ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Arrays and lists

Algorithms and data structures

Exercise 1

Array class in Java

```
public class Array<E>{
 private E data[];
 private int size;

 public Array(int capacity) {
        this.data = (E[]) new Object[capacity];
        this.size = 0;
 }

 public void insertLast(E o) {}
 public void insert(int position, E o) {}
 public void set(int position, E o) {}
 public E get(int position) {}
 public int find(E o) {}
 public int getSize() {}
 public void delete(int position) {}
 public void resize() {}

}
```

Basic array operations

```
public void insertLast(E o) {
    if(size + 1 > data.length)
        this.resize();
    data[size++] = o;
}

public void insert(int position, E o) {
    // before all we check if position is within range
    if (position >= 0 && position <= size) {
        //check if there is enough capacity, and if not - resize
        if(size + 1 > data.length)
            this.resize();
        //copy the data, before doing the insertion
        for(int i=size;i>position;i--) {
            data[i] = data[i-1];
        }
        data[position] = o;
        size++;
    } else {
        System.out.println("Ne mozhe da se vmetne element na taa pozicija");
    }
}
```

Basic array operations

```
public void set(int position, E o) {
    if (position >= 0 && position < size)
        data[position] = o;
    else
        System.out.println("Ne moze da se vmetne element na dadenata pozicija");
}

public E get(int position) {
    if (position >= 0 && position < size)
        return data[position];
    else
        System.out.println("Ne e validna dadenata pozicija");
    return null;
}
```

Basic array operations

```
public int find(E o) {
    for (int i = 0; i < size; i++) {
        if(o.equals(data[i]))
            return i;
    }
    return -1;
}

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

Basic array operations

```
public void delete(int position) {
    // before all we check if position is within range
    if (position >= 0 && position < size) {
        // first resize the storage array
        E[] newData = (E[]) new Object[size - 1];
        // copy the data prior to the delition
        for (int i = 0; i < position; i++)
            newData[i] = data[i];
        // move the data after the deletion
        for (int i = position + 1; i < size; i++)
            newData[i - 1] = data[i];
        // replace the storage with the new array
        data = newData;
        size--;
    }
}
```

Basic array operations

```
public void resize() {
    // first resize the storage array
    E[] newData = (E[]) new Object[size*2];
    // copy the data
    int copySize = size;;
    for (int i = 0; i < size; i++)
        newData[i] = data[i];
    // replace the storage with the new array
    this.data = newData;
}
```

Basic array operations usage

```
public static void main(String[] args) {
    Array<Integer> niza = new Array<Integer>( capacity: 4);

    niza.insertLast( o: 4);
    System.out.print("Nizata po vmetnuvanje na 4 kako posleden element: ");
    System.out.println(niza.toString());

    niza.insertLast( o: 7);
    niza.insertLast( o: 13);
    System.out.print("Nizata po dodavanje na 7 i 13 kako elementi: ");
    System.out.println(niza.toString());

    niza.insert( position: 1,  o: 3);
    System.out.print("Nizata po dodavanje na 3 kako element na pozicija 1: ");
    System.out.println(niza.toString());
```

Basic array operations usage

```
    niza.set(2, 6);
    System.out.print("Nizata po menuvanje na vrednosta na elementot na pozicija 2 vo 6: ");
    System.out.println(niza.toString());

    niza.delete( position: 0);
    System.out.print("Nizata po brishenje na elementot na pozicija 0 (prviot element): ");
    System.out.println(niza.toString());

    System.out.print("Na pozicija 2 vo nizata sega se naogja: ");
    System.out.println(niza.get(2));

    System.out.print("Brojot 3 sega se naogja vo nizata na pozicija: ");
    System.out.println(niza.find( o: 3));

    System.out.print("Sega na krajot goleminata na nizata e: ");
    System.out.println(niza.getSize());
}
```

Problem 1.

- Let there be given two arrays, which should be of the same size. Write a function that will make changes in both arrays so that if at a given position they have equal elements, they should be deleted in both arrays.

Problem 1 - solution

```
public class ChangeArrays<E> {
    public void compareAndChangeArrays(Array<E> niza1, Array<E> niza2) {
        if(niza1.getSize() != niza2.getSize()) {
            System.out.println("Nizite ne se so ista golemina!");
            return;
        }
        int size = niza1.getSize();
        int i = 0;
        while(i < size) {
            if(niza1.get(i).equals(niza2.get(i))) {
                niza1.delete(i);
                niza2.delete(i);
                size--;
            } else {
                i++;
            }
        }
    }
```

Problem 1 - solution - main

```
public static void main(String[] args) {
    Array<String> niza1 = new Array<~>( capacity: 4);
    niza1.insertLast( o: "nb11");
    niza1.insertLast( o: "b1");
    niza1.insertLast( o: "b2");
    niza1.insertLast( o: "nb12");

    Array<String> niza2 = new Array<~>( capacity: 4);
    niza2.insertLast( o: "nb21");
    niza2.insertLast( o: "b1");
    niza2.insertLast( o: "b2");
    niza2.insertLast( o: "nb22");

    System.out.println("Nizite pred primenuvanjeto na funkcijata: ");
    System.out.println(niza1.toString());
    System.out.println(niza2.toString());
```

Problem 1 - solution - main

```
Array<String> niza2 = new Array<~>( capacity: 4);
niza2.insertLast( o: "nb21");
niza2.insertLast( o: "b1");
niza2.insertLast( o: "b2");
niza2.insertLast( o: "nb22");

System.out.println("Nizite pred primenuvanjeto na funkcijata: ");
System.out.println(niza1.toString());
System.out.println(niza2.toString());

ChangeArrays<String> pom = new ChangeArrays<String>();
pom.compareAndChangeArrays(niza1, niza2);

System.out.println("Nizite po primenuvanjeto na funkcijata: ");
System.out.println(niza1.toString());
System.out.println(niza2.toString());

}
```

Problem 1 – solution with ArrayList

```
public void compareAndChangeArrays(ArrayList<E> niza1, ArrayList<E> niza2) {
    if(niza1.size() != niza2.size()) {
        System.out.println("Nizite ne se so ista golemina!");
    }
    int size = niza1.size();
    int i = 0;
    while(i < size) {
        if(niza1.get(i).equals(niza2.get(i))) {
            niza1.remove(i);
            niza2.remove(i);
            size--;
        } else {
            i++;
        }
    }
}
```

Before program (code) start class ArrayList has to be imported:

```
import java.util.ArrayList;
```

Problem 1 – solution with ArrayList - main

```
ArrayList<Integer> niza3 = new ArrayList<Integer>( initialCapacity: 3);
niza3.add(10);
niza3.add(13);
niza3.add(7);

ArrayList<Integer> niza4 = new ArrayList<Integer>( initialCapacity: 3);
niza4.add(5);
niza4.add(13);
niza4.add(3);
```

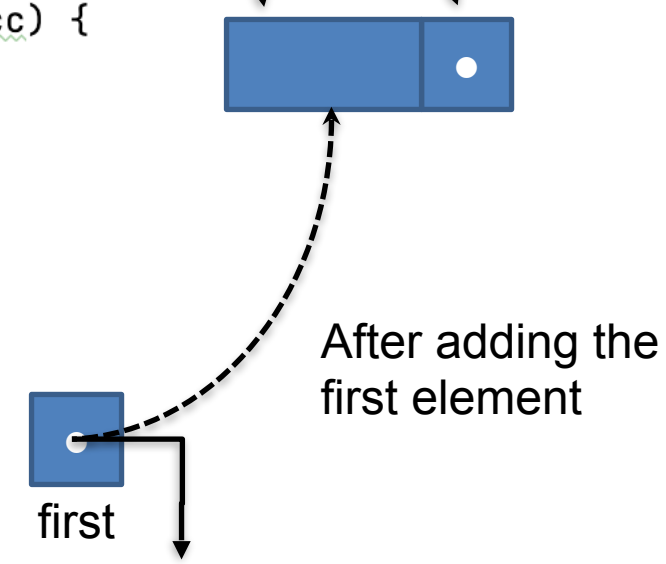
Problem 1 – solution with ArrayList - main

```
System.out.println("Nizite pred primenuvanjeto na funkcijata: ");
System.out.println(niza3.toString());
System.out.println(niza4.toString());

ChangeArrays<Integer> pom2 = new ChangeArrays<Integer>();
pom2.compareAndChangeArrays(niza3, niza4);

System.out.println("Nizite po primenuvanjeto na funkcijata: ");
System.out.println(niza3.toString());
System.out.println(niza4.toString());
```

Single linked list

```
public class SLLNode<E> {
    protected E element;
    protected SLLNode<E> succ;

    public SLLNode(E elem, SLLNode<E> succ) {
        this.element = elem;
        this.succ = succ;
    }
}
```

```
public class SLL<E> {
    private SLLNode<E> first;

    public SLL () {
        //kreiranje prazna lista
        this.first = null;
    }
  ...
}
```

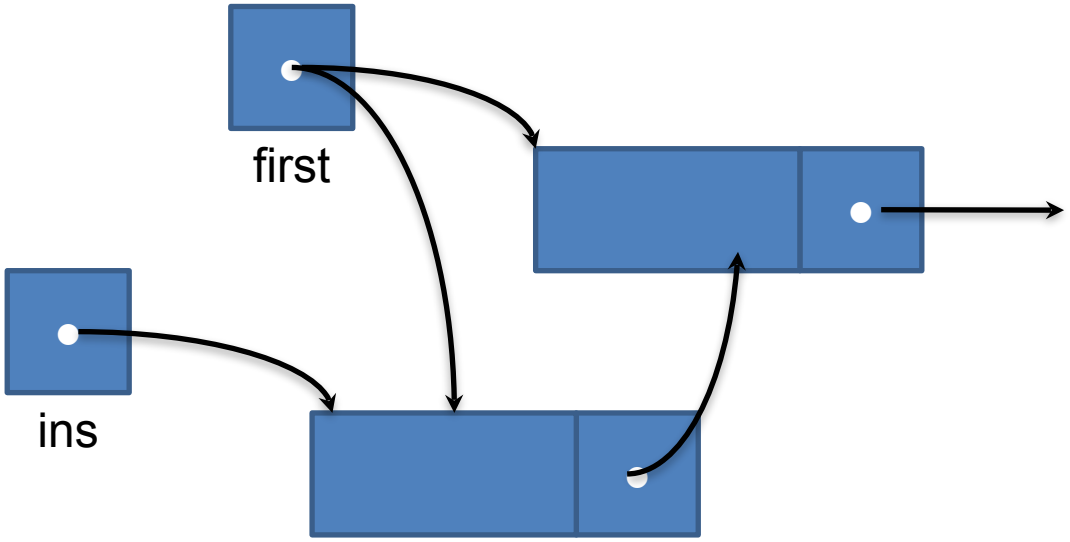


Single linked list in Java

```
public class SLL<E> {
  private SLLNode<E> first;

  public SLL (){
    // kreiranje prazna lista
    this.first = null;
  }

  public void insertFirst(E o)
  public void insertAfter(E o, SLLNode<E> after)
  public void insertBefore(E o, SLLNode<E> before)
  public E deleteFirst()
  public E delete(SLLNode<E> node)
  public SLLNode<E> getFirst()
}
```

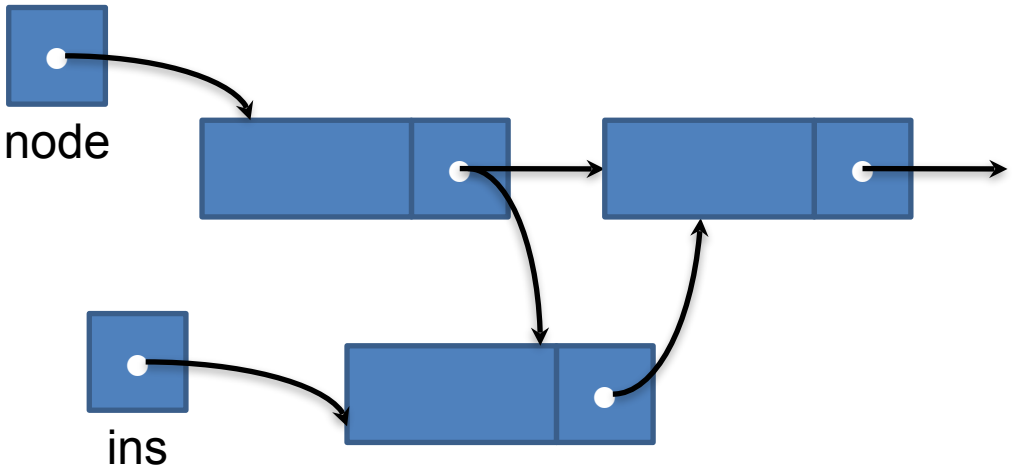
SLL node insertion at the beginning

```
public void insertFirst(E o) {
    SLLNode<E> ins = new SLLNode<E>(o, succ: null);
    ins.succ = first;
    //SLLNode<E> ins = new SLLNode<E>(o, first);
    first = ins;
}
```



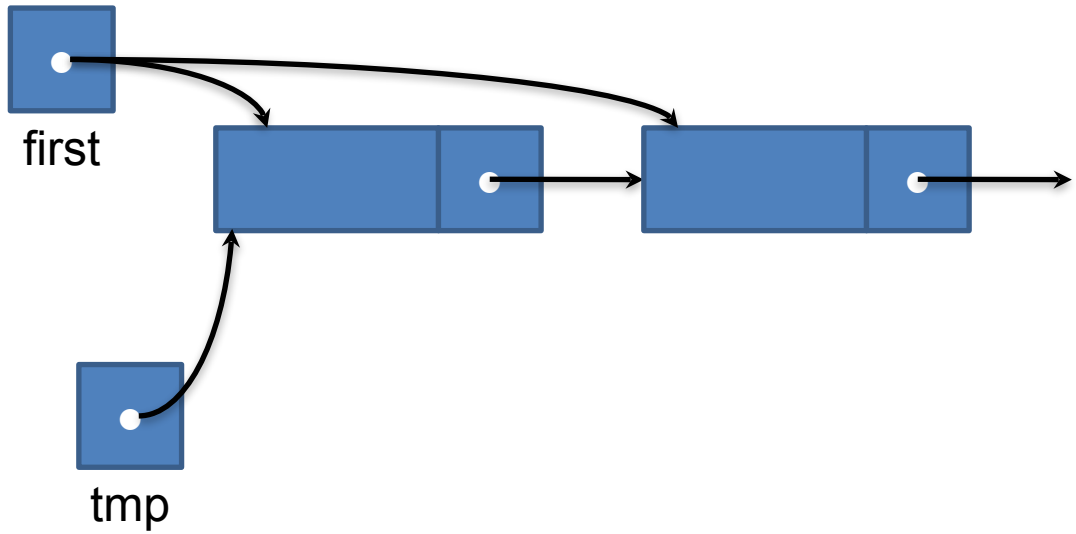
SLL node insertion after a given node

```
public void insertAfter(E o, SLLNode<E> node) {
    if (node != null) {
        SLLNode<E> ins = new SLLNode<E>(o, node.succ);
        node.succ = ins;
    } else {
        System.out.println("Dadenot jazol e null");
    }
}
```



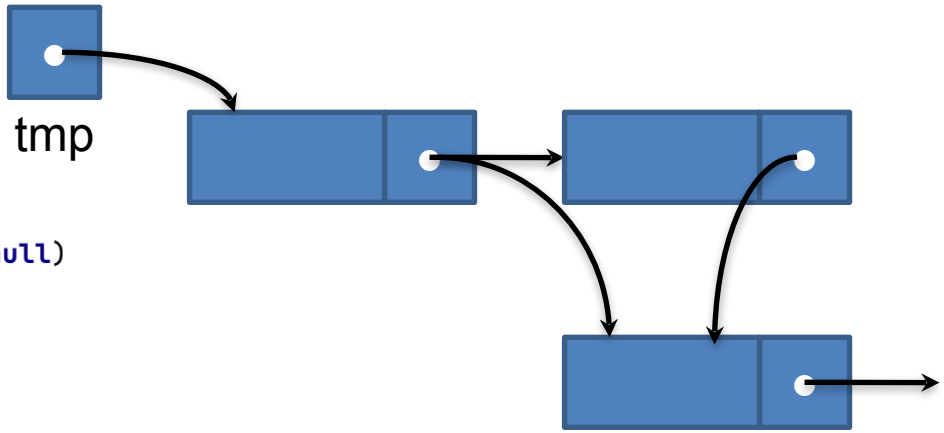
SLL node deletion – first

```
public E deleteFirst() {
    if (first != null) {
        SLLNode<E> tmp = first;
        first = first.succ;
        return tmp.element;
    } else {
        System.out.println("Listata e prazna");
        return null;
    }
}
```



SLL node deletion – given node

```
public E delete(SLLNode<E> node) {
    if (first != null) {
        SLLNode<E> tmp = first;
        if(first == node) {
            return this.deleteFirst();
        }
        while (tmp.succ != node && tmp.succ.succ != null)
            tmp = tmp.succ;
        if (tmp.succ == node) {
            tmp.succ = tmp.succ.succ;
            return node.element;
        } else {
            System.out.println("Elementot ne postoi vo listata");
            return null;
        }
    } else {
        System.out.println("Listata e prazna");
        return null;
    }

}
```



SLL remaining operations

```
public int size() {
    int listSize = 0;
    SLLNode<E> tmp = first;
    while(tmp != null) {
        listSize++;
        tmp = tmp.succ;
    }
    return listSize;
}
```

SLL remaining operations

```
public void insertBefore(E o, SLLNode<E> before) {

    if (first != null) {
        SLLNode<E> tmp = first;
        if(first==before){
            this.insertFirst(o);
            return;
        }
        //ako first!=before
        while (tmp.succ != before && tmp.succ!=null)
            tmp = tmp.succ;
        if (tmp.succ == before) {
            tmp.succ = new SLLNode<E>(o, before);;
        } else {
            System.out.println("Elementot ne postoi vo listata");
        }
    } else {
        System.out.println("Listata e prazna");
    }
}
```

SLL remaining operations

```
public void insertLast(E o) {
    if (first != null) {
        SLLNode<E> tmp = first;
        while (tmp.succ != null)
            tmp = tmp.succ;
        tmp.succ = new SLLNode<E>(o, succ: null);
    } else {
        insertFirst(o);
    }
}
```

SLL remaining operations

```
public SLLNode<E> find(E o) {
    if (first != null) {
        SLLNode<E> tmp = first;
        while (tmp.element != o && tmp.succ != null)
            tmp = tmp.succ;
        if (tmp.element.equals(o)) {
            return tmp;
        } else {
            System.out.println("Elementot ne postoi vo listata");
        }
    } else {
        System.out.println("Listata e prazna");
    }
    return null;
}
```

SLL remaining operations

```
public void merge (SLL<E> in){
    if (first != null) {
        SLLNode<E> tmp = first;
        while(tmp.succ != null)
            tmp = tmp.succ;
        tmp.succ = in.getFirst();
    }
    else{
        first = in.getFirst();
    }
}
```

SLL operations usage

```
public static void main(String[] args) {
    SLL<Integer> lista = new SLL<Integer>();
    lista.insertLast( o: 5);
    System.out.print("Listata po vmetnuvanje na 5 kako posleden element: ");
    System.out.println(lista.toString());

    lista.insertFirst( o: 3);
    System.out.print("Listata po vmetnuvanje na 3 kako prv element: ");
    System.out.println(lista.toString());

    lista.insertLast( o: 1);
    System.out.print("Listata po vmetnuvanje na 1 kako posleden element: ");
    System.out.println(lista.toString());
```

SLL operations usage

```
lista.insertLast( o: 1);
System.out.print("Listata po vmetnuvanje na 1 kako posleden element: ");
System.out.println(lista.toString());

lista.deleteFirst();
System.out.print("Listata po brishenje na prviot element: ");
System.out.println(lista.toString());

SLLNode<Integer> pom = lista.find( o: 5);
lista.insertBefore( o: 2, pom);
System.out.print("Listata po vmetnuvanje na elementot 2 pred elementot 5: ");
System.out.println(lista.toString());

pom = lista.find( o: 1);
lista.insertAfter( o: 3, pom);
System.out.print("Listata po vmetnuvanje na elementot 3 posle elementot 1: ");
System.out.println(lista.toString());
```

SLL operations usage

```
        System.out.println("Momentalna dolzina na listata: " + lista.size());

        System.out.print("Listata po prevrtuvanje: ");
        lista.mirror();
        System.out.println(lista.toString());

        pom = lista.find( o: 2);
        lista.delete(pom);
        System.out.print("Listata po brishenje na elementot 2: ");
        System.out.println(lista.toString());

        System.out.println("Momentalna dolzina na listata: " + lista.size());

        lista.deleteList();
        System.out.print("Pecatenje na listata po nejzino brishenje: ");
        System.out.println(lista.toString());
        System.out.println("Momentalna dolzina na listata: " + lista.size());

}
```

Problem 2.

- Write a function that for a given single linked list of integers, will return the count of even numbers in that list.

Problem 2 - solution

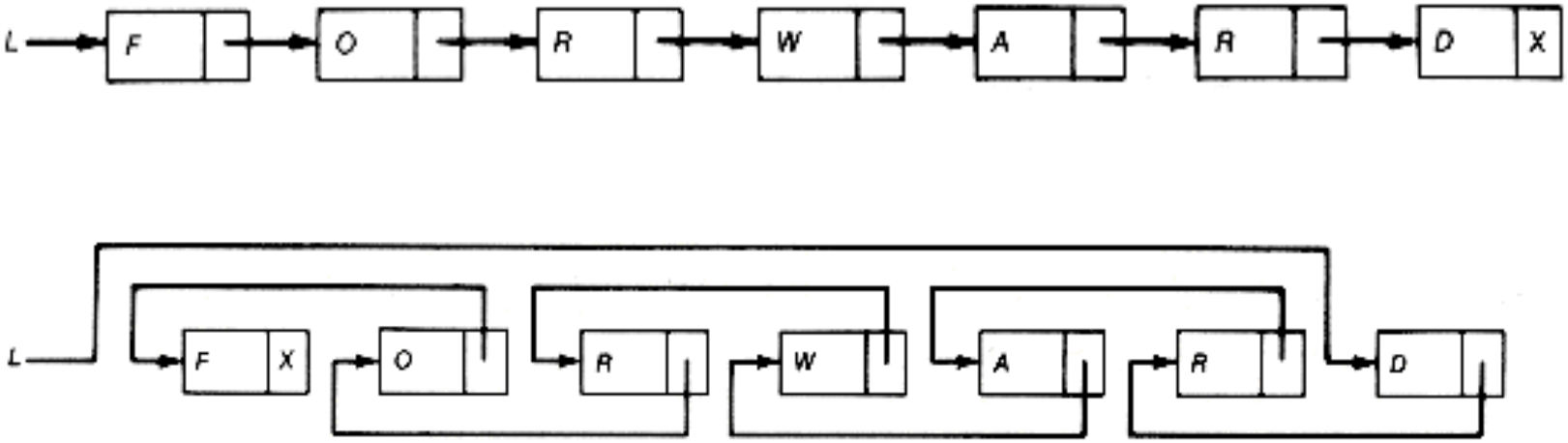
```
import java.util.Scanner;

public class EvenNumbersSLL {
    public static int evenNumbers(SLL<Integer> list) {
        SLLNode<Integer> tmp = list.getFirst();
        int res = 0;

        while(tmp!=null) {
            if(tmp.element%2==0) {
                res++;
            }
            tmp = tmp.succ;
        }

        return res;
    }
```

Problem 2 - solution - main

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Vnesete go brojot na elementi vo listata:");
        int n = sc.nextInt();

        SLL<Integer> list = new SLL<>();
        System.out.println("Vnesete gi elementite na listata (celi broevi):");
        for(int i=0;i<n;i++) {
            list.insertLast(sc.nextInt());
        }

        System.out.println("Brojot na parni elementi vo vnesenata lista e: " + evenNumbers(list));
    }
}
```

Problem 3.

- Write a function that reverses all the links in a single linked list.



Problem 3 - solution

```
public void mirror() {
    if (first != null) {
        //m=nextsucc, p=tmp,q=next
        SLLNode<E> tmp = first;
        SLLNode<E> newsucc = null;
        SLLNode<E> next;

        while(tmp != null){
            next = tmp.succ;
            tmp.succ = newsucc;
            newsucc = tmp;
            tmp = next;
        }
        first = newsucc;
    }
}
```

Problem 3 - solution - main

```
public static void main(String[] args) {
    SLL<String> lista = new SLL<String>();
    lista.insertLast( o: "ovaa");
    lista.insertLast( o: "lista");
    lista.insertLast( o: "kje");
    lista.insertLast( o: "bide");
    lista.insertLast( o: "prevrtena");
    System.out.println("Listata pred da bide prevrtena: " + lista.toString());
    lista.mirror();
    System.out.println("Listata otkako e prevrtena: " + lista.toString());
}
```

Problem 4.

- There are given two single linked lists whose nodes are sorted in ascending order. Write a function that will merge/join the two lists into one so that the resulting list is sorted. The sort is a merge/join sort.

Problem 4 - solution

```java
public class JoinSortedLists<E extends Comparable<E>> {

    public SLL<E> join(SLL<E> list1, SLL<E> list2) {
        SLL<E> rezultat = new SLL<E>();
        SLLNode<E> jazol1 = list1.getFirst(), jazol2 = list2.getFirst();
        //SLLNode<E> jazol2 = list2.getFirst();

        while (jazol1 != null && jazol2 != null) {
            if (jazol1.element.compareTo(jazol2.element) < 0) { //jazol1<jazol2
                rezultat.insertLast(jazol1.element);
                jazol1 = jazol1.succ;
            } else {
                rezultat.insertLast(jazol2.element);
                jazol2 = jazol2.succ;
            }

        }

        if (jazol1 != null) {
            while (jazol1 != null) {
                rezultat.insertLast(jazol1.element);
                jazol1 = jazol1.succ;
            }
        }

        if (jazol2 != null) {
            while (jazol2 != null) {
                rezultat.insertLast(jazol2.element);
                jazol2 = jazol2.succ;
            }
        }

        return rezultat;
    }
}
```

Problem 4 - solution - main

```
public static void main(String[] args){
    SLL<String> lista1 = new SLL<String>();
    lista1.insertLast( o: "Ana");lista1.insertLast( o: "Bojana");lista1.insertLast( o: "Dejan");
    SLL<String> lista2 = new SLL<String>();
    lista2.insertLast( o: "Andrijana");lista2.insertLast( o: "Biljana");lista2.insertLast( o: "Darko");

    JoinSortedLists<String> js = new JoinSortedLists<String>();
    System.out.println(js.join(lista1, lista2));

}
```