


# ABSTRAKTNI PODATKOVNI TIPI



**Podatkovni tip** definira:

1. Množico možnih vrednosti objektov tega tipa
2. Vse možne operacije na objektih tega

Pri strukturiranih podatkovnih objektih se **operacije na posameznih delih strukture dedujejo**.

- to omogoča prožno programiranje, nad katerim pa ima nadzor samo programer in ne prevajalnik
  - programer se mora cel čas zavedati, kako je podatkovna struktura implementirana, kar upočasni programiranje
  - največje število napak pri programiranju je zaradi napačnih (nenadzorovanih) dostopov do podatkov
- 

# ABSTRAKTNI PODATKOVNI TIPI (ADT)

**Abstraktni podatkovni tip** (Abstract Data Type) definira:

1. Strukturo podatkovnega objekta
2. Množico možnih vrednosti za vsak del strukture
3. Vse možne operacije na objektih tega tipa – uporabniški vmesnik

(ni avtomatskega dedovanja!!!)

**Prednosti ADT:**

1. **Varnost** programiranja:

- nadzorovan dostop do podatkov – **napačen dostop prepreči prevajalnik**
- ločen razvoj in testiranje – **varna koda**

2. **Hitrost** programiranja:

- ADT lahko uporabljam v različnih delih programa
- pri uporabi me implementacija ADT ne zanima

# ABSTRAKTNI PODATKOVNI TIPI (ADT)



Različni programski jeziki nudijo različne koncepte za implementacijo ADT:

- modul (npr. modula)
- paket (npr. ada)
- razred objektov (objektno orientirani jeziki, npr. java)

Za vsak jezik se je treba zavedati (ne)zmožnosti idealne implemetacije ADT in morajo biti temu primerno programerji “samodisciplinirani”, da uporabljajo ADT v skladu z definicijo.

(v takih jezikih jih prevajalnik ne more popolnoma nadzirati)



# ABSTRAKTNI PODATKOVNI TIPI

Osnovni abstraktni podatkovni tipi, ki jih potrebujemo za razvoj algoritmov so:

- **seznam (list)** – zbirka elementov, ki se lahko *ponavljajo*; vrstni red elementov je *pomemben*
- **množica (set)** – zbirka elementov, kjer vrstni red *ni pomemben*; elementi se *ne ponavljajo*
- **vrsta (queue)** – zbirka, kjer elemente vedno dodajamo na konec vrste in jih vedno brišemo na začetku vrste
- **sklad (stack)** – zbirka, kjer se elementi dodajajo in brišejo vedno na vrhu (enem koncu) sklada
- **preslikava (map)** – vsakemu elementu  $d$  iz domene priredi ustrezen element  $r$  iz zaloge vrednosti



# ABSTRAKTNI PODATKOVNI TIPI



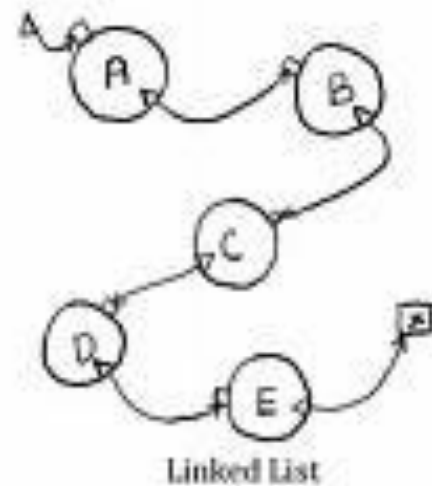
---

V javi so osnovni ADT že implementirani  
**(javanske zbirke – collections)**

Kljub temu bomo razvili nekaj lastnih osnovnih  
implementacij za vsak osnovni ADT, zato da spoznamo  
osnovne principe implementacij in njihove lastnosti.



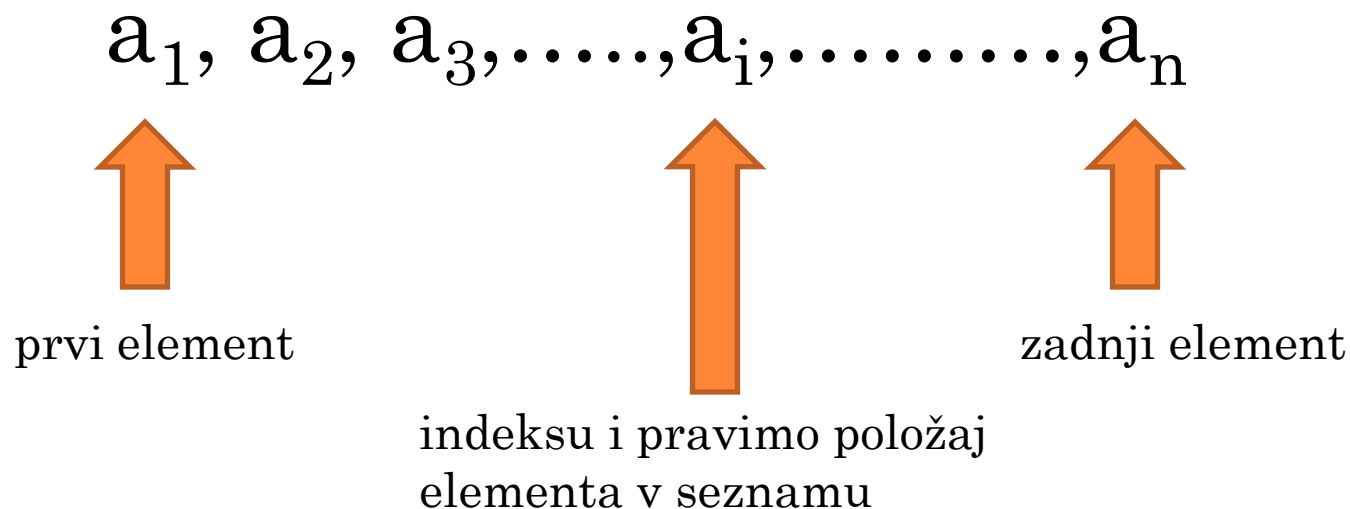
# SEZNAM (List)



# SEZNAM

Seznam je zaporedje 0 ali več elementov, pri čemer velja:

- vrstni red elementov je pomemben
- elementi v seznamu se lahko ponavljajo



# ADT LIST

- **MAKENULL(L)** – naredi prazen seznam L
- **FIRST(L)** – vrne položaj prvega elementa v seznamu
- **LAST(L)** – vrne položaj zadnjega elementa v seznamu
- **NEXT(p, L)** – vrne naslednji položaj položaja p
- **PREVIOUS(p, L)** – vrne predhodni položaj položaja p
- **RETRIEVE(p, L)** – vrne element  $a_p$  na položaju p
- **INSERT(x, p, L)** – vstavi element x na položaj p
- **INSERT(x, L)** – vstavi element x na poljuben položaj
- **DELETE(p, L)** – zbriše element  $a_p$  na položaju p
- **EMPTY(L)** – preveri, če je seznam prazen
- **END(L)** – vrne položaj, ki sledi zadnjemu elementu seznama
- **OVEREND(p, L)** – preveri, če je  $p = \text{END}(L)$
- **LOCATE(x, L)** – poišče položaj elementa x v seznamu
- **PRINTLIST(L)** – po vrsti izpiše vse elemente seznama





# ADT LIST – ABSTRAKTNI RAZRED

```
public abstract class List {  
    public abstract void makenull() ;  
    public abstract Object first() ;  
    public abstract Object last() ;  
    public abstract Object next(Object pos) ;  
    public abstract Object previous(Object pos) ;  
    public abstract Object end() ;  
    public abstract Object retrieve(Object pos) ;  
    public abstract void insert(Object x) ;  
    public abstract void insert(Object x, Object pos) ;  
    public abstract void delete(Object pos) ;  
    public abstract boolean overEnd(Object pos) ;  
    public abstract boolean empty() ;  
} // class List
```



# ADT LIST – ABSTRAKTNI RAZRED

```
public abstract class List {  
  
    public Object locate(Object x) {  
        for (Object iter= first() ; !overEnd(iter) ; iter=next(iter))  
            if (x.equals(retrieve(iter)))  
                return iter ;  
        return null ;  
    }  
  
    public void printList() {  
        for (Object iter = first() ; ! overEnd(iter) ; iter=next(iter))  
            System.out.print(retrieve(iter)+ ", ") ;  
        System.out.println();  
    }  
  
} // class List
```

# PRIMER: DOLŽINA SEZNAMA

---

## Iterativno:

```
public int len() {  
    int n = 0 ;  
    for (Object iter = first() ; ! overEnd(iter) ; iter=next(iter))  
        n++ ;  
    return n ;  
}
```

## Rekurzivno:

- Seznam je sestavljen iz prvega elementa (glave) in iz repa, ki je tudi seznam, le da ima en element manj.
- Seznam je podan z položajem prvega elementa (first).
- Rep je podan s položajem next(first).



# PRIMER: DOLŽINA SEZNAMA

## Rekurzivno:

- 1) Rekurzijska spremenljivka: `pos = položaj elementa na začetku je pos=first`
- 2) Kaj mi pomaga dolžina repa seznama `len(next(first))`?  
`Dolžina celega seznama je len+1.`
- 3) Robni pogoj? `overEnd(pos) → len = 0`
- 4) Splošni primer? `Glej 2)`

```
public int lenRec() {  
    return lenRec(first());  
}
```

```
public int lenRec(Object pos) {  
    if (overEnd(pos)) return 0 ;  
    else return lenRec(next(pos))+1;  
}
```



# PRIMER: VSOTA ELEMENTOV SEZNAMA



## Iterativno:

```
public int sum() {  
    int n = 0 ;  
    for (Object iter = first() ; ! overEnd(iter) ; iter=next(iter))  
        n = n + (Integer)(retrieve(iter)) ;  
    return n ;  
}
```



# PRIMER: VSOTA ELEMENTOV SEZNAMA

## Rekurzivno:

- 1) Rekurzijska spremenljivka: `pos = položaj elementa na začetku je pos=first`
- 2) Kaj mi pomaga vsota elementov repa `sum(next(first))`?  
`Vsota elementov celega seznama je sum+retrieve(first).`
- 3) Robni pogoj? `overEnd(pos) → sum = 0`
- 4) Splošni primer? `Glej 2)`

```
public int sumRec() {  
    return sumRec(first());  
}
```

```
public int sumRec(Object pos) {  
    if (overEnd(pos)) return 0;  
    else return sumRec(next(pos)) + (Integer)retrieve(pos);  
}
```

# ADT LIST



---

## Implementacije ADT LIST:

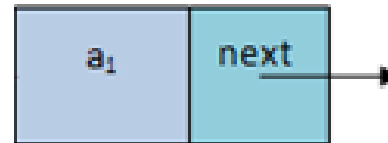
- enosmerni seznam s kazalci
  - dvosmerni seznam s kazalci
  - s poljem
  - z indeksnimi kazalci (kurzorji)
- 
- Implementacije se razlikujejo po časovni in prostorski zahtevnosti posameznih operacij.
  - Imajo poudarek na različnih lastnostih seznama in na različnih podmnožicah operacij.



# ENOSMERNI SEZNAM S KAZALCI

Podatkovna struktura je podana z enim elementom seznama:

```
class ListLinkedNode {  
    Object element;  
    ListLinkedNode next;  
    ...  
}
```



ter s samim seznamom elementov:

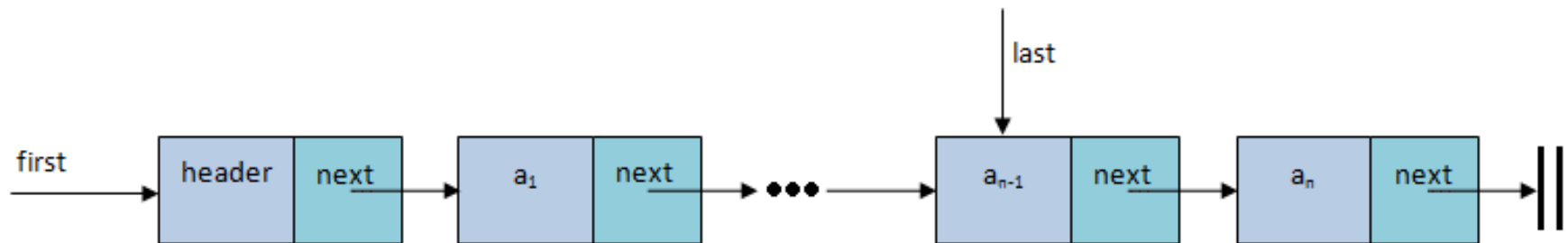
```
public class ListLinked {  
    protected ListLinkedNode first, last;  
    ...  
}
```

Seznam je definiran s položajem prvega in zadnjega elementa.





# ENOSMERNI SEZNAM S KAZALCI



## Položaj elementa zamaknjen!

Zaradi učinkovite implementacije vstavljanja in brisanja je položaj elementa seznama podan s kazalcem na celico, ki vsebuje kazalec (next) na celico z elementom.

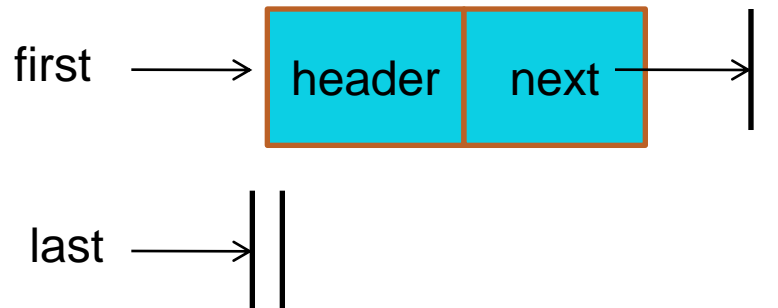


# ENOSMERNI SEZNAM S KAZALCI

Kreiranje praznega seznama

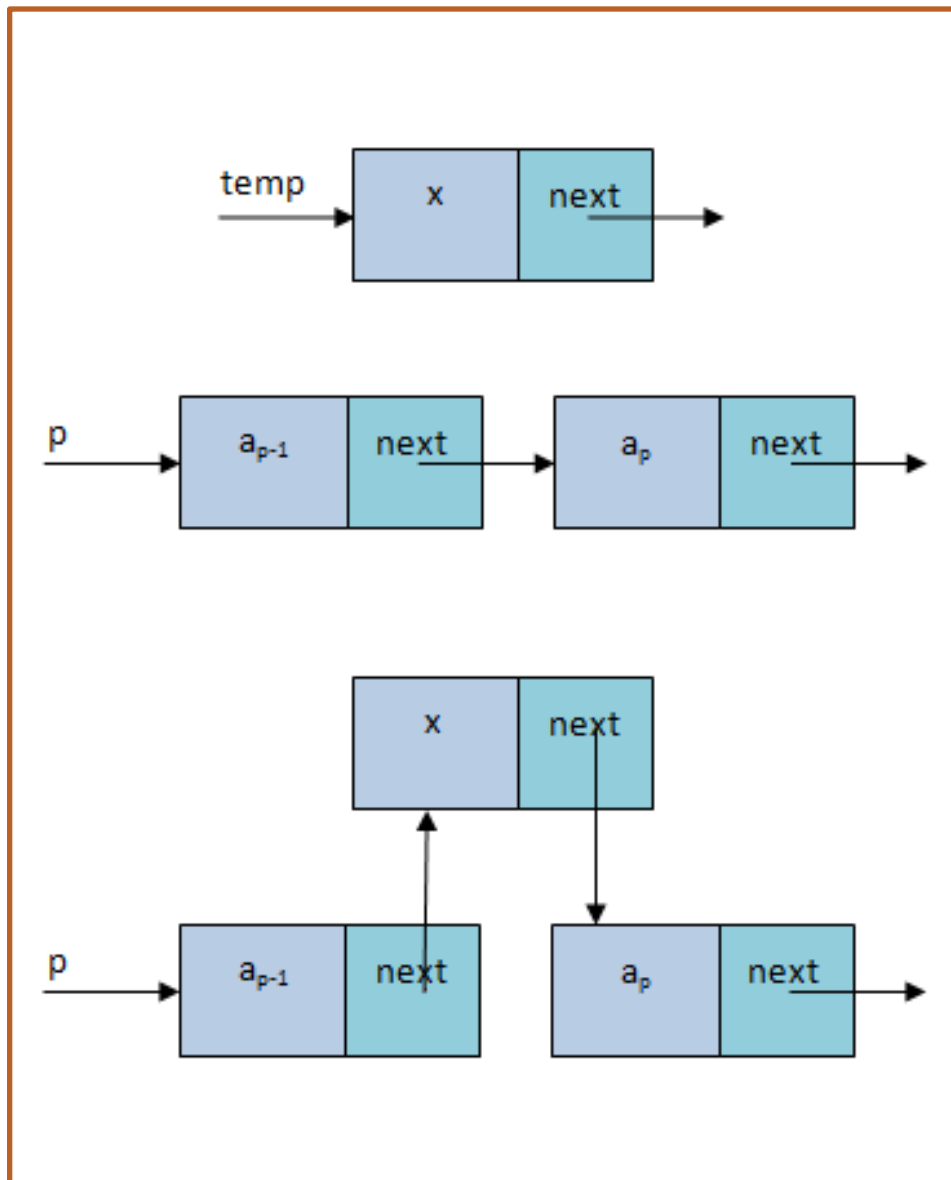
**MAKENULL(L)**

```
public void makenull() {  
    first = new ListLinkedNode(null,null) ;  
    last = null ;  
}
```



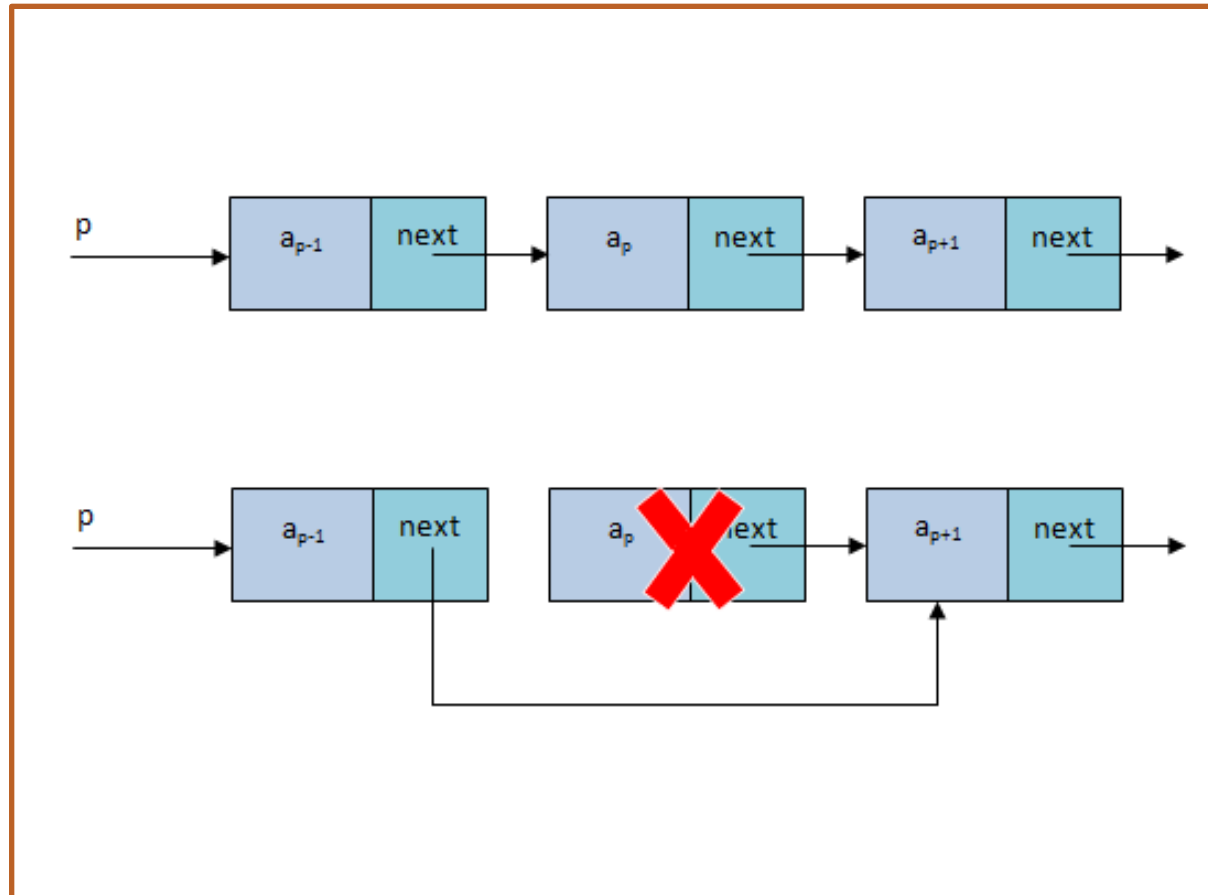
# ENOSMERNI SEZNAM S KAZALCI

Vstavljanje



# ENOSMERNI SEZNAM S KAZALCI

## Brisanje



# ENOSMERNI SEZNAM S KAZALCI

MAKENULL(L)	$O(1)$
FIRST(L)	$O(1)$
LAST(L)	$O(1)$
NEXT(p, L)	$O(1)$
PREVIOUS(p, L)	$O(n)$
RETRIEVE(p, L)	$O(1)$
INSERT(x, p, L)	$O(1)$
INSERT(x, L)	$O(1)$
DELETE(p, L)	$O(1) - O(n)$
EMPTY(L)	$O(1)$
END(L)	$O(1)$
OVEREND(p, L)	$O(1)$
LOCATE(x, L)	$O(n)$
PRINTLIST(L)	$O(n)$



# ENOSMERNI SEZNAM S KAZALCI

Lastnosti enosmernega seznama:

- učinkovito vstavljanje elementa na znan položaj –  $O(1)$ ,
- učinkovito brisanje elementa na znanem položaju –  $O(1)$ ,  
(razen v primeru brisanja zadnjega elementa, ko je  $O(n)$ )
- seznam zaseda samo toliko pomnilnika, kolikor ga zares potrebuje,
- počasna operacija iskanja predhodnika v seznamu –  $O(n)$ ,
- zaporedno premikanje po seznamu v eno smer.

