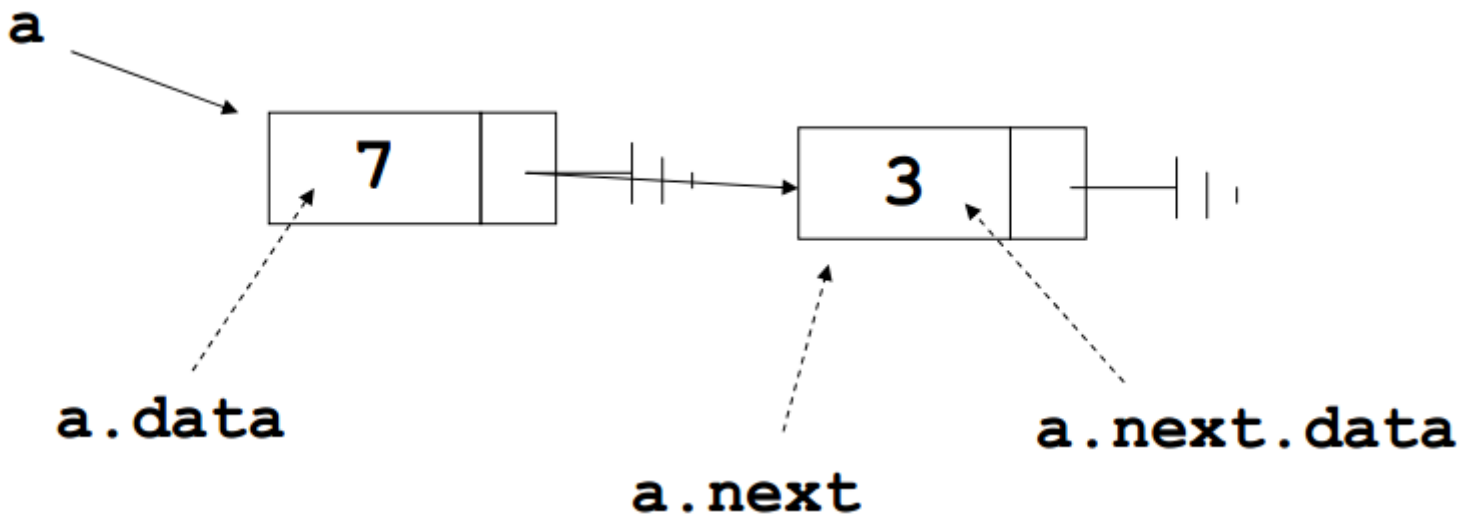
КРЕИРАЊЕ НА ЕДИНСТВЕН ЈАЗОЛ

Node <Integer> a = new Node <Integer> (7, null);



ПОВРЗУВАЊЕ НА ДВА ЈАЗЛИ

a. next = new Node <Integer> (3,null);



КРЕИРАЊЕ НА ПРАЗНА ЕДИНЕЧНО ПОВРЗАНА ЛИСТА

```
class SLinkedList <E> {
    private Node<E> head;
    public SLinkedList(){
        head = null;
}... }
```

```
SLinkedList <Integer> list = new SLinkedList();
```

ОПЕРАЦИИ КАЈ ЕДИНЕЧНО ПОВРЗАНА ЛИСТА

- Креирај празна листа
- Додади јазол (на почеток, на крај, на позиција)

```
public void insertFirst(E e)
public void insertLast (E e)
```

- Избриши јазол (на почеток, на крај, на позиција)

```
public void deleteFirst ()
```

- Измини ја листата
- Врати ја големината на листата

ЕДИНЕЧНО ПОВРЗАНА ЛИСТА

```
class SLinkedList <E> {

   private Node<E> head;

   public SLinkedList(){
        head = null;
    }

   public Node<E> getHead()
   {
       return head;
   }
```

ДОДАВАЊЕ НА ЈАЗОЛ НА ПОЧЕТОК НА Е.П. ЛИСТА

```
public void insertFirst(E e)
   {
       Node <E> first = new Node (e,head);
       head=first;
   }
```

ДОДАВАЊЕ НА ЈАЗОЛ НА КРАЈ НА Е.П. ЛИСТА

```
public void insertLast (E e)
 {
    if(head!=null)
    {
       Node <E> tmp=head;
       while(tmp.next!=null)
          tmp=tmp.next;
       Node <E> last=new Node(e,null);
       tmp.next=last;
    }
    else
       this.insertFirst(e);
 }
```

ПЕЧАТЕЊЕ НА Е.П. ЛИСТА

```
public void printList()
 {
    Node <E> tmp = this.head;

    while(tmp.next!=null)
    {
       System.out.print(tmp.data+" -> ");
       tmp=tmp.next;
    }
    System.out.println(tmp.data);
 }
```

УПОТРЕБА НА Е. П. ЛИСТА

```
public static void main(String[] args) throws NodeException {
    SLinkedList <Integer> list = new SLinkedList();
    /* kreirame prazna lista */
    list.insertLast(10);
    list.insertLast(11);
    list.insertLast(13);
    list.printList();
    list.deleteFirst();
    list.printList();
    list.insertFirst(1);
    list.insertFirst(2);
    list.insertFirst(3);
    list.printList();          }
```

Izlez:

10 -> 11 -> 13

11 -> 13

3 -> 2 -> 1 -> 11 -> 13

ЗАДАЧА 1 (1)

- Да се напише функција која ги превртува сите врски во единечно поврзана листа

```
public void prevrti()
   {       if(head!=null)
      {    Node <E> current = head;              /* current е јазолот на кој моментално му ги менуваме врските */
           Node <E> previous = null;       /* previous е јазолот со кој ќе го повземе pom */
           Node <E> next = null;                 /* next е јазолот следбеник на pom во оригиналната листа */
         while(current!=null)
         {   next=current.next;
             current.next=previous;
             previous=current;
             current=next;
         }
          head = previous;
      }
   }
```

ЗАДАЧА 1 (2)

```
public static void main(String[] args) {

    SLinkedList <Integer> list = new SLinkedList();
    list.insertFirst(15);
    list.insertLast(18);
    list.insertLast(25);
    list.insertLast(30);

    list.prevrti();
    Node <Integer> pom = list.getHead();
    while(pom!=null)
    {
        System.out.println(pom.data);
        pom=pom.next;
    }
}
```

ЗАДАЧА 2 (1)

- Да се напише програма која во произволна единечно поврзана листа ќе ги исфрли сите јазли што се повторуваат. За секој јазол да се води информација колку пати се повторувал

```
class Node <E extends Comparable<E>> { // klasa so funkcii za compare
    protected E data1; // podatoci - info
    protected int data2; //podatoci - info
    protected Node <E> next; // sleden jazol
    public Node() //default konstruktor
     {    data1 = null;
         data2=0;
         next = null;      }
    public Node(E data1, int data2, Node <E> next) // parametarski konstruktor
        {    this.data1 = data1;
             this.data2=data2;
             this.next = next;       }
}
```

ЗАДАЧА 2 (2)

```
public class SLinkedListWithTwoInfos <E extends Comparable<E>>  { // definiranje na lista
   private Node <E> head;
   public SLinkedListWithTwoInfos(){    //konstruktor
        head = null;
    }
   public Node<E> getHead() // vrati go prviot element
   {       return head;   }
   public void insertFirst(E e1, int e2) // vnesi nov jazol
   {
      Node <E> first = new Node (e1,e2,head);
      head=first;
   }
```

ЗАДАЧА 2 (3)

```
public void insertLast (E e1, int e2) // vnesi posleden (jazel)
  {
     if(head!=null)
     {
        Node <E> tmp=head;  // temp promenliva za iteriranje niz niza
        while(tmp.next!=null)
           tmp=tmp.next;
        Node <E> last=new Node(e1,e2,null);  // kreiram nov jazel
        tmp.next=last; // azhuriram na posleden

     }
     else
        this.insertFirst(e1,e2);
  }
```

ЗАДАЧА 2 (4)

```
 public void isfrliDuplikati()
{
    if(head!=null)
    {
        Node <E> tmp1=head;
        Node <E> tmp2=head.next;
        Node <E> prethodnik=tmp1;
        /* prethodnik секогаш ќе е претходникот на tmp2
         * затоа што листата е единечно поврзана и мора да чуваме
      * информација за претходникот на даден јазол
      */
```

ЗАДАЧА 2 (5)

```
while(tmp1.next!=null)
        {
            while(tmp2!=null)
            {
            if(tmp1.data1.compareTo(tmp2.data1)==0) // dokolku broevite se isti
            {
                tmp1.data2=tmp1.data2+1;  // stavi deka se povtoruva
                prethodnik.next=tmp2.next; // brisenje na jazel
                tmp2=tmp2.next;
            }
            else
            {
                prethodnik=tmp2;
                tmp2=tmp2.next;
            }
        }
```

ЗАДАЧА 2 (6)

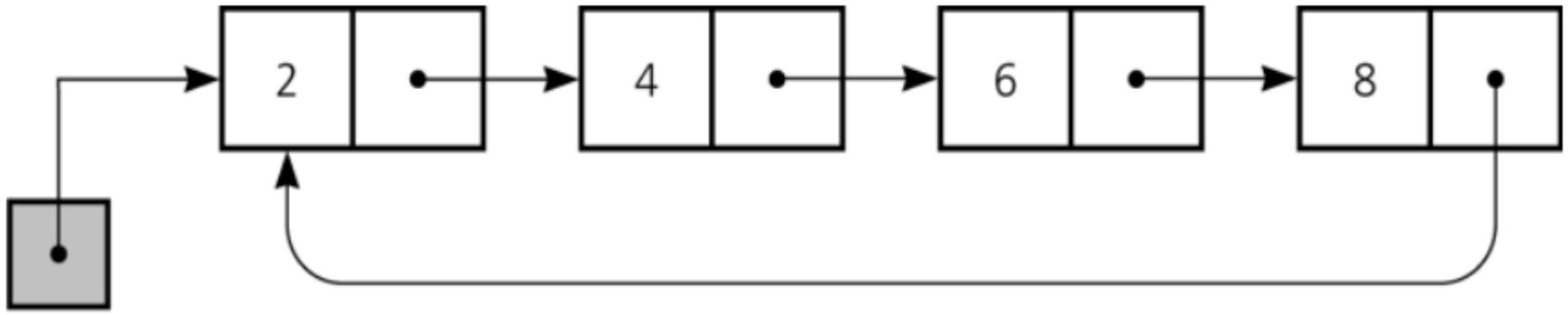
```
tmp1=tmp1.next;
        prethodnik=tmp1;
        tmp2=tmp1.next;
    }
}
    else
    {
        System.out.println("Listata e prazna");
    }
}
```

ЗАДАЧА 2 (7)

```
public static void main(String[] args) {
        SLinkedListWithTwoInfos <Integer> list = new SLinkedListWithTwoInfos();
        list.insertLast(5, 0);
        list.insertLast(5, 0);
        list.insertLast(4, 0);
        list.insertLast(5, 0);
        list.insertLast(3, 0);
        list.insertLast(4, 0);
        list.insertLast(5, 0);
        list.isfrliDuplikati();
        Node <Integer> pom = list.head;
        while(pom!=null)
         {
            System.out.println("("+pom.data1+", "+pom.data2+") ");
            pom=pom.next;
         }
    }
}
```

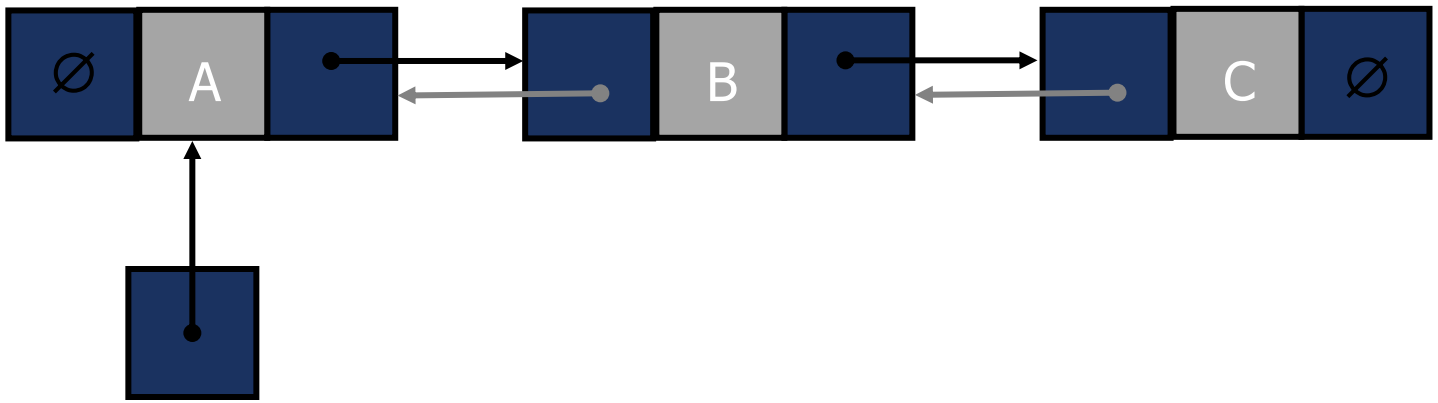
КРУЖНО ПОВРЗАНА ЛИСТА

- Последниот јазол покажува кон првиот јазол
- Секој јазол има следбеник



ДВОЈНО ПОВРЗАНА ЛИСТА

- Секој јазол има врска и кон неговиот претходник
- Предности:
  - Брзо додавање/бришење на елементи на двата краеви од листата
  - Брзо додавање/бришење на елементи претходник на даден елемент



ДВОЈНО ПОВРЗАНА ЛИСТА (1)

```
class Node <E> {
protected E data;
protected Node <E> next;
protected Node <E> prev;
public Node()
{   data = null;
    next = null;
    prev=null; }
public Node(E data, Node <E> prev, Node <E> next)
{   this.data = data;
    this.prev=prev;
    this.next = next;}
}
```

ДВОЈНО ПОВРЗАНА ЛИСТА (2)

```
class DLinkedList <E> {
   private Node<E> head, tail;
   public DLinkedList(){
        head = null;
        tail = null;
    }
public Node <E> getHead()
   {
       return head;
   }
    public Node <E> getTail()
   {
       return tail;
   }
```

ДВОЈНО ПОВРЗАНА ЛИСТА (3)

```
public void insertFirst(E e)
  {
      Node <E> first = new Node (e,null,head);
      if(head!=null)
          head.prev=first;
      if(tail==null)
          tail=first;
      head=first;
  }
```

ДВОЈНО ПОВРЗАНА ЛИСТА (4)

```
public void insertLast (E e)
 {
    if(head!=null)
    {
        Node <E> last=new Node(e,tail,null);
        tail.next=last;
        tail=last;
    }
    else
        this.insertFirst(e);
 }
```

ДВОЈНО ПОВРЗАНА ЛИСТА (5)

```
public static void main(String[] args) throws IOException {

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        int brElementi = Integer.parseInt(in.readLine());

        DLinkedList<Integer> list = new DLinkedList<Integer>();
        for (int i = 0; i < brElementi; i++) {
                list.insertLast(Integer.parseInt(in.readLine()));
            }
        Node<Integer> tmp=list.getHead();
```

ДВОЈНО ПОВРЗАНА ЛИСТА (6)

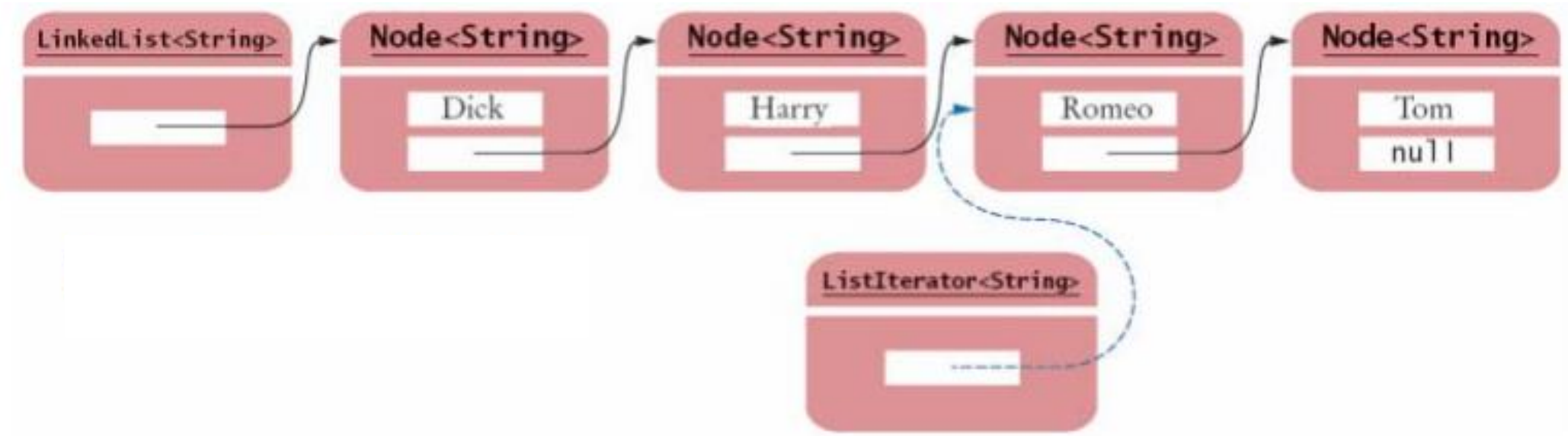
```
while(tmp!=null)
        {
            System.out.print(tmp.data+" ");
            tmp=tmp.next;
        }
        System.out.println();

        tmp=list.getTail();
        while(tmp!=null)
        {
            System.out.print(tmp.data+" ");
            tmp=tmp.prev;
        }
    }
}
```

КЛАСАТА LINKEDLIST

- Генеричка класа која се наоѓа во java.util
- Имплементацијата е двојно поврзана листа
  - Секој јазол има врска и со неговиот претходник и со неговиот следбеник
  - Додавањето и бришењето на јазли е многу ефикасно
- Класата LinkedList обезбедува методи за целосно симулирање на магацини и редови
- Некои дефинирани методи за LinkedList <E>:
  - void addFirst(E)
  - void addLast(E)
  - Void add(E)
  - E getFirst( )
  - E getLast( )
  - E removeFirst( )
  - E removeLast( )\

КЛАСАТА LINKEDLIST

- За да додаваме или бришеме елемент од внатрешноста на листата користиме ListIterator
- ListIterator ја чува позицијата



КЛАСАТА LINKEDLIST

- ListIterator објект добиваме од објект на класата LinkedList
  - LinkedList<String> employeeNames = . . .;
  - ListIterator<String> iterator = employeeNames.listIterator();
- Иницијално iterator покажува кон првиот елемент на листата. Низ листата се движиме со помош на методот next()
  - iterator.next();
- Методот next() исфрла NoSuchElementException доколку стигнеме до крај на листата. Затоа секогаш треба да го повикаме методот hasNext() пред да изминуваме со next().
  - if (iterator.hasNext())

    iterator.next();

КЛАСАТА LINKEDLIST

- Методот next() го враќа елементот кон кој покажува итераторот. Ако изминуваме листа со стрингови:
  - while iterator.hasNext()

    { String name = iterator.next(); //Do something with name }
- Ако сакаме да ги посетиме сите елементи во листа од String објекти:
  - for (String name : employeeNames)

    { //Do something with name }
  - Во позадина for each јамката користи итератор

КЛАСАТА LINKEDLIST

- Може да се употребат методите од ListIterator класата: previous() и hasPrevious() за да се пристапува кон претходникот
- Методот add() додава елемент после тековниот
  - iterator.add("Juliet");
- Методот remove() го отстранува објектот вратен со последниот повик на next() или previous()
  - //Remove all names that fulfill a certain condition

```
while (iterator.hasNext())
{ String name = iterator.next();
if (name fulfills condition)
iterator.remove(); }
```

ЕДНОСТАВНА ПРОГРАМА СО LINKEDLIST

- Вметнува стрингови во листа, итерира низ листата, додавајќи и бришејќи објекти, ја печати листата

```
import java.util.LinkedList;
import java.util.ListIterator;

/**
   A program that demonstrates the LinkedList class
*/
public class ListTester
{
   public static void main(String[] args)
   {
      LinkedList<String> staff = new LinkedList<String>();
      staff.addLast("Dick");
      staff.addLast("Harry");
      staff.addLast("Romeo");
      staff.addLast("Tom");

      // | in the comments indicates the iterator position
```

ЕДНОСТАВНА ПРОГРАМА СО LINKEDLIST

```
ListIterator<String> iterator
      = staff.listIterator(); // |DHRT
iterator.next(); // D|HRT
iterator.next(); // DH|RT

// Add more elements after second element

iterator.add("Juliet"); // DHJ|RT
iterator.add("Nina"); // DHJN|RT

iterator.next(); // DHJNR|T

// Remove last traversed element

iterator.remove(); // DHJN|T
```

ЕДНОСТАВНА ПРОГРАМА СО LINKEDLIST

```
        // Print all elements

        for (String name : staff)
            System.out.println(name);
    }
}
```

Output:

```
Dick
Harry
Juliet
Nina
Tom
```