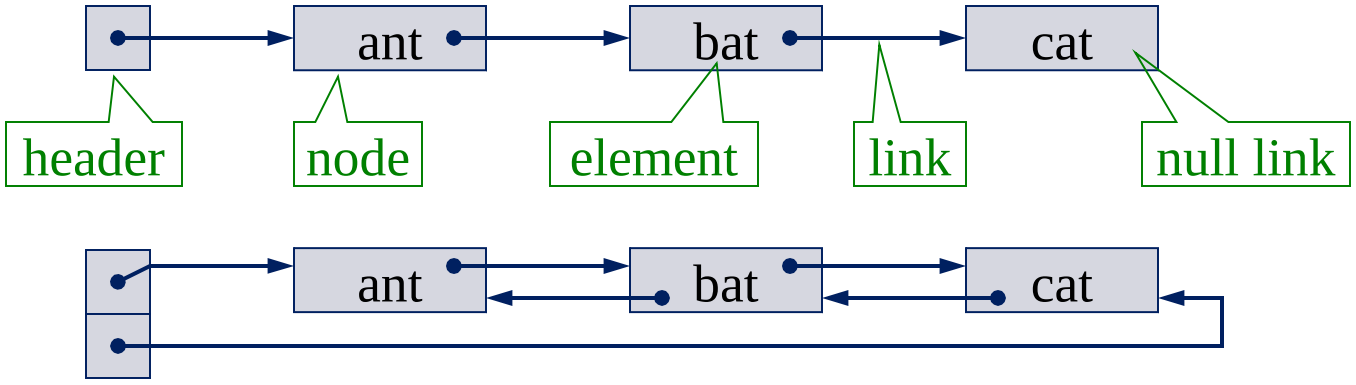
Kazalci in dinamične podatkovne strukture: seznami

Povezani seznami

- Enojno povezani, dvojno povezani.
- Vstavljanje.
- Brisanje.
- Iskanje.

Povezani seznami*(1)*

- Povezan seznam (linked list) vsebuje zaporedje vozlišč, povezanih s povezavami (links), vsebuje pa tudi kazalec na prvo vozlišče (header, glava).
- Vsako vozlišče (razen zadnjega) ima svojega naslednika in vsako vozlišče (razen prvega) ima svojega predhodnika.
- Vsako vozlišče vsebuje en element (objekt ali vrednost) in povezavo (link) na naslednika (in morda predhodnika).

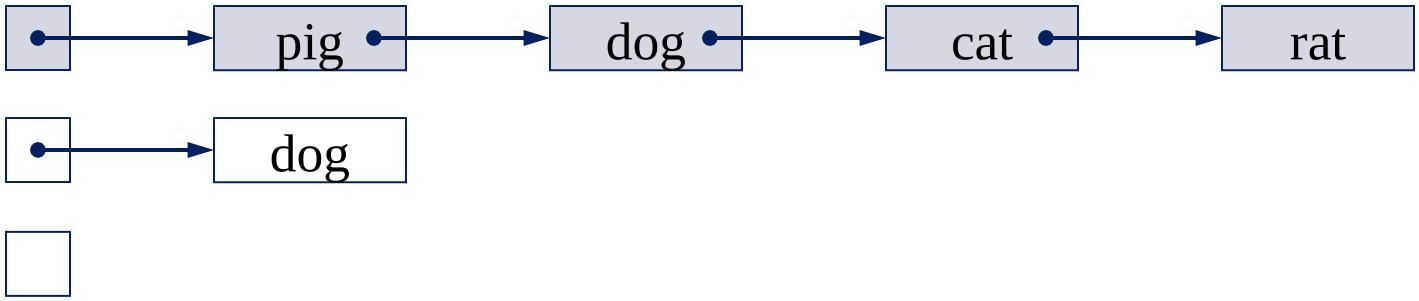


Povezani seznami *(2)*

- Dolžina povezanega seznama je število vozlišč.
- Prazen seznam nima vozlišč.
- V povezanem seznamu:
  - Lahko rokujemo s posameznimi elementi.
  - Lahko rokujemo s povezavami in tako celo strukturo spreminjamo (kar je pri poljih nemogoče.)

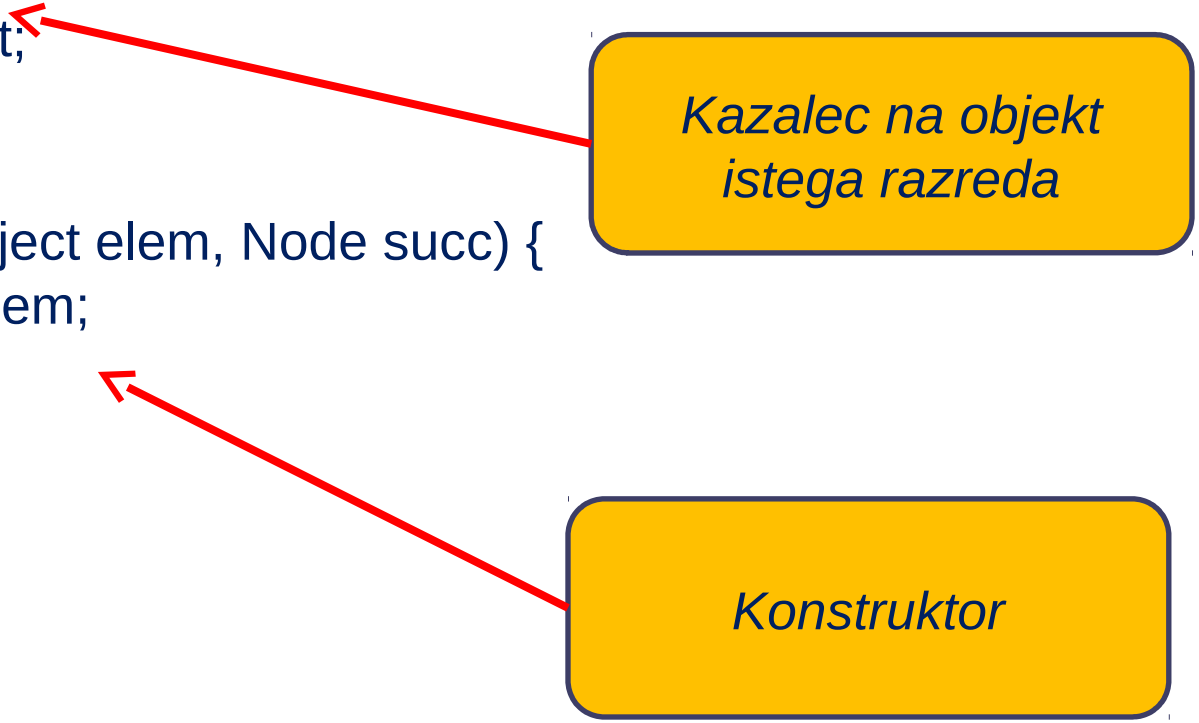
Enojno povezani seznami*(1)*

- Enojno povezani seznami imajo povezave le v eno smer.
- Vsako vozlišče ima element in povezavo na naslednje vozlišče. Povezava je v bistvu kazalec.
- Pri zadnjem vozlišču, ki nima naslednika, je ta kazalec enak null.
- Glava (header) na prvi element je tudi kazalec. Če je seznam prazen, je ta kazalec enak null.



Implementacija vozlišča v Javi

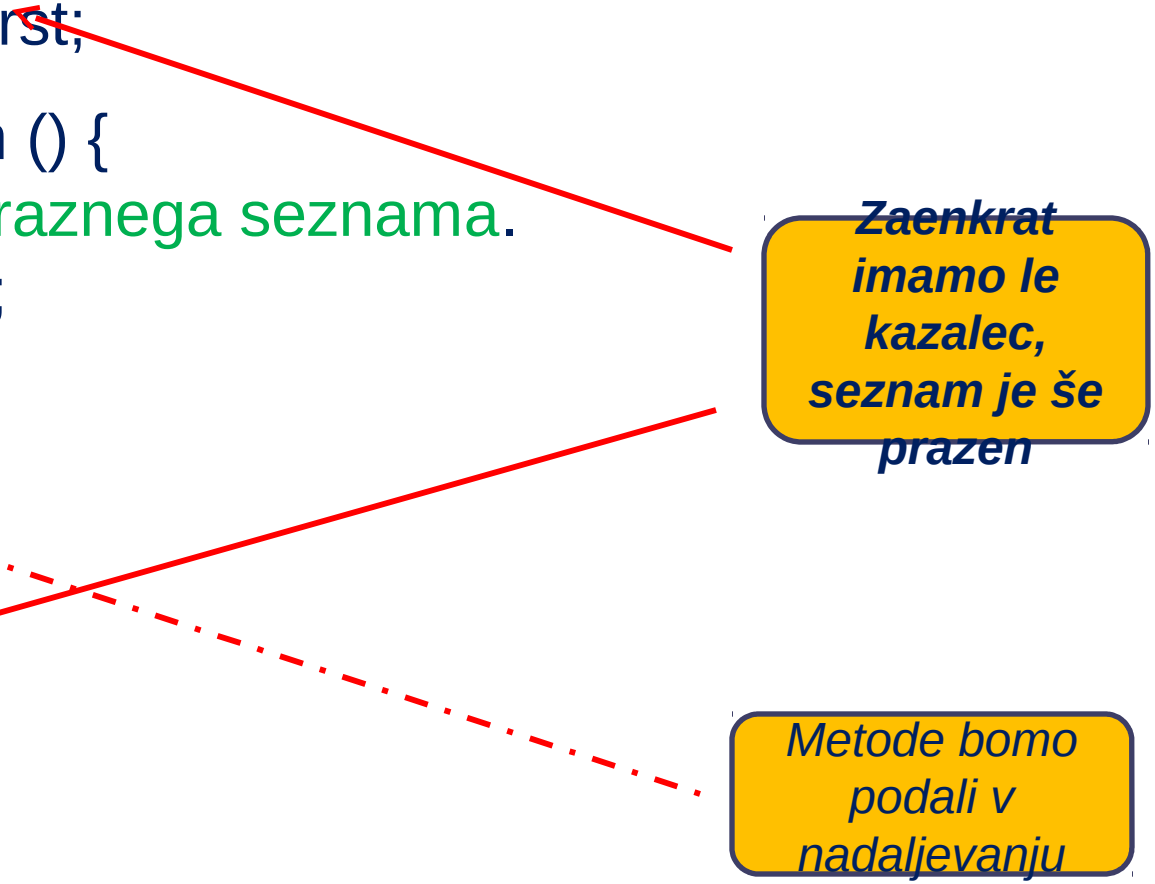
```
public class Node {

    protected Object element;
protected Node next;



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

}
```



Študijski primer: razred z enojno povezanim seznamom

```
public class Seznam {

    private Node first;

    public Seznam () {
    // konstruktor praznega seznama.
    this.first = null;
}

    …

}
```

first



Študijski primer: Prehod povezanega seznama

```
        public void printFirstToLast () {
// Zaporedni izpis elementov povezanega seznama.
        for (Node curr = first;  curr != null; curr = curr.next)
            System.out.println(curr.element);
}
```

Animacija:

Študijski primer: Brisanje prvega s seznama

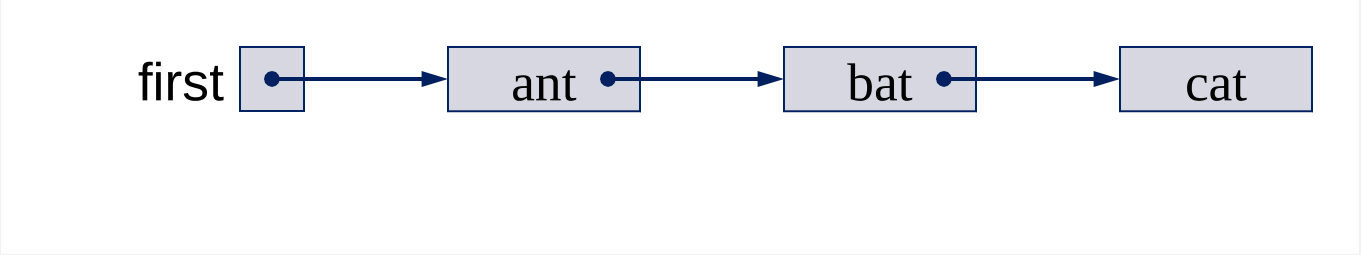
```
        public void deleteFirst () {
// Brisanje prvega vozlisca (ob predpostavki dolzine> 0).
        this.first = this.first.¸next;
}
```

Animacija:

Študijski primer: brisanje drugega v seznamu

```
    public void deleteSecond () {
  //(velja, ce je dolzina > 1).
    Node second = first.next;
    first.next= second.next;
}
```

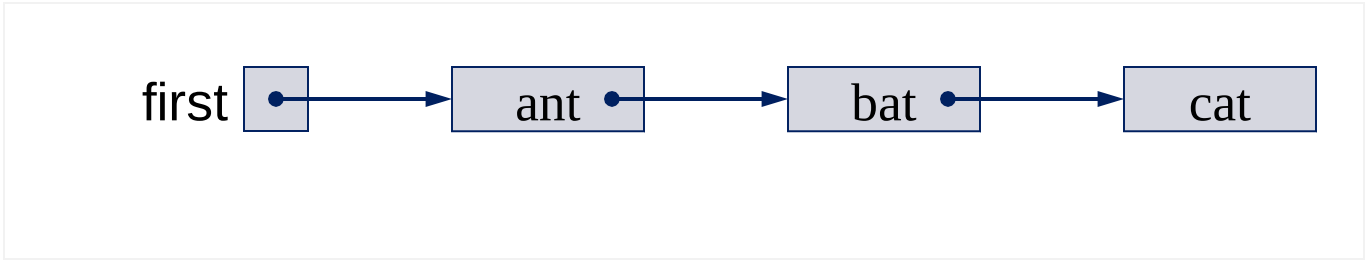
Animacija:



Študijski primer: Zamenjava dveh vozlišč

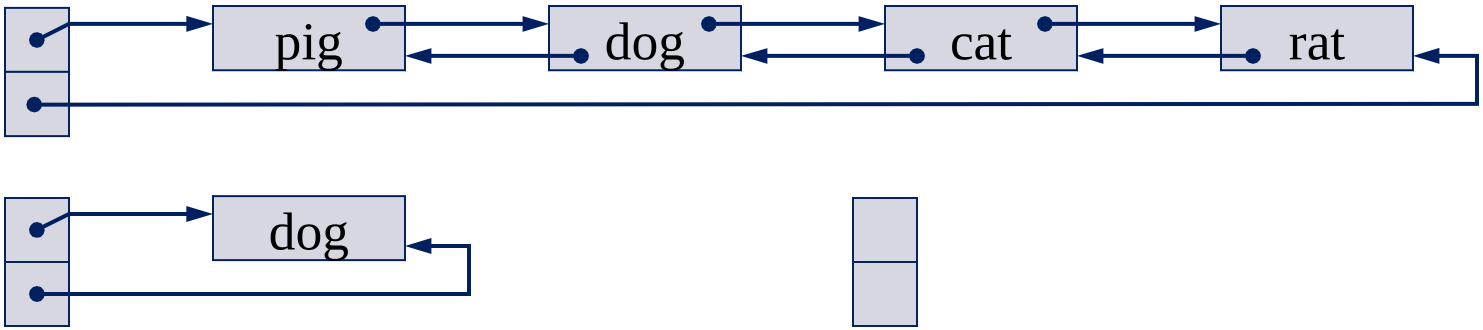
```
        public void swapFirstTwo () {
        Node second =  first.next;
        first.next= second.next;
        second.next = first;
        first = second;
}
```

Animacija:



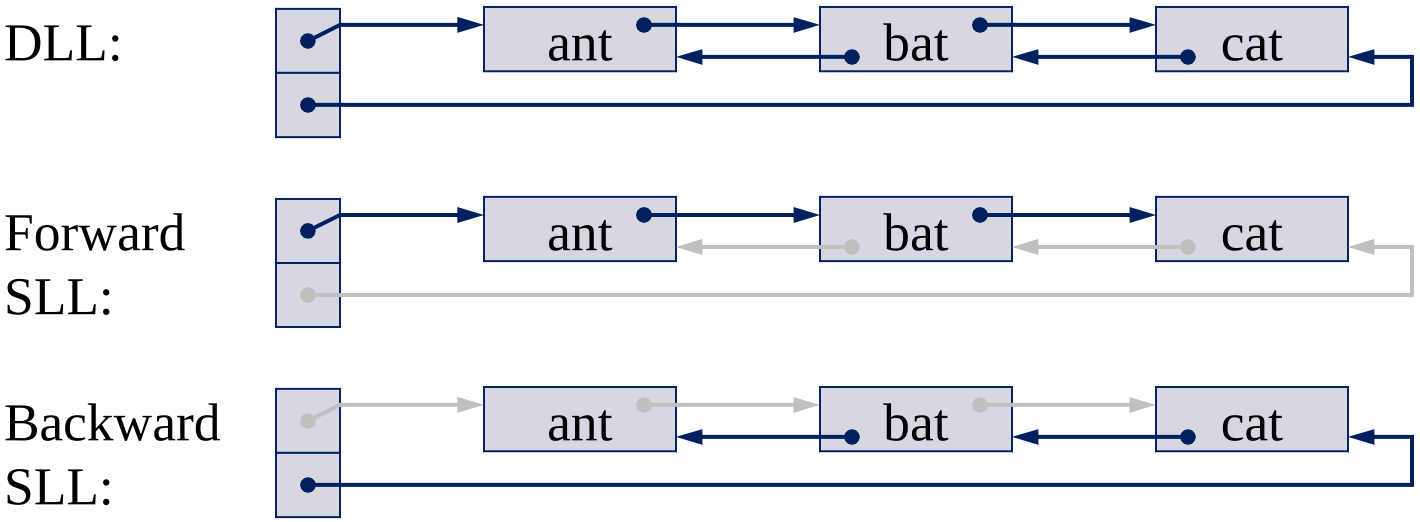
Dvojno povezan seznam

- Dvojno povezan seznam vsebuje zaporedje vozlišč, povezanih z linki v obeh smereh.
- Vsako vozlišče vsebuje en element in povezavi na predhodnika in naslednika. Če teh ni, so take povezave (kazalci) enaki null.
- Glava dvojno povezanega seznama ima kazalca na prvo in zadnje vozlišče.
- Pri praznem seznamu sta oba kazalca enaka null.



DLL = forward SLL + backward SLL

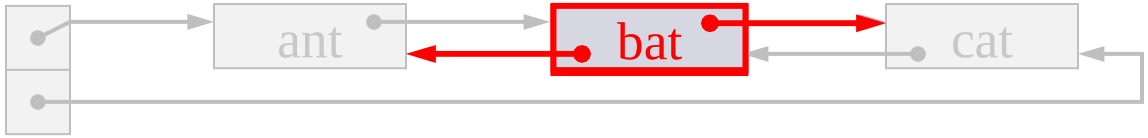
Na dvojno povezani seznam glejmo kot na superpozicijo naprej enojno povezanega in nazaj enojno povezanega seznama:



Implementacija vozlišča dvojno povezanega seznama v Javi

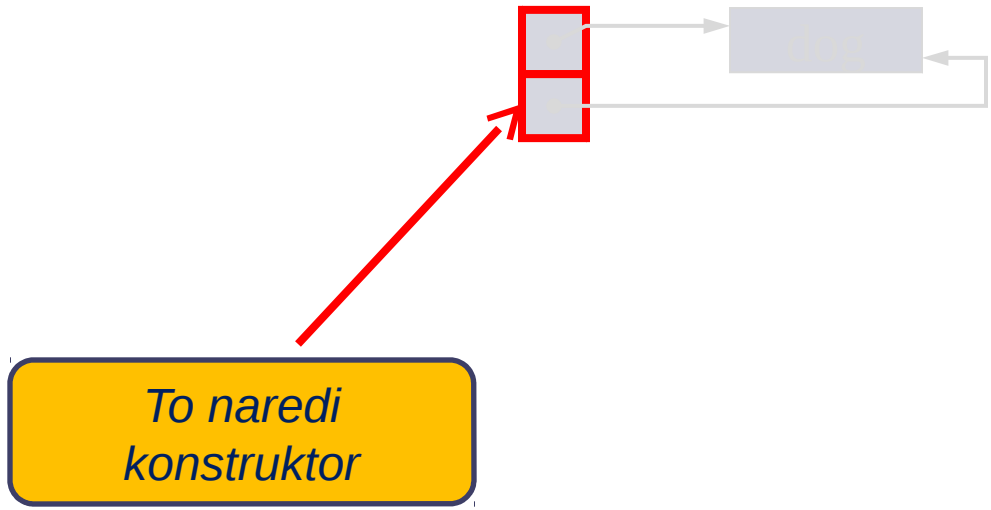
```
public class Node {

        protected Object element;
protected Node pred, succ;


    public Node (Object elem, Node pred, Node succ) {
                this.element = elem;
                this.pred = pred;
                this.succ = succ;


    }
}
```

DLL:



Java razred z implementacijo DLL glave

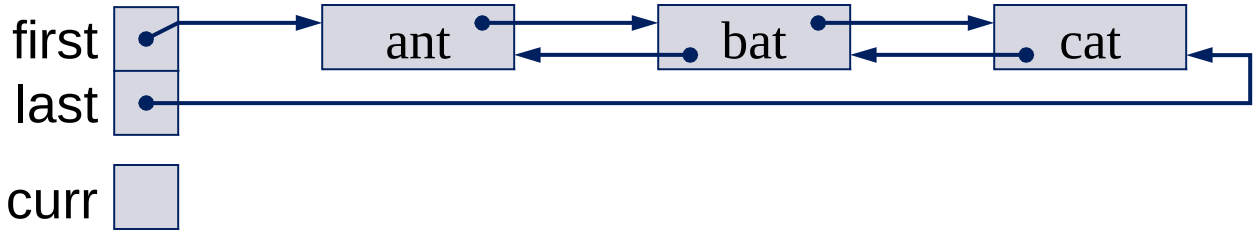
```
public class Seznam {

private Node first, last;

public Seznam () {
// konstruktor za prazen seznam.
this.first = null;
this.last = null;
}

…

}
```



Primer: Vzvratni prehod po seznamu

```
        public void printLastToFirst () {
// Izpis vseh elementov od zadnjega do prvega
        for (Node curr = last;
                curr != null; curr = curr.pred)
            System.out.println(curr.element);
}
```

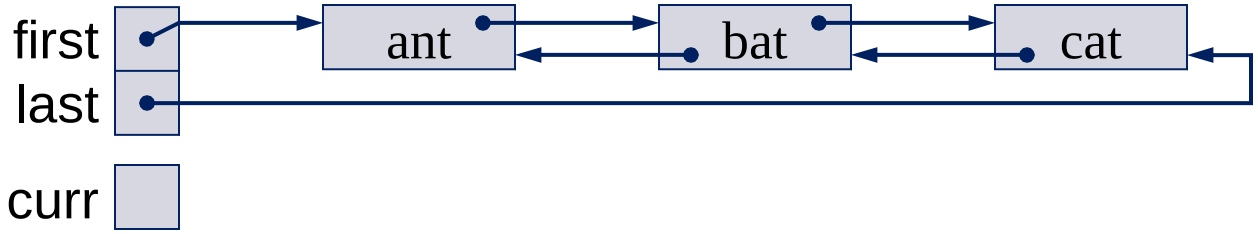
Animacija:



Primer: Vzvratni prehod po seznamu

```
        public void printLastToFirst () {
// Izpis vseh elementov od zadnjega do prvega
        for (Node curr = last;
                curr != null; curr = curr.pred)
            System.out.println(curr.element);
}
```

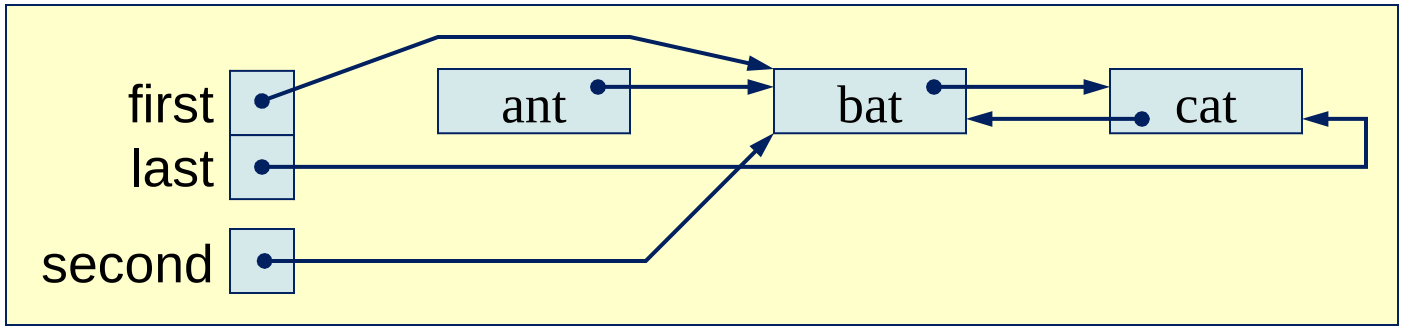
Animacija:



Primer: brisanje prvega vozlišča v DLL

```
        public void deleteFirst () {
// Delete this DLL's first node (assuming length > 0).
        Node second = this.first.succ;
        second.pred = null;
        this.first = second;
}
```

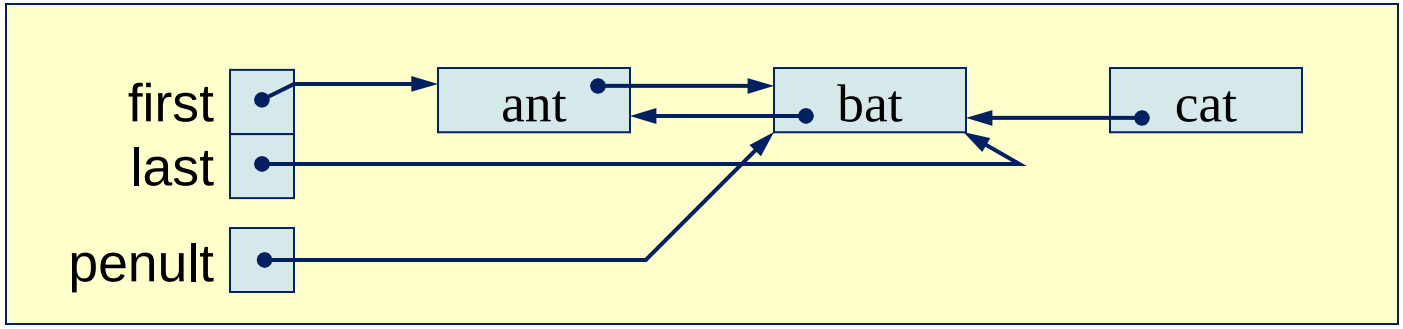
Animacija:



Primer: brisanje zadnjega vozlišča v DLL

```
        public void deleteLast () {
// Delete this DLL's last node (assuming length > 0).
        Node penult = last.pred;
        penult.succ = null;
        last = penult;
}
```

Animacija:



Vstavljanje novega elementa v seznam

- Primeri:
  1) Vstavljanje v prazen seznam;
  2) Vstavljanje pred prvo vozlišče nepraznega seznama;
  3) Vstavljanje za zadnje vozlišče v nepraznem seznamu;
  4) Vstavljanje med vozlišča v nepraznem seznamu.
- Algoritem vstavljanja mora popraviti povezave v predhodniku in nasledniku.

Algoritem vstavljanja pri enojnem seznamu (SLL)

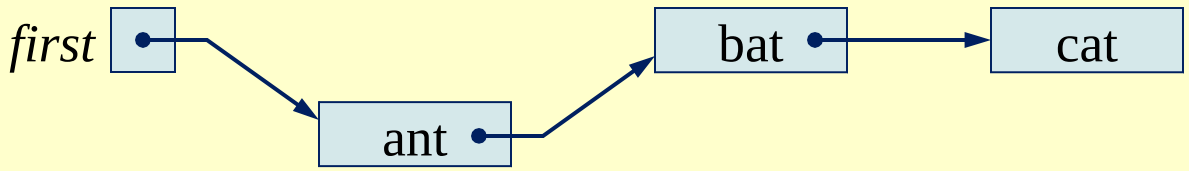
To insert *elem* at a given point in the SLL headed by *first*:

1. Make *ins* a link to a newly-created node with element *elem* and successor null.
2. If the insertion point is before the first node:
   2.1. Set node *ins*'s successor to *first*.
   2.2. Set *first* to *ins*.
3. If the insertion point is after the node *pred*:
   3.1. Set node *ins*'s successor to node *pred*'s successor.
   3.2. Set node *pred*'s successor to *ins*.
4. Terminate.

Vstavljanje pred prvo vozlišče (animacija)

To insert *elem* at a given point in the SLL headed by *first*:

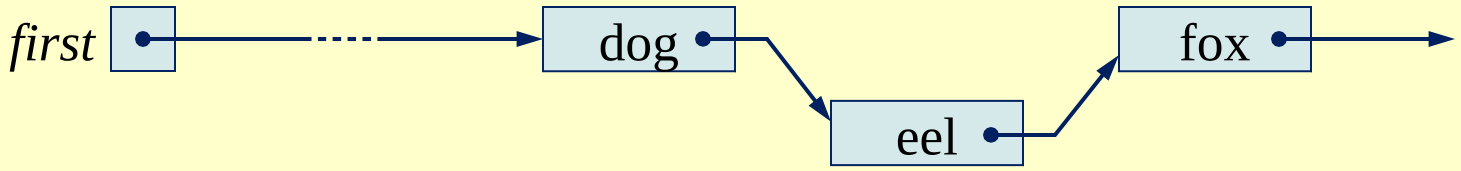
1. Make *ins* a link to a newly-created node with element *elem* and successor null.
2. If the insertion point is before the first node:
   2.1. Set node *ins*'s successor to *first*.
   2.2. Set *first* to *ins*.
3. If the insertion point is after the node *pred*:
   3.1. Set node *ins*'s successor to node *pred*'s successor.
   3.2. Set node *pred*'s successor to *ins*.
4. Terminate.



Vstavljanje za danim vozliščem (animacija)

To insert *elem* at a given point in the SLL headed by *first*:

1. Make *ins* a link to a newly-created node with element *elem* and successor null.
2. If the insertion point is before the first node:
   2.1. Set node *ins*'s successor to *first*.
   2.2. Set *first* to *ins*.
3. If the insertion point is after the node *pred*:
   3.1. Set node *ins*'s successor to node *pred*'s successor.
   3.2. Set node *pred*'s successor to *ins*.
4. Terminate.



Vstavljanje: Implementacija v Javi

```
      public void insert (Object elem Node pred) {
// Insert elem at a given point in this SLL, either after the node
// pred, or before the first node if pred is null.
      Node ins = new Node(elem, null);
      if (pred == null) {
         ins.next = first;
         first = ins;
      } else {
         ins.next = pred.next;
         pred.next = ins;
      }
}
```

Vstavljanje v dvojni seznam

To insert *elem* at a given point in the DLL headed by (*first*, *last*):

1. Make *ins* a link to a newly-created node with element *elem*, predecessor null, and successor null.
2. Insert *ins* at the insertion point in the forward SLL headed by *first*.
3. Let *succ* be *ins*'s successor (or null if *ins* has no successor).
4. Insert *ins* after node *succ* in the backward SLL headed by *last*.
5. Terminate.

DLL insertion *(2)*

- Auxiliary forward SLL insertion algorithm:

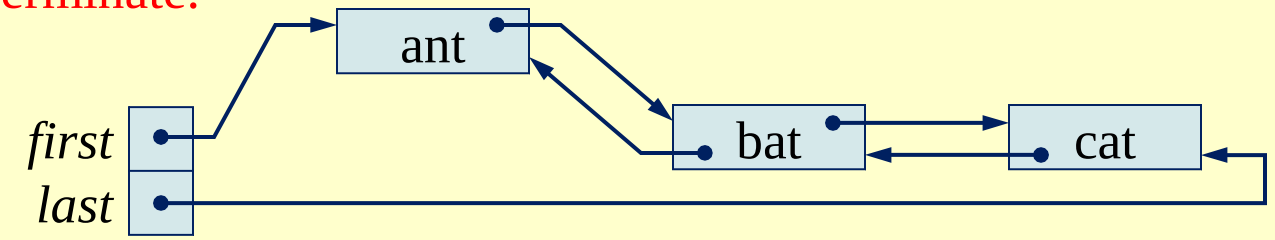
To insert node *ins* at a given point in the forward SLL headed by *first*:

1. If the insertion point is before the first node:
   1.1. Set node *ins*'s successor to *first*.
   1.2. Set *first* to *ins*.
2. If the insertion point is after the node *pred*:
   2.1. Set node *ins*'s successor to node *pred*'s successor.
   2.2. Set node *pred*'s successor to *ins*.
3. Terminate.

Vstavljanje pred prvo vozlišče v DLL (animacija)

To insert *elem* at a given point in the DLL headed by (*first*, *last*):

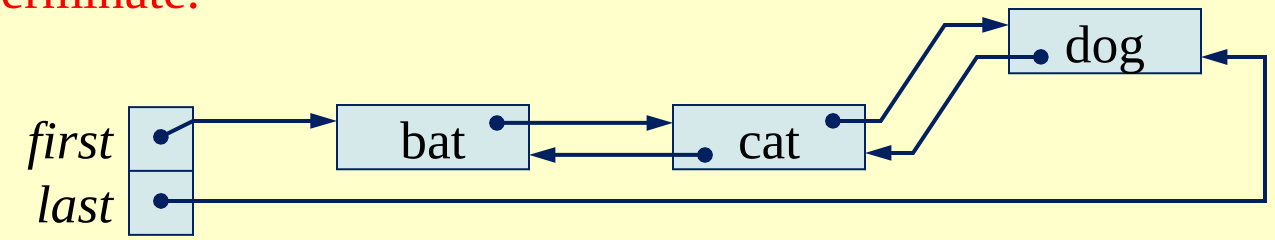
1. Make *ins* a link to a newly-created node with element *elem*, predecessor null, and successor null.
2. Insert *ins* at the insertion point in the forward SLL headed by *first*.
3. Let *succ* be *ins*'s successor (or null if *ins* has no successor).
4. Insert *ins* after node *succ* in the backward SLL headed by *last*.
5. Terminate.



Vstavljanje za zadnje vozlišče v DLL (animacija)

To insert *elem* at a given point in the DLL headed by (*first*, *last*):

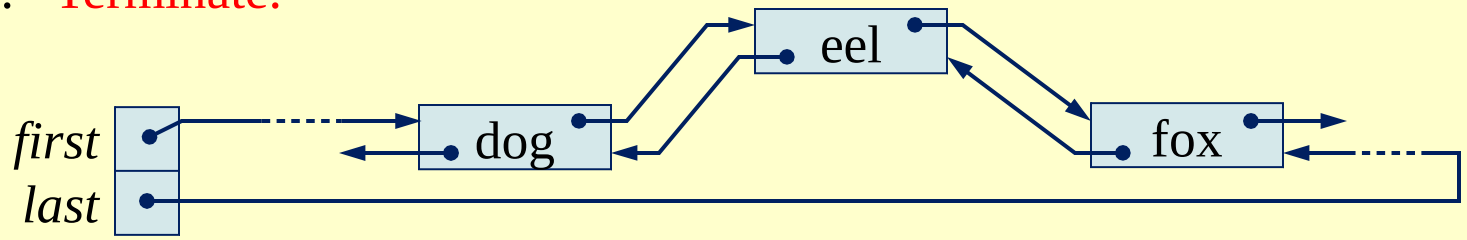
1. Make *ins* a link to a newly-created node with element *elem*, predecessor null, and successor null.
2. Insert *ins* at the insertion point in the forward SLL headed by *first*.
3. Let *succ* be *ins*'s successor (or null if *ins* has no successor).
4. Insert *ins* after node *succ* in the backward SLL headed by *last*.
5. Terminate.



DLL : vstavljanje med vozlišči

To insert *elem* at a given point in the DLL headed by (*first*, *last*):

1. Make *ins* a link to a newly-created node with element *elem*, predecessor null, and successor null.
2. Insert *ins* at the insertion point in the forward SLL headed by *first*.
3. Let *succ* be *ins*'s successor (or null if *ins* has no successor).
4. Insert *ins* after node *succ* in the backward SLL headed by *last*.
5. Terminate.



Brisanje danega vozla s seznama

- Primeri:
  1) deletion of a singleton node;
  2) deletion of the first (but not last) node;
  3) deletion of the last (but not first) node;
  4) deletion of an intermediate node.
- The deletion algorithm needs links to the deleted node’s successor and predecessor.

Brisanje v enojno povezanem seznamu

To delete node *del* from the SLL headed by *first*:

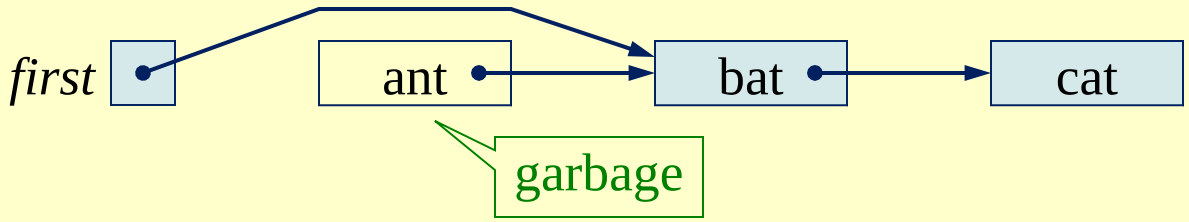
1. Let *succ* be node *del*'s successor.
2. If *del* = *first*:
   2.1. Set *first* to *succ*.
3. Otherwise (if *del* ≠ *first*):
   3.1. Let *pred* be node *del*'s predecessor.
   3.2. Set node *pred*'s successor to *succ*.
4. Terminate.

- But there is no link from node *del* to its predecessor, so step 3.1 can access *del*'s predecessor only by following links from *first*!

Brisanje prvega vozlišča (animacija)

To delete node *del* from the SLL headed by *first*:

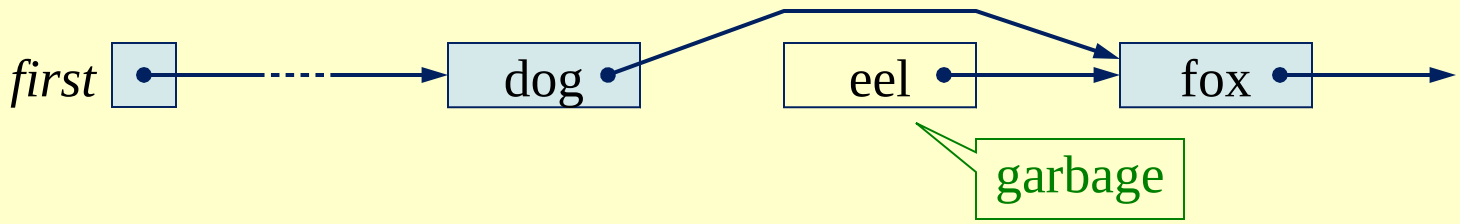
1. Let *succ* be node *del*'s successor.
2. If *del* = *first*:
   2.1. Set *first* to *succ*.
3. Otherwise (if *del* ≠ *first*):
   3.1. Let *pred* be node *del*'s predecessor.
   3.2. Set node *pred*'s successor to *succ*.
4. Terminate.



Brisanje vmesnega ali zadnjega vozlišča

To delete node *del* from the SLL headed by *first*:

1. Let *succ* be node *del*'s successor.
2. If *del* = *first*:
   2.1. Set *first* to *succ*.
3. Otherwise (if *del* ≠ *first*):
   3.1. Let *pred* be node *del*'s predecessor.
   3.2. Set node *pred*'s successor to *succ*.
4. Terminate.



Časovna kompleksnost brisanja

- Analiza:

  Naj bo n dolžina enojno povezanega seznama.

  S korakom 3.1 moramo obiskati vsa vozlišča od prvega do brisanega. Takih vozlišč je med 0 in n–1.

  Povprečno število obiskanih vozlišč= $(n - 1)/2$

  Časovna kompleksnost je torej $O(n)$.

Implementacija brisanja v Javi

```
        public void delete (Node del) {
// Delete node del from this SLL.
        Node succ = del.next;
        if (del == first) {
            first = succ;
        } else {
            Node pred = first;
            while (pred.next != del)
                pred = pred.next;
            pred.next = succ;
        }
}
```

Algoritem brisanja v dvojno povezanem seznamu (DLL)

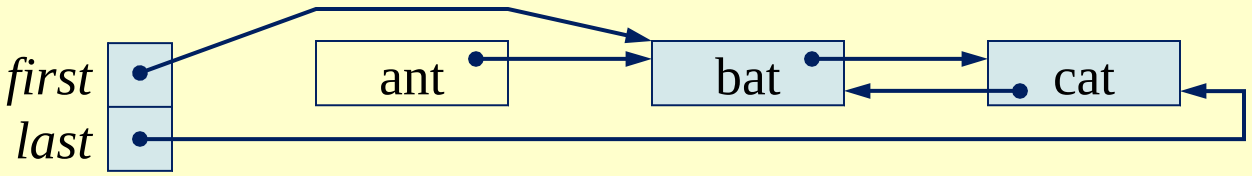
To delete node *del* from the DLL headed by (*first*, *last*):

1. Let *pred* and *succ* be node *del*'s predecessor and successor.
2. Delete node *del*, whose predecessor is *pred*, from the forward SLL headed by *first*.
3. Delete node *del*, whose successor is *succ*, from the backward SLL headed by *last*.
4. Terminate.

Animacija brisanja prvega (ne pa zadnjega) vozlišča

To delete node *del* from the DLL headed by (*first*, *last*):

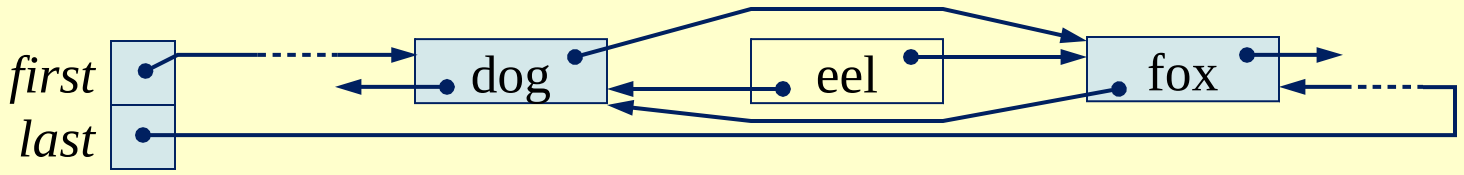
1. Let *pred* and *succ* be node *del*'s predecessor and successor.
2. Delete node *del*, whose predecessor is *pred*, from the forward SLL headed by *first*.
3. Delete node *del*, whose successor is *succ*, from the backward SLL headed by *last*.
4. Terminate.



Brisanje vmesnega vozlišča (animacija)

To delete node *del* from the DLL headed by (*first*, *last*):

1. Let *pred* and *succ* be node *del*'s predecessor and successor.
2. Delete node *del*, whose predecessor is *pred*, from the forward SLL headed by *first*.
3. Delete node *del*, whose successor is *succ*, from the backward SLL headed by *last*.
4. Terminate.



Primerjava algoritmov vstavljanja in brisanja

Algoritem	SLL	DLL
Vstavljanje	$O(1)$	$O(1)$
Brisanje	$O(n)$	$O(1)$

Iskanje dane vrednosti v seznamu

- Algoritem iskanja v neurejenem enojno povezanem seznamu (SLL):

Iščemo obstoj vozlišča z vsebino *target* v seznamu SLL, ki je naslovljen s kazalcem *first*:

1. For each node *curr* in the SLL headed by *first*, repeat:
1.1. If *target* is equal to node *curr*'s element, terminate with answer *curr*.
2. Terminate with answer *none*.

- Linearno iskanje v DLL je podobno, le smer lahko izbiramo.

Kompleksnost iskanja

- Analiza (štejmo primerjave):

  Naj bo n dolžina enojno povezanega seznama.
- Če je iskanje uspešno:

  Povprečno število primerjav= $(n + 1)/2$
- Če je iskanje neuspešno:

  Število primerjav = n
- V obeh primerih imamo časovno kompleksnost $O(n)$.

Implementacija v Javi (primer SLL)

```
    public Node search (Object target) {
// Find which (if any) node of this SLL contains an element equal
to
// target. Return a link to the matching node (or null if there is
// none).
    for (Node curr = this.first;
         curr != null; curr = curr.succ) {
       if (target.equals(curr.element))
         return curr;
    }
    return null;
}
```