

Jednostruko spregnuta lista

```
/**
 * Čvor jednostruko spregnute liste.
 * Svaki čvor liste u sebi čuva podatak (kod ovog čvora to je ceo broj)
 * i pokazivač na sledeći čvor.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class CvorJSListe {
    // podatak koji se čuva u čvoru.
    // Ako čvor treba da čuva neki drugi tip elementa, ovde to treba promeniti
    int podatak;

    // "pokazivač" na sledeći čvor u listi
    CvorJSListe sledeci;

    /**
     * Konstruktor. Prihvata i postavlja sve elemente čvora.
     * @param p Podatak koji se čuva u čvoru.
     * @param sled Pokazivač na sledeći element za čvor koji se kreira
     */
    public CvorJSListe(int p, CvorJSListe sled) {
        podatak = p;
        sledeci = sled;
    }
}

/**
 * Jednostruko spregnuta lista. Dinamička opšta lista - elementi mogu da se
 * dodaju bilo gde u strukturu i bilo koji element može da se izbaci.
 * Redosled ubacivanja i izbacivanja nije definisan. Elementi su uvezani preko jedne veze "unapred",
 * tako da je moguće direktno kretanje kroz strukturu samo u jednom smeru od prvog do poslednjeg. Na
 * svaki čvor pokazuje tačno jedan pokazivač, te je pri ubacivanju novog čvora potrebno povezati
 * jedan pokazivač na njega, dok je pri izbacivanju potrebno razvezati jedan
 * pokazivač.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
 */
public class JSLista {

    /**
     * Pokazivač na prvi čvor u listi. Pošto su čvorovi međusobno uvezani,
     * dovoljno je pamtititi gde se nalazi prvi čvor, pa preko njega pristupiti ostalim
     * čvorovima.
     */
    CvorJSListe prvi;

    public JSLista(){
        prvi = null;
    }

    /**
     * Proverava da li je lista prazna. Lista je prazna kada ne postoji
     * pokazivač na prvi čvor.
     * @return true ako je lista prazna. u suprotnom false
     */
    public boolean praznaLista(){
        return prvi == null;
    }
}
```

```

/**
 * Ubacivanje novog elementa na početak liste.
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciNaPocetak(int podatak){
    // Kreiranje novog čvora. Pošto se novi čvor kreira na početku
    // strukture, njegov sledeći element će biti trenutno prvi element.
    CvorJSListe novi = new CvorJSListe(podatak, prvi);
    // nakon kreiranja čvora postavlja se da je novokreirani čvor prvi čvor.
    prvi = novi;
}

/**
 * Ubacivanje novog elementa na kraj liste. Potrebno je naći poslednji čvor
 * i ubaciti novi nakon njega.
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciNaKraj(int podatak){
    if (praznaLista()){
        // ako je lista prazna, algoritam za ubacivanje novog čvora na kraj
        // liste je isti kao i za ubacivanje na početak, jer se ubacuje
        // prvi koji je istovremeno i poslednji čvor
        ubaciNaPocetak(podatak);
    } else {
        // pomoćni pokazivač preko koga se kreće kroz listu
        CvorJSListe pom = prvi;
        // pronalazak poslednjeg čvora u listi (onaj čvor koji nema sledbenika)
        while(pom.sledeci != null){
            pom = pom.sledeci;
        }
        // kreiranje novog čvora. Pošto se ubacuje na kraj liste,
        // onda on nema sledbenika.
        CvorJSListe novi = new CvorJSListe(podatak, null);
        // povezivanje da je novokreirani čvor sledbenik do tada poslednjem čvoru
        pom.sledeci = novi;
    }
}

/**
 * Izbacivanje elementa sa početka liste. Element se izbacuje tako što se pokazivač na prvi
 * čvor preveže da pokazuje na drugi čvor. Time niko više ne pokazuje na taj čvor čime se on briše
 * iz memorije.
 *
 * @return Vrednost iz čvora koji je izbačen.
 * @throws Exception Ako je lista prazna baca se izuzetak.
 */
public int izbaciSaPocetka() throws Exception{
    // Ako je lista prazna baca se izuzetak
    if (praznaLista())
        throw new Exception("lista je prazna");

    // sačuvamo podatak iz prvog čvora u pomoćnu promenljivu
    int podatak = prvi.podatak;
    // prebacujemo prvi na drugi čvor
    prvi = prvi.sledeci;
    // vraćamo sačuvani podatak
    return podatak;
}

```

```

/**
 * Izbacivanje elementa sa kraja liste. Da bismo izbacili poslednji čvor potrebno je da
 * nađemo pretposlednji čvor i da razvežemo vezu
 * između pretposlednjeg i poslednjeg čvora.
 *
 * @return Vrednost iz čvora koji je izbačen.
 * @throws Exception Ako je lista prazna baca se izuzetak.
 */
public int izbaciSaKraja() throws Exception{
    if (praznaLista())
        throw new Exception("lista je prazna");

    // ako lista ima samo jedan element, onda se izbacuje prvi
    if (prvi.sledeci == null)
        return izbaciSaPocetka();

    // pronalazimo pretposlednji čvor
    CvorJSListe pom = prvi;
    // pretposlednji čvor ima samo jedan element ispred sebe
    while(pom.sledeci.sledeci != null){
        pom = pom.sledeci;
    }
    int podatak = pom.sledeci.podatak;
    // razvezujemo vezu između pretposlednjeg i poslednjeg
    pom.sledeci = null;
    return podatak;
}

/**
 * Ubacivanje elementa u sortiranu listu tako da ona ostane sortirana u
 * rastućem redosledu. Prolazi se kroz listu i traži onaj čvor kod koga je podatak u
 * njegovom sledećem veći od elementa koji se ubacuje.
 * Nakon toga se novi čvor umetne između trenutnog i njegovog sledećeg.
 *
 * @param podatak Podatak koji će se čuvati u novom čvoru
 */
public void ubaciUSort(int podatak){
    // ako je lista prazna ili ako je podatak u prvom čvoru nije manji od
    // podatka koji se ubacuje, onda se vrši ubacivanje na početak liste
    if (praznaLista() || prvi.podatak >= podatak)
        prvi = new CvorJSListe(podatak, prvi);
    else {

        CvorJSListe pom = prvi;
        // tražimo element čiji sledeći čvor ima podatak koji je
        // veći od elementa koji ubacujemo ili dok ne dođemo do kraja liste
        while(pom.sledeci != null &&
            pom.sledeci.podatak < podatak){
            pom = pom.sledeci;
        }
        // umetanje novog čvora
        CvorJSListe novi = new CvorJSListe(podatak, pom.sledeci);
        pom.sledeci = novi;
    }
}

/**
 * Prolazak kroz listu i ispisivanje svih elemenata u čvorovima od
 * prvog do poslednjeg.
 */
public void ispisi(){
    CvorJSListe pom = prvi;
    while (pom != null){
        System.out.println(pom.podatak);
        pom = pom.sledeci;
    }
}

```

```

}

/**
 * Ispisivanje svih elemenata u čvorovima u obrnutom redosledu - od poslednjeg
 * do prvog. Rekurzivna metoda. Ideja algoritma: prvo je potrebno ispisati
 * sve elemente nakon trenutnog u obrnutom redosledu, a na kraju
 * ispisati trenutni element. Ovo se rekurzivno ponavlja za svaki čvor.
 */
public void ispisiObrnuto(CvorJSListe pom){
    if (pom == null) return;
    if (pom.sledeci != null)
        ispisiObrnuto(pom.sledeci);
    System.out.println(pom.podatak);
}
}

```