Rešitev sedme domače naloge (Stanovanjski blok)

Tale bo dolga. Nekatere metode se nekoliko poenostavijo, če namesto navadnih tabel uporabimo knjižnico vsebovalnikov (razrede, ki implementirajo vmesnike `List`, `Set` in `Map` iz paketa `java.util`). Ker pa je še ne poznamo, bomo delali s tabelami.

Razred `Oseba`

Ta razred je enostaven in ne potrebuje posebnih opomb.

```
public class Oseba {

    private String ip;
    private char spol;
    private int starost;

    public Oseba(String ip, char spol, int starost) {
        this.ip = ip;
        this.spol = spol;
        this.starost = starost;
    }

    public String toString() {
        return String.format("%s, %c, %d", this.ip, this.spol, this.starost);
    }

    public boolean jeStarejsaOd(Oseba os) {
        return (this.starost > os.starost);
    }
}
```

Razred `Stanovanje`

V razredu `Stanovanje` potrebujemo vsaj dva atributa: tabelo stanovalcev (`stanovalci`) in tabelo sosednjih stanovanj (`sosedi`):

```
public class Stanovanje {
    private Oseba[] stanovalci;
    private Stanovanje[] sosedi;
    ...
}
```

Spremenljivka `sosedi` bo na začetku `null`, po klicu metode `nastaviSosede` pa bo kazala na tabelo s štirimi kazalci na objekte. Prvi objekt bo predstavljal levega, drugi zgornjega, tretji desnega, četrti pa spodnjega soseda stanovanja `this`.

Konstruktor nastavi atributa `stanovalci` in `sosedi`. Stavek `this.sosedi = null` je, kot vemo, pravzaprav odveč, a ne škoduje.

```
public Stanovanje(Oseba[] stanovalci) {
    this.stanovalci = stanovalci;
    this.sosedi = null;
}
```

Število oseb v stanovanju `this` je enako dolžini tabele `this.stanovalci`.

```
public int steviloStanovalcev() {
    return this.stanovalci.length;
}
```

Pri metodi `steviloStarejsihOd` se sprehodimo po tabeli stanovalcev in štejemo osebe, starejše od podane osebe `os`.

```
public int steviloStarejsihOd(Oseba os) {
    int n = 0;
    for (Oseba p: this.stanovalci) {
        if (p.jeStarejsaOd(os)) {
            n++;
        }
    }
    return n;
}
```

V metodi `mz` si pripravimo tabelo dveh števcev, nato pa med sprehodom po tabeli stanovalcev vsakokrat povečamo ustrezni števec. Ker mora metoda za vsako osebo ugotoviti njen spol, atribut `spol` pa je v razredu `Oseba` privaten, bomo razred `Oseba` obogatili z »getterjem« za atribut `spol`:

```
public class Oseba {
    ...
    public char vrniSpol() {
        return this.spol;
    }
    ...
}

public class Stanovanje {
    ...
    public int[] mz() {
        int[] tab = {0, 0};
        for (Oseba p: this.stanovalci) {
            int ix = (p.vrniSpol() == 'M') ? 0 : 1;
            tab[ix]++;
        }
        return tab;
    }
    ...
}
```

Metoda `starosta` poišče najstarejšega stanovalca po dobro znanem postopku: med sprehodom po tabeli stanovalcev vzdržuje kazalec na najstarejšega doslej obravnavanega stanovalca.

```
public Oseba starosta() {
    if (this.stanovalci.length == 0) {
        return null;
    }
    Oseba naj = this.stanovalci[0];
    for (Oseba p: this.stanovalci) {
        if (p.jeStarejsaOd(naj)) {
            naj = p;
        }
    }
    return naj;
}
```

Metoda `nastaviSosede` ustvari novo tabelo `this.sosedi` in njene elemente obenem nastavi na podane parametre.

```
public void nastaviSosede(Stanovanje levi, Stanovanje zgornji,
        Stanovanje desni, Stanovanje spodnji) {

    this.sosedi = new Stanovanje[]{
        levi, zgornji, desni, spodnji
    };
}
```

Starosta soseščine stanovanja `this` bo bodisi starosta stanovanja `this` bodisi starosta enega od njegovih neposrednih sosedov. V metodi `starostaSosescine` se bomo zato sprehodili po vseh obstoječih sosedih stanovanja `this` in za vsakega poklicali metodo `starosta`, prav tako pa bomo to metodo poklicali za stanovanje `this`. Najstarejši starosta bo proglašen za starosto celotne soseščine.

```
public Oseba starostaSosescine() {
    Oseba naj = this.starosta();
    for (Stanovanje s: this.sosedi) {
        if (s != null) {
            Oseba najSosed = s.starosta();
            if (najSosed != null) {
                if (naj == null || najSosed.jeStarejsaOd(naj)) {
                    naj = najSosed;
                }
            }
        }
    }
    return naj;
}
```

Metoda `sosedjeSosedov` obišče vse sosede sosedov stanovanja `this` in vse stanovalce v teh stanovanjih doda v tabelo `osebe`. Pri tem mora paziti, da vsako stanovanje obišče

natanko enkrat in da ne upošteva stanovanja `this`, ki je dejansko prav tako sosed lastnega soseda. V ta namen ustvari tabelo `obiskanaStanovanja` in vanjo najprej shrani stanovanje `this`, v glavni zanki pa vanjo doda vsakega obravnavanega soseda soseda stanovanja `this`. Preden obravnava soseda soseda stanovanja `this`, preveri, ali se že nahaja v tabeli `obiskanaStanovanja`.

Tabela `obiskanaStanovanja` hrani kazalce na že obstoječe objekte; nikakršne potrebe ni po izdelavi kopij. V metodi `vsebuje`, ki preveri, ali se podano stanovanje nahaja v tabeli stanovanj, primerjamo stanovanja z operatorjem `==`, saj nas zanima, ali je trenutno obravnavano stanovanje v tabeli *istovetno* s podanim stanovanjem.

```
import java.util.Arrays;

public class Stanovanje {
    private static final int MAX_ST_STANOVALCEV = 10000;
    ...
    public Oseba[] sosedjeSosedov() {
        Stanovanje[] obiskanaStanovanja = new Stanovanje[20];
        int stObiskanihStanovanj = 0;
        obiskanaStanovanja[stObiskanihStanovanj++] = this;

        Oseba[] osebe = new Oseba[MAX_ST_STANOVALCEV];
        int stOseb = 0;    // število oseb v sosedih sosedov stanovanja this
        for (Stanovanje s: this.sosedi) {
            if (s != null) {
                for (Stanovanje ss: s.sosedi) {
                    if (ss != null && !vsebuje(obiskanaStanovanja, ss)) {
                        obiskanaStanovanja[stObiskanihStanovanj++] = ss;
                        System.arraycopy(ss.stanovalci, 0, osebe, stOseb,
                                         ss.steviloStanovalcev());
                        stOseb += ss.steviloStanovalcev();
                    }
                }
            }
        }
        return Arrays.copyOf(osebe, stOseb);
    }

    private static boolean vsebuje(Stanovanje[] tabela, Stanovanje stanovanje) {
        for (Stanovanje s: tabela) {
            if (s == stanovanje) {
                return true;
            }
        }
        return false;
    }
}
```

Da bi kodo nekoliko skrajšali, smo si pomagali z metodo `System.arraycopy(t1, p1, t2, p2, d)`, ki elemente `t1[p1]`, ..., `t1[p1 + d - 1]` skopira v celice `t2[p2]`, ..., `t2[p2 + d - 1]`.

Ker število sosedov sosedov stanovanja ne more biti večje od $4^2 = 16$, je več kot dovolj, če

dolžino tabele `obiskanaStanovanja` nastavimo na, recimo, 20. Dolžino tabele `osebe` smo nastavili na `MAX_ST_STANOVALCEV`, torej na največje možno število stanovalcev v bloku.

Tabela, ki jo vrne metoda `sosedjeSosedov`, mora biti dolga toliko, kot je dejansko skupno število stanovalcev v iskanih stanovanjih. Zato ne smemo vrniti tabele `osebe`, pač pa njeno »odsekano« kopijo. Pomagamo si z metodo `Arrays.copyOf(t, d)`, ki vrne kopijo podtabele, ki jo tvorijo elementi `t[0]`, ..., `t[d - 1]`.

Razred `Blok`

Objekt razreda `Blok` ob svoji inicializaciji sprejme kazalec na eno od stanovanj; recimo mu *izhodiščno stanovanje*. Metodi `starosta` in `razporeditev` pa potrebujeta podatke o vseh stanovanjih v bloku. Zato bomo prej ali slej morali prečesati celoten blok. To lahko storimo ob vsakem klicu metode `starosta` ali `razporeditev`, veliko bolj učinkovito pa je, če sprehod po bloku opravimo enkrat za vselej. Ob prvem klicu metode `starosta` ali `razporeditev` bomo poklicali metodo `obisciVsaStanovanja`, ki bo prečesala celoten blok in njegova stanovanja shranila v tabelo `this.vsaStanovanja`. Metoda bo ob sprehodu nastavila tudi relativne koordinate posameznih stanovanj — relativne glede na izhodiščno stanovanje. Izhodiščno stanovanje bo imelo relativni koordinati (0, 0) (vrstica, stolpec), koordinati njegovega zgornjega soseda bosta enaki $(-1, 0)$, desni sosed tega stanovanja bo imel koordinati $(-1, 1)$ itd.

Da bomo lahko hranili koordinate posameznih stanovanj, bomo razred `Stanovanje` dopolnili z atributoma `vrstica` in `stolpec` ter z metodama za njuno nastavljanje in vračanje:

```
public class Stanovanje {
    ...
    private int vrstica;
    private int stolpec;
    ...
    public void nastaviKoordinati(int vrstica, int stolpec) {
        this.vrstica = vrstica;
        this.stolpec = stolpec;
    }

    public int[] vrniKoordinati() {
        return new int[]{this.vrstica, this.stolpec};
    }
}
```

V razredu `Blok` bomo poleg »očitnega« atributa, ki hrani izhodiščno stanovanje v bloku, deklarirali tudi že omenjeni atribut `vsaStanovanja`:

```
import java.util.Arrays;

public class Blok {

    // maksimalno možno število stanovanj
    public static final int MAX_ST_STANOVANJ = 1000;

    // izhodiščno stanovanje v bloku this
    private Stanovanje izhodiscnoStanovanje;
```

```
    // vsa stanovanja v bloku this
    private Stanovanje[] vsaStanovanja;

    public Blok(Stanovanje stanovanje) {
        this.izhodiscnoStanovanje = stanovanje;
        this.vsaStanovanja = null;
    }

    private void obisciVsaStanovanja() {
        ...
    }
    ...
}
```

Metoda `obisciVsaStanovanja` najprej preveri, ali je svoje delo že opravila. To ugotovi na podlagi atributa `this.vsaStanovanja`: če ima vrednost `null`, potem se po bloku še ni sprehodila. V tem primeru pripravi tabelo `vsaStanovanjaTemp`. Vanjo bo kasneje shranjevala stanovanja, na katera bo naletela med obhodom po bloku.

```
private void obisciVsaStanovanja() {
    if (this.vsaStanovanja != null) {
        return;
    }
    Stanovanje[] vsaStanovanjaTemp = new Stanovanje[MAX_ST_STANOVANJ];
    int stVseh = 0;   // število doslej obiskanih stanovanj
    ...
}
```

Metoda bo vzdrževala tudi tabelo `neobdelana`, ki bo hranila vsa stanovanja, na katera je že naletela, ni pa še preverila njihovih sosedov. Ta tabela bo na začetku vsebovala samo izhodiščno stanovanje. Nato bo metoda vstopila v zanko, v kateri bo v vsakem obhodu odstranila eno od stanovanj (zadnje, ker bo to najenostavneje) iz tabele `neobdelana` in v tabelo namesto njega dodala vse njegove še neobiskane sosede. Odstranjeno stanovanje bo dodala tudi v tabelo `vsaStanovanjaTemp`. Zanka bo tekla tako dolgo, dokler se tabela `neobdelana` ne bo izpraznila.[1] Takrat bomo vedeli, da smo preiskali vse sosede vseh stanovanj in s tem celoten blok.

Za vsako stanovanje bomo morali hraniti tudi status obiskanosti. V ta namen bomo v razred `Stanovanje` dodali atribut, ki bo povedal, ali je stanovanje `this` že bilo obiskano ali ne. Seveda bomo potrebovali tudi metodi za nastavljanje oziroma preverjanje tega atributa.

```
public class Stanovanje {
    ...
    private boolean zeObiskano;
    ...
    public void nastaviObiskanost(boolean obiskanost) {
        this.zeObiskano = obiskanost;
    }
```

[1] Ker dolžine tabele ne moremo spreminjati, bomo vzdrževali števec `stNeobdelanih`, ki bo hranil efektivno število elementov v njej.

```
    public boolean zeObiskano() {
        return this.zeObiskano;
    }
}
```

Med sprehodom po bloku shranjujemo tudi relativne koordinate obiskanih stanovanj. Izhodiščno stanovanje ima koordinati (0, 0), sosedje stanovanja s koordinatama (v, s) pa imajo koordinate $(v, s-1)$ (levi), $(v-1, s)$ (zgornji), $(v, s+1)$ (desni) in $(v+1, s)$ (spodnji).

Ko se iskalna zanka izteče, s pomočjo metode `Arrays.copyOf` skopiramo relevanten del tabele `vsaStanovanjaTemp` v tabelo `this.vsaStanovanja`.

```
private void obisciVsaStanovanja() {
    ...
    Stanovanje[] neobdelana = new Stanovanje[MAX_ST_STANOVANJ];
    int stNeobdelanih = 0;

    neobdelana[stNeobdelanih++] = this.izhodiscnoStanovanje;
    this.izhodiscnoStanovanje.nastaviObiskanost(true);
    this.izhodiscnoStanovanje.nastaviKoordinati(0, 0);

    while (stNeobdelanih > 0) {
        // odstrani zadnje stanovanje iz tabele
        Stanovanje trenutno = neobdelana[--stNeobdelanih];
        int[] koordinati = trenutno.vrniKoordinati();
        vsaStanovanjaTemp[stVseh++] = trenutno;

        // odmiki koordinat sosedov glede na trenutno stanovanje
        int[] dVrstica = {0, -1, 0, 1};
        int[] dStolpec = {-1, 0, 1, 0};

        // dodaj še ne obiskane sosede v tabelo neobdelana
        int ixSoseda = 0;
        for (Stanovanje sosed: trenutno.vrniSosede()) {
            if (sosed != null && !sosed.zeObiskano()) {
                sosed.nastaviObiskanost(true);
                int vr = koordinati[0] + dVrstica[ixSoseda];
                int st = koordinati[1] + dStolpec[ixSoseda];
                sosed.nastaviKoordinati(vr, st);
                neobdelana[stNeobdelanih++] = sosed;
            }
            ixSoseda++;
        }
    }
    this.vsaStanovanja = Arrays.copyOf(vsaStanovanjaTemp, stVseh);
}
```

Metoda pokliče tudi »getter« za tabelo sosedov podanega stanovanja:

```
public class Stanovanje {
    ...
```

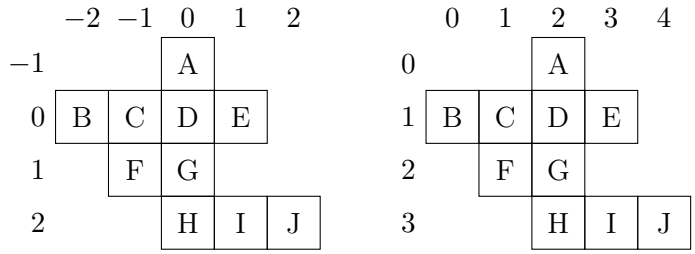
```
    public Stanovanje[] vrniSosede() {
        return this.sosedi;
    }
    ...
}
```

Metoda `starosta` je sedaj enostavna: najprej prečeše celoten blok (če to še ni bilo opravljeno), nato pa za vsako stanovanje v tabeli `this.vsaStanovanja` pokliče metodo `starosta` in po potrebi posodobi kazalec na najstarejšo doslej obravnavano osebo.

```
public Oseba starosta() {
    this.obisciVsaStanovanja();

    Oseba naj = null;
    for (Stanovanje stanovanje: this.vsaStanovanja) {
        Oseba starosta = stanovanje.starosta();
        if (starosta != null) {
            if (naj == null || starosta.jeStarejsaOd(naj)) {
                naj = starosta;
            }
        }
    }
    return naj;
}
```

Metoda `razporeditev` na začetku prav tako po potrebi preišče celoten blok. Da bi lahko izdelala dvodimenzionalno tabelo s podatki o zasedenosti posameznih stanovanj, mora ugotoviti višino in širino bloka, relativne koordinate stanovanj pa pretvoriti v absolutne. Recimo, da se ukvarjamo z blokom iz besedila naloge in da stanovanje D služi kot naše izhodiščno stanovanje. V tem primeru je višina bloka enaka 4, širina 5, relativne koordinate pa v absolutne pretvorimo takole:



Naj $v_{\min}$ in $v_{\max}$ označujeta najmanjšo in največjo relativno vrstično koordinato (-1 in 2 v gornjem primeru), $s_{\min}$ in $s_{\max}$ pa najmanjšo in največjo relativno stolpčno koordinato (-2 in 2 v gornjem primeru). Relativne koordinate pretvorimo v absolutne preprosto tako, da od relativne vrstične koordinate odštejemo $v_{\min}$, od relativne stolpčne pa $s_{\min}$. Višina bloka znaša $v_{\max} - v_{\min} + 1$, širina pa $s_{\max} - s_{\min} + 1$.

Ko metoda `razporeditev` izračuna $v_{\min}$, $v_{\max}$, $s_{\min}$ in $s_{\max}$, izdela tabelo `razporeditev` ustrezne velikosti, nato pa za vsako stanovanje v tabeli `this.vsaStanovanja` pokliče metodo `vrniKoordinati`, dobljeni relativni koordinati pretvori v absolutni in v pripadajočo celico tabele `razporeditev` vpiše število stanovalcev v tem stanovanju.

```
public int[][] razporeditev() {
    this.obisciVsaStanovanja();

    // določi ekstreme relativnih koordinat
    int minVr = 0;
    int minSt = 0;
    int maxVr = 0;
    int maxSt = 0;
    for (Stanovanje stanovanje: this.vsaStanovanja) {
        int[] k = stanovanje.vrniKoordinati();
        minVr = Math.min(minVr, k[0]);
        maxVr = Math.max(maxVr, k[0]);
        minSt = Math.min(minSt, k[1]);
        maxSt = Math.max(maxSt, k[1]);
    }

    // izdelaj dvodimenzionalno tabelo
    int stVrstic = maxVr - minVr + 1;
    int stStolpcev = maxSt - minSt + 1;
    int[][] razporeditev = new int[stVrstic][stStolpcev];
    for (int i = 0;  i < stVrstic;  i++) {
        for (int j = 0;  j < stStolpcev;  j++) {
            razporeditev[i][j] = -1;
        }
    }

    // napolni tabelo
    for (Stanovanje stanovanje: this.vsaStanovanja) {
        int[] k = stanovanje.vrniKoordinati();
        razporeditev[k[0] - minVr][k[1] - minSt] = stanovanje.steviloStanovalcev();
    }
    return razporeditev;
}
```