UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Nina PAVLIN

Verižni seznam

Diplomska naloga

Ljubljana, 2007

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Vsebinsko kazalo

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Zahvala mentorju, profesorju mag. Matiji Lokarju za napotke, strokovno pomoč in predvsem podporo pri izdelovanju diplomske naloge. Zahvaljujem se tudi vsem, ki so kakorkoli pripomogli pri izdelavi diplomske naloge. Posebna zahvala staršem, Juretu, Jasni in Anji.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Program diplomske naloge

V diplomski nalogi predstavite verižni seznam. Omejite se na enojno povezani verižni seznam. Ustrezne algoritme za delo s seznamom razvijte v programskem jeziku java.

mentor

mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

Diplomska naloga govori o osnovnih značilnostih verižnih seznamov. Cilj diplomske naloge je bralcu na razumljiv in enostaven način predstaviti osnove verižnih seznamov.

Diplomsko nalogo sem razdelila v štiri večje dele. Poglavja 1.1, 1.2 in 1.3 diplomske naloge vsebujejo predstavitev verižnega seznama, osnovnih pojmov, lasnosti in osnovnega razreda Vozel, kot elementa verižnega seznama. V ostalih poglavjih pa sem pod drobnogled vzela enojno povezan verižni seznam, enojno povezan verižni seznam s kazalcem na začetek ter enojno povezan verižni seznam s kazalcem na začetek in konec.

Math. Subj. Class. (2000): 68-01, 68P05, 68Q65
Computing Review Class. System (1998): D.1.5, D.3.3., E.1, E.2

Ključne besede: verižni seznam, vozel, prvi vozel, zadnji vozel, podatek, kazalec
Keywords: linked list, node, first node, last node, data element, pointer

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

1 VERIŽNI SEZNAM

1.1 PREDSTAVITEV VERIŽNEGA SEZNAMA

Verižni seznam je podatkovna struktura. Je zaporedje elementov, kjer vsak element vsebuje informacijo, potrebno, da pridemo do naslednjega (in/ali prejšnjega) elementa. Verižni seznam je torej sestavljen iz zaporedja vozlišč, ki so med seboj povezana s povezavami. V posameznem vozlišču so podatki, ki jih hranimo v verižnem seznamu in povezava na eno ali obe sosednji vozlišči. Verižne sezname si lahko predstavljamo kot verigo. Prvi člen verige je povezan z drugim, se pravi, da vsebuje t.i. kazalec na drugi element. Tako je vsak člen verige povezan z naslednjim, oz. vsebuje kazalec na naslednji element. Iz prvega člena verige lahko pridemo do drugega člena, iz drugega do tretjega, itd. Preko prvega člena verige lahko pridemo do vseh ostalih členov, tudi do zadnjega, če bomo le verigo vlekli dovolj dolgo oz. se premikali po njenih členih.

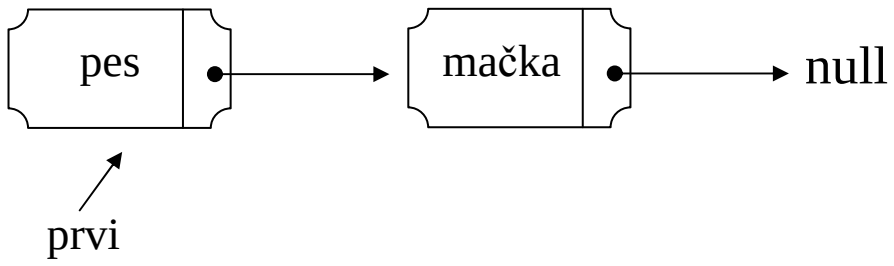
Načinov, kako verižni seznam v izbranem programskem jeziku predstavimo, je več. Povezave lahko izvedemo s pomočjo indeksov v tabeli. Zelo pogosto pa kot povezave uporabimo pomnilniške naslove (reference). Verižni seznam je takrat sestavljen iz vozlov, ki jih povezujejo kazalci. Vsak vozel vsebuje podatek in kazalec na enega ali oba sosednja vozla.

Ločimo dve osnovni obliki verižnih seznamov:

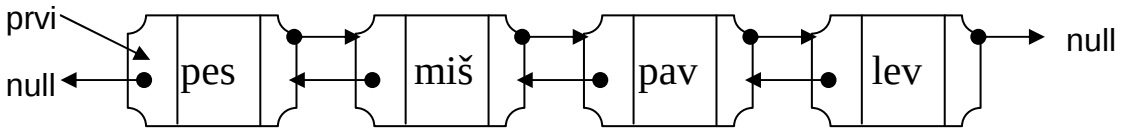
- **Enojno povezani VS**
- **Dvojno povezani VS**

Pri prvi obliki vozel vsebuje kazalec le na naslednji vozel, pri dvojno povezanem seznamu pa vozel vsebuje povezavo tako na naslednji, kot tudi na predhodnji vozel v zaporedju.

Verižne sezname pogosto prikažemo grafično s pomočjo diagramov, kjer povezave predstavimo s puščicami. Če vozel nima svojega naslednika (ali prednika v primeru dvojno povezanega seznama), puščica kaže v prazno. To običajno označimo tako, da jo usmerimo v besedico **`null`**, uporabimo kakšen poseben znak (≡, ▒, ≈), ali pa puščico končamo kako drugače(npr.: ——◆)



SLIKA 1: Enojno povezan verižni seznam



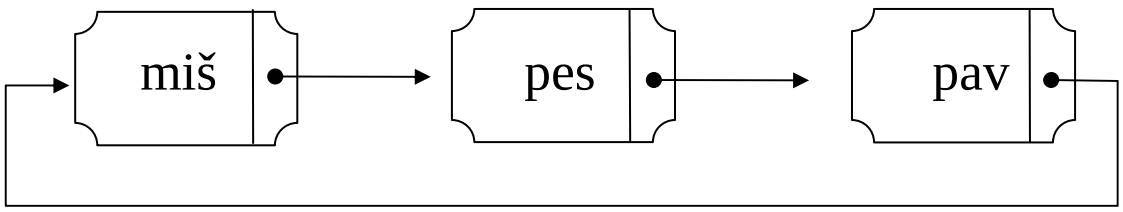
SLIKA 2: Dvojno povezan verižni seznam

Prvi vozel vsebuje povezavo le do svojega **naslednika**. Vozel, katerega kazalec kaže na vozel x, se imenuje **prednik** vozla x. Prvi vozel v verižnem seznamu predhodnika nima, zadnji vozel v verižnem seznamu pa nima naslednika. Pri zgornjem primeru (slika 1) ima vozel s podatkom `mačka` za predhodnika vozel s podatkom `pes`. Naslednika nima, saj je zadnji vozel v verižnem seznamu. Vozel s podatkom `pes` pa nima predhodnika, saj je prvi vozel v verižnem seznamu, ima pa naslednika. To je vozel s podatkom `mačka`.

Dolžina verižnega seznama je število vozlov, ki jih vsebuje. Verižni seznam je **prazen**, če je njegova dolžina 0. Takrat ne vsebuje nobenega vozla. Verižni seznam je lahko poljubno dolg oz. lahko vsebuje poljubno število vozlov.

Pomembna lastnost verižnih seznamov je, da praviloma samih vozlov ne prestavljamo po pomnilniku. Spreminjanje zaporedja vozlov opravljamo z manipulacijo s kazalci (naslovi, kje v pomnilniku vozli dejansko so). Pri spreminjanju kazalcev moramo biti previdni in paziti, da jih prenastavimo pravilno in da kakšnega kazalca ne izgubimo oz. ga ne izbrišemo. Če bi na primer izgubili kazalec prvega vozla verižnega seznama s slike 1 (mu spremenili vrednost tako, da kaže kam drugam), do vozla s podatkom `mačka` ne bi mogli več priti (seveda, če ne bi nanj kazal še kak drug kazalec).

Kot smo že povedali, imamo možnost, da v verižnem seznamu vsak vozel povežemo z enim ali obema sosednjima vozloma. Tako dobimo enojno povezane verižne sezname (slika 1) in dvojno povezane verižne sezname (slika 2). V enojno in dvojno povezanih verižnih seznamih zadnji vozel naslednika nima, saj njegov kazalec kaže na `null`. Lahko pa določimo, da ima zadnji vozel za naslednika prvi vozel. Takemu verižnemu seznamu rečemo **krožni verižni seznam**. Takrat je predhodnik prvega vozla zadnji vozel. Krožni verižni seznam ima seveda lahko tudi samo en element. Takrat njegov kazalec kaže na ta isti vozel.



SLIKA 3: Krožno povezan verižni seznam

1.2 NEKAJ OSNOVNIH POJMOV IN LASTNOSTI VERIŽNIH SEZNAMOV

Povzemimo nekaj osnovnih pojmov in lastnosti verižnih seznamov, ki jih bomo uporabljali v nadaljevanju:

- **Naslednik**

Tako imenujemo naslednji vozel v zaporedju. Vsak element enojno povezanega verižnega seznama vsebuje le kazalec na drug element. Ta element imenujemo njegov naslednik. Zadnji element verižnega seznama nima naslednika. Njegov kazalec kaže v prazno. Če je seznam krožni, je naslednik zadnjega elementa prvi element verižnega seznama.

- **Prednik**

Tako imenujemo predhodnje vozlišče v zaporedju. Prvi element enojno povezanega seznama prednika nima, razen, če je to krožni seznam. Tam je prednik prvega elementa zadnji element.

- **Ničta povezava**

Če kazalec ne kaže na noben vozel, uporabimo ničto povezavo, oziroma povezavo v prazno. Kot smo omenili, jo pogosto označimo z `null`. To je namreč v več programskih jezikih rezervirana beseda, ki jo uporabljamo za označevanje povezav, ki ne kažejo na noben konkreten pomnilniški naslov. Kazalec zadnjega elementa verižnega seznama je torej ničta povezava in določa konec verižnega seznama.

- **Prvo vozlišče**

To je prvi element verižnega seznama. Nima prednika, temveč samo naslednika. V krožnem seznamu se pojem vrstnega reda vozlov izgubi. Zato je prvo vozlišče tisto, ki ga določimo sami oz. kamor kaže povezava za prvo vozlišče v zaporedju. V krožnem seznamu ima prvo vozlišče tako naslednika kot tudi prednika, saj je prednik tega vozlišča »zadnji« element krožnega seznama.

- **Zadnje vozlišče**

Je zadnji element dvojno ali enojno povezanega verižnega seznama. Ima prednika, ne pa tudi naslednika. Njegov kazalec je ničta povezava. V krožnem seznamu je zadnje vozlišče tisto, ki je prednik prvega vozlišča.

Pomembni lastnosti verižnega seznama, ki smo ju že omenili, sta:

- **Število elementov verižnega seznama je poljubno**

Dolžina verižnega seznama je število elementov (vozlov) v njem. Verižni seznam nima omejitve glede dolžine, kot jih ima npr. tabela. Zaradi te lastnosti lahko elemente poljubno dodajamo in odstranjujemo iz verižnega seznama, ne da bi imeli probleme s prostorom, ki je na voljo.

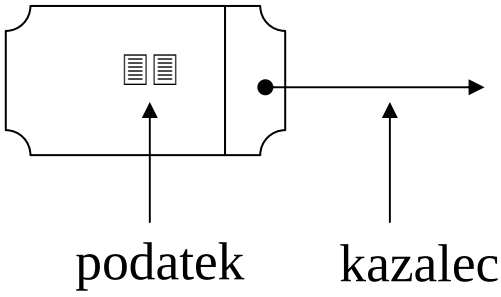
- **Delo s kazalci**

Vse spremembe vrstnega reda vozlišč izvajamo le s spremembami povezav oz. pravilnim prenastavljanjem kazalcev. S to lastnostjo element enostavno odstranimo, ga dodamo ali samo

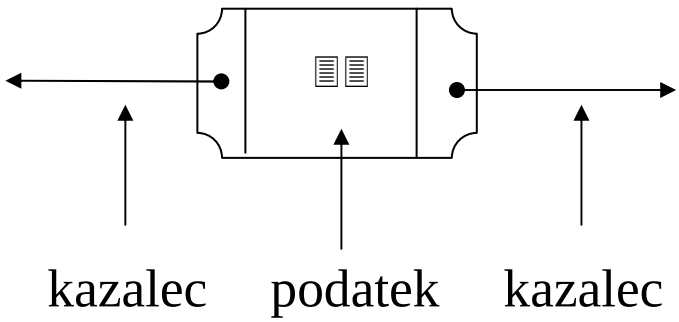
spremenimo vrstni red vozlišč. Paziti moramo le, da med tem, ko spreminjamo vrstni red, ne izgubimo še zadnjega kazalca na določeno vozlišče. Do takega vozlišča takrat ne moremo več priti.

1.3 VOZEL

Kot smo dejali, je verižni seznam sestavljen iz zaporedja elementov – vozlov, kjer vsak vozel poleg podatkov vsebuje tudi kazalec (ali kazalca, če imamo opraviti z dvojno povezanim seznamom) na naslednji (in prejšnji pri dvojno povezanem seznamu) element zaporedja. Vozel je torej osnovni sestavni del vsakega verižnega seznama. Ima komponente, v katerih so podatki, ki jih želimo hraniti v verižnem seznamu in povezavo na naslednji vozel – na naslednika. V primeru dvojno povezanega seznama vsebuje še povezavo na prejšnji vozel v verižnem seznamu – na prednika. Če naslednika (ali prednika) ni, uporabimo ničto povezavo.



SLIKA 4: Vozel v enojno povezanem verižnem seznamu



SLIKA 5: Vozel v dvojno povezanem verižnem seznamu

Omenili smo že, da je način predstavitve verižnega seznama v programskem jeziku naša izbira. Lahko ga predstavimo s pomočjo indeksov v tabeli, najpogosteje pa kot povezave uporabimo pomnilniške naslove (reference). Slednji način predstavitve verižnih seznamov bomo uporabili tudi mi. Za nadaljno delo z verižnimi seznami bomo uporabili reference, rekli jim bomo tudi kazalci.

1.4 VOZEL ENOJNO POVEZANEGA VERIŽNEGA SEZNAMA

Začnimo z enostavnim zgledom. Predpostavimo, da bomo v vsakem vozlu hranili po eno celo število. Oglejmo si, kako bi v programskem jeziku java sestavili razred, ki bi predstavljal vozel v enojno povezanem verižnem seznamu.

Sestavili bomo razred `Vozel`, ki bo vseboval konstruktorje in metode, ki jih potrebujemo za delo z njim. Objekt tipa `Vozel` bo imel dve komponenti. Prva, imenovali jo bomo `podatek`, bo tipa `int`. V njej bomo hranili dejanski podatek. Drugo komponento bomo poimenovali `naslednji` in bo tipa `Vozel` - torej kazalec na istovrstni objekt in bo določala naslednika. Obe komponenti bosta imeli dostop `private`, tako da do njiju lahko dostopamo le znotraj tega razreda, izven razreda pa le preko ustreznih metod.

```
private int podatek; // v vozlu hranimo cela števila – podatek
private Vozel naslednji; // referenca – povezava na naslednji vozel
                         // (naslednik)
```

1.4.1 Konstruktorji

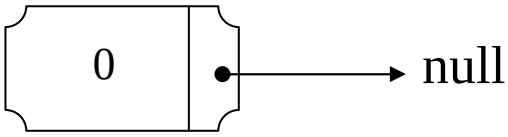
Razred bo vseboval tudi nekaj konstruktorjev. Poleg osnovnega konstruktorja brez parametrov bomo razred `Vozel` opremili še z nekaj dodatnimi konstruktorji, ki nam bodo olajšali delo pri izpeljavi različnih metod za delo z verižnim seznamom.

- `Vozel()`: naredi vozel, kjer hranimo podatek 0 in nima naslednika

Za povezavo naslednji moramo torej uporabiti ničto povezavo. V programskem jeziku java to naredimo tako, da za vrednost spremenljivke uporabimo rezervirano besedo **null**.

Vozel vsebuje komponenti `podatek` in `naslednji`. V prvo shranimo vrednost 0 in v drugo `null`.

```
public Vozel(){
   podatek = 0;
   naslednji = null;
}
```

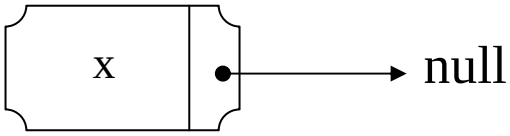


SLIKA 6: Objekt narejen s konstruktorjem `Vozel()`

- `Vozel(int x)`: naredi vozel, kjer hranimo podatek x in ki nima naslednika

Za ničto povezavo uporabimo rezervirano besedo `null`. V komponento `podatek` shranimo dobljeno vrednost x.

```
public Vozel(int x){
   podatek = x;
   naslednji = null;
}
```

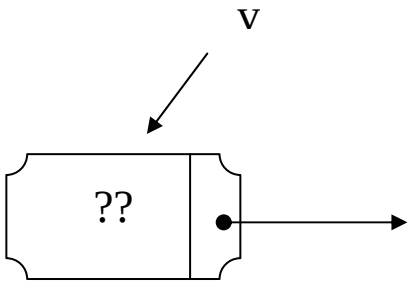


SLIKA 7: Objekt narejen s konstruktorjem `Vozel(int x)`

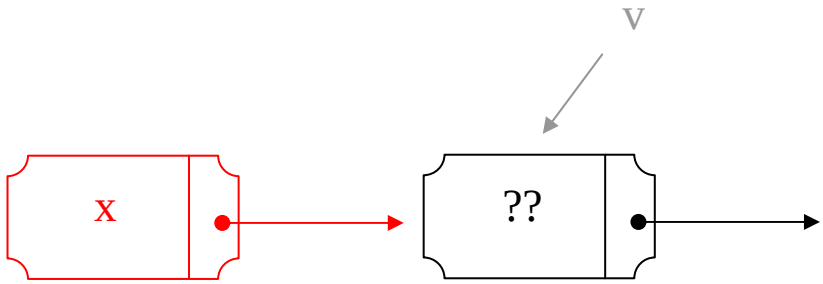
- `Vozel(int x, Vozel v)`: naredi vozel z vrednostjo x in kazalcem na vozel v

S tem konstruktorjem v komponento `podatek` shranimo vrednost x, v komponento `naslednji` pa kazalec `v`.

```
public Vozel(int x, Vozel v){
   podatek = x;
   naslednji = v;
}
```

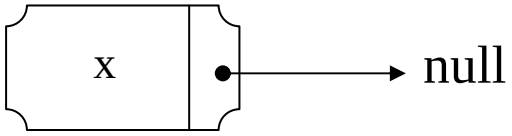


SLIKA 8: Objekt `v` uporabljen kot argument konstruktorja `Vozel(int x, Vozel v)`



SLIKA 9: Objekt (označen rdeče) narejen s konstruktorjem `Vozel(int x, Vozel v)`

Tukaj je potrebno omeniti, da je pravilen tudi klic metode, pri katerem kot referenco uporabimo ničto povezavo (`null`). Takrat kazalec našega novo nastalega objekta ne kaže na objekt `v`, temveč na `null`. Klic metode bo `Vozel(x, null)`.



SLIKA 10: Objekt narejen z uporabo konstruktorja `Vozel(int x, null)`

1.4.2 Primeri uporabe konstruktorjev razreda Vozel

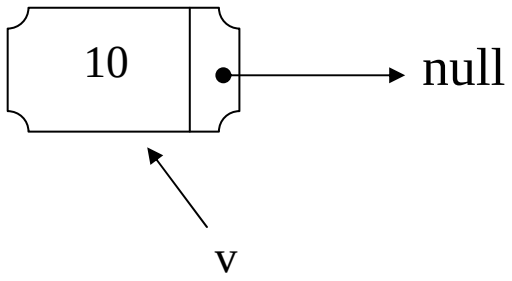
Konstruktorje, s katerimi ustvarimo nov objekt, poznamo. Kako pa nov objekt tipa `Vozel` naredimo? Da bomo vedeli, kateri konstruktor potrebujemo, moramo vedeti, kakšne naj bodo vrednosti njegovih komponent. Recimo, da potrebujemo objekt z vrednostjo 10, vendar ne vemo, kdo je njegov naslednik. Takrat uporabimo konstruktor oblike `Vozel(int x)`, ki nastavi vrednost spremenljivke `podatek` na `x`, vrednost spremenljivke `naslednji` pa je `ničta` povezava (ima torej vrednost `null`). Splošna oblika je

```
Vozel v = new Vozel(<izraz tipa int>);
```

za naš primer pa

```
Vozel v = new Vozel(10);
```

S tem smo naredili spremenljivko `v`, ki pokaže na nov objekt tipa `Vozel`, katerega vrednost podatka je 10, njegov kazalec pa je ničta povezava. V tem primeru (ko je kazalec ničta povezava) pogosto rečemo, da kazalec kaže v prazno.



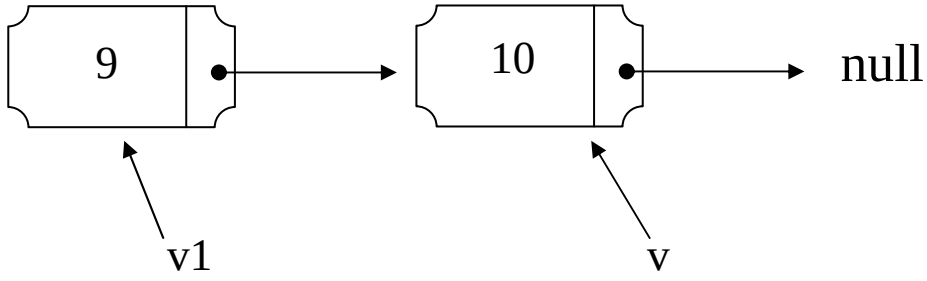
SLIKA 11: Objekt narejen s konstruktorjem `Vozel(10)`

Je z danimi konstruktorji mogoče narediti verižni seznam, oziroma natančneje povedano, povezano zaporedje vozlov? Z uporabo konstruktorja `Vozel(int x)` smo dobili vozel z vrednostjo 10 in s kazalcem v prazno. Samo z enim elementom v našem zaporedju vozlov nismo zadovoljni. Naredimo nov objekt vozel z vrednostjo 9 in ga dodamo na začetek. Uporabimo konstruktor tipa `Vozel(int x, Vozel v)`, torej

```
Vozel v1 = new Vozel(<izraz tipa int>, <izraz tipa Vozel>);
```

za naš primer pa

```
Vozel v1 = new Vozel(9, v);
```

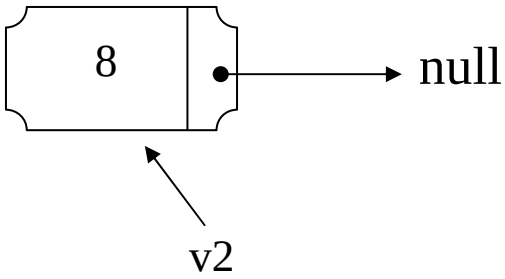


SLIKA 12: Zaporedje vozlov narejeno s konstruktorjema `Vozel(10)` in `Vozel(9, v)`

Dobili smo nov vozel z vrednostjo `9`, njegov kazalec pa kaže na vozel `v`.

Sedaj pa želimo na konec našega zaporedja vozlov dodati nov objekt s podatkom 8. Najprej bomo uporabili konstruktor vrste `Vozel(int x)`. Klic metode bo:

```
Vozel v2 = new Vozel(8);
```

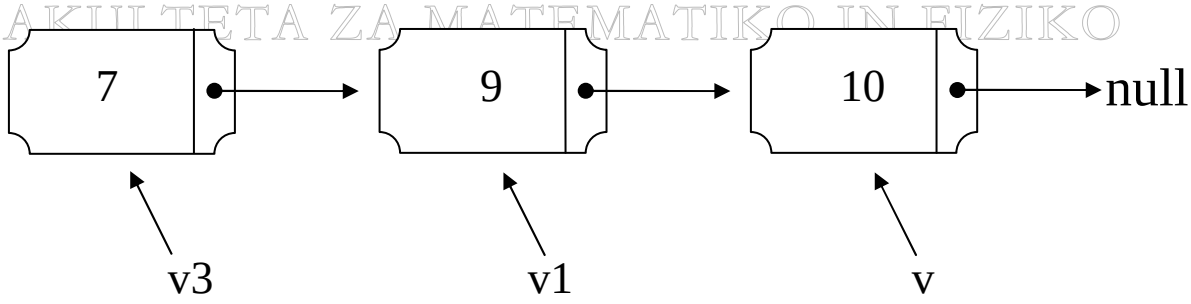


SLIKA 13: Slika narejena s konstruktorjem `Vozel(8)`

Z uporabo do sedaj znanih konstruktorjev s kazalcem vozla `v` (slika 12) ne moremo pokazati na vozel `v2`. Lahko pa bi nov vozel dodali na začetek našega zaporedja vozlov, vendar bi morali uporabiti drug konstruktor. Vozla `v2`, ko smo ga enkrat že naredili, samo z danimi konstruktorji ne moremo več dodati v naše zaporedje vozlov, saj konstruktor vedno ustvari nov objekt. Vozla `v2` torej ne moremo več dodati v zaporedje.

Če želimo v verižni seznam dodati nov element in imamo na voljo le konstruktorje, ga lahko dodamo le na začetek. Klic konstruktorja:

```
Vozel v3 = new Vozel(7, v1);
```



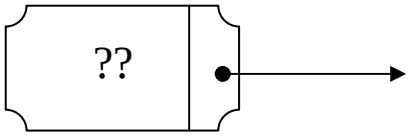
SLIKA 14: Dodan vozel na prvo mesto k zaporedju s slike 12

Ker konstruktorji vedno ustvarijo nov objekt, z njimi kazalcev obstoječih vozlov ne moremo preusmerjati. Potrebujemo dodatne metode. Te metode bodo omogočale, da bomo lahko spremenili podatek v vozlu, prenakazali njegov kazalec, vrnili vrednost podatka v vozlu ali vrnili njegovo referenco oz. naslednika (zvedeli, kam kaže kazalec v vozlu).

1.4.3 Metode

- `nastaviPodatek(int x)`: nastavi podatek v vozlu

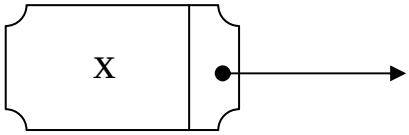
Vozlu spremenimo podatek na vrednost `x`. Kazalca ne spreminjamo.



SLIKA 15: Vozel pred uporabo metode `nastaviPodatek(int x)`

```
public void nastaviPodatek(int x){
   podatek = x;
}
```

Nova vrednost podatka je `x`.

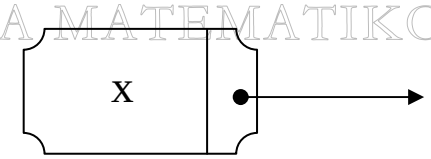


SLIKA 16: Vozel po uporabi metode `nastaviPodatek(int x)`

- `vrniPodatek()`: vrne podatek, ki je v vozlu

Vrne podatek, ki je shranjen v vozlu. Vozel ostane tak, kot je bil pred uporabo metode.

```
public int vrniPodatek(){
   return podatek;
}
```

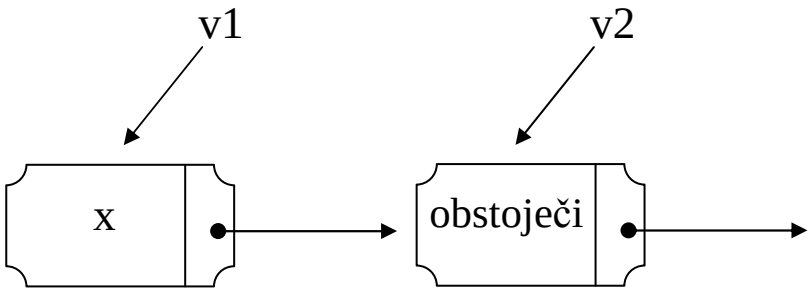


SLIKA 17: Vozel za uporabo metode `vrniPodatek()`

Metoda vrne vrednost podatka, torej `x`.

- `nastaviNasled(Vozel v)`: spremeni naslednika

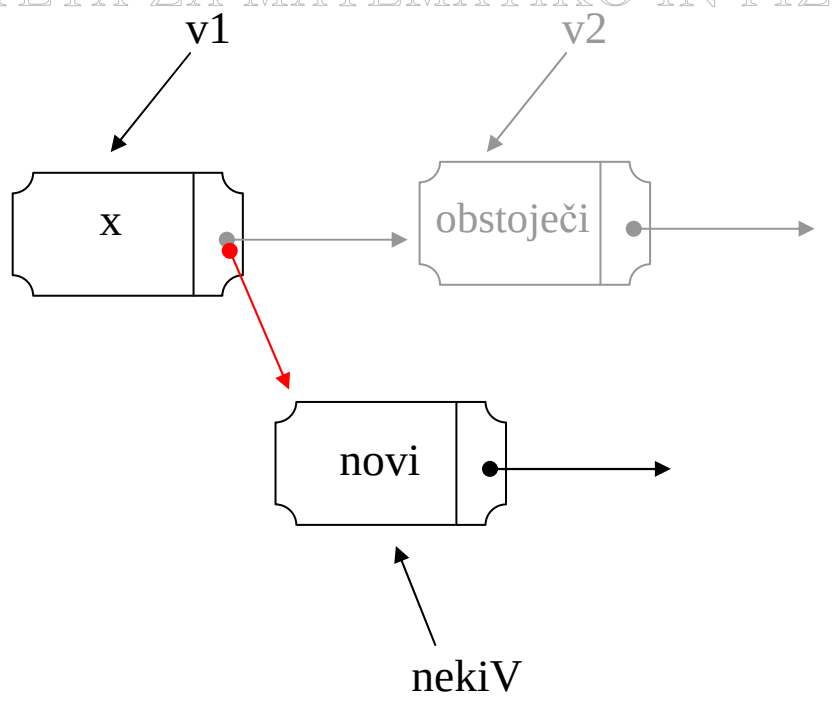
Spremenimo naslednika vozlu `v1`. Naj bo njegov naslednik pred uporabo metode vozel `v2`.



SLIKA 18: Vozel `v1` pred uporabo metode `nastaviNasled(Vozel v)`

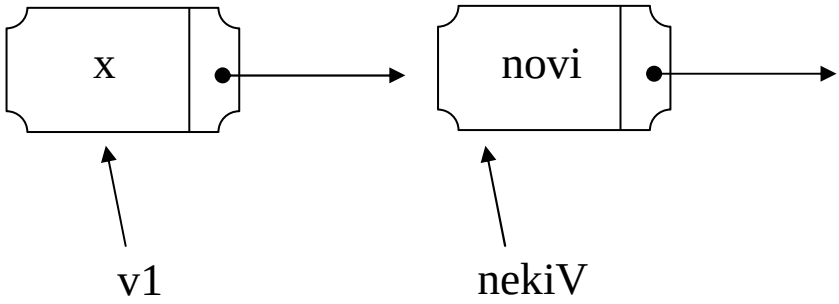
```
public int nastaviNasled(Vozel v){
   naslednji = v;
}
```

Po uporabi metode (s klicem `v1.nastaviNasled(nekiV)`) se kazalec vozla `v1` spremeni. Ne kaže več na vozel `v2`, temveč na vozel `nekiV`. Naslednik vozla `v1` je sedaj vozel `nekiV`.



SLIKA 19: Prikaz delovanja metode `nastaviNasled(Vozel v)` ob klicu `v1.nastaviNasled(nekiV).` Z rdečo barvo je označena nastala sprememba.

Po izvedbi ukaza imamo povezavo med vozloma `v1` in `nekiV`. Vozel `v2` še obstaja, le v naši verigi ga ni.

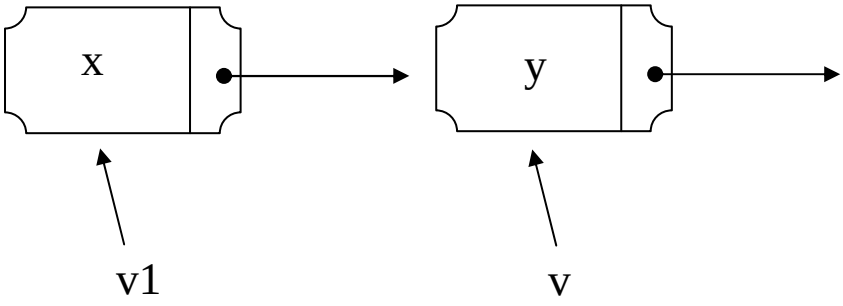


SLIKA 20: Zaporedje vozlov po uporabi metode `nastaviNasled(Vozel v)`

- `vrniNasled()`: vrne naslednika (naslednji vozel) vozla `v`

```
public Vozel vrniNasled(){
   return naslednji;
}
```

Ponazorimo delovanje metode. Denimo, da imamo vozla povezana kot kaže slika:



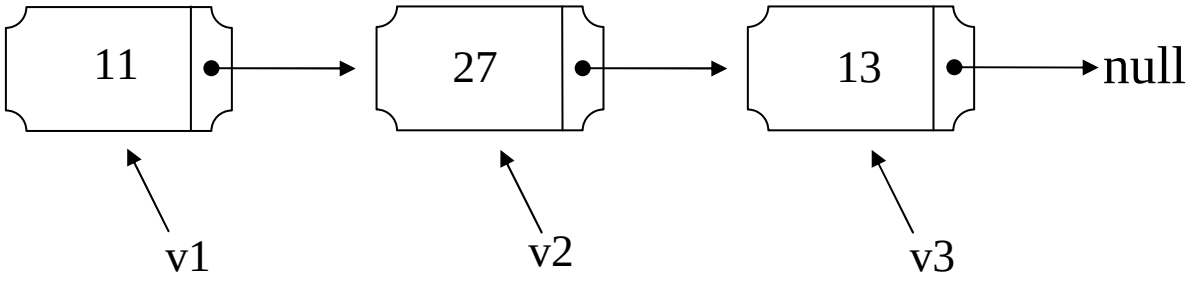
SLIKA 21: Prikaz uporabe metode `vrniNasled()`

Če metodo `vrniNasled()` uporabimo nad vozlom s podatkom x (torej kot `v1.vrniNasled()`), nam vrne vrednost komponente `naslednji` tega vozla, torej referenco na vozel s podatkom y (vozel `v`).

Če je vozel nad katerim metodo uporabimo, zadnji v našem zaporedju, metoda vrne `null`.

1.4.4 Primeri uporabe metod razreda Vozel

Pogledali bomo, kako metode uporabljamo nad obstoječim zaporedjem vozlov. Klic in delovanje metod bomo prikazali na naslednjem zaporedju vozlov:



SLIKA 22: Primer zaporedja vozlov za prikaz uporabe metod nad vozli

metoda `vrniPodatek()`

Klicanje metode nad določenim vozlom nam omogoča kazalec, ki nanj kaže. Tako lahko metodo `vrniPodatek()` pokličemo nad vsemi vozli s slike 22.

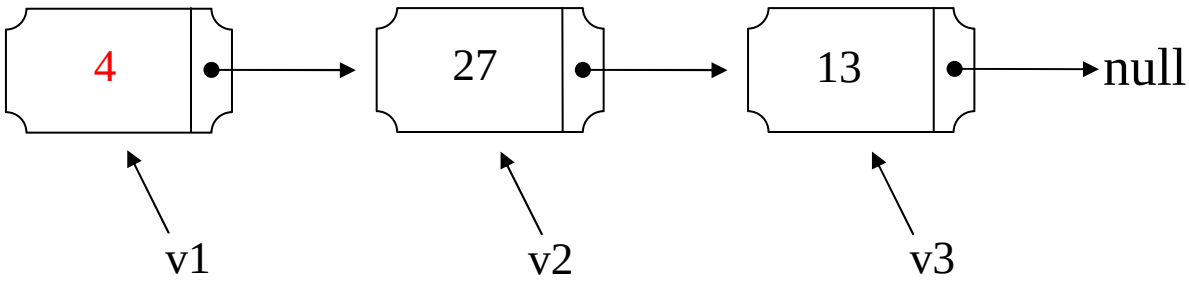
- Klic metode za prvi vozel: `v1.vrniPodatek()`
  Metoda vrne vrednost podatka vozla v1, torej 11.
- Klic metode za drugi vozel: `v2.vrniPodatek()`
  Metoda vrne vrednost podatka vozla v2. Kot rezultat dobimo število 27.

metoda `nastaviPodatek(int x)`

Metoda spremeni vrednost podatka v vozlu. Poglejmo, kaj se zgodi ob klicu metode nad prvim vozlom. Spremenimo podatek iz 11 na 4.

- Klic metode za prvi vozel: `v1.nastaviPodatek(4)`

Metoda spremeni vrednost podatka. Nova vrednost je 4. Na sliki označimo, kaj se zgodi po klicu metode.



SLIKA 23: Stanje po klicu `v1.nastaviPodatek(4)`

metoda `vrniNasled()`

Metoda vrne vrednost komponente `naslednji` vozla, nad katerim metodo pokličemo. Pokličimo metodo nad vozlom `v1`. Vrnila nam bo naslednji vozel, se pravi referenco nanj. V našem primeru torej referenco na vozel, na katerega kaže tudi kazalec `v2` (vozel z vrednostjo 27).

- Klic metode za prvi vozel: `v1.vrniNasled()`

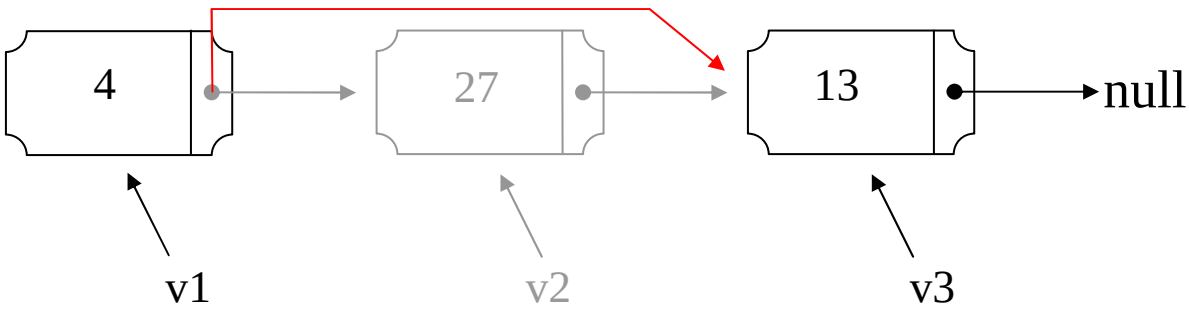
Dobimo referenco na vozel, na katerega kaže kazalec `v2`, torej naslednika vozla `v1`.

metoda `nastaviNasled(Vozel v)`

Metoda objektu (vozlu) nastavi (spremeni) naslednika. Metodo pokličimo nad vozlom `v1`. Kot argument metode bomo izbrali `v3`.

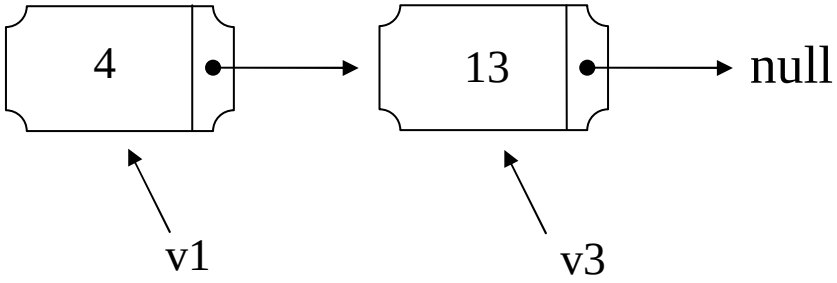
- Klic metode za prvi vozel: `v1.nastaviNasled(v3)`

Po klicu metode se kazalec vozla `v1` (torej komponenta `naslednji`) prestavi (pokaže) na vozel `v3`.



SLIKA 24: Zaporedje vozlov po klicu `v1.nastaviNasled(v3)`

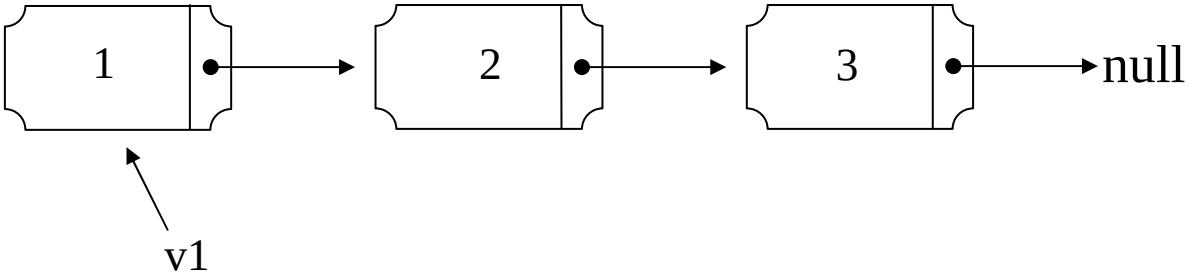
Na zgornji sliki je nova referenca vozla `v1` prikazana z rdečo barvo in kaže na vozel `v3`. Novo zaporedje vozlov je:



SLIKA 25: Verižni seznam po klicu `v1.nastaviNasled(v3)`

Če predpostavimo, da je začetek našega verižnega seznama (prvi vozel) vozel `v1`, potem ima naš verižni seznam po klicu metode dva elementa. Prvi je vozel `v1`, drugi vozel `v3`. Referenco vozla `v1` smo spremenili z vozla `v2` na `v3`.

Ogledali smo si nekaj osnovnih primerov uporabe metod. Na vseh primerih smo imeli kazalce na vsak posamezen vozel v zaporedju. Kaj pa naredimo v primeru, ko kazalca na vozel ni? Kako potem dostopamo do željenega vozla? Recimo, da imamo zaporedje vozlov, kot je prikazano na spodnji sliki.



SLIKA 26: Verižni seznam s kazalcem na prvi vozel

Neposredno lahko dostopamo do prvega vozla v zaporedju, saj nanj kaže kazalec `v1`. Kako bi izpisali podatek vmesnega vozla, torej vrednost 2?

Uporabili bomo metodo `vrniNasled()`, ki nam vrne naslednika vozla, nad katerim pokličemo metodo. Metodo bomo poklicali nad vozlom `v1` in kot rezultat dobili drugi vozel v našem zaporedju (to je naslednik vozla `v1`). To je ravno tisti vozel, katerega podatek želimo izpisati. Tako klic metode `v1.vrniNasled()` vrne ravno vozel, ki ga potrebujemo. Nad njim le še pokličemo metodo `vrniPodatek()`. Če vse skupaj združimo, dobimo

```
v1.vrniNasled().vrniPodatek()
```

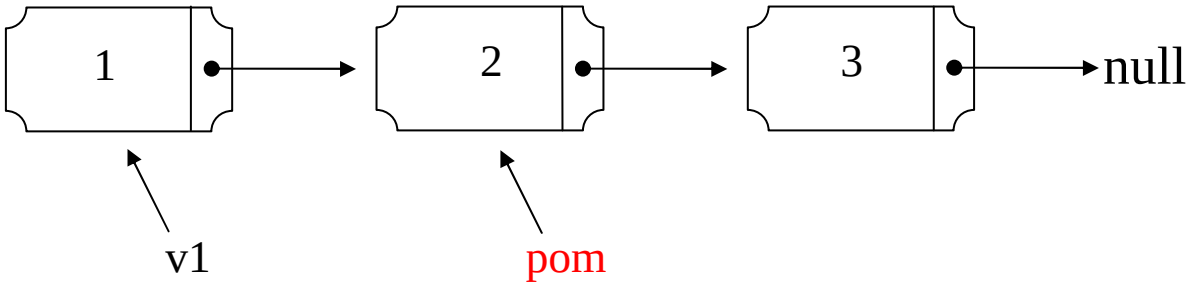
Ukaz je dolg in nepregleden. Raje ga razbijmo v dva koraka in uporabimo pomožno spremenljivko. Deklarirali bomo novo spremeljivko `pom`, v kateri lahko hranimo naslove objektov tipa `Vozel`.

```
Vozel pom;
```

Potem bomo s pomočjo klica metode `vrniNasled()` nad vozlom `v1` naslov drugega vozla shranili v spremenljivko `pom`. Rekli bomo, da smo se premaknili do našega vozla.

```
pom = v1.vrniNasled();
```

Sedaj je stanje takšno:



SLIKA 27: Verižni seznam s pomožno spremenljivko

Nato bomo izpisali vrednost podatka tistega vozla, na katerega sedaj kaže `pom` (v našem primeru je to drugi vozel v zaporedju).

Kazalec `pom` kaže na naš drugi vozel. Metodo `vrniPodatek()` pokličemo nad njim.

```
pom.vrniPodatek();
```

Končni rezultat je isti, le način je veliko bolj pregleden.

Verižni seznam, kot ga poznamo do sedaj, ima kazalec le na prvi element. Če želimo ohraniti dostop do celotnega zaporedja vozlov, kazalca na prvi element ne smemo prestavljati. S pomočjo kazalca na prvi element v zaporedju in pomožnega kazalca (v našem primeru `pom`) lahko dostopamo do vseh ostalih elementov v zaporedju. Kazalec na prvi element vedno kaže na začetek, kazalec `pom` pa premikamo po zaporedju. Tako bomo takrat, ko bomo želeli pregledati vse vozle v verigi, uporabili zanko:

```
Vozel pom;
pom = v1; //pokažemo na prvi vozel
while (pom != null) { //dokler ne "pademo" z verige
   //nekaj naredimo z vozlom, na katerega kaže pom
   pom = pom.vrniNasled(); //prestavimo se naprej
}
```

Uporaba pomožnega kazalca je zelo uporaben način in si ga je vredno zapomniti.

Vidimo, da z uporabo metode `vrniNasled()` zlahka pridemo do naslednika vozlišča. Kako pa je s prednikom? Ali lahko dostopamo do prednika določenega vozla? Če torej opazujemo naš primer - ali lahko preko vozla `v3` oz. zadnjega vozla dostopamo do drugega vozla v seznamu. Ker imamo opraviti z enojno povezanim verižnim seznamom, to ne gre. Po kazalcu se lahko »premikamo« le v smeri puščice. Zakaj? Spomnimo se, da so te puščice naslovi. Naslovnik ne ve, od kod prihaja tisti, ki kaže nanj. Poljubni vozel v enojno povezanem verižnem seznamu ve le, kdo je njegov naslednik.

O svojem predhodniku ne ve ničesar. Tako tudi vozel v3 ne ve, da je njegov predhodnik vozel s podatkom 2. Pri dvojno povezanih seznamih je zgodba seveda drugačna, saj vozel pozna svojega predhodnika. Vendar bomo v tej diplomski nalogi podrobneje obravnavali le enojno povezane verižne sezname.

1.4.5 Razred Vozel

Strnimo naše znanje v razred, ki bo vseboval vse zgoraj opisane konstruktorje in metode.

```
public class Vozel{
   private Vozel naslednji; //referenca – povezava na naslednji vozel
   private int   podatek; //v vozlu hranimo cela števila

   //kostruktorji

   public Vozel(){ //vozel s podatkom 0 in referenco null
      podatek = 0;
      naslednji = null;
   }

   public Vozel(int x){ //vozel s podatkom x in referenco null
      podatek = x;
      naslednji = null;
   }

   public Vozel(Vozel nasl){ //vozel s podatkom 0 in referenco nasl
      podatek = 0;
      naslednji = nasl;
   }

   public Vozel(int x, Vozel nasl){ //vozel s podatkom x in referenco nasl
      podatek = x;
      naslednji = nasl;
   }

   // metode

   public void nastaviPodatek(int x){ //spremeni podatek vozla
      podatek = x;
   }
```

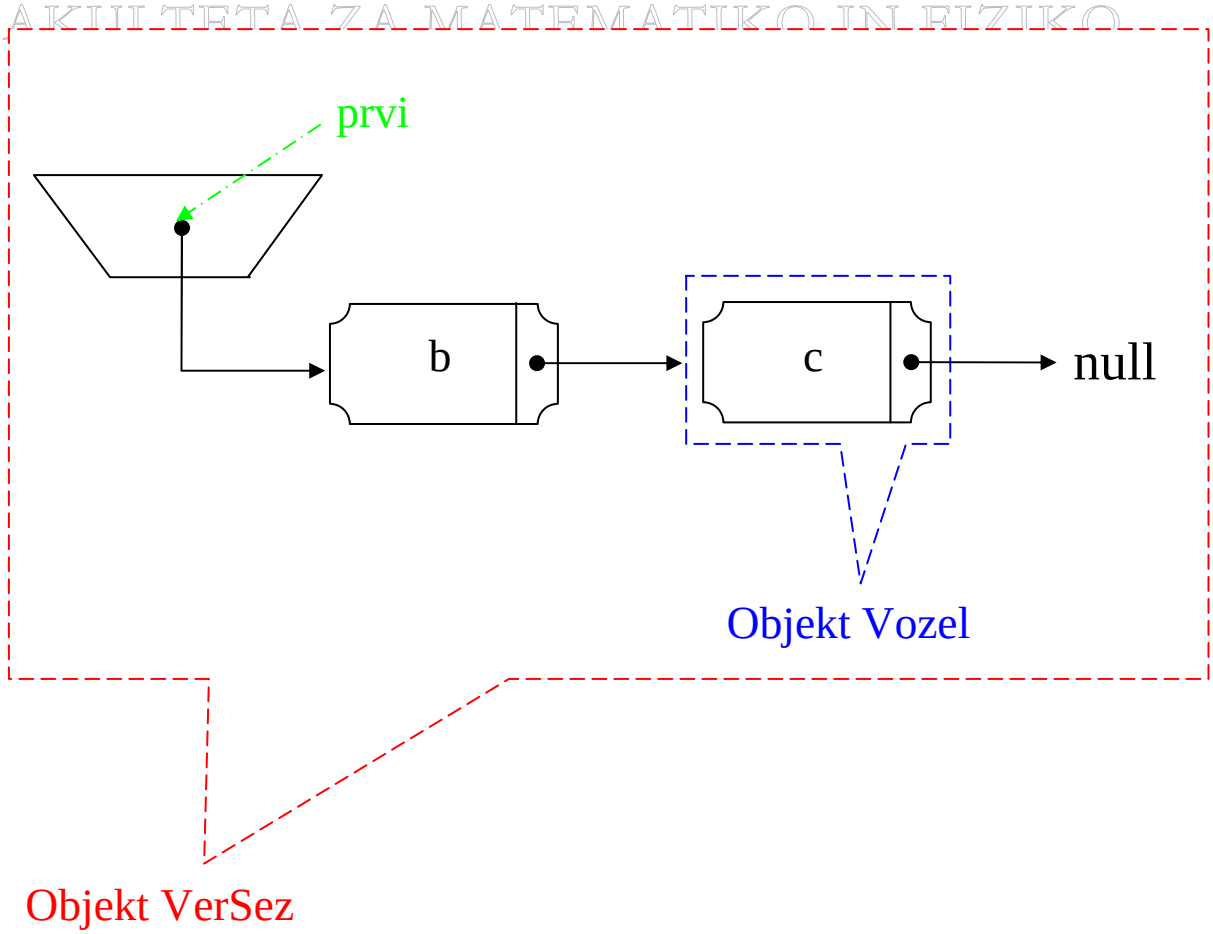
```
    public int vrniPodatek(){ //vrne vrednost vozla
        return podatek;
    }

    public void nastaviNasled(Vozel v){ //vozlu nastavi naslednika
        naslednji = v;
    }

    public Vozel vrniNasled(){ //vrne naslednika vozla x
        return naslednji;
    }
}
```

1.5 ENOJNO POVEZAN VERIŽNI SEZNAM S KAZALCEM NA ZAČETEK

Osnove dela z verižnim seznamom smo obravnavali v razdelku 1.1 in videli, da so osnovne sestavine verižnega seznama vozli. Ogledali smo si tudi že razred `Vozel`, ki nam omogoča delo z vozli. Njegove metode bomo uporabili za delo z verižnim seznamom in pripravili še razred `VerSez`, ki bo omogočal delo z verižnim seznamom.

Oglejmo si razred `VerSez` podrobneje. Sedaj že zelo dobro vemo, da je kazalec na prvi vozel v enojno povezanem verižnem seznamu nujno potreben. Brez njega do prvega elementa ne moremo priti. Preko njega lahko pridemo do poljubnega elementa v seznamu, zato za verižni seznam v najosnovnejši obliki potrebujemo torej le kazalec na prvi element (slika 28). V razredu `VerSez` bo zato naša edina komponenta tipa `Vozel`, referenca na prvi element v verižnem seznamu.

```
private Vozel prvi; //referenca na prvi element verižnega seznama
```



SLIKA 28: Prikaz verižnega seznama

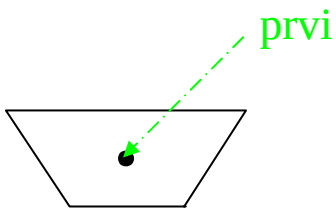
Verižni seznam kot objekt iz razreda `VerSez` sestavlja le del, na sliki 28 narisan kot trapez. Kot smo rekli, bomo v objektu tipa `VerSez` hranili le kazalec na začetek verižnega seznama. Z zeleno barvo smo označili le to, da se ustrezna komponenta v verižnem seznamu imenuje `prvi`. Zeleni prvi ni kazalec (zato smo tudi uporabili črtkano puščico) ampak le opomnik, kako se imenuje ta komponenta!

Ker lahko preko kazalca na prvi element pridemo do vseh ostalih elementov verižnega seznama, si lahko predstavljamo, da je naš objekt sestavljen tako iz kazalca na prvi element (označeno s trapezom), kot tudi z vseh povezanih vozlov. Z objektom tipa `VerSez` bomo torej prišli do reference na prvi vozel in s tem do vseh elementov, ki sestavljajo naš verižni seznam.

1.5.1 Konstruktor

- `public VerSez()`: naredi prazen verižni seznam

Ustvari prazen verižni seznam.



SLIKA 29: Prazen verižni seznam

Na sliki 29 vidimo, kako prikažemo prazen verižni seznam. Ker je prazen, nima nobenega elementa (nobenega vozla). Komponenta `prvi` torej kaže v prazno (ima vrednost `null`).

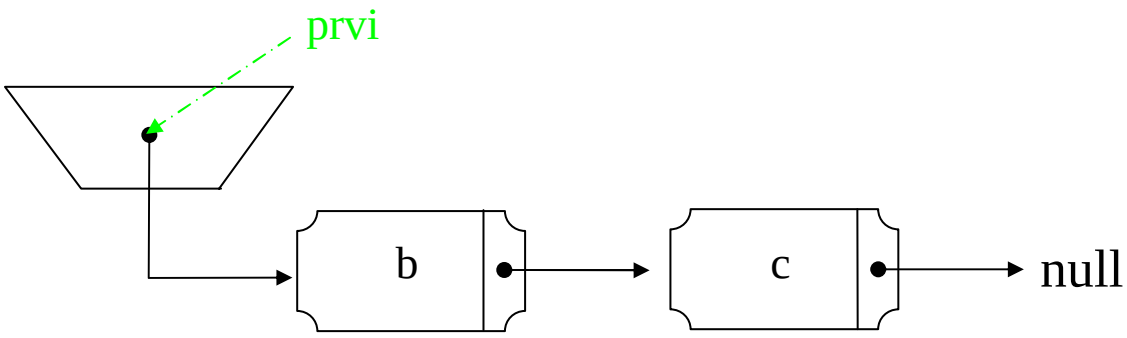
1.5.2 Metode:

- `vstavi_prvega(Vozel v)`: vstavi vozel `v` na začetek verižnega seznama

Vozel `v` že obstaja. Vstavimo ga na začetek obstoječega verižnega seznama. Referenca `prvi` pokaže na vozel `v`.

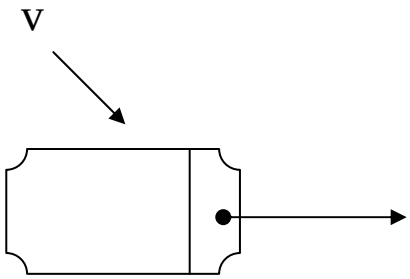
```
public void vstavi_prvega(Vozel v){
   //referenca vozla v pokaže na "starega" prvega v verižnem
   //seznamu
   v.nastaviNasled(prvi);
   prvi = v; //novi prvi element
}
```

Oglejmo si po korakih, kaj metoda naredi na seznamu s slike 30. Ne pozabimo, da imamo z zeleno barvo označeno le to, da se ustrezna komponenta v verižnem seznamu imenuje `prvi` in da zelena črtkana črta, označena s `prvi`, ni kazalec (zato smo tudi uporabili črtkano puščico!).



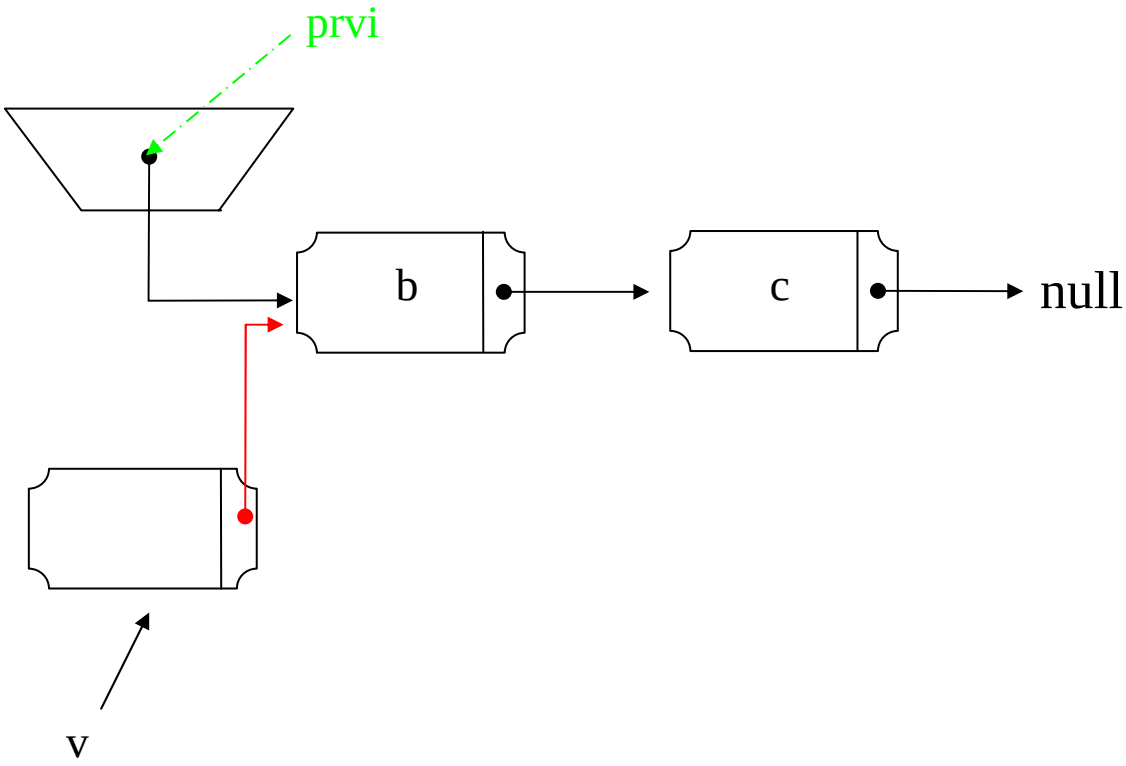
SLIKA 30: Verižni seznam za prikaz delovanja metod

Imamo tudi vozel `v`:



SLIKA 31: Vozel `v`

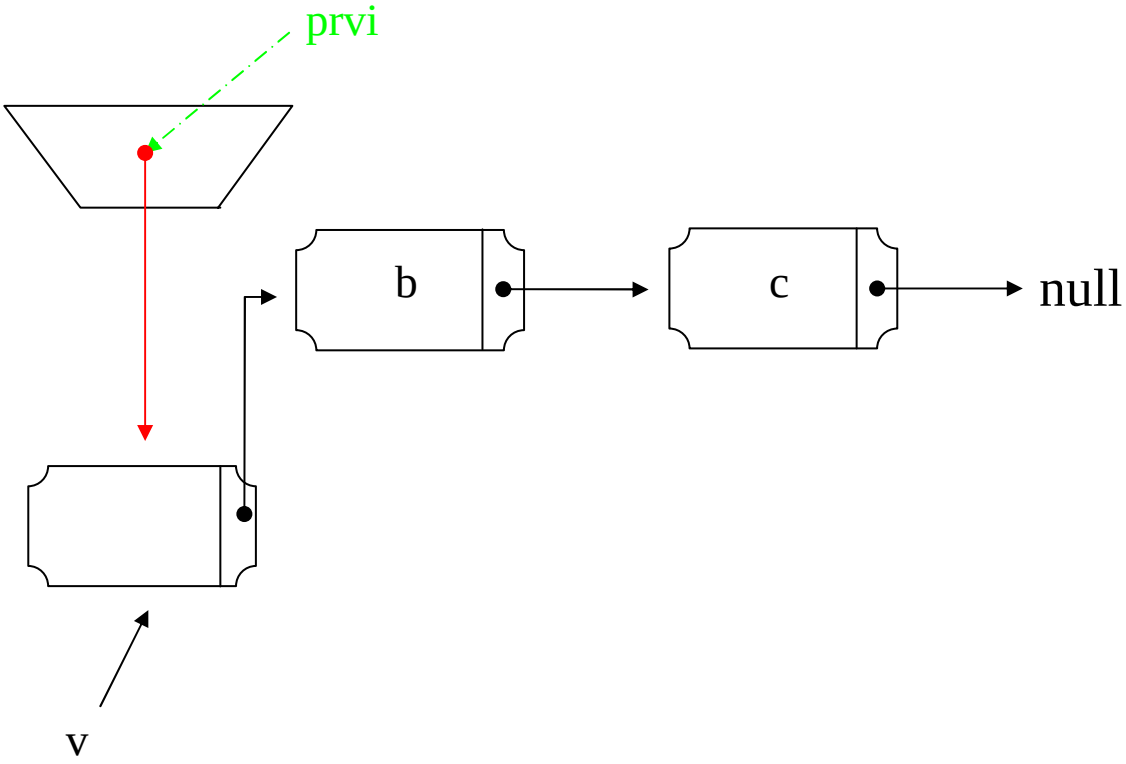
Najprej se izvede metoda `v.nastaviNasled(prvi)`. Vozlu `v` nastavimo kot naslednika tisti vozel, na katerega kaže kazalec `prvi`. V našem primeru `prvi` kaže na vozel z vrednostjo b. Začetek verižnega seznama še vedno kaže na vozel z vrednostjo b.



SLIKA 32: Vozlu `v` nastavimo naslednika

Vozel `v` sedaj kaže na vozel z vrednostjo b. Tja prav tako kaže tudi začetek verižnega seznama. Sedaj moramo še začetek verižnega seznama preusmeriti na pravi vozel. Želimo, da je novi začetek verižnega seznama vozel `v`. To dosežemo z ukazom:

```
prvi = v;
```



SLIKA 33: Verižni seznam po klicu metode `vstavi_prvega(Vozel v)`

Verižnemu seznamu s slike 30 smo na začetek dodali vozel `v`.

- `vstavi_prvega(int x)`: vstavi vozel na začetek verižnega seznama

Metoda naredi nov objekt `Vozel`, v katerem se hrani podatek `x`. Nov vozel postavi na začetek verižnega seznama.

```
public void vstavi_prvega(int x){
   //naredimo nov objekt in ga postavimo na začetek
   Vozel v = new Vozel(x, prvi);
   //referenca vozla v pokaže na prvega v v.s.
   prvi = v; //novi prvi element
}
```

Tudi tu si oglejmo, kako deluje metoda, če bi jo izvedli nad verižnim seznamom s slike 30. Metoda deluje podobno kot prejšnja metoda. Razlika je le-ta, da mora ta metoda objekt `Vozel`, v katerem bomo hranili vrednost x, še ustvariti. V prejšnji metodi smo objekt `Vozel` že imeli podan kot argument metode.

Prva ukazna vrstica v metodi je:

```
Vozel v = new Vozel(x, prvi);
```

Tu smo uporabili konstruktor iz razreda `Vozel`, ki naredi nov objekt s podatkom x in katerega kazalec pokaže na prejšnji prvi element v verižnem seznamu. Če pogledamo sliko 30, vidimo, da je prvi element vozel z vrednostjo b. Na ta vozel bo pokazal kazalec novega vozla `v`. Dobimo stanje kot je na sliki 32.

Sedaj moramo ukrepati enako kot pri prejšnji metodi - vozel `v` nastaviti za prvi vozel v verižnem seznamu. Izvedemo prireditveni stavek:

```
prvi = v;
```

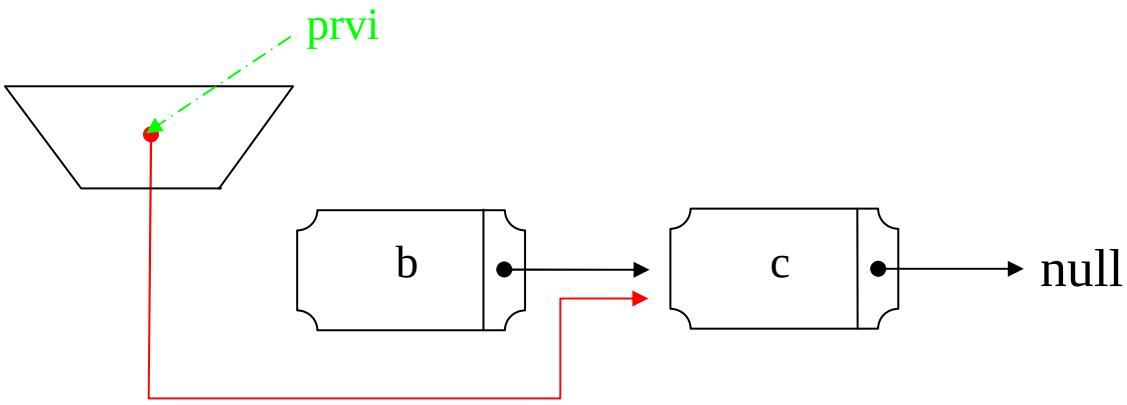
in dobimo verižni seznam, kot ga prikazuje slika 33.

- `odstrani_prvega()`: odstrani prvi element iz verižnega seznama

Predpostavimo, da verižni seznam ni prazen. Elementa fizično ne odstranimo, le kazalec `prvi` preusmerimo na drugi element (na naslednika prvega vozla) v verižnem seznamu.

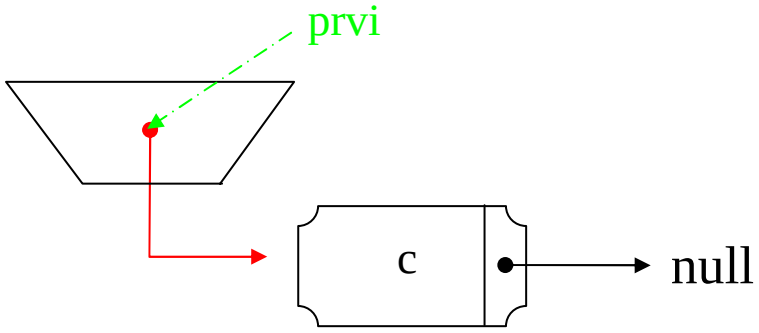
```
public void odstrani_prvega(){
   //predpostavka: seznam NI prazen
   //prvi pokaže tja, kamor kaže naslednik
   prvi = prvi.vrniNasled(); //novi prvi element
}
```

Naj bo verižni seznam s slike 30 tisti, na katerem bomo izvedli metodo. Kazalec `prvi` kaže na vozel z vrednostjo b. Ta vozel želimo odstraniti iz verižnega seznama. S kazalcem `prvi` bomo pokazali na njegovega naslednika. Začetek našega verižnega seznama bo torej pokazal na vozel z vrednostjo c.



SLIKA 34: Verižni seznam med klicem metode `odstrani_prvega()`

Vozel z vrednostjo b še obstaja, vendar ni v našem verižnem seznamu. Začetek našega verižnega seznama kaže na vozel z vrednostjo c. Sedaj ima verižni seznam le en element. Narišimo ustrezno sliko. Ne pozabimo pa, da je vozel s podatkom c ostal na istem mestu v pomnilniku kot prej. Spremenil se je le naslov (kazalec) `prvi`:



SLIKA 35: Verižni seznam po klicu metode `odstrani_prvega()`

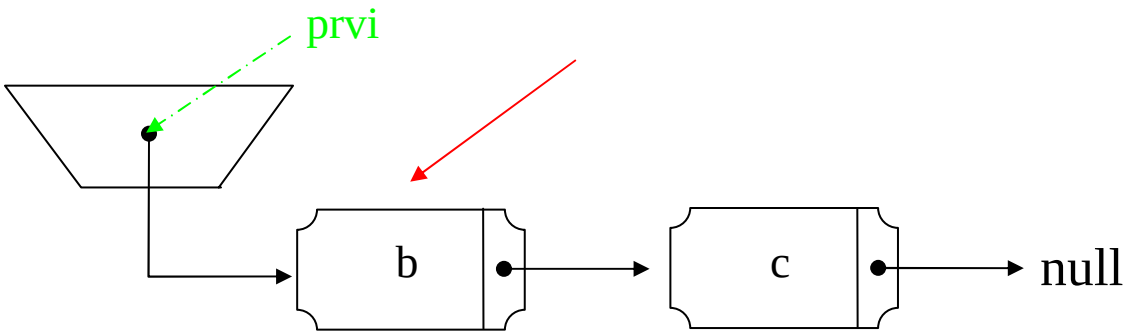
Če je verižni seznam prazen, potem ob klicu `prvi.vrniNasled()` pride do napake, saj vozel, na katerem bi izvedli metodo `vrniNasled()` ne obstaja (prvi ima vrednost `null`).

- `prvi()`: vrne prvi vozel

Metoda vrne referenco na objekt, na katerega kaže referenca `prvi`.

```
public Vozel prvi(){
   //vrne prvi vozel
   return prvi;
}
```

Če pokličemo metodo nad verižnim seznamom s slike 30, nam vrne referenco na vozel z vrednostjo b. Prikažimo na sliki, kaj nam metoda vrne:



SLIKA 36: Rdeča puščica je tisto, kar vrne metoda `prvi()`

Z rdečo puščico je označen naslov, ki ga metoda vrne. Rečemo, da nam vrne prvi vozel.

Če je verižni seznam prazen, metoda vrne vrednost `null`. V praznem verižnem seznamu ima namreč komponenta `prvi` vrednost `null`.

- `vrednost_prvega()`: vrnemo podatek iz objekta, na katerega kaže referenca `prvi`

Spet predpostavimo, da verižni seznam ni prazen. Metoda vrne podatek iz objekta, na katerega kaže referenca `prvi`.

```
public int vrednost_prvega(){
   //predpostavka: seznam NI prazen
   //vrnemo podatek iz objekta, na katerega kaže prvi
   return prvi.vrniPodatek(); //novi prvi element
}
```

Metodo pokličimo nad objektom s slike 30. Metoda vrne vrednost b.

Če je verižni seznam prazen, potem ob klicu `prvi.vrniPodatek()` pride do napake. Ker ima `prvi` vrednost `null`, ne obstaja vozel, na katerem bi izvedli metodo `vrniPodatek()`.

- `prazen()`: metoda preveri, če je verižni seznam prazen

Če je seznam prazen, metoda vrne `true`, sicer `false`. V kodi le pogledamo, če kazalec `prvi` kaže v prazno.

```
public boolean prazen(){
   return (prvi == null);
}
```

Pri pisanju kode metod vidimo, da je poglavitna točka uporabe vseh metod kazalec `prvi`. Zato moramo zelo paziti nanj. Vedno mora kazati na prvi element v verižnem seznamu. Če z njim pokažemo drugam, metode ne bodo pravilno delovale.

1.5.3 Razred VerSez

Zložimo opisane konstruktorje in metode v razred VerSez:

```
public class VerSez {
   private Vozel prvi; //referenca na prvi element seznama

   //konstruktorji

   public VerSez(){
      prvi = null; //prazen v. sez.
   }
```

```
    //metode

        public void vstavi_prvega(Vozel v){
           //refrenca vozla v pokaže na prvega v v.s.
           //referenca prvi pokaže na vozel v
           v.nastaviNasled(prvi);
           prvi = v; //novi prvi element
        }

        public void vstavi_prvega(int x){
           //naredimo nov objekt in ga postavimo na začetek
           Vozel v = new Vozel(x, prvi);
           //referenca vozla v pokaže na prvega v v.s.
           prvi = v; //novi prvi element
        }

        public void odstrani_prvega(){
           //predpostavka: seznam NI prazen
           //prvi pokaže tja, kamor kaže naslednik
           prvi = prvi.vrniNasled(); //novi prvi element
        }

        public Vozel prvi(){
           //vrne prvi vozel
           return prvi;
        }

        public int vrednost_prvega(){
           //predpostavka: seznam NI prazen
           //vrnemo podatek iz objekta, na katerega kaže prvi
           return prvi.vrniPodatek();
        }

        public boolean prazen(){
           return (prvi == null);
        }
    }
```

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

1.6 OSNOVNE OPERACIJE

Nad verižnim seznamom in vozli torej lahko izvajamo različne operacije. Zavedati se moramo, kdaj uporabljamo verižni seznam kot objekt, kdaj pa na verižni seznam gledamo izključno kot na zbirko s pomočjo referenc povezanih vozlov. Poleg osnovnih operacij, ki smo jih navedli pri obeh razredih (`Vozel` in `VerSez`), nad verižnimi seznami (kot objekti ali kot zbirkami vozlov) izvajamo nekatere tipične operacije. V praksi jih uporabimo v skoraj vsakem programu, zato si jih je dobro zapomniti. Najbolj pomembno pa je, da jih razumemo.

Te operacije so:

- Vstavljanje

Element vstavimo na začetek, konec ali pa nekje znotraj verižnega seznama.

- Brisanje

Zbrišemo element znotraj, na začetku ali koncu verižnega seznama.

- Iskanje

Poiščemo, če v verižnem seznamu hranimo določeno vrednost.

- Izpis

Izpišemo podatke, ki jih hranimo v verižnem seznamu.

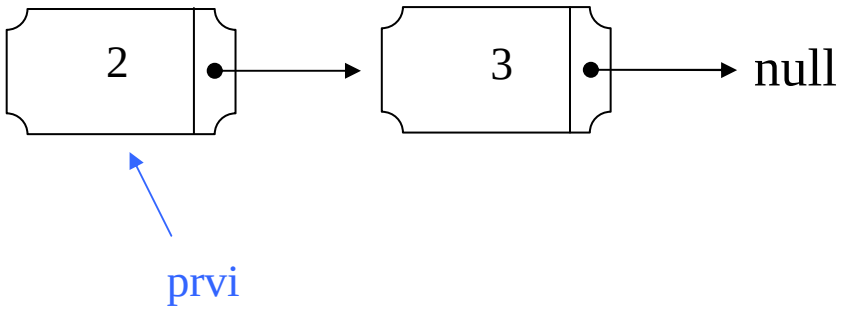
1.6.1 Vstavljanje

1.6.1.1 Vstavljanje na začetek

Na začetek zaporedja vozlov bomo vstavili nov vozel. Naredili bomo dva podobna primera. Pri prvem bomo predpostavili, da vozel, ki ga vstavljamo, že imamo. Vstavili bomo torej znan vozel. Pri drugem primeru pa bomo vstavili vozel, ki ga bomo naredili znotraj metode. Oba zgornja primera bomo naredili z uporabo razreda `Vozel` (na verižni seznam gledamo kot na zaporedje vozlov in poznamo kazalec na prvi vozel tega zaporedja) in z uporabo razreda `VerSez` (dan je verižni seznam kot objekt razreda `VerSez`).

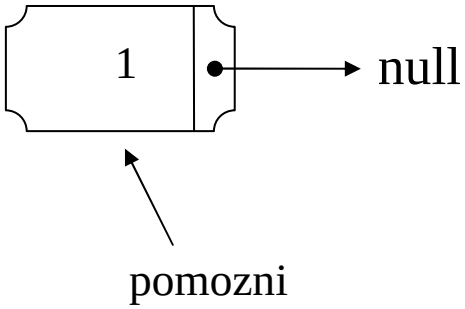
- **S pomočjo razreda Vozel**

Denimo, da imamo dano naslednje zaporedje vozlov:



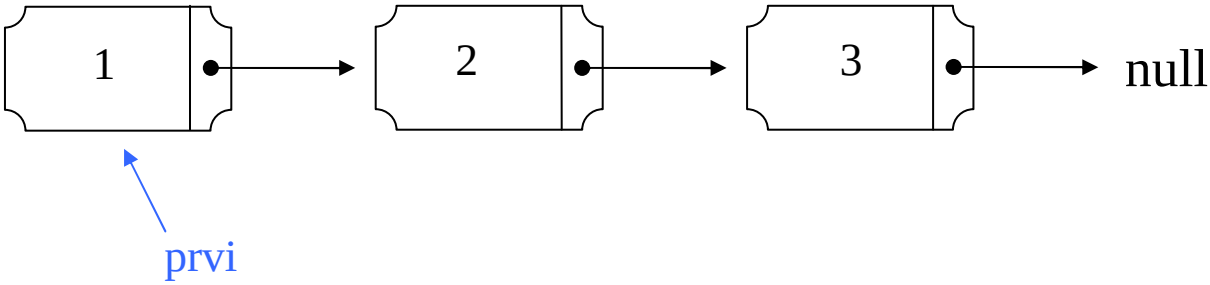
SLIKA 37: Primer zaporedja vozlov

V tem zaporedju vozlov nam na prvi vozel kaže kazalec `prvi`. Prikazali smo ga z modro barvo. Tako bomo lažje videli kam kaže in bomo nanj bolj pozorni. V zgornje zaporedje bomo na začetek vstavili vozel, na katerega kaže kazalec `pomozni` (spodnja slika).



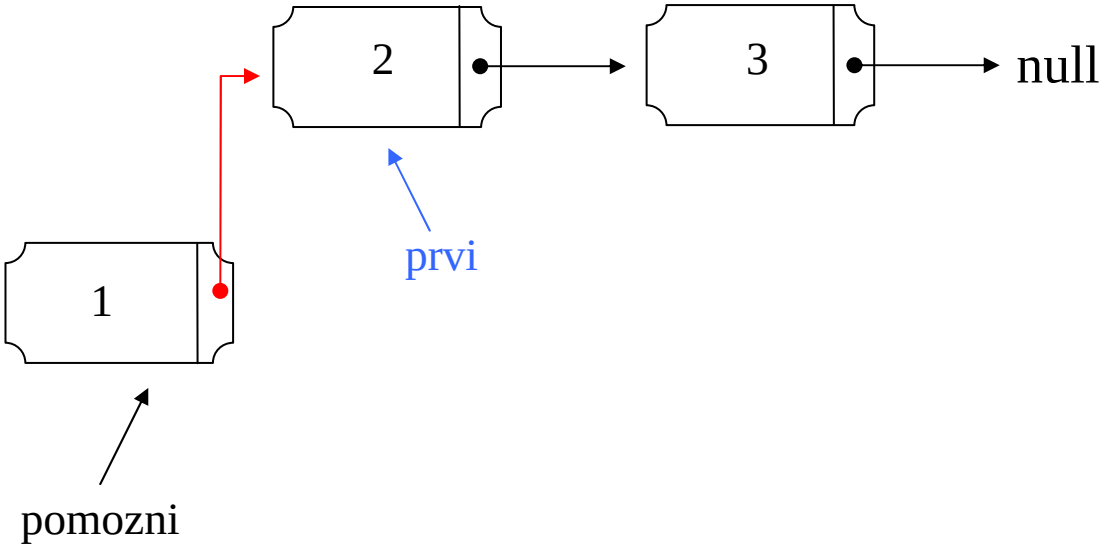
SLIKA 38: Vozel `pomozni`

Želimo, da po vstavljanju dobimo naslednje stanje:



SLIKA 39: Vstavljanje na začetek

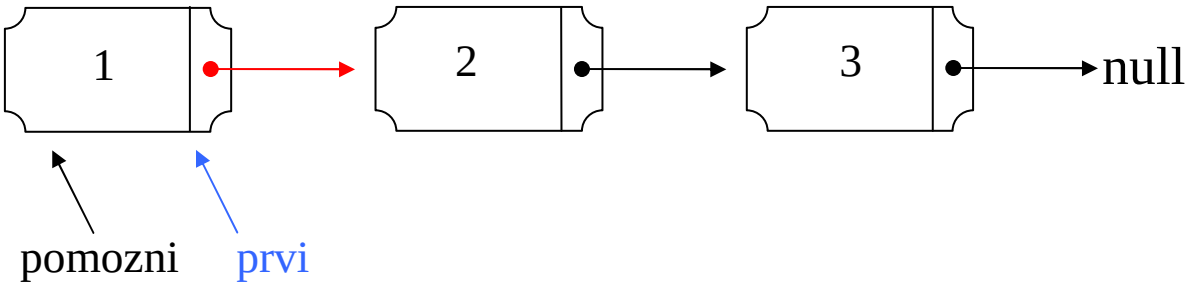
Kako vozel `pomozni` vstavimo na začetek zaporedja vozlov? Postopek vstavljanja v zaporedje, na katerega začetek kaže kazalec `prvi`, prikažimo slikovno. Najprej bomo vozlu `pomozni` nastavili naslednika. To bo vozel `prvi` iz zaporedja vozlov s slike 37. To bomo storili z metodo `nastaviNasled()` s klicem `pomozni.nastaviNasled(prvi)`. Dobimo:



SLIKA 40: Vozlu `pomozni` smo nastavili naslednika

Na zgornji sliki je z rdečo prikazana sprememba kazalca vozla `pomozni`. Sedaj kaže na vozel, na katerega kaže tudi `prvi`. Pri tem nismo naredili nobenega novega objekta (vozla) in obstoječih nismo prestavljali. Spremenili smo le en kazalec (referenco, oziroma naslov).

S kazalcem `prvi` moramo sedaj pokazati na novi prvi vozel v zaporedju. To je vozel `pomozni` (ali natančneje: vozel, na katerega kaže kazalec `pomozni`).



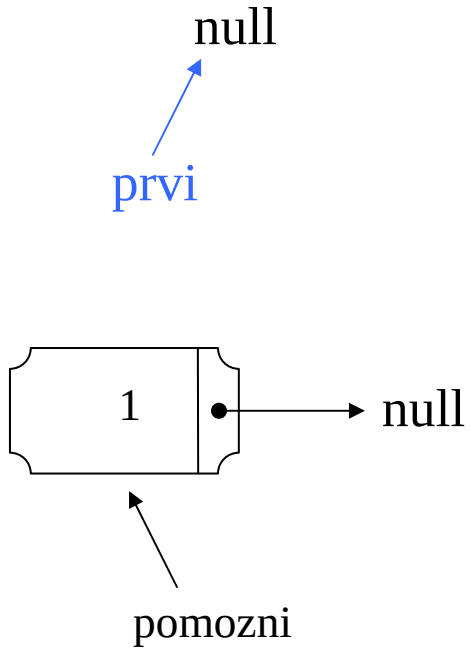
SLIKA 41: Novo zaporedje vozlov

Sedaj pa razmislimo, kako bi zgornji postopek napisali kot metodo. Jasno je, da potrebujemo kazalec `pomozni` (kaže na vozel, ki ga bomo vstavili v zaporedje) in kazalec `prvi` (kaže na prvi vozel v zaporedju vozlov). Dobili ju bomo kot argumenta naše metode. V metodi moramo vozel `pomozni` nastaviti za prvi vozel v danem zaporedju. To storimo tako, da najprej vozlu `pomozni` nastavimo naslednika z metodo `nastaviNasled()`. Nato bomo s kazalcem `prvi` pokazali na vozel `pomozni`, oz. na nov prvi vozel v zaporedju. Metoda bo vrnila rezultat tipa `Vozel` in sicer vozel `prvi`. Ob tem se še zadnjič spomnimo na to, da takrat, ko govorimo na primer o vozlu `prvi`, mislimo bodisi na samo spremenljivko `prvi`, ki vsebuje naslov (kazalec, referenco) nekega objekta tipa `Vozel`, bodisi mislimo na tisti objekt, na katerega kažemo s kazalcem `prvi`. Metoda seveda vrne naslov objekta tipa `Vozel`, čeprav govorimo, da metoda vrača `Vozel`.

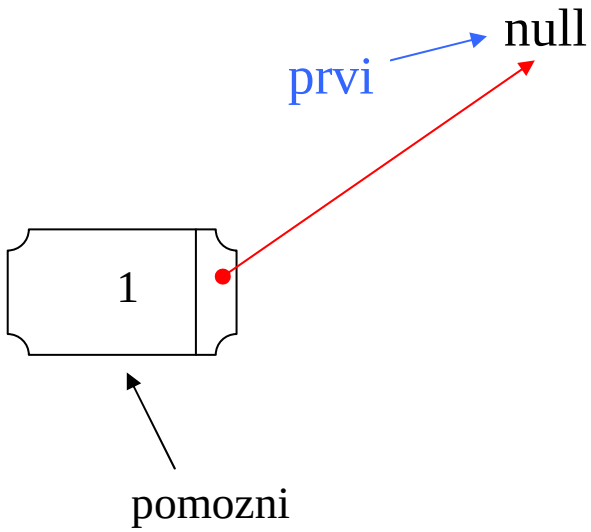
Zapišimo še metodo:

```
public static Vozel dodaj_prvega1(Vozel pomozni, Vozel prvi){
   //vozel vstavimo na začetek
   pomozni.nastaviNasled(prvi);
   prvi = pomozni;//pomozni je prvi element zaporedja
   return prvi;
}
```

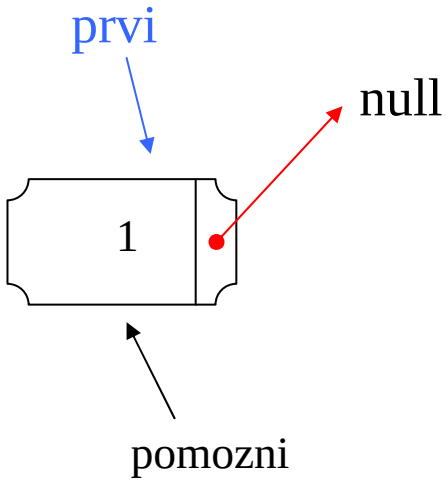
Tako. Metoda je napisana. Pa deluje prav? Kratek premislek pokaže, da bi bilo dobro preveriti, kako je z delovanjem te metode takrat, ko je prvotno zaporedje prazno (ko prvi kaže v prazno). Sledimo metodi in si delovanje v tem primeru zopet narišimo:



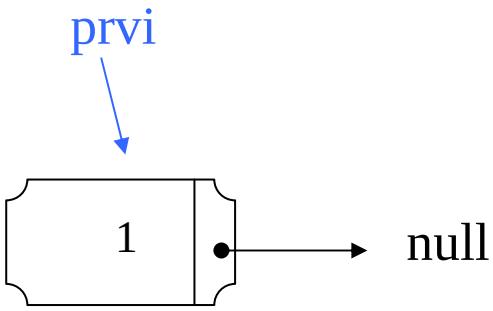
SLIKA 42: Pred vstavljanjem vozla `pomozni` v prazno zaporedje vozlov



SLIKA 43: Po ukazu `pomozni.nastaviNasled(prvi);`



SLIKA 44: Po ukazu `prvi = pomozni;`



SLIKA 45: Končno stanje

Sestavili smo torej metodo, s katero že obstoječi vozel vstavimo v zaporedje vozlov. Sedaj si oglejmo še primer, ko vozla še nimamo, imamo pa dan podatek `x`, ki ga želimo hraniti v vozlu, ki bo nov prvi vozel zaporedja vozlov.

Znotraj metode bomo poklicali konstruktor iz razreda `Vozel`, ki naredi nov vozel s podatkom `x`. Nadaljni del metode bo zelo podoben zgornjemu. Novemu vozlu, ki ga bomo imenovali `pomozni`, bomo nastavili naslednika. Njegov naslednik bo prvi vozel v zaporedju vozlov. Sledi še premik kazalca `prvi` na vozel `pomozni`, oz. na nov prvi vozel v verižnem seznamu. Metoda bo tudi v tem primeru vrnila objekt `Vozel`, oz. prvi vozel.

Metoda:

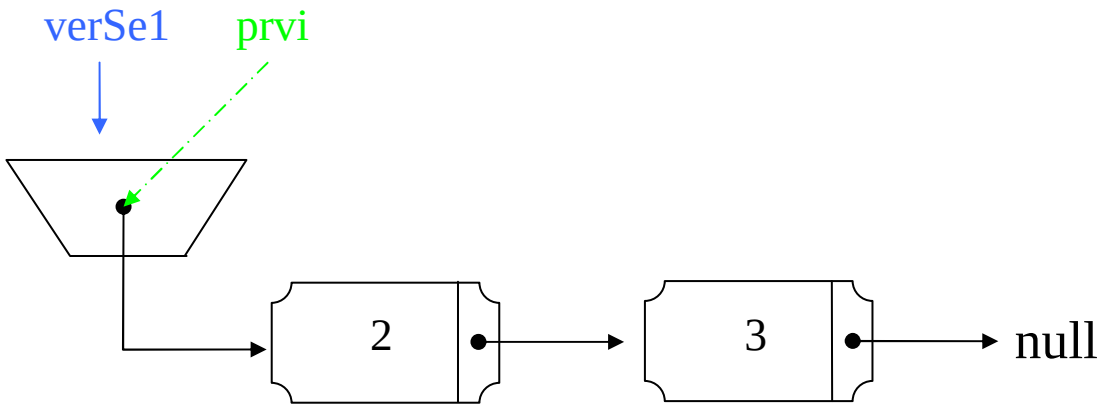
```
public static Vozel dodaj_prvega2(int x, Vozel prvi){
   Vozel pomozni = new Vozel(x);//nov vozel
   pomozni.nastaviNasled(prvi);//pomozni nastavimo kot prvi vozel
   prvi = pomozni; //nov prvi vozel
   return prvi;
}
```

Lahko pa uporabimo tudi konstruktor, ki novemu vozlu že nastavi naslednika in dobimo:

```
public static Vozel dodaj_prvega2a(int x, Vozel prvi){
   Vozel pomozni = new Vozel(x, prvi);//nov vozel, ki že kaže prav
   prvi = pomozni; //nov prvi vozel
   return prvi;
}
```

- **S pomočjo razreda VerSez**

Denimo, da imamo dan objekt `VerSez` iz razreda `VerSez`. V verižni seznam, ki ga predstavlja, bomo na začetek vstavili vozel `pomozni`.

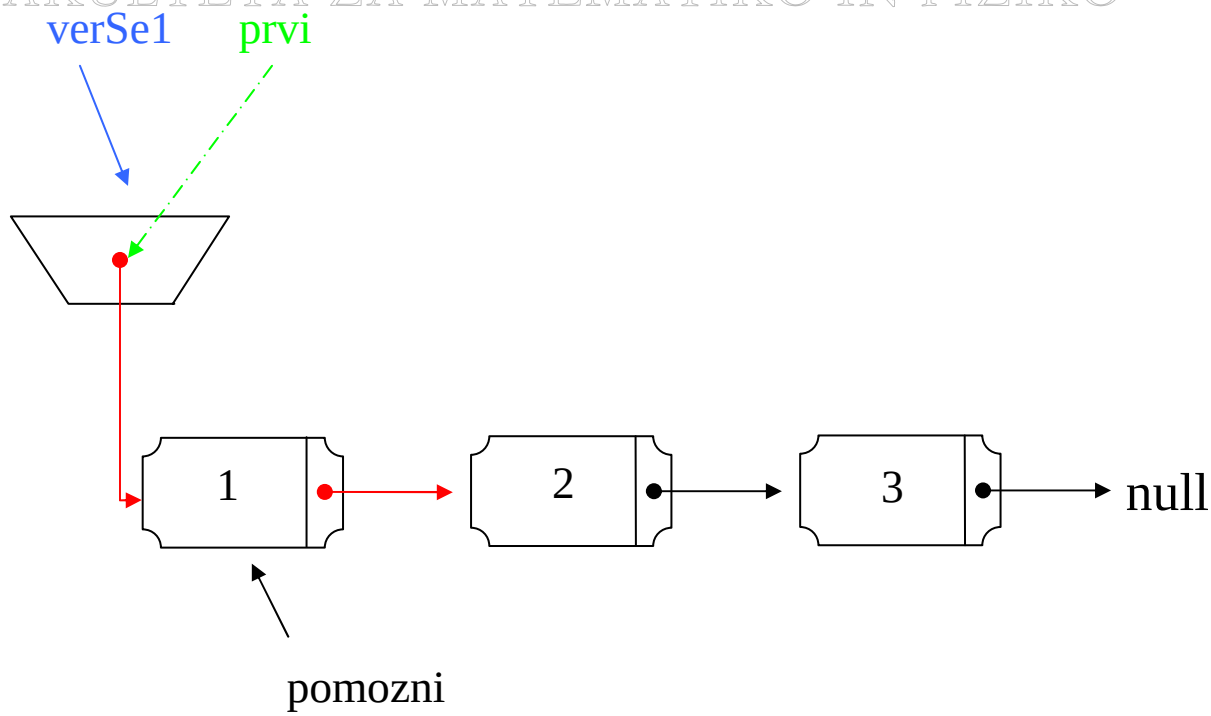


SLIKA 46: Primer verižnega seznama

Morda ne bo odveč znova opozoriti, da črtkana zelena puščica ni kazalec (zato smo tudi uporabili črtkano puščico), ampak le vizualni pripomoček, da se ustrezna komponenta v verižnem seznamu imenuje `prvi`.

V razredu `VerSez` že obstaja metoda, ki vozel vstavi na prvo mesto v verižni seznam. Metoda se imenuje `vstavi_prvega(Vozel v)`. Ta sama poskrbi, da se ustrezno popravijo reference (kazalci). Metodo bomo poklicali nad verižnim seznamom s slike 46. Argumenta metoda bosta kazalec `pomozni` in kazalec `verSe1`. Prvi bo kazalec na vozel, ki ga bomo vstavili v verižni seznam in bo tipa `Vozel` (slika 38). Drugi parameter pa bo kazalec na objekt tipa `VerSez`. Metoda bo vrnila objekt tipa `VerSez`. Ustrezna metoda bo:

```
public static VerSez dodaj_prvega3(Vozel pomozni, VerSez verSe1){
   //vozel pomozni vstavimo na začetek verižnega seznama
   verSe1.vstavi_prvega(pomozni);//pomozni vstavimo na začetek
   return verSe1;
}
```



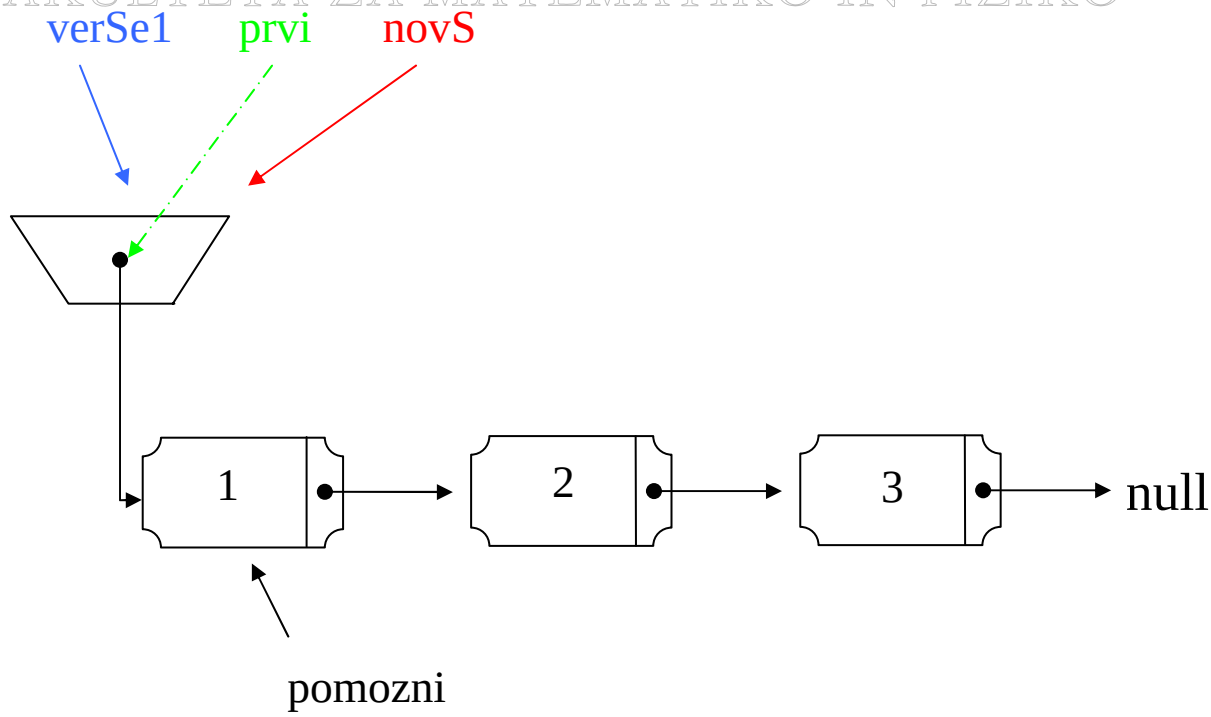
SLIKA 47: Verižni seznam po klicu metode `dodaj_prvega3(Vozel pomozni, VerSez verSe1)`

Na zgornji sliki sta prikazani obe referenci, ki se znotraj metode spremenita, oz. ki ju spremeni klic metode: `verSe1.vstavi_prvega(pomozni)`.

Metoda je torej enaka objektni metodi `vstavi_prvega` iz razreda `VerSez`. Razlika je v tem, da dobimo še dodatni kazalec na spremenjeni verižni seznam, ki nam ga da stavek `return`. Če torej na verižnem seznamu s slike 46 izvedemo metodo s prireditvenim stavkom

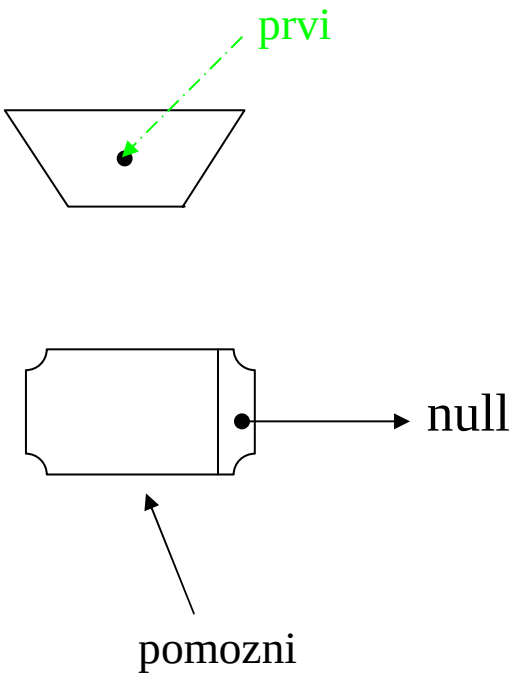
```
VerSez novS = dodaj_prvega3(pomozni, verSe1)
```

dobimo

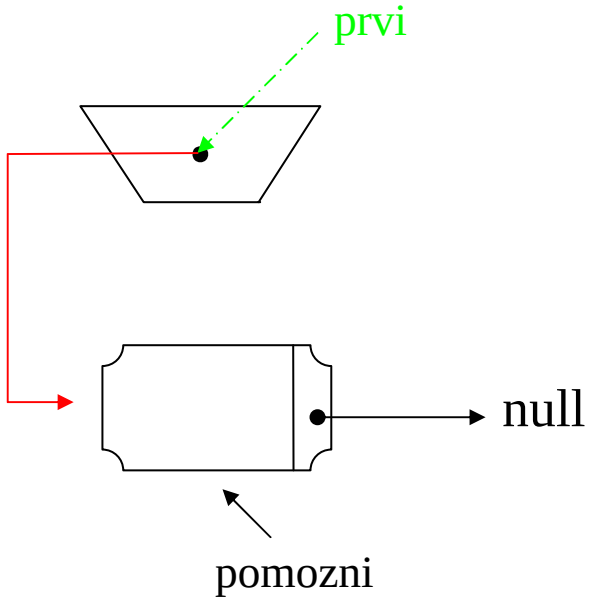


SLIKA 48: Stanje po stavku `VerSez novS = dodaj_prvega(pomozni, verSe1);`

Metoda je napisana. Oglejmo si, če deluje prav, kadar jo pokličemo nad praznim verižnim seznamom. Sledimo metodi in si delovanje metode zopet narišimo:



SLIKA 49: Pred vstavljanjem vozla `pomozni` v prazen verižni seznam



SLIKA 50: Stanje po klicu metode `dodaj_prvega3(Vozel pomozni, VerSez verSe1)`

Na sliki 50 je z rdečo puščico prikazana sprememba v verižnem seznamu. Kazalec `prvi` v vreižnem seznamu `verSe1` sedaj kaže na vozel `pomozni`, ki je novi prvi vozel v verižnem seznamu.

Naredimo še metodo, kjer bomo znotraj metode naredili vozel, ki ga bomo vstavili v verižni seznam. Metoda se od zgornje ne bo veliko razlikovala. Poleg argumenta tipa `VerSez`, ki bo verižni seznam, bomo podali še vrednost podatka, ki ga vstavljamo v verižni seznam. Znotraj metode bomo skonstruirali nov vozel in ga vstavili v verižni seznam. Metoda bo prav tako kot zgornja vrnila kazalec, ki kaže na prvi element v verižnem seznamu in je seveda objekt tipa `VerSez`.

```
public static VerSez dodaj_prvega4(int x, VerSez verSe1){
   verSe1.vstavi_prvega(x);
   return verSe1;//kazalec na verižni seznam
}
```

1.6.1.2 Vstavljanje na sredino

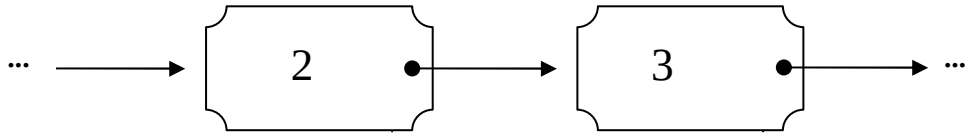
Ogledali si bomo, kako element vstavimo na sredino verižnega seznama. Podobno kot prej, bomo tudi sedaj stvar izpeljali v dveh osnovnih oblikah – če imamo verižni seznam podan kot neko zaporedje vozlov (poznamo kazalec na prvi vozel v zaporedju objektov tipa `Vozel`) in če imamo verižni seznam podan kot objekt tipa `VerSez`.

- **S pomočjo razreda Vozel**

Najprej si oglejmo, kako vstavimo vozel v verižni seznam, podan kot neko zaporedje vozlov, kjer poznamo kazalec na prvi vozel v zaporedju objektov tipa `Vozel`.

Vozel bomo vstavili znotraj zaporedja vozlov. Vedeti moramo, kam želimo vozel vstaviti. Ponavadi vozel vstavimo na določeno mesto na podlagi nekega pogoja. Nas trenutno pogoj ne zanima, temveč nas zanima sam postopek vstavljanja na sredino.

Imeli bomo zaporedje vozlov s kazalcem `prvi`, ki bo kazal na prvi element v zaporedju vozlov. Slikovno bomo prikazali le tista vozla iz zaporedja, med katera bomo vstavili vozel `pomozni` (slika 51).

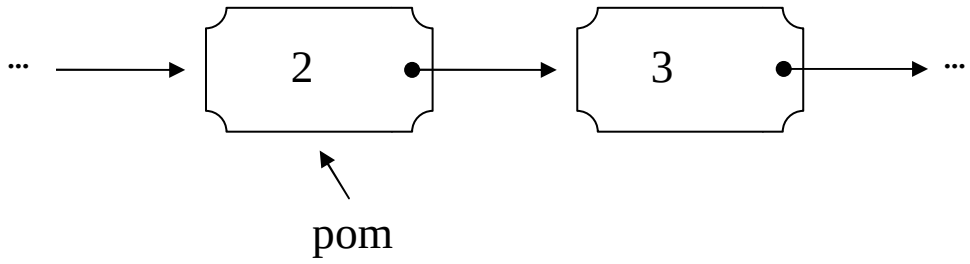


SLIKA 51: Del zaporedja vozlov, kamor bomo vstavili vozel `pomozni`

Zgornja vozla sta del naše verige in nanju še ne kaže noben kazalec. Če želimo vozel vstaviti med zgornja vozla, bomo potrebovali pomožni kazalec. Kazati mora na vozel z vrednostjo 2 oz. v splošnem na vozel, za katerim bomo vstavili nov vozel (torej tistega, na katerega v tem trenutku kaže `pom`). To bomo dosegli na sledeč način:

- naredili bomo pomožni kazalec tipa `Vozel` z imenom `pom`
- z njim bomo pokazali na vozel `prvi`
- s klicem metode `vrniNasled()` in ustrezno zanko `while` bomo kazalec `pom` premaknili na ustrezni vozel, torej na vozel, za katerega bomo vstavili vozel `pomozni`.

Poskrbeli bomo torej, da bo stanje tako:

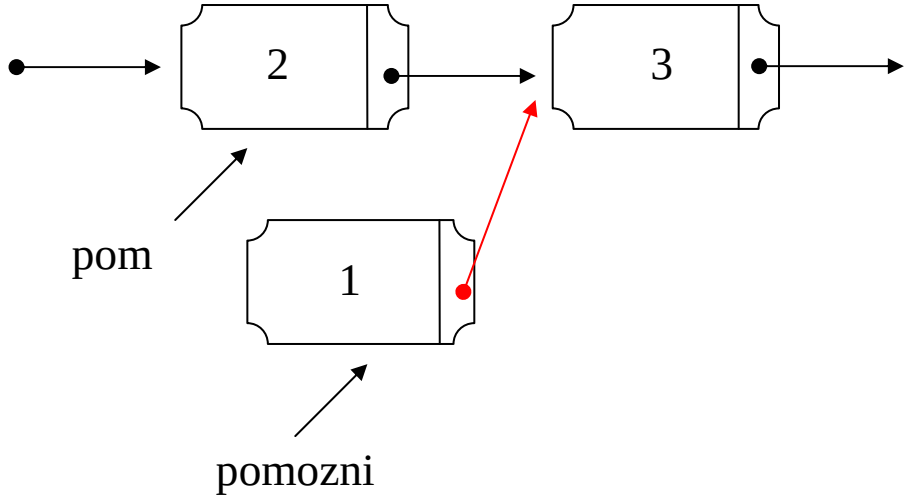


SLIKA 52: Del zaporedja vozlov s slike 51 z ustreznim kazalcem

Postopek vstavljanja vozla bomo najprej prikazali grafično. Imamo torej zaporedje vozlov s slike 52 in vozel, na katerega kaže kazalec `pomozni`. Najprej bomo vozlu `pomozni` z metodo `nastaviNasled(Vozel v)` nastavili za naslednika naslednika vozla, na katerega kaže `pom` (v našem primeru torej vozel z vrednostjo 3). Ustrezni klic bo

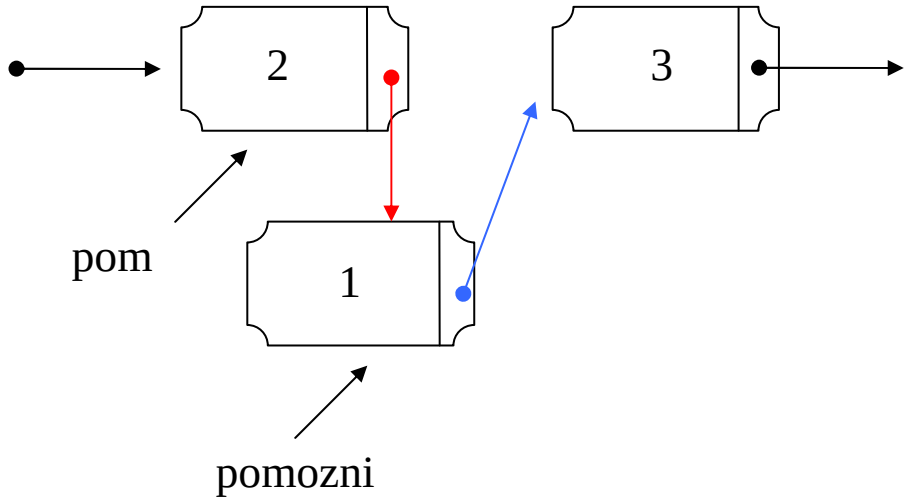
```
pomozni.nastaviNasled(pom.vrniNasled());
```

Po izvedbi tega klica je ustrezna slika:



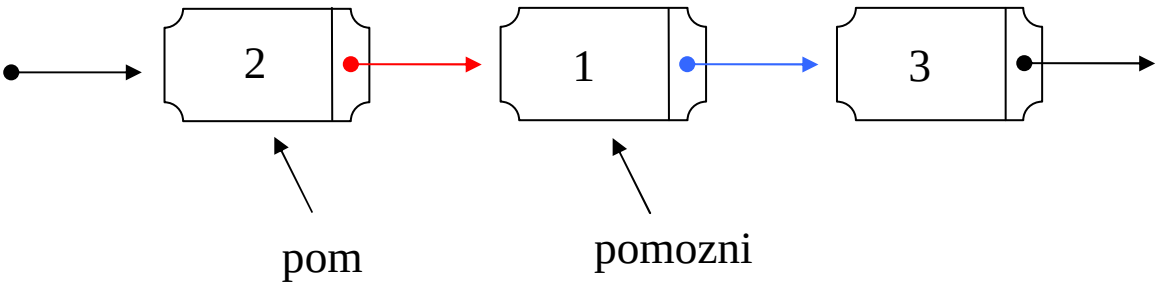
SLIKA 53: Vozlu `pomozni` smo nastavili naslednika na vozel z vrednostjo 3

Na zgornji sliki je z rdečo puščico prikazan kazalec, ki smo ga spremenili. Vidimo, da sedaj vozel `pomozni` kaže na vozel z vrednostjo 3. Sedaj moramo še vozlu `pom` nastaviti ustreznega naslednika, se pravi vozel `pomozni`. To naredimo z ukazom `pom.nastaviNasled(pomozni)`. Slika je sledeča



SLIKA 54: Vozel `pomozni` vstavljen v zaporedje vozlov I

Oziroma, če sliko narišemo "lepše" (in se pri tem seveda zavedamo, da so vozli ostali na tistih mestih v pomnilniku, kjer so bili):



SLIKA 55: Vozel `pomozni` vstavljen v zaporedje vozlov II

Na zgornjih dveh slikah (gre za isto stanje) je z modro barvo prikazana povezava spremenjena v prejšnjem koraku, z rdečo pa povezava, ki smo jo spremenili nazadnje. Sedaj je vozel `pomozni` vstavljen v zaporedje vozlov.

Metodo bomo napisali le za tisti del postopka, ki smo ga prikazali grafično. Kazalec `pom`, ki kaže na vozel za katerim bomo vstavili vozel `pomozni`, bomo podali kot argument metode. Metoda bo kot argument dobila torej dva kazalca. Prvi bo kazal na vozel, ki ga bomo v zaporedje vstavili, drugi pa na vozel, za katerim vstavljamo vozel. Metoda ne bo vrnila ničesar.

Postopek vstavljanja vozla je torej sledeč: poznamo kazalec `pom`, ki kaže na vozel, za katerim bomo vstavili vozel `pomozni`. Najprej bomo vozlu `pomozni` kot naslednika nastavili vozel, ki je naslednik vozla `pom`, potem pa naslednika vozla pom spremenili v vozel `pomozni`.

```
public static void dodaj_na_sredino(Vozel pomozni, Vozel pom){
   //vozel pomozni vstavimo v zaporedje
   pomozni.nastaviNasled(pom.vrniNasled());
   pom.nastaviNasled(pomozni);
}
```

- **S pomočjo razreda VerSez**

Razmislimo, kako delujejo metode v razredu `VerSez()`. Vse metode so vezane na kazalec `prvi`. Ta vedno kaže na prvi element v verižnem seznamu. Zato načeloma nimamo možnosti, da bi vstavili vozel v sredino verižnega seznama. Kot smo videli prej, bi vnaprej potrebovali kazalec na vozel, za katerim vstavljamo novo vozlišče. Do tega pa »od zunaj« ne moremo priti. Torej bo smiselno ustrezno metodo napisati le, če bomo podatek o tem, kam moramo vstaviti nov vozel, podali drugače. Na primer, da bomo napisali metodo, ki bo vstavila vozel v verižni seznam za element, ki ima določeno vrednost. Ali pa bi napisali tako metodo, ki bi vstavila vozel v verižni seznam pred element z določeno vrednostjo.

1.6.1.3 Vstavljanje za element z dano vrednostjo

V prejšnjem razdelku smo si ogledali, kako v verižni seznam, podan kot zaporedje vozlov, vstavimo element za vozel, na katerega kaže določeni kazalec. Povedali smo tudi, da bomo primer, če je verižni seznam podan kot objekt tipa `VerSez`, razdelili na primer vstavljanja za element z dano vrednostjo in na primer vstavljanja pred element z dano vrednostjo.

Oglejmo si najprej primer, ko je dano zaporedje vozlov.

Naredili bomo torej metodo, kjer bomo nov vozel vstavili v zaporedje vozlov za vozel z določeno vrednostjo. Metoda bo kot argument sprejela vrednost `x` (nov vozel bomo vstavili za vozel z vrednostjo `x`), kazalec na prvi element v verižnem seznamu ter vrednost `y` (vstavili bomo nov vozel z vrednostjo `y`). Seveda bi lahko podobno kot v prejšnjem primeru predpostavili, da že imamo vozel, v katerem hranimo podatek `y` in bi torej kot tretji parameter uporabili kazalec na ta vozel. A to različico metode lahko mirno prepustimo bralcu, saj se praktično čisto nič ne razlikuje od te, ki jo bomo opisali mi.

Odločiti se moramo, kako bomo ravnali, če se vrednost `x` v zaporedju pojavi večkrat (več kot en vozel v zaporedju ima vrednost podatka enako vrednosti `x`) ali če se ta vrednost v zaporedju sploh ne pojavi (noben vozel v zaporedju nima vrednost podatka enako vrednosti `x`). V prvem primeru se odločimo, da bomo podatek vstavili za prvi vozel z danim podatkom. Če pa podatka v zaporedju ni, bomo zaporedje vozlov pustili tako, kot je bilo podano kot argument metode. V zaporedje vozlov vozla s podatkom `y` torej ne bomo vstavili in samega zaporedja ne bomo spreminjali. Odločiti se moramo tudi, kaj bomo naredili v primeru, ko podamo prazno zaporedje vozlov. Tudi v tem primeru novega vozla v zaporedje ne bomo dodajali, tako bo zaporedje ostalo prazno.

Ideja metode je, da bomo s kazalcem najprej pokazali na tisti vozel, katerega podatek je enak `x`. S tem bomo prišli v enak položaj, kot smo ga opisali v razdelku 1.6.1.2 in problem je rešen.

Razdelajmo sedaj idejo podrobneje.

Kazalec `prvi` vedno kaže na prvi element v zaporedju. Zato bomo uporabili pomožni kazalec (`pom`), s pomočjo katerega se bomo premikali po verigi. Kazalec `prvi` bo še vedno kazal na prvi element v zaporedju. Tako ne bomo izgubili dostopa do prvega elementa zaporedja.

S pomožnim kazalcem `pom` bomo najprej pokazali na prvi element zaporedja, se pravi tja, kamor že kaže kazalec `prvi`. Prireditveni stavek se bo torej glasil:

```
Vozel pom = prvi;
```

S klicem konstruktorja tipa `Vozel(int x)` bomo ustvarili vozel z vrednostjo `y`. To bo ustvarilo vozel, ki ga bomo v zaporedje vstavili. Prireditveni stavek je:

```
Vozel pomozni = new Vozel(y);
```

Sedaj pa težji del. Vozel z vrednostjo `y` moramo vstaviti za prvi vozel v zaporedju, katerega vrednost je enaka `x`. Zato moramo zaporedoma pregledovati vrednosti (podatke vozlov) v zaporedju. Uporabili bomo zanko `while`. Sedaj moramo razmisliti, kdaj naj se zanka zaključi.

Izvaja naj se toliko časa, dokler v zaporedju ne najdemo vozla s podatkom z vrednostjo x. Vendar ne vemo, kateri vozel, če sploh kakšen, ima podatek z vrednostjo x. Zato moramo v splošnem pregledati vrednosti vseh vozlov v zaporedju. Torej naj se zanka izvaja toliko časa, dokler zaporedje ni prazno (v tem primeru bo kazalec pom kazal na referenco null). Zanka bo torej:

```
while(pom != null) {}
```

Znotraj zanke bomo preverili, ali je vrednost trenutnega vozla (torej vrednost tistega vozla, kamor trenutno kaže kazalec pom) enaka x. Torej:

```
if(pom.vrniVrednost() == x){}
```

Ko je pogoj stavka if izpolnjen, kazalec pom torej kaže na vozel z vrednostjo x. Takrat vozel pomozni vstavimo v zaporedje. V tem primeru imamo isto stanje kot v razdelku 1.6.1.2 in lahko uporabimo že narejeno metodo dodaj_na_sredino(Vozel pomozni, Vozel pom). Argumenta metode bosta vozel pomozni (vozel, ki ga vstavimo v zaporedje), ter vozel pom (vozel, z vrednostjo x, torej vozel, za katerega vstavimo vozel pomozni).

Našli smo torej prvi vozel v zaporedju, ki ima podatek z vrednostjo x, ter vozel pomozni vstavili v zaporedje. Ker smo rekli, da bomo vozel vstavili le za prvi vozel s podatkom x, zanko while zaključimo z ukazom break.

Če pa vozla s podatkom x še nismo našli, s kazalcem pom pokažemo na naslednika vozla, na katerega trenuno kaže kazalec pom in se tako premaknemo naprej po verigi.

Celotna metoda bi bila:

```
public static void vstavi1(int x, Vozel prvi, int y){
   //za vozel z vrednostjo x vstavi vozel z vrednostjo y
   //če se vrednost x pojavi večkrat, vstavimo za prvi vozel s
   //podatkom z vrednostjo x
   //če vozel prvi kaže v prazno (zaporedje je prazno) ali pa
   //podatka x v verigi ni, ne naredimo nič
   Vozel pom = prvi; //s pom pokažemo na prvi vozel v zaporedju
   Vozel pomozni = new Vozel(y); //skonstruiramo vozel, kjer
                                 //bo podatek, ki ga vstavljamo
   while(pom != null){ //dokler ne pregledamo vseh
      if(pom.vrniPodatek() == x){ //našli smo iskani podatek
         //y vstavimo za x
         MetodeNaZaporedju.dodaj_na_sredino(pomozni, pom);
         break; //opravili smo vstavljanje, lahko zaključimo
      }
      pom = pom.vrniNasled(); //prestavimo se na naslednji vozel
                              //v zaporedju
   }
}
```

Kaj pa, če imamo dan objekt tipa `VerSez`. V tem primeru bo metoda kot argument sprejela vrednost `x` (nov vozel bomo vstavili za vozel z vrednostjo `x`), kazalec na verižni seznam `vs` ter vrednost `y` (vstavili bomo vozel z vrednostjo `y`).

Pri kodiranju te metode bomo uporabili prej razvito metodo.

```
public static void vstavi2(int x, VerSez vs, int y){
   Vozel prvi = vs.prvi(); //prvi pokaže na začetek zaporedja vozlov
   vstavi1(x, prvi, y);
}
```

S pomočjo ustrezne metode iz objekta verižni seznam le vzamemo kazalec na prvi vozel v zaporedju. S tem imamo na voljo enake podatke in enako stanje kot prej, zato le še pokličemo prej razvito metodo.

1.6.1.4 Vstavljanje pred element z dano vrednostjo

Tu bomo pred vozel z vrednostjo `x` vstavili vozel z vrednostjo `y`. Ta primer je nekoliko težji od prvega in zahteva premislek. Kot prej, tudi tu najprej pogledamo, kaj bi naredili, če je dano zaporedje vozlov, potem pa si bomo ogledali še, kaj storiti, če imamo dan objekt tipa `VerSez`.

Najprej bomo naredili prostor za nov vozel, kamor bomo shranili `y`. Na ta vozel pokažimo s kazalcem `pomozni`. Nato si bomo zopet pomagali s pomožnim kazalcem `pom`, ki bo kazal na trenutni vozel (tako kot v prejšnjem razdelku). Če je vrednost trenutnega vozla (`pom`) enaka vrednosti `x`, potem moramo vozel `pomozni` vstaviti pred vozel `pom`. Predhodniku vozla `pom` moramo za naslednika nastaviti vozel `pomozni`. Kako pa bomo to naredili, če na predhodnika vozla `pom` ne kaže noben kazalec? Potrebovali bomo dodatni kazalec (`pom1`), ki bo vedno kazal na predhodnika vozla `pom`.

Če dobro razmislimo, ugotovimo, da je stanje tedaj, ko smo vozel s podatkom z vrednostjo `x` našli, podobno kot v razdelku 1.6.1.2. Vstavljanje pred vozel z vrednostjo `x` bomo prevedli na primer vstavljanja za predhodnika vozla z vrednostjo `x`. Uporabili bomo torej metodo `dodaj_na_sredino(Vozel pomozni, vozel pom)`, kjer bomo kot prvi argument uporabili vozel `pomozni` (torej tistega, ki vsebuje podatek, ki ga vstavljamo), kot drugi element pa vozel `pom1` (ta namreč kaže na predhodnika vozla z vrednostjo `x`). Torej bomo s klicem `dodaj_na_sredino(pomozni, pom1)` za vozel `pom1` (to je predhodnik vozla `pom`, oz. vozla z vrednostjo `x`) vstavili vozel `pomozni`.

Razdelajmo sedaj idejo podrobneje. Premislimo, kaj bomo storili v primerih, ko je zaporedje prazno, ko je več vozlov s podatkom z vrednostjo `x` ali ko takega vozla ni, ter v primeru, ko je vrednost podatka prvega vozla enaka `x`. Če je zaporedje prazno, ali če iskanega podatka v zaporedju ni, zaporedja ne bomo spreminjali. Tudi tukaj bomo v primeru, ko je iskanih vrednosti

več, vozel z vrednostjo `y` vstavili pred prvi vozel s podatkom z vrednostjo `x`. Posebej moramo razdelati tudi primer, ko je vrednost podatka prvega vozla enaka `x`. V tem primeru moramo vozel z vrednosto `y` vstaviti pred prvi vozel. Torej se bo spremenil kazalec na prvi vozel verige. Zato ta metoda ne bo tipa `void` kot zgornja, temveč bo vračala objekt tipa `Vozel`, torej kazalec na novi (ali stari, če ne bo sprememb) prvi vozel verige.

Vsakega koraka metode ne bomo podrobneje razlagali, saj bomo uporabili znanje, ki smo ga do sedaj že osvojili. Idejo vstavljanja pred vozel z vrednostjo `x` smo z razmislekom preoblikovali v idejo vstavljanja za predhodnika vozla z vrednostjo `x`. Podrobneje si bomo ogledali le tisti del metode, ki se razlikuje od metode `vstavi1(int x, Vozel prvi, int y)`. Potrebovali bomo dva pomožna kazalca, saj moramo imeti kazalec tudi na predhodnika vozla, ki vsebuje podatek z vrednostjo `x`. S kazalcem `pom1` bomo na začetku metode pokazali na prvi vozel, s kazalcem `pom` na njegovega naslednika. Vendar se moramo tukaj vprašati, kaj se zgodi, če kot argument metode podamo prazno zaporedje. Takrat s kazalcem `pom` ne moremo pokazati na naslednika vozla `pom1`, saj vozla `pom1` ni (je le kazalec z vrednostjo `null`). Problem rešimo tako, da najprej preverimo, če je zaporedje prazno, ter, kot smo se dogovorili, v tem primeru vrnemo prazno zaporedje. Če pa zaporedje ni prazno, s kazalcema `pom` ter `pom1` pokažemo na ustrezna vozla.

Omenili smo tudi, da je poseben primer, ko je vrednost podatka prvega vozla enaka `x`. Takrat uporabimo znanje iz razdelka 1.6.1.1., saj gre v tem primeru za vstavljanje podatka na začetek zaporedja.

Del metode, v katerem vozel s podatkom z vrednostjo `x` iščemo, ko ga najdemo pa vstavimo vozel z vrednostjo `y` v zaporedje, je enak kot v metodi `vstavi1(int x, Vozel prvi, int y)`.

```
public static Vozel vstavi3(int x, Vozel prvi, int y) {
//pred vozel z vrednostjo x vstavi vozel z vrednostjo y
//če se vrednost x pojavi večkrat, vstavimo za prvi vozel s
// podatkom z vrednostjo x
//če vozel prvi kaže v prazno (zaporedje je prazno),
//ne naredimo nič
   if(prvi == null){ //če je zaporedje prazno, vrnemo prav tako
      return null;
   }
   Vozel pom1 = prvi; //s pom1 pokažemo na prvi vozel v zaporedju
   Vozel pom = pom1.vrniNasled(); //s pom pokažemo na
                                  //naslednika vozla pom1
   Vozel pomozni = new Vozel(y); //skonstruiramo vozel, kjer bo
                                 //podatek, ki ga vstavljamo
   if(prvi.vrniPodatek() == x){ //če je vrednost podatka prvega vozla
                     //enaka x, vozel pomozni vstavimo pred prvi vozel
      pomozni.nastaviNasled(prvi);
      prvi = pomozni;
      return prvi; // zaključimo metodo
   }
   while(pom != null){ //dokler ne pregledamo vseh
      if(pom.vrniPodatek() == x){ //našli smo iskani podatek
                                  //y vstavimo za x
         dodaj_na_sredino(pomozni, pom1);
         return prvi; //opravili smo vstavljanje, lahko zaključimo
      }
      pom1 = pom; //novi predhodnik bo trenutni vozel
      pom = pom.vrniNasled(); //prestavimo se na naslednji vozel
                              //v zaporedju
   }
   return prvi;
}
```

Sedaj pa si oglejmo, kako ravnati, če je dan objekt tipa `VerSez`. Ideja metode je podobna kot v metodi `vstavi3(int x, Vozel prvi, int y)`. Razlika je v tem, da bo metoda kot argument poleg vrednosti `x` in `y` sprejela tudi objekt tipa `VerSez` in ne objekt tipa `Vozel`. Metoda bo tipa `void`, saj bo po potrebi (če bo prvi vozel že vseboval podatek `x`) spremenila obstoječi objekt tipa `VerSez`. S pomočjo ustrezne metode iz objekta verižni seznam bomo dobili kazalec na prvi vozel v zaporedju. Tega bomo imenovali `prvi`. Preverili bomo, ali je seznam prazen. V tem primeru ne bo potrebno narediti nič. V primeru, da je vrednost podatka prvega vozla

enaka x, bomo s pomočjo metode `vstavi_prvega(int x)` iz razreda `VerSez`, vstavili vozel `pomozni` (vozel z vrednostjo y) v verižni seznam na prvo mesto (oz. pred prvi element verižnega seznama). Če verižni seznam ni prazen in tudi prvi element ni enak x, potem postopoma pregledujemo vrednosti podatkov vozlov verižnega seznama. Ta del metode bo enak kot v metodi `vstavi3(int x, Vozel prvi, int y)` in si ga ne bomo podrobneje ogledali. Za to, da bomo vstavili vozel z vrednostjo y pred vozel z vrednostjo x, bomo uporabili metodo `dodaj_na_sredino(Vozel pomozni, Vozel pom)`.

Celotna metoda:

```
public static void vstavi4(int x, VerSez  verSe1, int y){
   if(verSe1.prazen()){//če je seznam prazen, končamo
      return;
   }
   Vozel prvi = verSe1.prvi(); //pokažemo na začetek zaporedja vozlov
   Vozel pom1 = prvi; //s pom1 pokažemo na prvi vozel v seznamu
   Vozel pom = pom1.vrniNasled();//s pom pokažemo na naslednika
   //vozla pom1
   Vozel pomozni = new Vozel(y); //skonstruiramo vozel, ki ga
                                 //vstavljamo
   if(prvi.vrniPodatek() == x){ //če je vrednost podatka prvega
            //vozla enaka x, vozel pomozni vstavimo pred prvi vozel
      verSe1.vstavi_prvega(pomozni);
      return; //konec
   }
   while(pom != null){ //dokler ne pregledamo vseh
      if(pom.vrniPodatek() == x){ //našli smo iskani podatek
         //vozel z vrednostjo y vstavimo za vozel z vrednostjo x
         MetodeNaZaporedju.dodaj_na_sredino(pomozni, pom1);
         return verSe1; //opravili smo vstavljanje, lahko
                        //zaključimo
      }
      pom1 = pom; //prestavimo se na naslednji vozel v zaporedju
      pom = pom.vrniNasled();//prestavimo se na naslednji vozel
                             //v zaporedju
   }
}
```

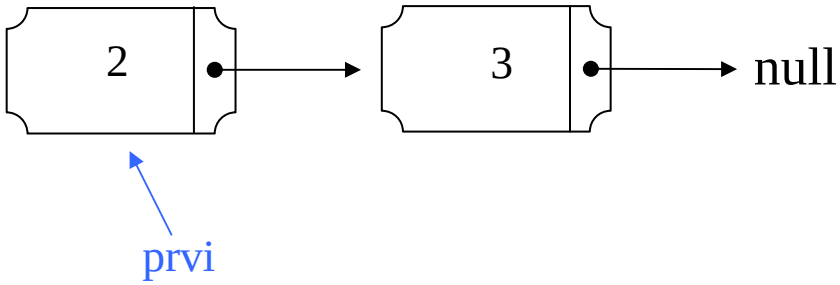
1.6.1.5 Vstavljanje na konec

Vozel bomo vstavili na konec zaporedja. Naredili bomo primer tako z uporabo zaporedja objektov razreda `Vozel`, kot z uporabo objektov razreda `VerSez`. Da bomo vozel lahko vstavili na konec zaporedja vozlov ali na konec verižnega seznama kot objekta, potrebujemo kazalec na zadnji vozel. Vendar pa imamo zaporedje vozlov podano le s kazalcem na prvi element, pri verižnem seznamu pa tako ali tako lahko neposredno dostopamo le do prvega vozla. Kako bomo torej prišli do kazalca na zadnji vozel? V prejšnjih razdelkih smo že spoznali koncept premikanja po vozlih verižnega seznama. Na ta način bomo rešili tudi ta problem in z ustrezno napisano zanko poskrbeli za kazalec na zadnji vozel.

- **S pomočjo razreda Vozel**

Najprej si oglejmo, kako bi vozel vstavili na konec zaporedja, če ima zaporedje le dva elementa. V tem primeru bomo kazalec na zadnji element postavili z enim samim prireditvenim stavkom in zanka `while` ne bo potrebna.

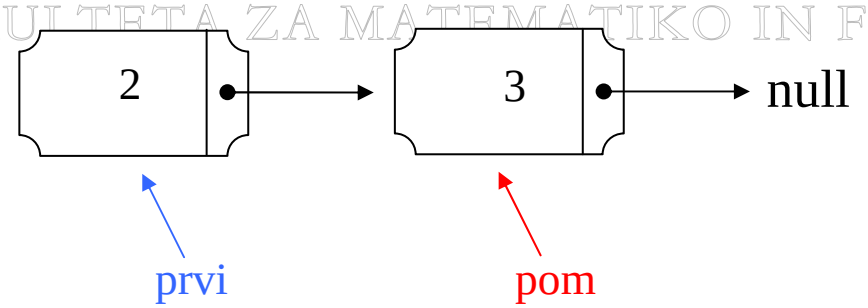
Imamo zaporedje vozlov:



SLIKA 56: Primer zaporedja vozlov

in osamljeni vozel `pomozni`, ki ga želimo vstaviti na konec tega zaporedja, torej za vozel s podatkom 3. Predpostavimo še, da komponenta naslednji v vozlu `pomozni` že kaže v prazno. `Prvi` je kazalec na prvi element. Če želimo vstaviti vozel na konec zaporedja, potrebujemo tudi kazalec na zadnji element v zaporedju. Tega še nimamo. Ker pa ima naša veriga le dva elementa, bomo do potrebnega kazalca prišli na zelo enostaven način. Uporabili bomo pomožno spremenljivko `pom` in jo s pomočjo klica metode `vrniNasled()` nad vozlom `prvi` nastavili na zadnji element v zaporedju.

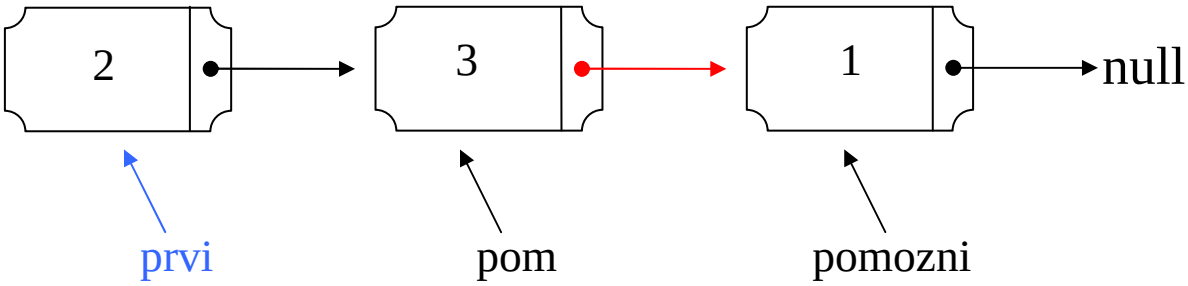
```
Vozel pom = prvi.vrniNasled();
```



SLIKA 57: S kazalcem `pom` smo pokazali na zadnji element v zaporedju vozlov s slike 56

Sedaj, ko imamo kazalec na zadnji element, vozel `pomozni` vstavimo v zaporedje na zadnje mesto. V ta namen vozlu `pom` nastavimo za naslednika vozel `pomozni`. Sedaj je vozel `pomozni` novi zadnji element v zaporedju. Ustrezni stavek je:

```
pom.nastaviNasled(pomozni);
```



SLIKA 58: Vstavili smo nov vozel na konec zaporedja

Metoda bo kot argument dobila objekt `Vozel`, ki ga bomo vstavili v zaporedje in objekt `Vozel`, ki bo kazal na prvi element v zaporedju.

Metoda:

```
public static void dodaj_na_konec1(Vozel pomozni, Vozel prvi){
//vstavljanje na konec verige z dvema vozloma
   Vozel pom = prvi.vrniNasled();
   pom.nastaviNasled(pomozni);//pomozni je novi zadnji vozel
}
```

Zaporedje vozlov pa ima lahko (in ponavadi je tako) več (ali manj) kot dva elementa. Verjetno ni smiselno, da bi za vsako možno število vozlov napisali svojo metodo. Velikokrat tudi ne vemo vnaprej, koliko elementov ima zaporedje. Zato bomo problem rešili kar v splošnem z uporabo zanke `while` in dveh pomožnih spremenljivk.

Kot smo napovedali, bomo uporabili dve pomožni spremenljivki. S prvo (`pom`) bomo na začetku pokazali na vozel `prvi`, s spremenjivko `pom1` pa na naslednika vozla `prvi`. Z zanko `while` se bomo z obema kazalcem hkrati premikali po elementih, dokler bo v zaporedju še kakšen element

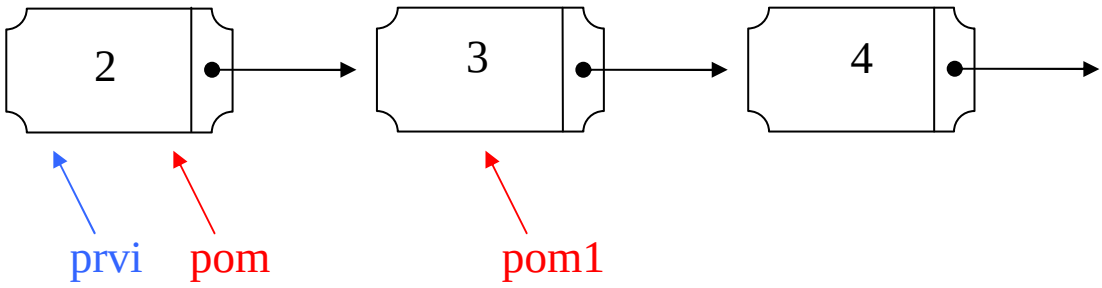
(dokler `pom1` ne bo imel vrednosti `null`). Ko se bo zanka `while` končala, bo spremenljivka `pom1` kazala v prazno (imela vrednost `null`), `pom` pa bo kazal na zadnji vozel v zaporedju. Nato bomo vozel `pomozni` vstavili kot zadnji element v zaporedju. Kako to storimo, smo že videli pri zgledu z dvema elementoma.

Metoda bo kot argument sprejela objekta tipa `Vozel` (`pomozni` in `prvi`) in bo tipa `void` (ne vrača nobene vrednosti).

Prikažimo zgornjo rešitev grafično po korakih. Najprej bomo nastavili kazalca `pom` in `pom1`. S kazalcem `pom` bomo pokazali na vozel `prvi`, s `pom1` pa na njegovega naslednika. Prireditvena stavka bosta:

```
Vozel pom = prvi;
Vozel pom1 = prvi.vrniNasled();
```

Recimo, da ima naše zaporedje osem elementov z vrednostmi podatkov od 2 do 9. Zaradi lepšega prikaza bomo grafično prikazali le tisti del zaporedja, v katerem bomo tisti trenutek spreminjali kazalce. Na prvi sliki (slika 59) tako vidimo kazalec `prvi`, ter kazalca `pom` in `pom1` (kamor kažeta, ko se izvedeta zgornja prireditvena stavka).

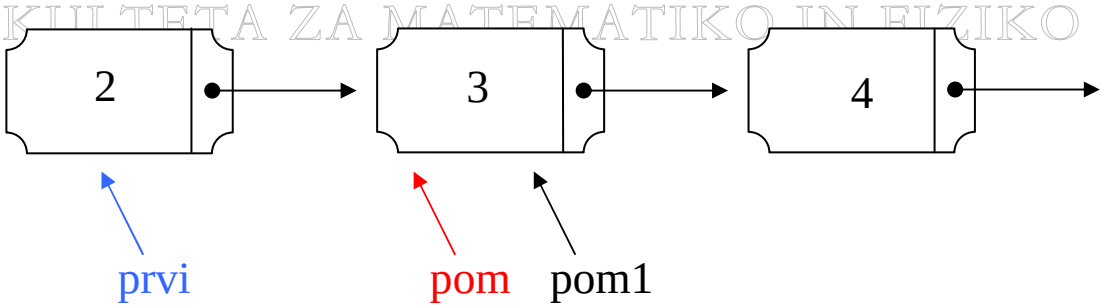


SLIKA 59: Začetek zaporedja v katerega bomo vstavili vozel na zadnje mesto

Sedaj se bomo s pomočjo zanke `while` s kazalcema `pom` in `pom1` premikali po zaporedju, dokler ne pridemo do konca. Ustrezna zanka bo:

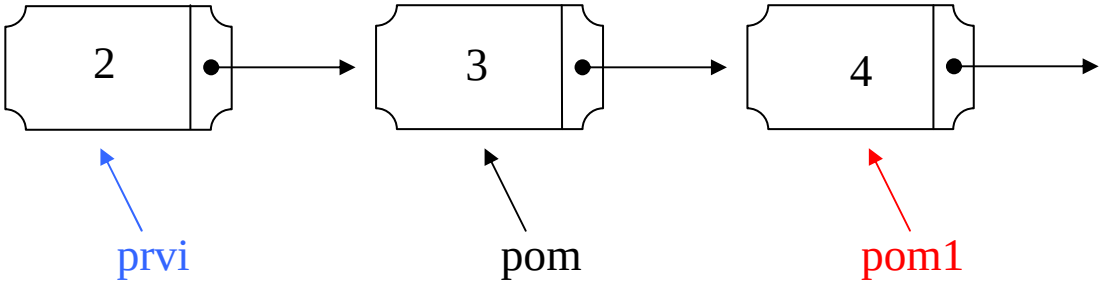
```
while(pom1 != null){
   pom = pom1;
   pom1 = pom1.vrniNasled();
}
```

Zanka se izvaja, dokler `pom1` ne kaže na `null`. Znotraj zanke najprej s kazalcem `pom` pokažemo na vozel `pom1`, s `pom1` pa pokažemo na njegovega naslednika. Ko se zanka `while` izvaja prvič, bo torej po prvem koraku (`pom = pom1;`) stanje tako:



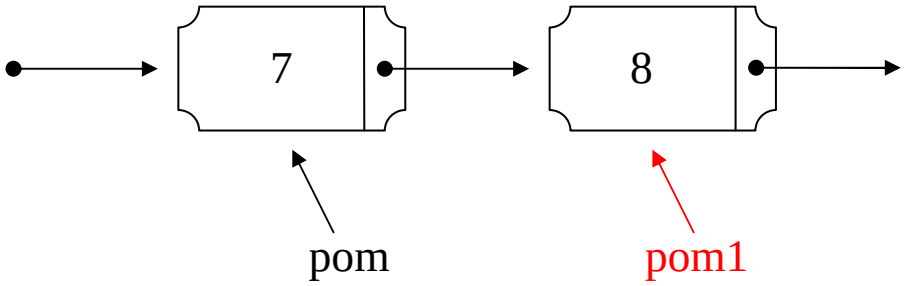
SLIKA 60: Prvi korak v zanki `while`

po drugem (`pom = pom1.vrniNasled();`) pa:



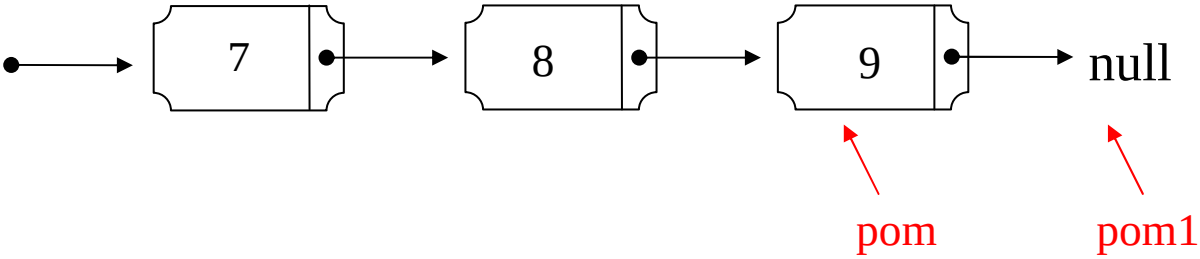
SLIKA 61: Drugi korak v zanki `while`

Ker `pom1` še ne kaže v prazno, nadaljujemo. Po nekaj korakih zanke, imamo naslednje stanje:



SLIKA 62: Pred koncem smo

Ker `pom1` še nima vrednosti `null`, nadaljujemo. Zanka se izvede še enkrat in dobimo:

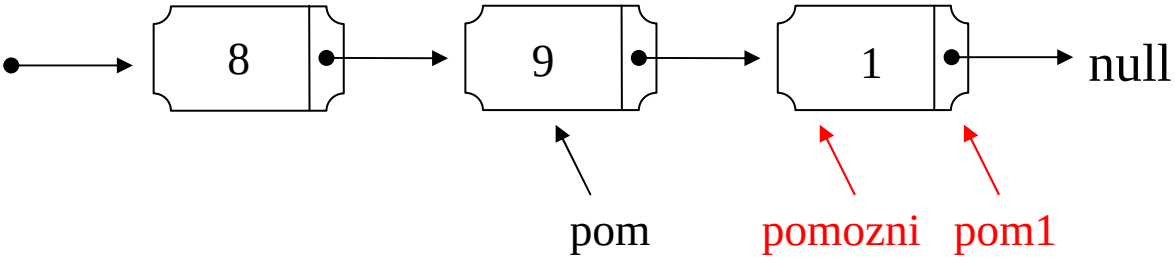


SLIKA 63: Zaporedje ob koncu izvajanja zanke `while`

Ker `pom1` kaže v prazno, smo z zanko končali. Sedaj imamo kazalec `pom` usmerjen na zadnji element v zaporedju. Vozel `pomozni` vstavimo v zaporedje tako, da vozlu `pom` nastavimo naslednika na vozel `pomozni`.

```
pom.nastaviNasled(pomozni);
```

Ustrezna slika:



SLIKA 64: Vozel `pomozni` smo vstavili na konec zaporedja

Razmislimo, kaj se zgodi v primeru, ko kot argument podamo prazno zaporedje. Takrat na vozlu `pom1` ne moremo izvesti metode `vrniNasled()`, saj vozla `pom1` ni (`pom1` ima vrednost `null`). Torej se metoda ne bo pravilno izvedla. Če podamo prazno zaporedje, potem vanj vozla `pomozni` ne bomo vstavili, temveč bomo pustili zaporedje tako, kot je. Če je vrednost kazalca `prvi` enaka `null`, bomo preverili s stavkom `if`. Ker vozla `pomozni` v zaporedje v tem primeru ne bomo vstavili, bomo znotraj stavka if uporabili stavek `return` in s tem zaključili metodo (nesmiselno bi bilo, da se metoda izvaja, saj je zaporedje prazno).

Napišimo metodo:

```
public static void dodaj_na_konec2(Vozel pomozni, Vozel prvi){
   if(prvi == null){ //zaporedje je prazno
      return;
   }
   Vozel pom = prvi;
   Vozel pom1 = prvi.vrniNasled();
   while(pom1 != null){ //s pom pokažemo na zadnji vozel
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   pom.nastaviNasled(pomozni);//vozel pomozni vstavimo na zadnje
                              //mesto
}
```

Pravzaprav ne bi potrebovali dveh pomožnih kazalcev. Dovolj bi bil le `pom`. Spremeniti moramo le pogoj tako, da preverjamo, če naslednik vozla `pom` ne obstaja več.

```
public static void dodaj_na_konec2a(Vozel pomozni, Vozel prvi){
   if(prvi == null){ //zaporedje je prazno
      return;
   }
   Vozel pom = prvi;
   while(pom.vrniNasled() != null){ //s pom pokažemo na zadnji vozel
      pom = pom.vrniNasled();
   }
   pom.nastaviNasled(pomozni);//vozel pomozni vstavimo na zadnje
                              //mesto
}
```

- **S pomočjo razreda VerSez**

Imamo dan objekt `verSe1` iz razreda `VerSez` in vozel `pomozni`. V verižni seznam, ki ga `verSe1` predstavlja, bomo na konec vstavili vozel `pomozni`.

Ob pisanju metode `dodaj_na_konec1(Vozel pomozni, Vozel prvi)` smo ugotovili, da je dodajanje elementa v zaporedje ali verižni seznam, ki ima le dva elementa, trivialno. Metoda bi kot argument sprejela objekt `VerSez` ter vrednost `x` (vrednost podatka vozla, ki bi ga vstavili v verižni seznam). Z uporabo metode `prvi()` iz razreda `VerSez`, bi pokazali kazalec na prvi element v seznamu. Uporabili bi pomožni kazalec `pom`, ki bi ga pokazali na prvi vozel. Nato bi s pomočjo klica metode `vrniNasled()` nad tem vozlom le tega preusmerili na njegovega naslednika. Seznam ima samo dva elementa, tako bi sedaj vozel `pom` kazal na zadnji element. Referenco vozla `pom` bi nato prestavili na vozel, ki ga želimo vstaviti v zaporedje (vozel `pomozni`). Tako bi ta vozel vstavili v zaporedje. Metoda je torej zelo podobna že omenjeni metodi.

```
public static void dodaj_na_konec3(VerSez verSe1, int x){
   Vozel pomozni = new Vozel(x);
   Vozel pom = verSe1.prvi(); //začetek verige
   pom = pom.vrniNasled(); //kažemo na drugi (zadnji) vozel
   pom.nastaviNasled(pomozni); //dodamo na konec
}
```

Oglejmo si sedaj splošno metodo, ki bo kot argument sprejela objekt `verSe1` iz razreda `VerSez`, in objekt Vozel `pomozni`, torej vozel, ki ga bomo vstavili v verižni seznam. Metoda bo praktično enaka metodi `dodaj_na_konec2(Vozel pomozni, Vozel prvi)`. Razlika bo le v tem, da bomo tu pomožni kazalec `pom` nastavili na prvi vozel v seznamu s klicem metode `prvi()` iz razreda `VerSez`. Ostali del kode pa bo enak kot v metodi `dodaj_na_konec2(Vozel pomozni, Vozel prvi)`.

Metoda bo:

```
public static void dodaj_na_konec4(Vozel pomozni, VerSez verSe1){
   if(verSe1.prazen()){ //verižni seznam je prazen
      return;
   }
   Vozel pom = verSe1.prvi();
   Vozel pom1 = pom.vrniNasled();
   while(pom1 != null){ //s pom pokažemo na zadnji vozel
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   pom.nastaviNasled(pomozni); //pomozni je novi zadnji vozel
}
```

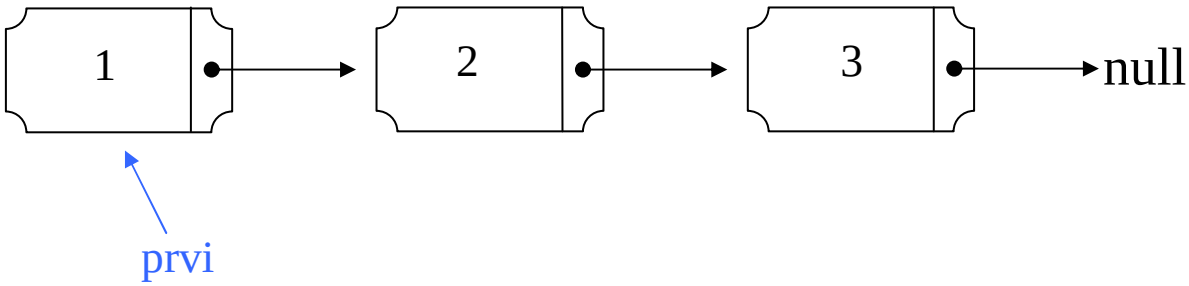
1.6.2 Brisanje

1.6.2.1 Brisanje prvega vozla

- **S pomočjo razreda Vozel**

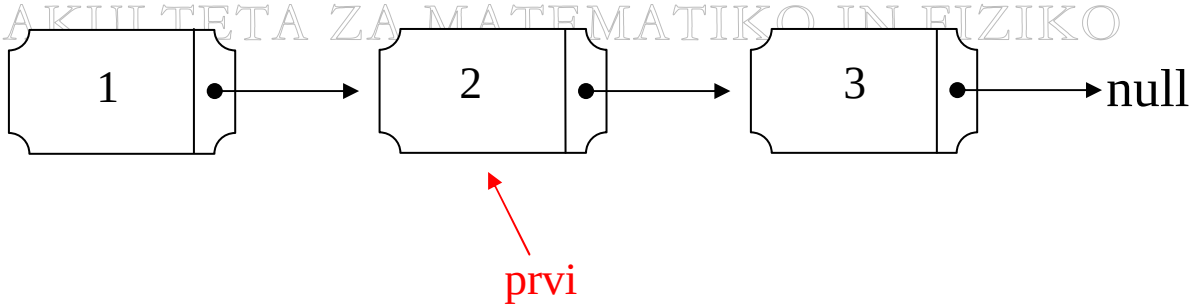
Iz zaporedja bomo izbrisali prvi vozel. Vozla dejansko ne bomo izbrisali, le kazalec na prvi element v zaporedju bomo prestavili. Z njim bomo pokazali na drugi element v zaporedju. Vozel, na katerega je na začetku kazal kazalec `prvi`, še vedno obstaja, vendar nanj ne kaže noben kazalec. Zato do njega ne moremo dostopati. Začetek zaporedja bo sedaj vozel, na katerega sedaj kaže kazalec `prvi`.

Denimo, da je dano zaporedje vozlov:



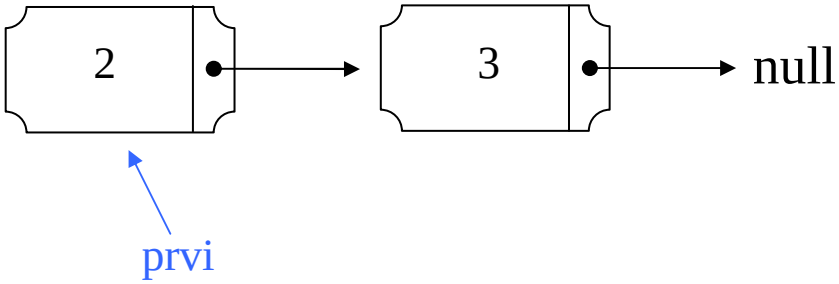
SLIKA 65: Primer zaporedja

Iz zgornjega zaporedja želimo odstraniti prvi element. Kot smo omenili že zgoraj, to naredimo tako, da s kazalcem `prvi` pokažemo na drugi element zaporedja.



SLIKA 66: Kazalec `prvi` smo preusmerili na drugi element danega zaporedja

Vozel s podatkom 1 še vedno obstaja, vendar, kot vidimo na sliki 66, nanj ne kaže noben kazalec. To pomeni, da do njega ne moremo dostopati. Kazalec `prvi` kaže na vozel z vrednostjo 2, ki je sedaj prvi vozel v našem zaporedju. Naše zaporedje ima sedaj dva elementa, in sicer:



SLIKA 67: Novo zaporedje

Po kratkem razmisleku ugotovimo, da je brisanje prvega vozla v zaporedju zelo enostavno. Le kazalec `prvi` mora pokazati na naslednika vozla, na katerega je kazal prej. To naredimo s klicem metode `vrniNasled()` nad vozlom `prvi`.

Razmislimo, kaj storiti v primeru, ko podamo prazno zaporedje vozlov. Problem bomo enostavno rešili tako, da bomo znotraj metode najprej preverili, ali je vrednost kazalca `prvi` enaka `null`. Če vrednost kazalca `prvi` ni enaka `null`, potem kazalec `prvi` prestavimo, sicer pa ne naredimo nič. Metoda bo kot argument dobila parameter tipa `Vozel`, ki bo kazalec na prvi element v zaporedju. Prav tako bo tak kazalec vrnila. Če bomo obstoječemu zaporedju želeli zbrisati prvi element, bomo metodo torej poklicali z

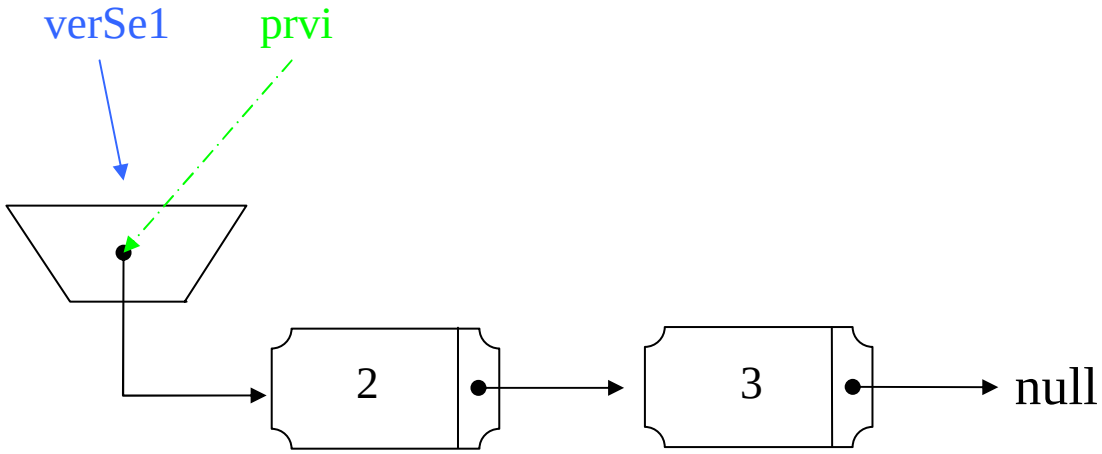
```
prviVZaporedju = brisi_prvega1(prviVZaporedju).
```

Poglejmo si sedaj metodo:

```
public static Vozel brisi_prvega1(Vozel prvi){
   if(prvi != null){ //zaporedje ni prazno
      prvi = prvi.vrniNasled(); //s kazalcem prvi pokažemo na drugi
                                //vozel
      return prvi;
   }
   return null; //vrnemo prazno zaporedje
}
```

❖ **S pomočjo razreda VerSez**

Imamo podan objekt verižni seznam iz razreda `VerSez`. Iz verižnega seznama, ki ga predstavlja, bomo izbrisali prvi vozel.



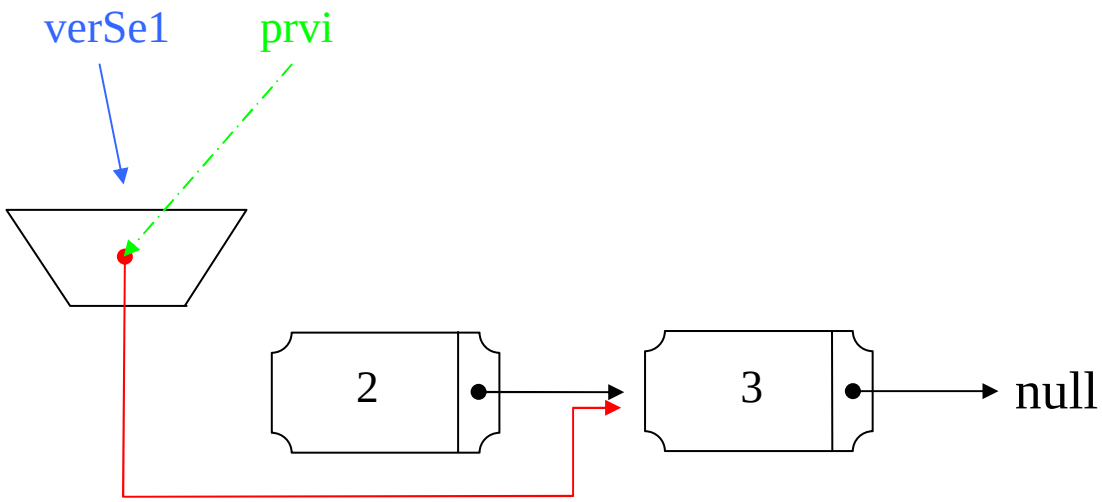
SLIKA 68: Primer verižnega seznama

V razredu `VerSez` metoda, ki odstrani prvi element, že obstaja. Imenuje se `odstrani_prvega()`. To je objektna metoda, ki spremeni verižni seznam, nad katerim jo izvajamo. A ko smo sestavljali tisti razred, smo predpostavili, da verižni seznam ni prazen. Tu odpravimo to pomanjkljivost in napišimo metodo, ki bo delovala vedno. Seveda metoda ne bo objektna, saj ne popravljamo razred `VerSez`, ampak bo statična. Vrnila bo nov objekt tipa `VerSez`, kjer bo kazalec `prvi` kazal na popravljeno zaporedje vozlov (prejšnje zaporedje brez prvega vozla). Dogovorimo se še, da če metodo pokličemo nad praznim verižnim seznamom, vrne nov prazen verižni seznam.

Ta metoda podobno, kot metoda `vstavi_prvega(Vozel v)` sama poskrbi, da se ustrezno popravijo reference (kazalci). Za klic metode bomo uporabili verižni seznam s slike 68. Metodo, ki jo bomo naredili, bomo imenovali `brisi_prvega2` in bo kot argument sprejela objekt tipa `VerSez` (natančneje – kazalec nanj). Metoda bo vrnila objekt tipa `VerSez` (kazalec na tak objekt). Če zanemarimo težave s praznim seznamom, oziroma morebitne težave s seznamom z enim vozlom, bi lahko poskusili takole:

```
public static VerSez brisi_prvega2(VerSez verSe1){
   verSe1.odstrani_prvega();
   return verSe1;
}
```

Ustrezna slika bo:



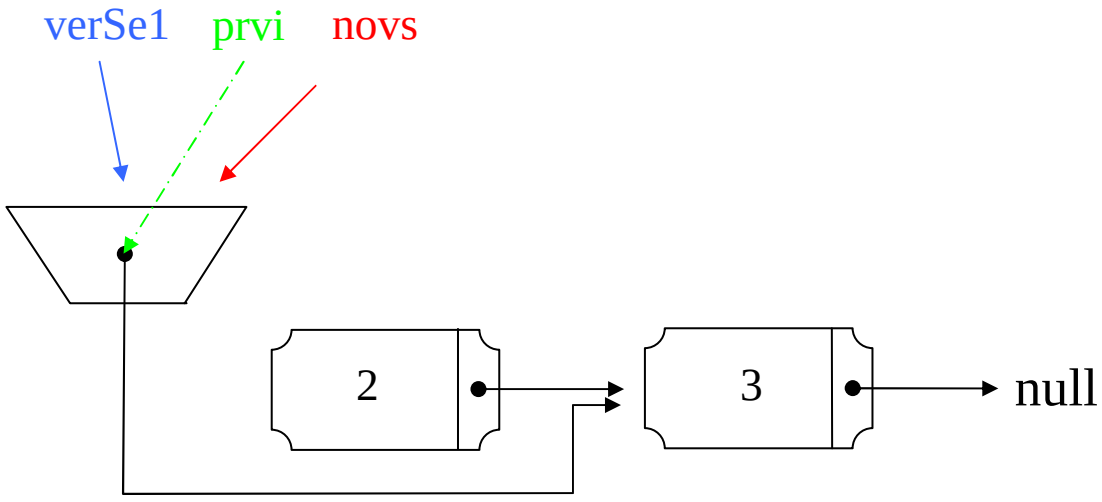
SLIKA 69: Verižni seznam po klicu metode `brisi_prvega2(VerSez verSe1)`

Na zgornji sliki je z rdečo barvo prikazana povezava, ki jo spremeni klic metode `verSe1.odstrani_prvega()`.

Metoda je po učinku enaka objektni metodi `odstrani_prvega()` iz razreda `VerSez`, le da dobimo dodatni kazalec na spremenjeni verižni seznam, ki nam ga da stavek `return`. Če torej metodo izvedemo s prireditvenim stavkom

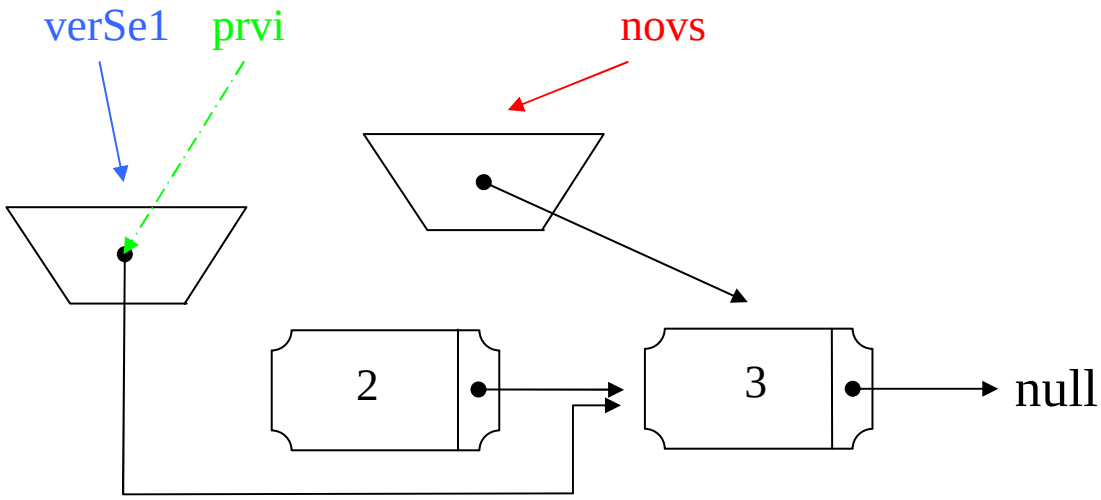
```
VerSez novs = brisi_prvega2(verSe1)
```

dobimo:



SLIKA 70: Stanje po stavku `VerSez novs = brisi_prvega2(verSe1)`

Vendar smo dejali, da bi radi, da naša metoda vrne nov verižni seznam, torej bi radi dobili tako stanje:



SLIKA 71: Stanje, kot ga želimo po klicu metode `brisi_prvega2(VerSez verSe1)`

Seveda bomo upoštevali tudi to, kaj se lahko zgodi, če je prvotni seznam prazen. Naš načrt je: najprej bomo znotraj metode naredili nov verižni seznam (praznega), saj v obeh primerih, ko seznam je in ko ni prazen, želimo vrniti nov verižni seznam. Nato bomo preverili ali je verižni seznam, podan kot argument, prazen. Če bo, vrnemo nov prazen verižni seznam. V nasprotnem primeru pa najprej iz verižnega seznama `verSe1` s klicem objektne metode `odstrani_prvega()` odstranimo prvi vozel, nato pa s kazalcem novega verižnega seznama pokažemo na spremenjeni verižni seznam.

```
public static VerSez brisi_prvega3(VerSez verSe1){
   VerSez novi = new VerSez();
   if(verSe1.prazen()){ //seznam je prazen
      return novi; //vrnemo nov verižni seznam
   }
   verSe1.odstrani_prvega(); //odstranimo prvi vozel
   //Tukaj imamo problem! Potrebovali bi:
   //novi.nastaviPrvega(verSe1.prvi());
   return novi; //vrnemo nov verižni seznam
}
```

Ko poskusimo napisati utrezno kodo, opazimo da ne gre! Manjka nam metoda, s pomočjo katere bi spremenili komponento `prvi` iz razreda `VerSez`. Ta se spremeni le z dodajanjem prvega vozla oziroma z brisanjem. Kam kaže `prvi`, lahko izvemo (metoda `prvi()`), enostavnega prenastavljanja komponenete `prvi` pa ni! V razredu `VerSez` bi torej potrebovali metodo `nastaviPrvega(Vozel noviPrvi)`, ki bi nastavila komponento `prvi` tako, da bo kazala

na nov prvi vozel verižnega seznama. Torej smo pri načrtovanju razreda `VerSez` naredili napako, saj bi bila ta metoda res nujna! Zato predpostavimo, da smo razred VerSez popravili tako, da vsebuje omenjeno metodo, ki jo navedimo tukaj:

```
public void nastaviPrvega(Vozel novPrvi){
   //metoda spremeni kazalec  na prvi element verižnega seznama
   this.prvi = novPrvi;
}
```

Potem lahko odkomentiramo predzadnjo vrstico metode `brisi_prvega3(VerSez verSe1).`

1.6.2.2 Brisanje na sredini

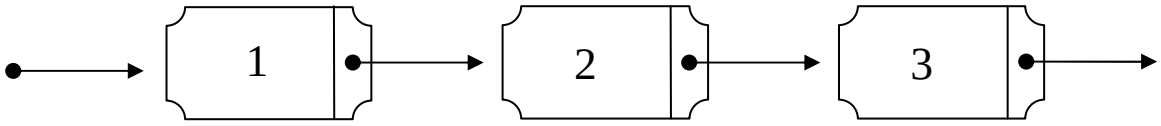
Denimo, da bi iz verižnega seznama radi odstranili vozel, ki vsebuje podatek z določeno vrednostjo. Kot prej, si bomo tudi tu najprej pogledali, kaj storimo, če imamo podan kazalec na prvi element nekega zaporedja vozlov in kaj storiti, če je dan verižni seznam kot objekt razreda `VerSez`. Osnovna ugotovitev je, da če želimo znotraj zaporedja odstraniti vozel, potrebujemo kazalec na njegovega predhodnika (in na njegovega naslednika, a če poznamo predhodnika, je pot do naslednika enostavna). Nato predhodniku kot njegovega naslednika nastavimo naslednika vozla, ki ga želimo odstraniti in na ta način odstranimo določen vozel iz zaporedja oz. verižnega seznama.

- **Zaporedje vozlov**

Vozel bomo odstranili iz verižnega seznama, podanega kot neko zaporedje vozlov s kazalcem na prvi vozel. Odstranili ga bomo nekje na sredini zaporedja na podlagi nekega pogoja (npr. odstrani vozel z vrednostjo 3).

Razmisliti moramo, kako bomo ravnali v primeru, če je v zaporedju samo en vozel, ali če je zaporedje prazno. Če je zaporedje prazno, vrnemo kar prazno zaporedje. Če ima zaporedje samo en element in je ta element ravno podatek, ki ga hočemo odstraniti, potem s kazalcem `prvi` pokažemo na naslednika tega elementa, se pravi, da `prvi` dobi vrednost `null`. S tem bomo zajeli tudi primer, ko želimo odstraniti prvi vozel iz zaporedja in to ni edini vozel v zaporedju. Takrat moramo s kazalcem `prvi` pokazati na naslednika vozla `prvi`. Potrebni ukrepi, kadar moramo pobrisati prvi vozel v verižnem seznamu, so torej enaki, če je ta vozel edini v verižnem seznamu, ali pa če imamo v verižnem seznamu več vozlov.

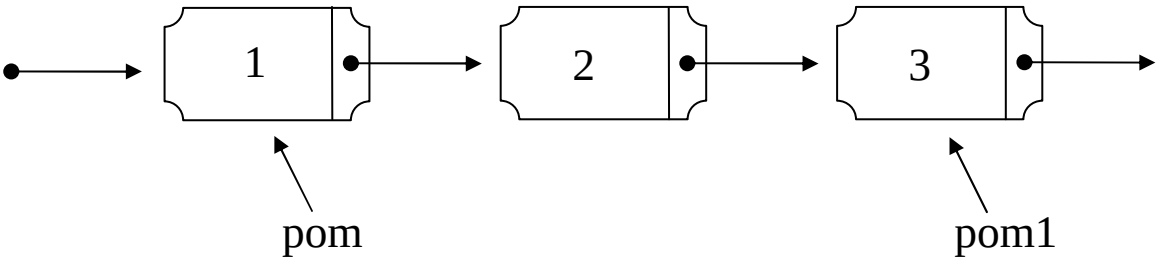
Sedaj si bomo ogledali, kako bi izbrisali element s sredine zaporedja. V tem primeru se ostali del zaporedja ne bo spremenil, zato bomo slikovno prikazali le del, kjer bomo vozel odstranili.



SLIKA 72: Del zaporedja vozlov iz katerega bomo odstranili vozel

Če želimo odstraniti vozel z vrednostjo 2, bomo potrebovali dva pomožna kazalca. Prvi, recimo mu `pom`, bo kazal na predhodnika vozla, ki ga želimo odstraniti. Drugi, `pom1`, bo kazal na naslednika vozla, ki ga želimo odstraniti. Torej, če želimo odstraniti vozel z vrednostjo 2, moramo imeti kazalec na vozel z vrednostjo 1, prav tako pa kazalec na vozel z vrednostjo 3. Privzemimo, da pomožna kazalca na ustrezna vozla (predhodnika in naslednika) že imamo.

Ustrezna slika bo:

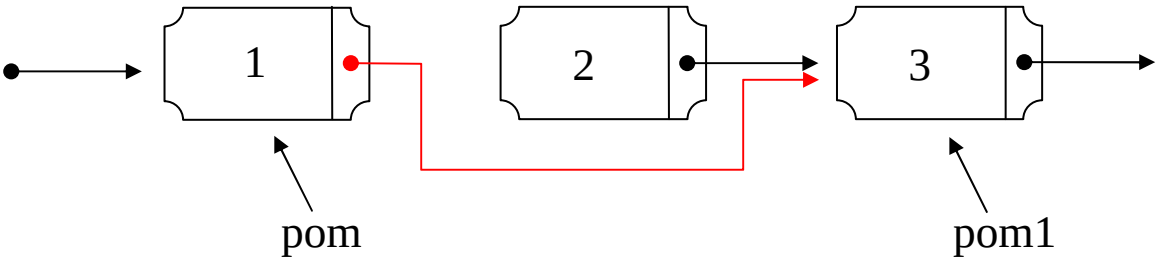


SLIKA 73: Del zaporedja vozlov z ustreznima kazalcema

Odstranimo sedaj vozel z vrednostjo 2 iz našega zaporedja. Kazalcu `pom` nastavimo za naslednika vozel `pom1`. Uporabimo metodo `nastaviNasled(Vozel v)` iz razreda `Vozel`. Ustrezen klic metode bo:

```
pom.nastaviNasled(pom1);
```

Novo stanje znotraj našega zaporedja bo:



SLIKA 74: Vozel `pom` ima novega naslednika

Z rdečo barvo je prikazan spremenjeni kazalec. Seveda moramo sedaj razmisliti, kako nastaviti pomožna kazalca. To bomo opisali kar spotoma, ko bomo opisovali, kako gradimo metodo.

Naredili bomo metodo, ki bo iz zaporedja odstranila vozel z vrednostjo `x`. Metoda bo kot argumenta sprejela parameter tipa `Vozel` (kazalec na prvi vozel), ter vrednost `x` (podatek vozla, ki ga bomo odstranili). Metoda se bo imenovala `brisi1(Vozel prvi, int x)`.

V primeru, ko je zaporedje vozlov prazno, vrnemo kazalec, ki kaže na prazno zaporedje. Če je v zaporedju več vozlov s podatkom z vrednostjo `x`, iz zaporedja odstranimo prvi tak vozel. V primeru, ko je v našem zaporedju le en vozel (in ima podatek z vrednostjo `x`) ali pa je prvi vozel tisti z vrednostjo `x`, s kazalcem `prvi`, ki ga dobimo kot parameter metode, pokažemo na njegovega naslednika. V pogoju stavka `if` preverimo, ali je vrednost prvega vozla enaka `x` in s

kazalcem `prvi` pokažemo na naslednika prvega vozla. Če v zaporedju vozla s podatkom z vrednostjo `x` ni, potem vrnemo kazalec na prvi element v zaporedju (podan kot argument metode). Metoda bo delovala na naslednji način. Najprej bomo s pomočjo stavka `if` preverili, ali je zaporedje prazno. Če je prazno, bomo vrnili kar kazalec na prazno zaporedje in končali z metodo.

Če zaporedje ni prazno (v zaporedju imamo vozle), bomo s kazalcema `pom` ter `pom1` pokazali na prvi vozel ter na njegovega naslednika. Nato bomo pogledali, če je vrednost prvega vozla enaka `x`. Če je to res, moramo prestaviti kazalec `prvi`. Z njim pokažemo na njegovega naslednika in se s tem prvega vozla znebimo. S tem smo tudi končali metodo in vrnemo vrednost prestavljenega kazalca `prvi`.

Če torej v seznamu iskani podatek ni v prvem vozlu, z uporabo zanke `while` pregledujemo zaporedje vozlov, dokler vozla z vrednostjo `x` ne najdemo (oziroma dokler ne pridemo do konca seznama). Ustavili se bomo, ko bo na ta vozel kazal kazalec `pom1`. Takrat bo `pom` kazal na predhodnika vozla s podatkom `x`. Ko ga najdemo, s kazalcem `pom1` pokažemo na njegovega naslednika in vozlu `pom` s pomočjo metode `nastaviNasled(Vozel v)` nastavimo vozel `pom1` za naslednika. Vozel z vrednostjo `x` smo tako iz našega zaporedja odstranili.

```
public static Vozel brisi_na_sredini1(Vozel prvi, int x){
   //iz verige vozlov z začetkom prvi zbriši prvi vozel s podatkom x
   if(prvi == null){ //zaporedje je prazno
      return prvi;
   }
   Vozel pom = prvi;
   Vozel pom1 = pom.vrniNasled();
   if(pom.vrniPodatek() == x){ //odstranimo prvi vozel
      prvi = prvi.vrniNasled(); //pokažemo na drugi vozel
      return prvi;
   }
   while(pom1 != null){ //pregledujemo vrednosti vozlov
      if(pom1.vrniPodatek() == x){ //našli smo vozel s podatkom x
         pom1 = pom1.vrniNasled(); //potrebujemo naslednika
         pom.nastaviNasled(pom1); //iskani vozel odstranimo
         return prvi;
      }
      //s kazalcem pom in pom1 pokažemo na njuna naslednika
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   return prvi; //če slučajno podatka x v verigi ni
}
```

Sedaj pa si oglejmo primer, ko bo metoda kot argument sprejela parameter tipa `VerSez` (kazalec na objekt tipa `VerSez`) ter vrednost `x` (podatek vozla, ki ga bomo odstranili). Primer bo zelo podoben prvemu. Metoda bo kot argument dobila podan verižni seznam kot objekt, in ne zaporedje vozlov s kazalcem na prvi element. V metodi `dodaj_na_konec4(Vozel pomozni,VerSez verSe1)` smo že opisali, kako s pomožnima kazalcema pokažemo na ustrezna vozla (z uporabo metode `prvi()` iz razreda `VerSez()`). Metoda bo delovala po istem principu kot metoda `brisi_na_sredini1(Vozel prvi, int x)`. A ker sedaj kot argument ne dobimo kazalca na prvi vozel, temveč kazalec na objekt tipa `VerSez`, v metodi `brisi_na_sredini1(Vozel prvi, int x)` dele, kjer uporabimo kazalec `prvi`, ustrezno popravimo.

S kazalcem `pom` s klicem metode `prvi()` nad objektom `verSe1` pokažemo na prvi vozel v verižnem seznamu. S kazalcem `pom1` pokažemo na naslednika vozla `pom`.

Razred `VerSez()` metodo, ki odstrani prvi vozel, že ima. Ta sama poskrbi, da se reference ustrezno popravijo. Seveda jo bomo uporabili. Metodo `odstani_prvega()` bomo poklicali nad objektom `verSe1`. Metoda bo za razliko od prejšnje tipa `void`, torej ne bo vračala ničesar.

Metoda bo torej:

```
public static void brisi_na_sredini2(VerSez verSe1, int x){
   Vozel pom = verSe1.prvi(); //začetek verige
   if(pom == null){
      return; //ni kaj narediti, končamo z metodo
   }
   Vozel pom1 = pom.vrniNasled();
   if(pom.vrniPodatek() == x){
      verSe1.odstrani_prvega(); //popravimo verižni seznam
      return;
   }
   while(pom1 != null){
      if(pom1.vrniPodatek() == x){
         pom1 = pom1.vrniNasled();
         pom.nastaviNasled(pom1);
         return; //verižni seznam je popravljen, končamo
      }
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   return; //če x ni bil v VS
}
```

1.6.2.3 Brisanje na koncu

- **S pomočjo razreda Vozel**

Iz zaporedja vozlov bomo odstranili zadnji vozel. Razmislimo, kako bi to storili in uporabimo dosedanje znanje. Najprej odpravimo »robna« primera. Če je prvotno zaporedje prazno (`prvi` ima vrednost `null`), vrnemo kar `null`. Isto vrednost vrnemo, če je v zaporedju le en element, saj bomo odstranili edini vozel in dobili prazno zaporedje.

V vseh drugih primerih pa odstraniti zadnji vozel iz zaporedja pomeni, da moramo predzadnjemu vozlu referenco nastaviti na vrednost `null`. Za to bomo potrebovali kazalec na predzadnji element v zaporedju. Na tem vozlu bomo uporabili metodo `nastaviNasled(Vozel v)` z argumentom metode `null`. Kako bomo s kazalcem pokazali na predzadnji vozel? Uporabili bomo dva pomožna kazalca ter seveda zanko `while`, s pomočjo katere se bomo premikali po zaporedju. Pogoj zanke bo, tako kot pri večini dosedanjih metod, imel vrednost true, dokler s kazalcem ne bomo pokazali na naslednika zadnjega vozla, torej na `null`. Začeli bomo s tem, da bomo s kazalecem `pom` pokazali na prvi vozel, s `pom1` pa na njegovega naslednika in se vrteli v zanki, dokler bo `pom1` različen od `null`. Znotraj zanke `while` bomo v vsakem koraku kazalca `pom` in `pom1` prestavili na naslednika vozlov, na katera kažeta.

Ko se bo zanka `while` izvedla zadnjič, bo kazalec `pom` kazal na predzadnji vozel, kazalec `pom1` pa na zadnji vozel. Takrat moramo kazalcu `pom` za naslednika nastaviti vrednost `null`. Ne smemo pa pozabiti vrniti kazalec na prvi element (čeprav bo enak kot je argument, s katerim smo poklicali metodo), saj metoda vrača kazalec. Metoda mora vračati kazalec, saj moramo v primeru, da je v verigi le en vozel javiti, da je kazalec na prvi vozel verige spremenjen.

Metoda:

```
public static Vozel brisi_na_koncu1(Vozel prvi){
   if(prvi == null){ //zaporedje je prazno
      return null;
   }
   if(prvi.vrniNasled() == null){ //zaporedje ima samo en element
      return null;
   }
   Vozel pom = prvi;
   Vozel pom1 = pom.vrniNasled();
   while(pom1.vrniNasled() != null){ //s pom1 iščemo zadnji element
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   //pom1 kaže na zadnji, pom pa na predzadnji element
   pom.nastaviNasled(null); //odstranimo zadnji vozel
   return prvi;
}
```

- **S pomočjo razreda VerSez**

Ideja metode se od zgornje metode `brisi_na_koncu1(Vozel prvi)` ne bo veliko razlikovala. Kot argument bo sprejela objekt `VerSez` ter ga spremenila. Metoda ne bo vračala ničesar.

Oglejmo si razliko med to metodo in metodo, kjer kot argument sprejmemo parameter tipa `Vozel`. Najprej bomo preverili, ali je verižni seznam prazen (stavek `if`). V tem primeru nam ni potrebno storiti ničesar, le zaključimo metodo.

Če pa seznam ni prazen, bomo s kazalcema `pom` in `pom1` pokazali na ustrezna vozla. Kazalec na prvi vozel bomo dobili z uporabo metode `prvi()` iz razreda `VerSez`. To bo kazalec `pom`. S kazalcem `pom1` bomo pokazali na njegovega naslednika. Nato bomo preverili, ali je v seznamu samo en element. Če je, ga bomo odstranili s klicem metode `odstrani_prvega()` nad verižnim seznamom, ki smo ga podali kot argument metode in zaključili. Drugače pa bomo poiskali predzadnji vozel. Zanka `while`, v kateri bomo s kazalcema pokazali na zadnji in predzadnji vozel, bo zelo podobna tisti iz metode `brisi_na_koncu1(Vozel prvi)`.

Metoda:

```
public static void brisi_na_koncu2(VerSez verSe1){
   if(verSe1.prazen()){ //seznam je prazen
      return; //končamo
   }
   Vozel pom = verSe1.prvi();
   Vozel pom1 = pom.vrniNasled();
   if(pom.vrniNasled == null){ //v seznamu je samo en element
      verSe1.odstrani_prvega();
      return; //končamo
   }
   while(pom1.vrniNasled() != null){ //s pom1 iščemo zadnji element
      pom = pom1;
      pom1 = pom1.vrniNasled();
   }
   pom.nastaviNasled(null); //odstranimo zadnji vozel
}
```

1.6.3 Iskanje elementa

S to metodo poiščemo, če v verižnem seznamu hranimo določeno vrednost. Ko smo razlagali metode za vstavljanje in brisanje elementov iz zaporedja ali verižnega seznama, smo že iskali elemente z določeno vrednostjo. Spoznali smo, kako se z zanko `while` premikamo znotraj zaporedja. Zato ne bomo ponovno razlagali, kako poiščemo element znotraj zaporedja. Ob morebitnih nejasnostih si le še enkrat ogledamo vse metode, ki smo jih opisali v razdelkih o vstavljanju na sredino ter brisanju na sredini.

1.6.4 Izpis

Izpišimo vrednosti v verižnem seznamu oz. zaporedju. Postopek je enostaven, saj smo več ali manj celotni postopek že naredili. S pomočjo pomožnega kazalca `pom` se bomo premikali po zaporedju in izpisali podatek trenutnega vozla. Uporabili bomo zanko `while`, ki se bo izvajala, dokler s `pom` ne bomo pregledali vseh vozlov v zaporedju. Podatek bomo izpisali s klicem metode `println`:

```
System.out.println(pom.vrniPodatek());
```

Metoda ne bo vračala nobene vrednosti in bo tipa `void`. Kot argument bo sprejela objekt tipa `Vozel`, ki bo kazalec na prvi vozel v zaporedju.

Metoda:

```
public static void izpisi1(Vozel prvi){
   Vozel pom = prvi;
   while(pom != null){ //pregledamo vse vozle v verigi
      System.out.println(pom.vrniPodatek());
      pom = pom.vrniNasled(); //naslednji vozel verige
   }
}
```

Metode, ki kot argument sprejme verižni seznam (kazalec na objekt tipa `VerSez`), ne bomo zapisali, saj razlik ni veliko. Takoj na začetku bi s pomočjo metode `prvi()` iz razreda `VerSez` nastavili `pom` tako, da bi kazal na prvi vozel. Ostali del metode bi bil enak kot v metodi `izpisi1(Vozel prvi)`.

Napišimo še metodo, ki bo poleg podatka izpisala tudi zaporedno številko vozla. V primeru, da imamo zaporedje s podatki 10, 12, 31 in 14, bi radi ob klicu metode dobili naslednji izpis:

```
Podatek 1. vozla je: 10
Podatek 2. vozla je: 12
Podatek 3. vozla je: 31
Podatek 4. vozla je: 14
```

Potrebovali bomo pomožno spremeljivko tipa `int`. Njena začetna vrednost je 1 (prvi vozel) in se znotraj zanke `while` z vsakim korakom poveča za ena.

Napišimo metodo:

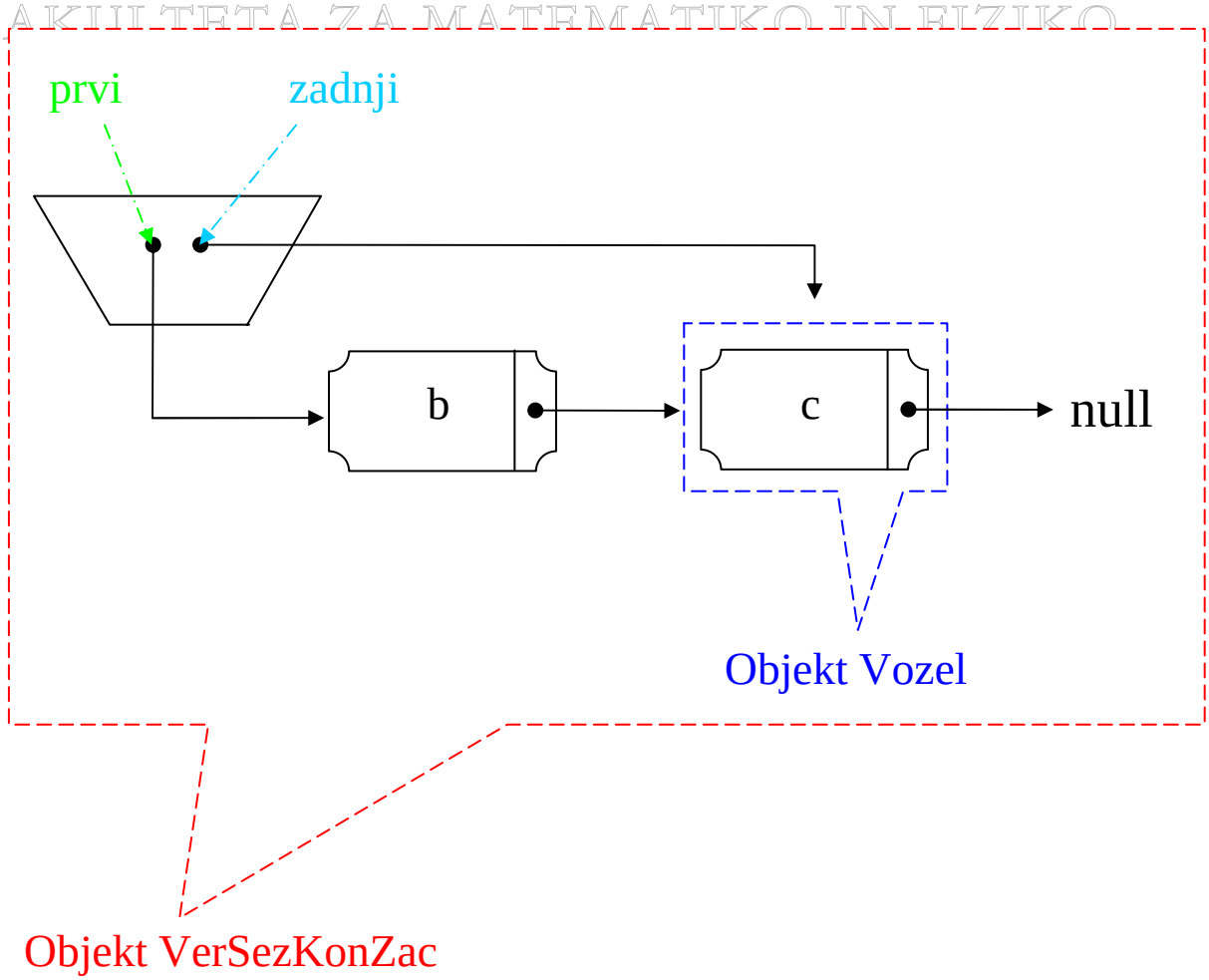
```
public static void izpisi2(Vozel prvi){
   Vozel pom = prvi;
   int x = 1;
   while(pom != null){
      System.out.println("Podatek " + x + ". vozla je: " +
      pom.vrniPodatek());
      pom = pom.vrniNasled();
      x++;
   }
}
```

1.7 ENOJNO POVEZAN VERIŽNI SEZNAM S KAZALCEM NA ZAČETEK IN KONEC

Razred `VerSez`, ki omogoča delo z verižnim seznamom, smo spoznali v razdelkih 1.5 in 1.6. Ker smo v objektih tega razreda vodili kazalec na prvi element v verigi vozlov, so bile operacije na začetku verige hitre in enostavne. Če smo želeli na primer dodati nov vozel na konec verige, smo se morali najprej sprehoditi preko vseh vozlov v verigi. Večkrat pa bi nam možnost, da bi tudi na koncu verige hitro vstavljali nove vozle, prišla še kako prav (na primer, če bi želeli z verigo vozlov predstaviti neko vrsto). Zato bomo na osnovi razreda `VerSez` naredili nov razred, ki bo omogočal delo z verižnim seznamom s kazalcema na prvi in zadnji vozel v verigi. Na ta način bo dodajanje novega vozla na konec enako enostavno, kot dodajanje vozla na začetek. V prej opisanem razredu imamo kazalec le na prvi element. Edina komponenta objekta tega tipa je tipa `Vozel` in je referenca na prvi element v verigi vozlov. Nov razred `VerSezKonZac` pa bo vseboval dve komponenti tipa `Vozel`. Prva bo referenca na prvi element v verigi, druga pa na zadnji element.

KOMPONENTI:

```
private Vozel prvi; //referenca na prvi element verižnega seznama
private Vozel zadnji; //referenca na zadnji element verižnega seznama
```



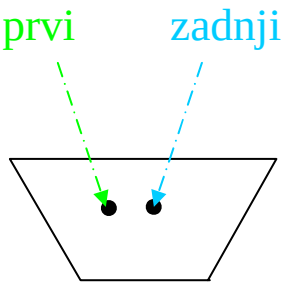
SLIKA 75: Prikaz verižnega seznama s kazalcema na začetek in konec

V objektu tipa `VerSezKonZac` bomo hranili kazalca na začetek in konec verižnega seznama. Tudi tu, kot pri verižnem seznamu tipa `VerSez`, bomo preko kazalca na prvi element prišli do vseh ostalih elementov verižnega seznama.

1.7.1 Konstruktor

- `public VerSezKonZac()`: naredi prazen verižni seznam

Ustvari prazen verižni seznam.



SLIKA 76: Prazen verižni seznam s kazalcem na začetek in konec

Na sliki 76 vidimo, kako prikažemo prazen verižni seznam s kazalcem na začetek in konec. Ker je prazen, nima nobenega elementa (nobenega vozla). Komponenti `prvi` in `zadnji` kažeta v prazno oz. imata vrednost `null`.

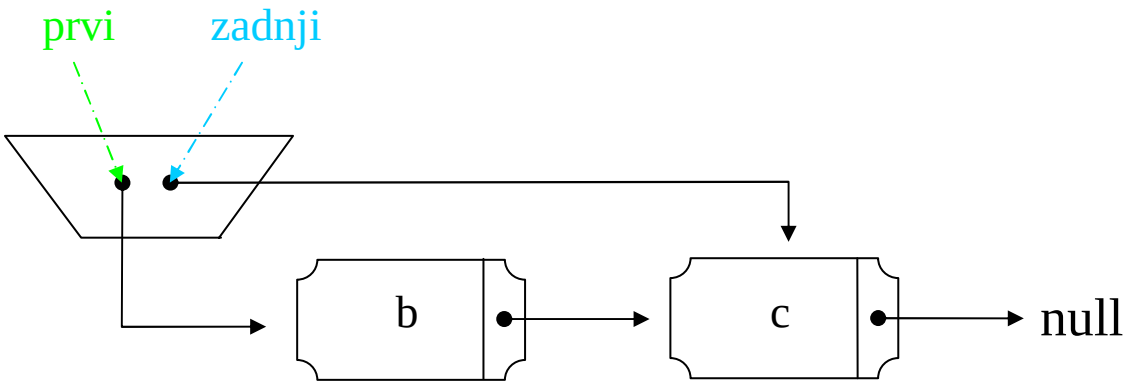
1.7.2 Metode

- `vstavi_prvega(Vozel v)`: vstavi vozel `v` na začetek verižnega seznama tipa `VerSezKonZac`

Vozel `v` že obstaja. Vstavimo ga na začetek obstoječega verižnega seznama. Kazalec `prvi` pokaže na ta novi vozel `v`. V primeru, ko je verižni seznam prazen, moramo na novo nastaviti tako kazalec `prvi` kot tudi kazalec `zadnji` (oba pokažeta na vozel `v`).

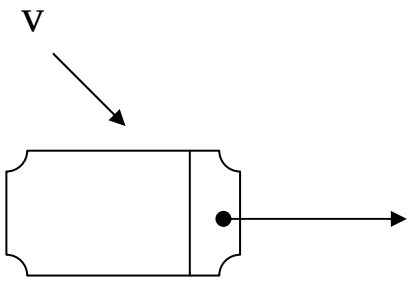
```
public void vstavi_prvega(Vozel v){
   //v vstavimo na začetek seznama
   if(this.prazen()){//če je verižni seznam prazen
                     //prvi in zadnji pokažeta na v
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      v.nastaviNasled(this.prvi); //naslednik v je prejšnji prvi
      this.prvi = v; //novi prvi element
   }
}
```

Oglejmo si po korakih, kaj metoda naredi na seznamu s slike:



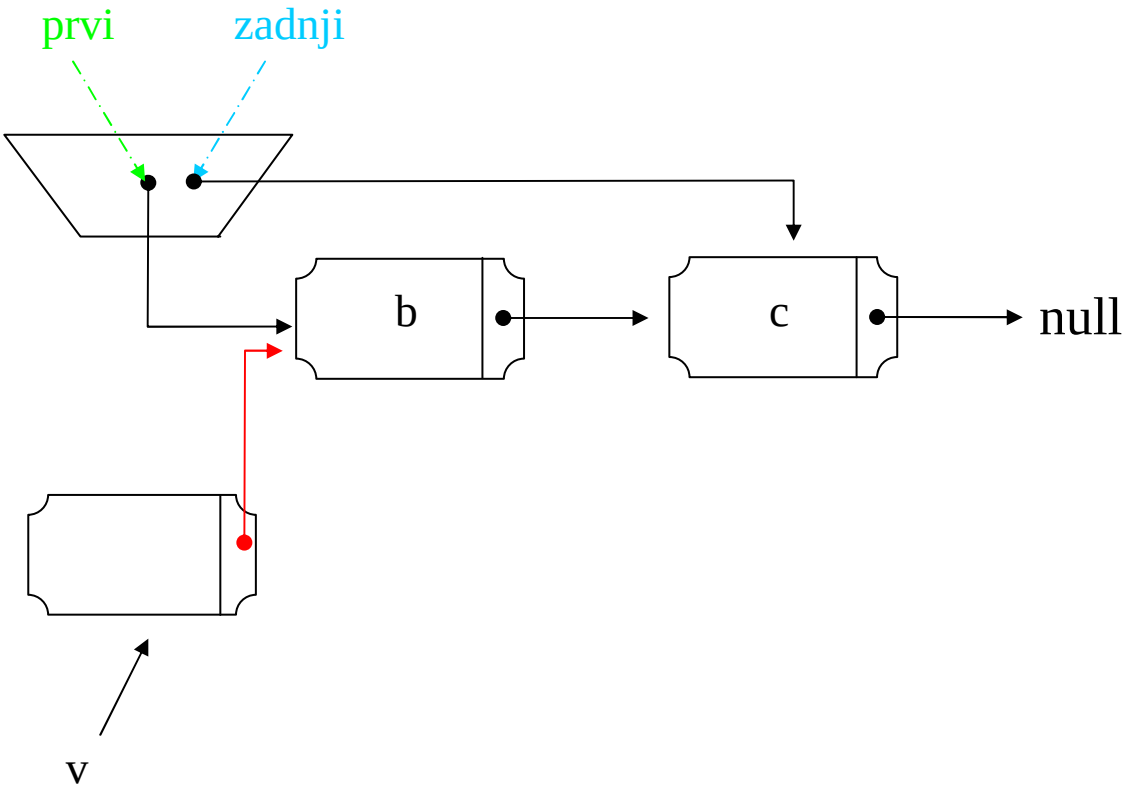
SLIKA 77: Verižni seznam za prikaz delovanja metode

Imamo tudi vozel `v`:



SLIKA 78: Vozel `v`

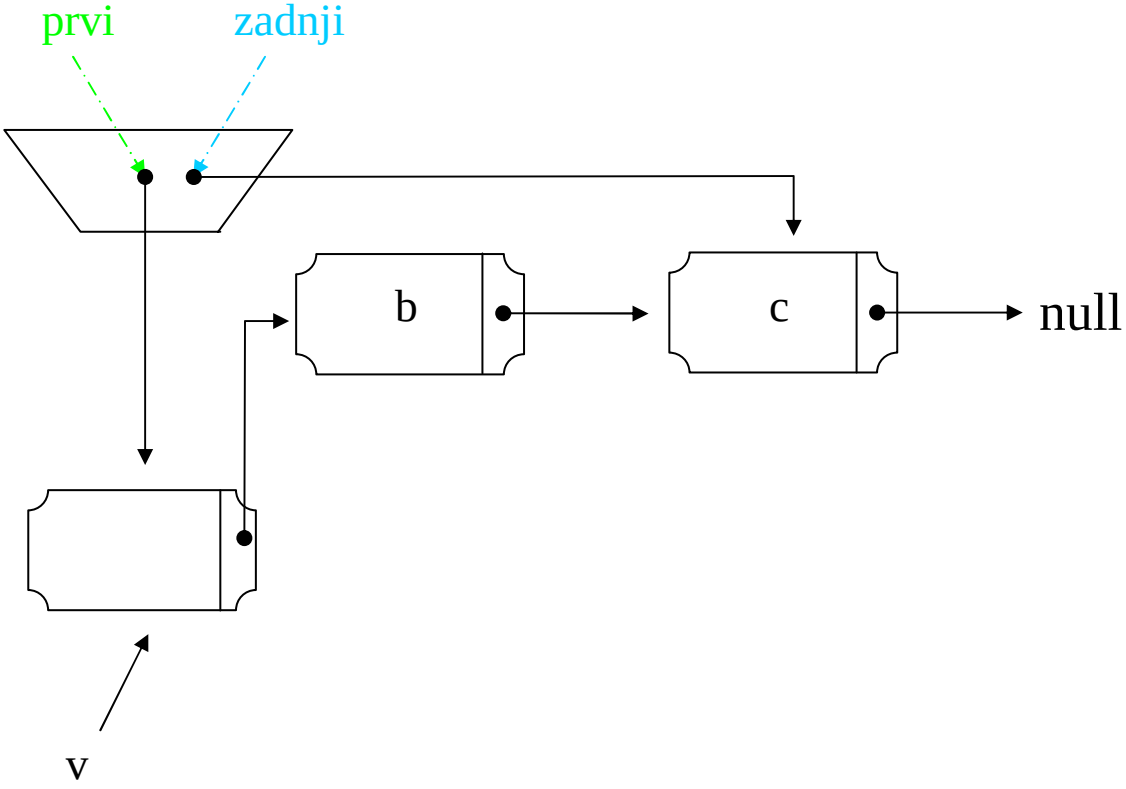
Verižni seznam ni prazen. Zato izvedemo stavke v delu `else`. Tu se najprej izvede metoda `v.nastaviNasled(this.prvi)`. Vozlu `v` nastavimo naslednika na tisti vozel, na katerega kaže kazalec `prvi`. V našem primeru kaže na vozel z vrednostjo b. Začetek verižnega seznama še vedno kaže na vozel z vrednostjo b.



SLIKA 79: Vozlu `v` nastavimo naslednika

Vozel `v` sedaj kaže na vozel z vrednostjo b, prav tako tja kaže tudi začetek verižnega seznama. Sedaj moramo še začetek verižnega seznama preusmeriti na pravi vozel. Želimo, da je novi začetek verižnega seznama vozel `v`. To dosežemo z ukazom:

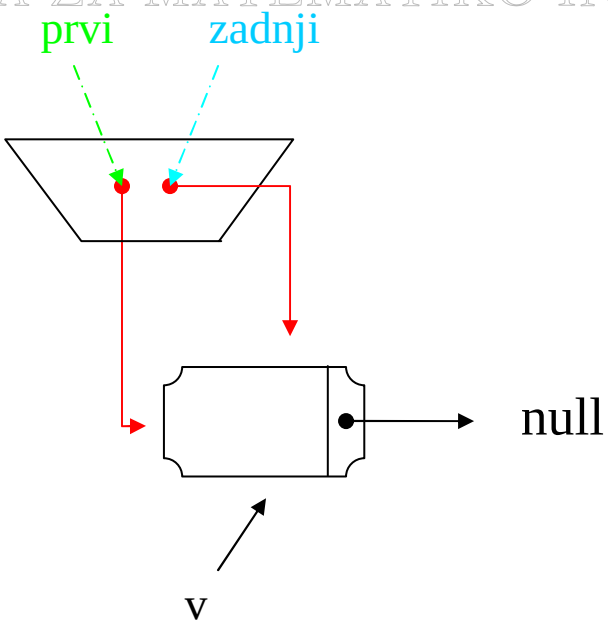
```
this.prvi = v;
```



SLIKA 80: Verižni seznam po klicu metode `vstavi_prvega(Vozel v)`

Verižnemu seznamu s slike 77 smo na začetek dodali vozel `v`.

Oglejmo si še kaj naredi metoda, če je verižni seznam prazen. Imamo torej prazen verižni seznam (slika 76). Zato nam `this.prazen()` vrne `true` in izvedejo se stavki v telesu stavka `if`. Na vozel `v` preusmerimo tako referenco `prvi` kot referenco `zadnji`. To je sedaj edini element verižnega seznama in nanj morata kazati oba referenci. Nov verižni seznam je prikazan na spodnji sliki.



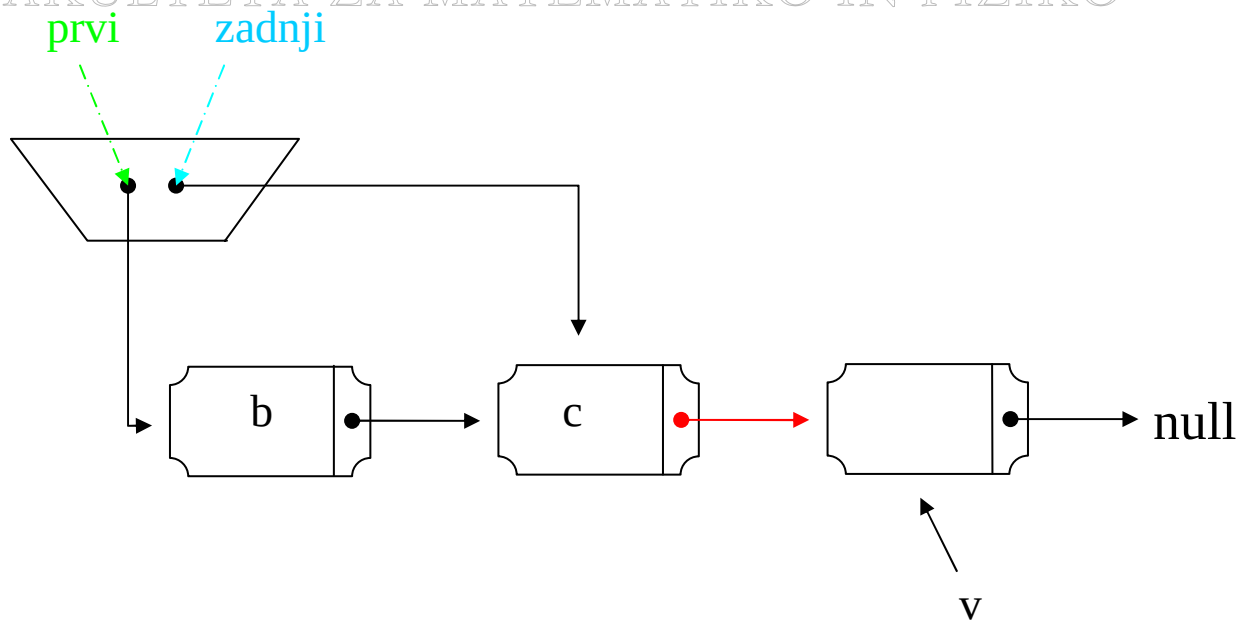
SLIKA 81: Prej prazen verižni seznam po klicu metode `vstavi_prvega(Vozel v)`

- `vstavi_zadnjega(Vozel v)`: vstavi vozel `v` na konec verižnega seznama tipa `VerSezKonZac`

Vozel `v` vstavimo na konec verižnega seznama. Referenca `zadnji` pokaže na vozel `v`. Če je verižni seznam prazen, bo ta zadnji vozel hkrati tudi prvi. Zato takrat na vozel `v` usmerimo obe referenci (`prvi` in `zadnji`).

```
public void vstavi_zadnjega(Vozel v){
   //vozel v vstavinmo kot zadnji vozel v verižni seznam
   if(this.prazen()){//če je verižni seznam prazen,
                     //prvi in zadnji pokažeta na v
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      this.zadnji.nastaviNasled(v);
      this.zadnji = v; // novi zadnji element
   }
}
```

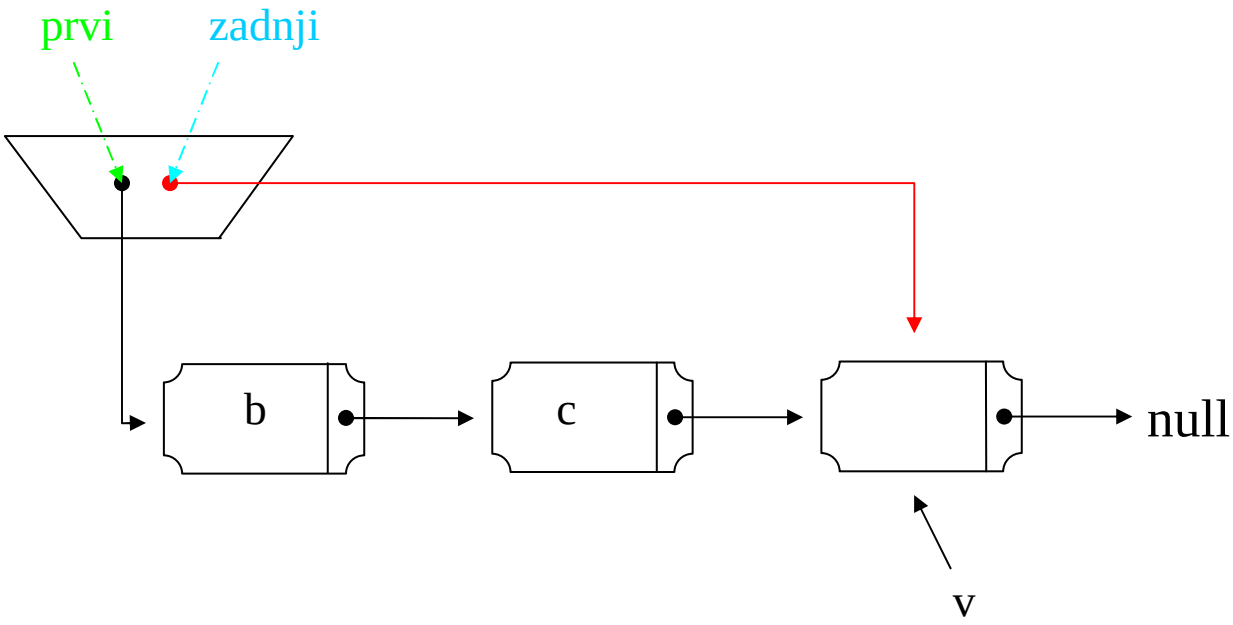
Po korakih si oglejmo, kaj metoda naredi na seznamu s slike 77. Imamo tudi vozel v s slike 78. Najprej se izvede `this.zadnji.nastaviNasled(v)`. V našem primeru kazalec `zadnji` kaže na vozel z vrednostjo c. Temu vozlu nastavimo naslednika na vozel `v` . Referenca `zadnji` še vedno kaže na vozel z vredostjo c.



SLIKA 82: Vozlu z vrednostjo c nastavimo naslednika na vozel `v`

Referenca vozla z vrednostjo c sedaj nima več vrednosti `null`, temveč ima ta vozel naslednika in sicer vozel `v`. S kazalcem `zadnji` moramo še pokazati na pravi vozel oz. določiti nov zadnji vozel. Želimo, da je novi konec verižnega seznama vozel `v`. To dosežemo z ukazom:

```
this.zadnji = v;
```



SLIKA 83: Verižni seznam po klicu metode `vstavi_zadnjega(Vozel v)`

Ob klicu metode `vstavi_zadnjega(Vozel v)` nad praznim seznamom, dobimo stanje kot je na sliki 81.

- `vstavi_prvega(int x)`: vstavi vozel na začetek verižnega seznama tipa `VerSezKonZac`

Metoda naredi nov objekt `Vozel` z vrednostjo `x`. Nov vozel postavi na začetek verižnega seznama. V primeru, ko je verižni seznam prazen, moramo referenci `prvi` in `zadnji` prestaviti na vozel v.

```
public void vstavi_prvega(int x){
   //naredimo nov objekt in ga postavimo na začetek
   Vozel v = new Vozel(x);
   if(this.prazen()){ //če je seznam prazen, usmerimo
                      //prvi in zadnji na vozel v
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      v.nastaviNasled(this.prvi);
      this.prvi = v; // novi prvi element
   }
}
```

Seveda pa lahko uporabimo tudi prej napisano metodo:

```
public void vstavi_prvega(int x){
   //naredimo nov objekt in ga postavimo na začetek
   Vozel v = new Vozel(x);
   this.vstavi_prvega(v);
}
```

Metoda naredi nov vozel v z vrednostjo x in ga vstavi na začetek verižnega seznama. V primeru, da metodo pokličemo nad seznamom s slike 77, dobimo verižni seznam kot je na sliki 80. Če je seznam prazen, pa dobimo stanje kot je na sliki 81.

- `vstavi_zadnjega(int x)`: vstavi vozel na konec verižnega seznama tipa `VerSezKonZac`

Metoda naredi nov objekt `Vozel` z vrednostjo `x`. Nov vozel postavi na konec verižnega seznama. Tudi tu moramo v primeru, da je verižni seznam prazen, s kazalcema `prvi` in `zadnji` pokazati na ta novi vozel.

```
public void vstavi_zadnjega(int x){
   //v seznam kot zadnjega dodamo nov vozel s podatkom x
   Vozel v = new Vozel(x);
   if(this.prazen()){//če je verižni seznam prazen,
                     //spremenimo referenci prvi in zadnji
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      this.zadnji.nastaviNasled(v);
      this.zadnji = v; // novi zadnji element
   }
}
```

Seveda pa lahko uporabimo prej napisano metodo:

```
public void vstavi_zadnjega(int x){
   //naredimo nov objekt in ga postavimo na konec
   Vozel v = new Vozel(x);
   this.vstavi_zadnjega(v);
}
```

Klic metode nad verižnim seznamom tipa `VerSezKonZac` s slike 77 nam vrne verižni seznam s slike 83 (vozel `v` ima vrednost x). Klic metode nad praznim verižnim seznamom pa seznam s slike 81 (vozel `v` ima vrednost x).

- `odstrani_prvega()`: odstrani prvi element verižnega seznama tipa `VerSezKonZac`

Elementa fizično ne odstranimo, le kazalec `prvi` preusmerimo na drugi element (na naslednika prvega vozla) v verižnem seznamu. Če metodo uporabimo na praznem verižnem seznamu, se seznam ne spremeni.
V primeru, ko ima verižni seznam samo en element, moramo na novo nastaviti tako referenco `prvi` kot tudi referenco `zadnji` (obema vrednost nastavimo na `null`).

```
public void odstrani_prvega(){
   //odstranimo prvi vozel iz seznama
   //če ima verižni seznam en element, referenci prvi in zadnji
   //nastavimo na null
   //če je seznam prazen, se ne spremeni
   if(this.prazen()){ //seznam je prazen
      return; //končamo metodo brez sprememb
   }
   if(this.prvi.vrniNasled() == null){//seznam vsebuje samo en vozel
      this.prvi = null;
      this.zadnji = null;
   }
   else{
      this.prvi = this.prvi.vrniNasled(); //novi prvi element
   }
}
```

- odstrani_zadnjega(): odstrani zadnji element iz verižnega seznama tipa `VerSezKonZac`

Tudi tu elementa fizično ne odstranimo (zanj bo poskrbel javanski smetar), le kazalec `zadnji` preusmerimo na predzadnji element v seznamu (na predhodnika zadnjega elementa).

Vozel odstranimo tako, da referenco predzadnjega vozla postavimo na `null`, ter da s kazalcem `zadnji` pokažemo na ta predzadnji vozel. Vendar kazalca na predzadnji vozel nimamo. Tudi vozel `zadnji` ne ve, kdo je njegov predhodnik, saj govorimo o enojno povezanih verižnih seznamih. Zato bomo uporabili zanko `while` in se z njeno pomočjo ter s pomožno spremenljivko `pom` pomikali po seznamu do predzadnjega elementa.

Zanka `while` se bo izvajala, dokler naslednik vozla `pom` ne bo vozel `zadnji`. Do takrat se s kazalcem `pom` premikamo za en vozel naprej. Takrat, ko bo naslednik vozla `pom` vozel `zadnji`, kazalec `pom` kaže na predzadnji vozel. Nato še referenco vozla `pom` postavimo na vrednost `null`, ter kazalec `zadnji` usmerimo na vozel `pom`.

V primeru, ko podamo prazen verižni seznam, se verižni seznam ne spremeni. Če ima verižni seznam samo en element, moramo na novo nastaviti tako referenco `prvi`, kot tudi referenco `zadnji` (z obema pokažemo na `null`).

```
public void odstrani_zadnjega(){
   //če je seznam prazen, se seznam ne spremeni
   //če ima seznam en vozel, referenci prvi in zadnji nastavimo
   //na null
   //zadnji pokaže na predhodnika zadnjega vozla
   if(this.prazen()){ //seznam je prazen
      return; //končamo metodo brez sprememb
   }
   if(this.prvi.vrniNasled() == null){ //seznam ima samo en vozel
      this.prvi = null;
      this.zadnji = null;
   }
   else{
      Vozel pom = this.prvi;
      //poiščemo predzadnjega
      while(pom.vrniNasled() != this.zadnji){
         pom = pom.vrniNasled();
      }
      pom.nastaviNasled(null); //izločimo zadnjega
      this.zadnji = pom; //novi zadnji
   }
}
```

- `prvi()`: vrne kazalec na prvi vozel v verižnem seznamu tipa `VerSezKonZac`

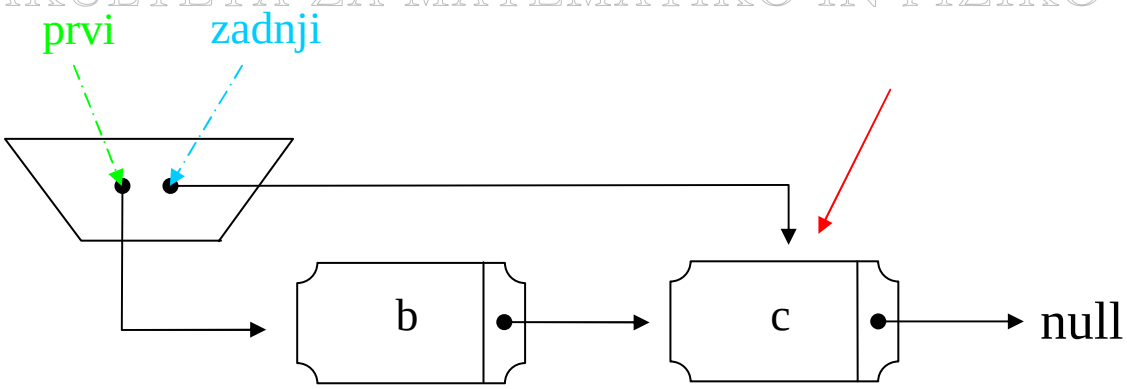
Metoda deluje tako kot metoda `prvi()` iz razreda `VerSez`. Metoda vrne objekt na katerega kaže referenca `prvi`. Če je verižni seznam prazen, metoda vrne vrednost `null`.

- `zadnji()`: vrne kazalec na zadnji vozel

Metoda vrne refernco na objekt, na katerega kaže referenca `zadnji`.

```
public Vozel zadnji(){//vrne zadnji vozel
   return this.zadnji;
}
```

Če pokličemo metodo nad verižnim seznamom s slike 77, nam vrne kazalec na vozel z vrednostjo c. Na spodnji sliki bomo z rdečo puščico prikazali, kaj nam metoda vrne.



SLIKA 84: Rdeča puščica označuje, kaj vrne metoda `zadnji()`

Če je verižni seznam prazen, metoda vrne vrednost `null`.

- `vrednost_prvega()`: vrne podatek iz objekta, na katerega kaže referenca `prvi`

Metoda vrne podatek iz objekta, na katerega kaže referenca `prvi`. Če je verižni seznam tipa `VerSezKonZac` prazen, metoda vrne največje celo število, s katerim programski jezik java še zna računati.

```
public int vrednost_prvega(){
   //vrnemo podatek iz objekta, na katerega kaže prvi
   if(this.prazen()){
      return Integer.MAX_VALUE();
   }
   return prvi.vrniPodatek();
}
```

Če metodo pokličemo na objektu s slike 77, potem nam metoda vrne vrednost b.

- `vrednost_zadnjega()`: vrne podatek iz objekta, na katerega kaže referenca `zadnji`

Metoda vrne podatek iz objekta, na katerega kaže referenca `zadnji`. Če je verižni seznam prazen, metoda vrne največje celo število, s katerim programski jezik java še zna računati.

```
public int vrednost_zadnjega(){
   //vrnemo podatek iz objekta, na katerega kaže zadnji
   if(this.prazen()){
      return Integer.MAX_VALUE;
   }
   return this.zadnji.vrniPodatek();
}
```

Metodo pokličimo nad objektom s slike 77. Metoda vrne vrednost c.

- `prazen()`: metoda preveri, ali je verižni seznam prazen

Metoda preveri, če kazalec `prvi` kaže v prazno. Če je seznam prazen, vrne `true`, sicer `false`. V primeru, da je verižni seznam prazen, imata obe referenci (`prvi` in `zadnji`) vrednost `null`. Ne more se namreč zgoditi, da bi referenca `prvi` bila `null`, referenca `zadnji` pa ne. Zato je dovolj preveriti le vrednost reference `prvi`.

1.7.3 Razred VerSezKonZac

Zložimo konstruktorje in metode v razred VerSezKonZac.

```
public class VerSezKonZac {
   private Vozel prvi; //referenca na prvi element seznama
   private Vozel zadnji; //referenca za zadnji element seznama

   //konstruktorji

   public VerSezKonZac(){
      prvi = null; //prazen verižni seznam
      zadnji = null;
   }

   //metode

   public void vstavi_prvega(Vozel v){
      //v vstavimo na začetek seznama
      if(this.prazen()){//če je verižni seznam prazen,
                        //prvi in zadnji pokažeta na v
         this.prvi = v;
         this.zadnji = v;
      }
      else{
         v.nastaviNasled(this.prvi); //naslednik v
                                     //je prejšnji prvi
         this.prvi = v; //novi prvi element
      }
   }
```

```
public void vstavi_zadnjega(Vozel v){
   //referenca zadnjega vozla pokaže na vozel v
   //referenca zadnji pokaže na v
   if(this.prazen()){ //če je verižni seznam prazen
                      //prvi in zadnji pokažeta na v
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      this.zadnji.nastaviNasled(v);
      this.zadnji = v; //novi zadnji element
   }
}

public void vstavi_prvega(int x){
   //naredimo nov objekt in ga postavimo na začetek
    Vozel v = new Vozel(x);
   if(this.prazen()){ //če je seznam prazen, usmerimo
                      //prvi in zadnji na vozel v
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      this.prvi.nastaviNasled(v);
      this.prvi = v; //novi prvi element
   }
}

public void vstavi_zadnjega(int x){
   //zadnji pokaže na vozel z vrednostjo x
    Vozel v = new Vozel(x);
   if(this.prazen()){ //če je verižni seznam prazen
                      //spremenimo prvi in zadnji
      this.prvi = v;
      this.zadnji = v;
   }
   else{
      this.zadnji.nastaviNasled(v);
      this.zadnji = v; //novi zadnji element
   }
}
```

```
public void odstrani_prvega(){
   //odstranimo prvi vozel iz seznama
   //če ima verižni seznam en element, referenci prvi
   //in zadnji nastavimo na null
   //če je seznam prazen, se ne spremeni
   if(this.prazen()){ //seznam je prazen
      return; //končamo metodo brez sprememb
   }
   if(this.prvi.vrniNasled() == null){
   //seznam vsebuje samo en vozel
      this.prvi = null;
      this.zadnji = null;
   }
   else{
      this.prvi = this.prvi.vrniNasled(); //novi prvi element
   }

public void odstrani_zadnjega(){
   //če je seznam prazen, vrnemo prazen verižni seznam
   //če ima seznam en vozel, referenci prvi in
   //zadnji nastavimo na null
   //zadnji pokaže na predhodnika zadnjega vozla
   if(this.prazen()){ //seznam je prazen
      return; //končamo metodo brez sprememb
   }
   if(this.prvi.vrniNasled() == null){ //seznam ima samo en vozel
      this.prvi = null;
      this.zadnji = null;
   }
   else{
      Vozel pom = this.prvi;
      //poiščemo predzadnjega
      while(pom.vrniNasled() != this.zadnji){
      //naslednik pom še ni zadnji
         pom = pom.vrniNasled();
      }
      pom.nastaviNasled(null); //izločimo zadnjega
      this.zadnji = pom; //novi zadnji
   }
}
```

```
    public Vozel prvi(){ //vrne prvi vozel
       return this.prvi;
    }

    public Vozel zadnji(){ //vrne zadnji vozel
       return this.zadnji;
    }

    public int vrednost_prvega(){
       //vrnemo podatek iz objekta, na katerega kaže prvi
       if(this.prazen()){
          return Integer.MAX_VALUE();
       }
       else{
          return prvi.vrniPodatek();
       }
    }

    public int vrednost_zadnjega(){
       //vrnemo podatek iz objekta, na katerega kaže zadnji
       if(this.prazen()){
          return Integer.MAX_VALUE;
       }
       else{
          return this.zadnji.vrniPodatek();
       }
    }

    public boolean prazen(){
       return (prvi == null);
    }
}
```

1.8 VERIŽNI SEZNAM IN TABELA

Zanimiva je primerjava med verižnimi seznami in tabelami. Osnovna prednost verižnega seznama v primerjavi s tabelo je, da lahko število elementov verižnega seznama poljubno spreminjamo, ne ozirajoč se na njegovo predhodno dolžino. Njegova dolžina se avtomatsko poveča, če element dodamo in zmanjša, če element odstranimo. Nima fiksne dolžine, kot jo ima tabela. Tabeli velikost (in s tem dolžino) določimo vnaprej, elemente pa lahko poljubno dodajamo le, dokler ne presežemo njene velikosti. Prav tako v verižnem seznamu lažje vrivamo elemente. Naredili smo veliko primerov, kjer smo element dodali ali odstranili na sredini verižnega seznama. Vedno smo element (vozel) dodali obstoječemu verižnemu seznamu. To smo lahko naredili v konstantnem času, neodvisno od števila elementov. Če bi želeli dodati nov element v tabelo, bi morali elemente preložiti. Ta operacija pa bi bila časovno odvisna od mesta, kamor bi želeli nov element vstaviti in od števila elementov v tabeli.

Po drugi strani pa je prednost tabele ta, da lahko do njenih elementov dostopamo direktno oz. na podlagi indeksov. Pri verižnem seznamu lahko pridemo do elementov le tako, da začnemo pri prvem in se premikamo po verižnem seznamo do elementa, ki ga želimo.

Verižni seznam uporabimo, če imamo veliko vstavljanja in brisanja elementov ter, če elemente praviloma potrebujemo zaporedoma. Smiselno ga je uporabiti, če ne vemo koliko prostora za podatke bomo potrebovali. Tabelo pa uporabimo, če bomo pogosto potrebovali neposredni dostop do elementov.

1.9 ZAKLJUČEK

V okviru diplomske naloge so podrobneje razloženi vozel enojno povezanega verižnega seznama, enojno povezani verižni seznam s kazalcem na začetek ter enojno povezan verižni seznam s kazalcema na začetek in konec. Namen diplomske naloge je, da bralec pridobi osnovno znanje o verižnih seznamih.

Pri verižnih seznamih moramo biti pri izrazoslovju zelo previdni. Pomembno je ali govorimo o verižnem seznamu kot objektu ali o verižnem seznamu kot zbirki s pomočjo referenc povezanih vozlov. Ko govorimo o verižnem seznamu kot objektu, le-ta, v odvisnosti od tega ali govorimo o objektu tipa VerSez ali VerSezKonZac, vsebuje kazalec na začetek ali tudi na konec verižnega seznama. Preko kazalca na prvi element pridemo do vseh ostalih elementov verižnega seznama. V objektu tipa Vozel, poleg kazalca na istovrstni objekt, ki določa naslednika, hranimo še komponento podatek, ki je tipa int in v kateri hranimo dejanski podatek.

Nad verižnimi seznami in vozli lahko izvajamo različne operacije. Seveda je tudi tukaj pomembno, ali uporabljamo verižni seznam kot objekt VerSez ali VerSezKonZac, ali pa uporabljamo verižni seznam kot zaporedje vozlov tipa Vozel. Nekaj operacij kot so vstavljanje, brisanje, iskanje in izpis sem v diplomskem delu tudi opisala. Pri vstavljanju (kot tudi pri drugih opearcijah) je pomembno vedeti kaj želimo vstaviti (pri brisanju kateri vozel želimo izbrisati, pri iskanju kaj iščemo in izpisu kaj želimo izpisati) in na katero mesto oz. kam želimo vozel vstaviti.

Verižne sezname je smiselno uporabiti, če ne vemo koliko prostora za podatke bomo potrebovali. Njegova prednost je, da lahko število elementov poljubno spreminjamo, saj se njegova dolžina avtomatsko poveča, če element dodamo ali zmanjša če ga odstranimo. Do elementov verižnega seznama dostopamo preko prvega elementa in se postopoma premikamo do elementa, ki ga želimo.

Poznamo tudi dvojno povezane verižne sezname ter krožne sezname. Pri dvojno povezanih verižnih seznamih vsak vozel povežemo z obema sosednjima vozloma. Pri dvojno povezanih verižnih seznamih zadnji vozel naslednika nima, saj njegov kazalec, prav tako kot pri enojno povezanih verižnih seznamih, kaže na null. Pri krožno povezanih verižnih seznamih pa določimo, da ima zadnji vozel za naslednika prvi vozel.

Pri vseh operacijah znotraj verižnega seznama spreminjanje zaporedja vozlov opravljamo z manipulacijo s kazalci. Pri tem moramo paziti, da jih prenastavimo pravilno in da kakšnega kazalca ne izgubimo oz. ga ne izbrišemo. Če kazalec na določen vozel izgubimo in nanj ne kaže noben drug kazalec, do tega vozla ne moremo več dostopati.

LITERATURA IN VIRI

Goodrich, M. T., Tomassia, R., 2001: *Data structures and algorithms in java*, J. Wiley & Sons: New York, 641 str.

Watt, D. A., Brown, D. F., 2001: *Java collections : an introduction to abstract data types, data structures and algorithms*, J. Wiley & Sons: Chichester, 549 str.

Sedgewick, R., 2004: *Algorithms in Java*, Addison-Wesley: Boston, 497 str.

University of Saskatchewan, Department of computer science, priročnik narejen s strani študentov, ki so bili del projekta TEL project, 2000/2001: Linked Lists (online). Dostopno na: *http://www.cs.usask.ca/resources/tutorials/csconcepts/2000_1/Tutorial/1.0Introduction.html*

Universitet I Kobenhavn, Search engine project, 2005: Implementing sequences using linked-lists (online). Dostopno na: *http://www.itu.dk/research/theory/SearchEngine/E2005/mirrors/wiscdocs/notes/LINKED-LIST.html*

Tina Ivaniš, 2003/2004: Seminarska naloga Linearni seznam (online). Dostopno na: *http://rc.fmf.uni-lj.si/matija/predavanja/2003-2004/Seminarska3/sem3/Ivanis/*

Stanford CS Education Library, 2006: Linked list Basics, Linked list Problems (online). Dostopno na:
http://cslibrary.stanford.edu/103/
http://cslibrary.stanford.edu/105/

mag. Matija Lokar, 2003/2004: Računalništvo 2, vaje in predavanja, linearni seznami (online). Dostopno na:
http://rc.fmf.uni-lj.si/matija/predavanja/2003-2004/R2_VSS.htm

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO