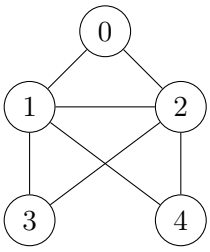
Primer izpita pri predmetu Programiranje I

Tik pred začetkom ...

- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) Enostaven neusmerjen graf z vozlišči 0, 1, 2, ..., $n-1$ ($n \geqslant 2$) lahko predstavimo z *matriko sosednosti* (tabela tipa `boolean[][]` velikosti $n \times n$, v kateri element `[i][j]` pove, ali sta vozlišči i in j povezani) ali pa s *seznamom sosednosti* (nepravokotna tabela tipa `int[][]` z n vrsticami, v kateri vrstica `[i]` vsebuje indekse vozlišč, povezanih z vozliščem i). Na primer:




```
// Matrika sosednosti:
boolean[][] m = {
    {F, T, T, F, F},
    {T, F, T, T, T},
    {T, T, F, T, T},
    {F, T, T, F, F},
    {F, T, T, F, F}
}; // (T = true, F = false)
```

```
// Seznam sosednosti:
int[][] s = {
    {1, 2},
    {0, 2, 3, 4},
    {0, 1, 3, 4},
    {1, 2},
    {1, 2}
};
```

V razredu `Graf` dopolnite sledeče metode:

- `public static boolean imaEulerjevObhod(boolean[][] m)`: [1–16]

  Vrne `true` natanko v primeru, če ima graf, podan z matriko sosednosti `m`, Eulerjev obhod. To velja natanko tedaj, ko ima vsako vozlišče sodo število sosedov. Na primer, graf na gornji sliki ima Eulerjev obhod.

- `public static boolean[][] vMatriko(int[][] s)`: [17–33]

  Vrne matriko sosednosti, ki opisuje isti graf kot podani seznam sosednosti `s`.

- `public static boolean istiGraf(boolean[][] m, int[][] s)`: [34–50]

  Vrne `true` natanko v primeru, če matrika sosednosti `m` in seznam sosednosti `s` opisujeta isti graf.

(2) Zaporedje k-kotnih števil je definirano takole:

$$a_1 = 1 \xrightarrow{+k-1} a_2 \xrightarrow{+2k-3} a_3 \xrightarrow{+3k-5} a_4 \xrightarrow{+4k-7} a_5 \xrightarrow{+5k-9} a_6 \ldots$$

Na primer, zaporedje 5-kotnih števil je 1, 5, 12, 22, 35, 51, ...:

$$1 \xrightarrow{+4} 5 \xrightarrow{+7} 12 \xrightarrow{+10} 22 \xrightarrow{+13} 35 \xrightarrow{+16} 51 \ldots$$

Razred `Zaporedje` dopolnite s sledečimi konstruktorji in metodami (ter po potrebi z atributi):

- `public Zaporedje(int k)`: [1–50]

  Ustvari objekt razreda `Zaporedje`, s pomočjo katerega bo mogoče (z zaporednimi klici metode `naslednje`) tvoriti zaporedje `k`-kotnih števil.

- `public int naslednje()`: [1–50]

  Vrne naslednje število v zaporedju. Ko se metoda nad nekim objektom tipa `Zaporedje` pokliče prvič, naj vrne prvo število (to je vedno 1), ko se pokliče drugič, naj vrne drugo število v zaporedju itd.

- `public int katero()`: [17–50]

  Vrne zaporedno številko števila, ki ga bo vrnil naslednji klic metode `naslednje`.

- `public void ponastavi(int n)`: [34–50]

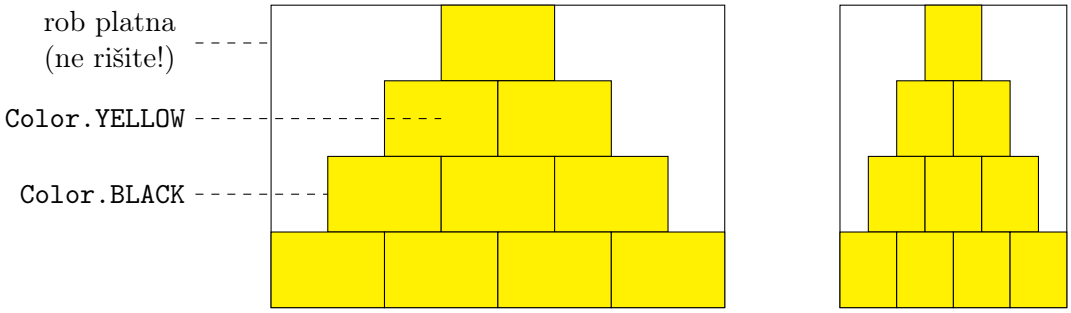
  Naredi vse, kar je treba, da bo naslednji klic metode `naslednje` vrnil `n`-to število v zaporedju. Velja $\texttt{n} \geqslant 1$.

Za lažje razumevanje si oglejmo primer uporabe razreda `Zaporedje`:

```
Zaporedje z = new Zaporedje(5);          // tvorili bomo 5-kotna števila
System.out.println( z.naslednje() );     // 1
System.out.println( z.naslednje() );     // 5
System.out.println( z.naslednje() );     // 12
System.out.println( z.katero() );        // 4 (metoda naslednje bo naslednjič
                                         //    vrnila četrto število v zaporedju)
System.out.println( z.naslednje() );     // 22
System.out.println( z.katero() );        // 5

z.ponastavi(3);
System.out.println( z.katero() );        // 3
System.out.println( z.naslednje() );     // 12
System.out.println( z.naslednje() );     // 22
```

(3) Dopolnite metodo `narisi` v razredu `Piramida` tako, da bo narisala »piramido« višine `visina`. Število `visina` (podano je kot atribut) je v vseh primerih enako najmanj 2. Sledeča slika prikazuje piramido višine 4 pri dveh različnih velikostih platna:



Piramida naj bo sestavljena iz enako velikih blokov (obrobljenih pravokotnikov) in naj po obeh dimenzijah zavzema celotno površino platna.

Dopolnite tudi sledeči metodi:

- `public double sirinaBloka(double wp, double hp)` [1–5]
- `public double visinaBloka(double wp, double hp)` [6–10]

Metodi naj vrneta širino oziroma višino posameznega bloka piramide v odvisnosti od širine (`wp`) in višine (`hp`) platna.

Primer izpita pri predmetu Programiranje I (na podlagi izpita z dne 4. februarja 2015)

Tik pred začetkom ...

- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) Mojster Vahid stoji pred steno velikosti $n \times n$, v škatli pa ima k ploščic velikosti $h_i \times w_i$ (h_i je višina, w_i pa širina i-te ploščice za $i \in \{1, \ldots, k\}$). Na vrh stene bi rad položil čimveč ploščic z najmanjšo višino, pod njimi čimveč ploščic z drugo najmanjšo višino itd. Znotraj ene vrste ploščic so torej vse ploščice enako visoke, višine vrst pa strogo naraščajo. Največ koliko ploščic lahko položi na opisani način?

Vhod: V prvi vrstici vhoda je podano celo število $n \in [1, 10^9]$, v drugi celo število $k \in [1, 10^3]$, nato pa sledi k vrstic s pari celih števil $h_i \in [1, n]$ in $w_i \in [1, n]$ (števili sta ločeni s presledkom).

V primerih 1–10 so vse ploščice enako velike, v primerih 11–25 pa enako visoke. (V primerih 1–25 bo torej na steni samo ena vrsta ploščic.) V primerih 11–20 in 26–35 so ploščice na vhodu urejene po naraščajočih višinah, pri enakih višinah pa po naraščajočih širinah.

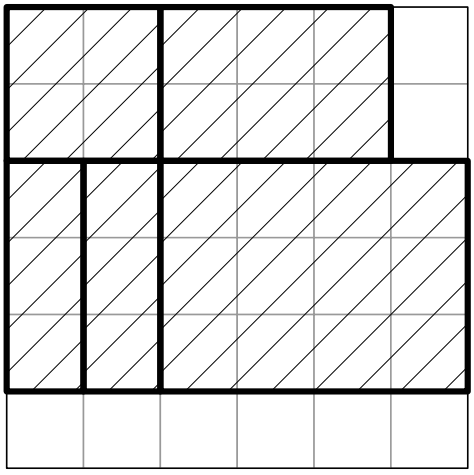
Izhod: Na izhodu izpišite samo iskano število ploščic.

Primer (vhod/izhod):

```
6
7
3 1
2 3
3 4
4 2
3 1
2 3
2 2
```

```
5
```

V gornjem primeru lahko položi vse ploščice razen ploščice 4×2 in ene od obeh 2×3:



(2) Podani so sledeči razredi in vmesniki (prikazani so samo atributi):

```
class Sestavina {
    private String naziv;      // naziv sestavine, npr. "moka"
    private double kalorije;   // število kcal na 100 gramov sestavine
    private int oh;            // delež ogljikovih hidratov (v odstotkih)
    private int beljakovine;   // delež beljakovin (v odstotkih)
    private int mascobe;       // delež maščob (v odstotkih)
}
class Jed {
    private String naziv;            // naziv, npr. "prekmurska gibanica"
    private Sestavina[] sestavine;   // vse sestavine, ki tvorijo jed
    private double[] mase;           // mase posameznih sestavin v gramih
    private String vrsta;            // npr. "glavna jed", "sladica"
}
class Obrok {
    private Jed[] jedi;   // jedi, ki sestavljajo obrok
}
interface Primerjalnik {}
```

V vseh objektih razreda `Jed` je tabela `mase` enako dolga kot tabela `sestavine`. Element `mase[i]` podaja maso sestavine `sestavine[i]` v jedi.

Dopolnite sledeče metode:

- `public boolean jeBeljakovinska()` v razredu `Jed` [1–10]:

  Vrne `true` natanko v primeru, če je jed `this` beljakovinska, kar pomeni, da vsaj ena sestavina jedi `this` vsebuje najmanj 10% beljakovin.

- `public double kalorije()` v razredu `Jed` [11–20]:

  Vrne skupno število kcal za jed `this`.

- `public int indeksNajboljKaloricneBeljakovinske()` v razredu `Obrok` [21–30]:

  Vrne indeks najbolj kalorične beljakovinske jedi v tabeli jedi obroka `this`. Če je takih jedi več, naj metoda vrne indeks prve od njih. Če beljakovinskih jedi ni, naj metoda vrne vrednost -1.

- `public double masaSestavine(String naziv)` v razredu `Obrok` [31–40]:

  Vrne skupno maso sestavine (ne jedi!) s podanim nazivom v obroku `this`.

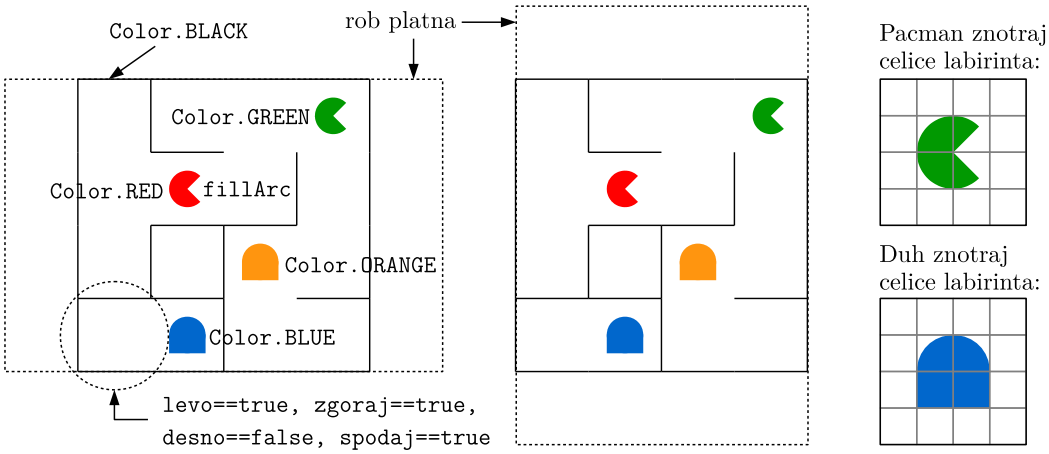
Napišite še razred `PrimerjalnikBK`, in to tako, da bo metoda `urediJedi` v razredu `Obrok`, če ji podamo objekt tega razreda, jedi obroka `this` uredila najprej po »beljakovinskosti« (v urejeni tabeli bodo najprej nanizane vse nebeljakovinske, nato pa vse beljakovinske jedi), obe kategoriji posebej pa bo naraščajoče uredila po številu kalorij [primeri 41–50].

(3) Dopolnite metodo `narisi` v razredu `Igra` tako, da bo na podlagi tabele `celice` narisala labirint, sestavljen iz enako velikih kvadratnih celic, na podlagi tabele `figure` pa posamezne figure (*pacmane* in *duhove*) znotraj labirinta. Tabela `celice` je kvadratna tabela tipa `Celica[][]`, v kateri element na indeksih `[`i`][`j`]` pove, na katerih stranicah je celica v vrstici z indeksom i in stolpcu z indeksom j obdana s steno. Tabela `figure` (tipa `Figura[]`) pa vsebuje podatke o posameznih figurah. Podatki so predstavljeni z objekti razredov `Pacman` in `Duh`.

```
class Celica {    // notranji razred v razredu Igra
    // vrednost true pove, da se na pripadajoči stranici celice nahaja stena
    private boolean levo, zgoraj, desno, spodaj;
}
abstract class Figura {    // notranji razred v razredu Igra
    private int vrstica;   // indeks vrstice celice, kjer se nahaja figura this
    private int stolpec;   // indeks stolpca celice, kjer se nahaja figura this
    private Color barva;   // barva figure this
}
class Pacman extends Figura {}   // objekt tega razreda predstavlja pacmana
class Duh extends Figura {}      // objekt tega razreda predstavlja duha
```

Vsi pacmani in duhovi se nahajajo na veljavnih in medsebojno različnih položajih. Velikost tabele `celice` znaša vsaj 1×1. Podatki o celicah labirinta so konsistentni.

Labirint naj zavzema celotno krajšo stranico platna, po daljši pa naj bo narisan na sredini platna. Pri risanju se zgledujte po sledeči sliki, ki se nanaša na javne testne primere 7–10 ter na tabeli `CELICE` in `FIGURE` v razredu `Igra`:



Poleg metode `narisi` dopolnite tudi sledeči metodi:

- `public double stranicaCelice(double wp, double hp)` [1–5]:

  Vrne dolžino stranice celice v odvisnosti od širine (`wp`) in višine (`hp`) platna.

- `public double[] zgorajLevo(double wp, double hp)` [6–10]:

  Vrne tabelo z dvema elementoma, ki podajata koordinati (najprej x, nato y) zgornjega levega kota labirinta.

V primerih 11–20 velja `wp == hp`. V primerih 11–30 velja `figure.length == 0`, kar pomeni, da se vam za do 60% točk s pacmani in duhovi ni treba ukvarjati.

Primer izpita pri predmetu Programiranje I (na podlagi izpita z dne 18. februarja 2015)

Tik pred začetkom ...

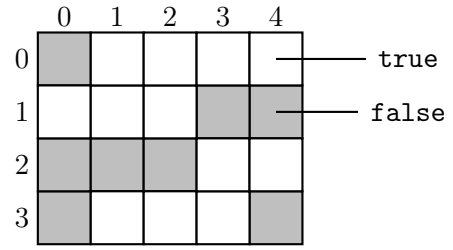
- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) V razredu `Park` dopolnite spodaj navedene metode. **Pozor:** morda se vam bosta nalogi (c) in (d) zdeli lažji od naloge (b).

(a) `public static int steviloProstih(boolean[][] p)` [1–12]:

Vrne število prostih parkirnih mest na pravokotnem parkirišču. Neprazna pravokotna tabela `p` podaja razpoložljivosti posameznih mest (`true`: prosto; `false`: zasedeno). V sledečem primeru bi metoda vrnila vrednost 12.



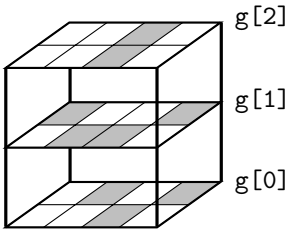
(b) `public static boolean zaporedje(boolean[][] p, int k)` [13–25]:

Vrne `true` natanko v primeru, če se v vsaj eni vrstici parkirišča nahaja vsaj `k` zaporednih prostih mest (`k` $\geqslant 1$). V gornjem primeru bi metoda vrnila `true` za vse `k` $\leqslant 4$.

(c) `public static int steviloMest(boolean[][][] g)` [26–37]:

Parkirišče je »zraslo« v garažno hišo v obliki kvadra. Prva dimenzija neprazne pravilne tabele `g` se nanaša na nadstropja, druga in tretja pa na zasedenosti parkirnih mest v posameznih nadstropjih.

Metoda `steviloMest` naj vrne število *vseh* parkirnih mest (prostih in zasedenih) v garažni hiši. V sledečem primeru bi bil rezultat enak 24.



(d) `public static int najboljProstoNadstropje(boolean[][][] g)` [38–50]:

Vrne indeks nadstropja z največ prostimi mesti; če je takih indeksov več, naj vrne najmanjšega. V gornjem primeru bi bil rezultat enak 2.

(2) Podani so sledeči razredi (prikazani so samo atributi):

```
class Recept {    // kuharski recept
    private Korak[] koraki;    // koraki recepta
}
class Korak {    // korak recepta
    private Snov[] vhodi;      // jedilne snovi, ki vstopajo v korak
    private Snov[] izhodi;     // jedilne snovi, ki so rezultat koraka
    private String akcija;     // akcija, npr. "zmesaj"
    private int trajanje;      // trajanje koraka v minutah
}
class Snov {     // jedilna snov
    private String naziv;      // unikaten naziv, npr. "moka" ali "palacinke"
}
```

Dopolnite sledeče metode:

(a) `public int trajanje()` v razredu `Recept` [1–12]:

Vrne trajanje recepta v minutah (torej vsoto trajanj vseh korakov).

(b) `public int prviKorakZAkcijo(String akcija)` v razredu `Recept` [13–25]:

Vrne indeks prvega koraka s podano akcijo oziroma -1, če takšnega koraka v receptu ni.

(c) `public boolean naIzhodu(Recept recept)` v razredu `Snov` [26–37]:

Vrne `true` natanko v primeru, če snov `this` nastopa na izhodu vsaj enega koraka podanega recepta.

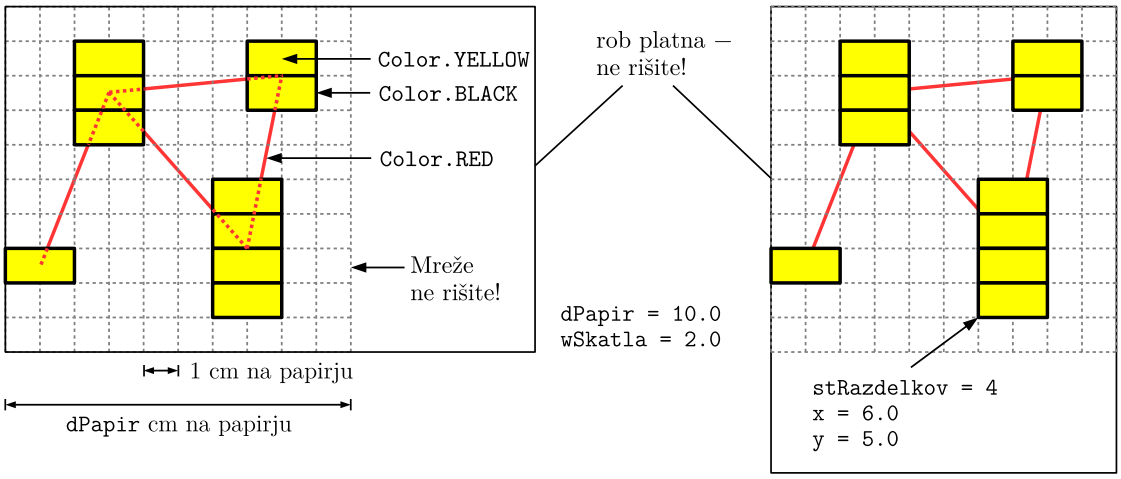
(d) `public int steviloVstopnihSnovi()` v razredu `Recept` [38–50]:

Vrne število snovi, ki se pojavljajo *samo* na vhodih v korake recepta. Lahko predpostavite, da vsaka snov nastopa na vhodu v kvečjemu enem koraku recepta.

(3) Na kvadratnem listu papirja s stranico `dPapir` cm je narisan diagram s »škatlami« in povezavami med njimi. Vse škatle so široke `wSkatla` cm, po višini pa so razdeljene na razdelke višine 1 cm. Povezave povezujejo središča škatel. Podatki o škatlah in povezavah so zbrani v tabelah `skatle` (tipa `Skatla`) in `povezave` (tipa `Povezava`):

```
private static class Skatla {
    private int stRazdelkov;   // število razdelkov (≥ 1)
    private double x, y;       // odmik (v cm) zgornjega levega kota škatle
                               //       od zgornjega levega kota lista papirja
}
private static class Povezava {
    private Skatla prva, druga;    // škatli, ki ju povezava povezuje
}
```

Dopolnite metodo `narisi` v razredu `Diagram` tako, da bo list z diagramom »preslikala« na platno. List naj bo postavljen v zgornjem levem kotu, zavzema pa naj celotno krajšo stranico platna. Zgledujte se po sledeči sliki, ki se nanaša na primer v razredu `Diagram` in na javna primera 8–9:



Dopolnite tudi sledeči metodi:

- `public double sirinaSkatle(double wp, double hp)` [1–5]:

  Na podlagi širine in višine platna vrne širino škatle v slikovnih pikah (»pikslih«) — ne v centimetrih!

- `public double[] sredisceSkatle(Skatla sk, double wp, double hp)` [6–15]:

  Vrne koordinati v »pikslih« (v obliki tabele $\{x, y\}$) središča podane škatle.

Namig: Kaj se vam splača narisati prej — škatle ali povezave?

Opombe: Škatle se med seboj ne prekrivajo. Za primere 16–25 velja `wp == hp`, za primere 16–35 pa (tudi) `povezave.length == 0`.

Primer izpita pri predmetu Programiranje I

Tik pred začetkom ...

- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) Mojster Vahid stoji pred steno velikosti $n \times n$, v škatli pa ima k ploščic velikosti $h_i \times w_i$ (h_i je višina, w_i pa širina i-te ploščice za $i \in \{1, \ldots, k\}$). Na vrh stene bi rad položil čimveč ploščic z najmanjšo višino, pod njimi čimveč ploščic z drugo najmanjšo višino itd. Znotraj ene vrste ploščic so torej vse ploščice enako visoke, višine vrst pa strogo naraščajo. Največ koliko ploščic lahko položi na opisani način?

Vhod: V prvi vrstici vhoda je podano celo število $n \in [1, 10^9]$, v drugi celo število $k \in [1, 10^3]$, nato pa sledi k vrstic s pari celih števil $h_i \in [1, n]$ in $w_i \in [1, n]$ (števili sta ločeni s presledkom).

V primerih 1–10 so vse ploščice enako velike, v primerih 11–25 pa enako visoke. (V primerih 1–25 bo torej na steni samo ena vrsta ploščic.) V primerih 11–20 in 26–35 so ploščice na vhodu urejene po naraščajočih višinah, pri enakih višinah pa po naraščajočih širinah.

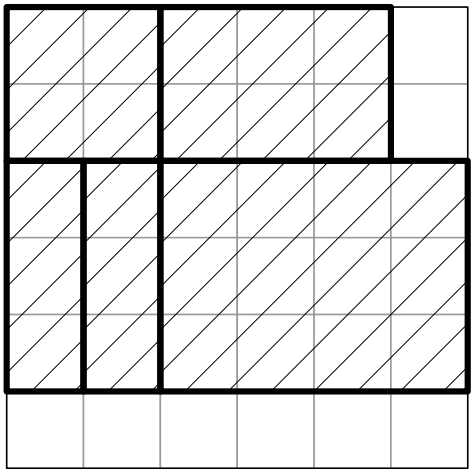
Izhod: Na izhodu izpišite samo iskano število ploščic.

Primer (vhod/izhod):

```
6
7
3 1
2 3
3 4
4 2
3 1
2 3
2 2
```

```
5
```

V gornjem primeru lahko položi vse ploščice razen ploščice 4×2 in ene od obeh 2×3:



(2) Podani so sledeči razredi in vmesniki (prikazani so samo atributi):

```
class Sestavina {
    private String naziv;      // naziv sestavine, npr. "moka"
    private double kalorije;   // število kcal na 100 gramov sestavine
    private int oh;            // delež ogljikovih hidratov (v odstotkih)
    private int beljakovine;   // delež beljakovin (v odstotkih)
    private int mascobe;       // delež maščob (v odstotkih)
}
class Jed {
    private String naziv;            // naziv, npr. "prekmurska gibanica"
    private Sestavina[] sestavine;   // vse sestavine, ki tvorijo jed
    private double[] mase;           // mase posameznih sestavin v gramih
    private String vrsta;            // npr. "glavna jed", "sladica"
}
class Obrok {
    private Jed[] jedi;   // jedi, ki sestavljajo obrok
}
interface Primerjalnik {}
```

V vseh objektih razreda `Jed` je tabela `mase` enako dolga kot tabela `sestavine`. Element `mase[i]` podaja maso sestavine `sestavine[i]` v jedi.

Dopolnite sledeče metode:

- `public boolean jeBeljakovinska()` v razredu `Jed` [1–10]:

  Vrne `true` natanko v primeru, če je jed `this` beljakovinska, kar pomeni, da vsaj ena sestavina jedi `this` vsebuje najmanj 10% beljakovin.

- `public double kalorije()` v razredu `Jed` [11–20]:

  Vrne skupno število kcal za jed `this`.

- `public int indeksNajboljKaloricneBeljakovinske()` v razredu `Obrok` [21–30]:

  Vrne indeks najbolj kalorične beljakovinske jedi v tabeli jedi obroka `this`. Če je takih jedi več, naj metoda vrne indeks prve od njih. Če beljakovinskih jedi ni, naj metoda vrne vrednost -1.

- `public double masaSestavine(String naziv)` v razredu `Obrok` [31–40]:

  Vrne skupno maso sestavine (ne jedi!) s podanim nazivom v obroku `this`.

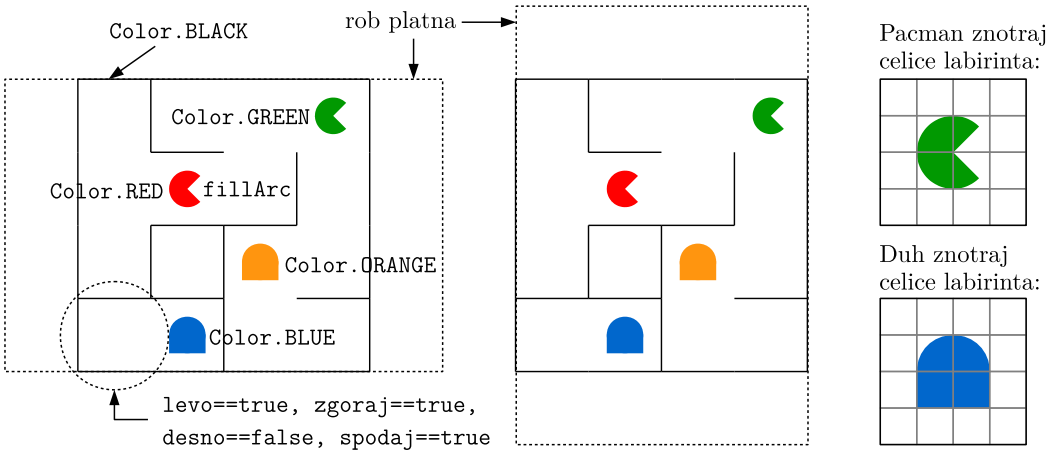
Napišite še razred `PrimerjalnikBK`, in to tako, da bo metoda `urediJedi` v razredu `Obrok`, če ji podamo objekt tega razreda, jedi obroka `this` uredila najprej po »beljakovinskosti« (v urejeni tabeli bodo najprej nanizane vse nebeljakovinske, nato pa vse beljakovinske jedi), obe kategoriji posebej pa bo naraščajoče uredila po številu kalorij [primeri 41–50].

(3) Dopolnite metodo `narisi` v razredu `Igra` tako, da bo na podlagi tabele `celice` narisala labirint, sestavljen iz enako velikih kvadratnih celic, na podlagi tabele `figure` pa posamezne figure (*pacmane* in *duhove*) znotraj labirinta. Tabela `celice` je kvadratna tabela tipa `Celica[][]`, v kateri element na indeksih `[`i`][`j`]` pove, na katerih stranicah je celica v vrstici z indeksom i in stolpcu z indeksom j obdana s steno. Tabela `figure` (tipa `Figura[]`) pa vsebuje podatke o posameznih figurah. Podatki so predstavljeni z objekti razredov `Pacman` in `Duh`.

```
class Celica {    // notranji razred v razredu Igra
    // vrednost true pove, da se na pripadajoči stranici celice nahaja stena
    private boolean levo, zgoraj, desno, spodaj;
}
abstract class Figura {    // notranji razred v razredu Igra
    private int vrstica;   // indeks vrstice celice, kjer se nahaja figura this
    private int stolpec;   // indeks stolpca celice, kjer se nahaja figura this
    private Color barva;   // barva figure this
}
class Pacman extends Figura {}   // objekt tega razreda predstavlja pacmana
class Duh extends Figura {}      // objekt tega razreda predstavlja duha
```

Vsi pacmani in duhovi se nahajajo na veljavnih in medsebojno različnih položajih. Velikost tabele `celice` znaša vsaj 1×1. Podatki o celicah labirinta so konsistentni.

Labirint naj zavzema celotno krajšo stranico platna, po daljši pa naj bo narisan na sredini platna. Pri risanju se zgledujte po sledeči sliki, ki se nanaša na javne testne primere 7–10 ter na tabeli `CELICE` in `FIGURE` v razredu `Igra`:



Poleg metode `narisi` dopolnite tudi sledeči metodi:

- `public double stranicaCelice(double wp, double hp)` [1–5]:

  Vrne dolžino stranice celice v odvisnosti od širine (`wp`) in višine (`hp`) platna.

- `public double[] zgorajLevo(double wp, double hp)` [6–10]:

  Vrne tabelo z dvema elementoma, ki podajata koordinati (najprej x, nato y) zgornjega levega kota labirinta.

V primerih 11–20 velja `wp == hp`. V primerih 11–30 velja `figure.length == 0`, kar pomeni, da se vam za do 60% točk s pacmani in duhovi ni treba ukvarjati.

Primer izpita pri predmetu Programiranje I

Tik pred začetkom ...

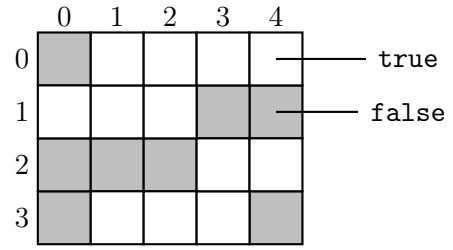
- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) V razredu `Park` dopolnite spodaj navedene metode. **Pozor:** morda se vam bosta nalogi (c) in (d) zdeli lažji od naloge (b).

(a) `public static int steviloProstih(boolean[][] p)` [1–12]:

Vrne število prostih parkirnih mest na pravokotnem parkirišču. Neprazna pravokotna tabela `p` podaja razpoložljivosti posameznih mest (`true`: prosto; `false`: zasedeno). V sledečem primeru bi metoda vrnila vrednost 12.



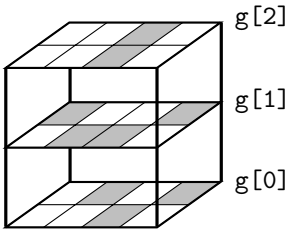
(b) `public static boolean zaporedje(boolean[][] p, int k)` [13–25]:

Vrne `true` natanko v primeru, če se v vsaj eni vrstici parkirišča nahaja vsaj `k` zaporednih prostih mest (`k` $\geqslant 1$). V gornjem primeru bi metoda vrnila `true` za vse `k` $\leqslant 4$.

(c) `public static int steviloMest(boolean[][][] g)` [26–37]:

Parkirišče je »zraslo« v garažno hišo v obliki kvadra. Prva dimenzija neprazne pravilne tabele `g` se nanaša na nadstropja, druga in tretja pa na zasedenosti parkirnih mest v posameznih nadstropjih.

Metoda `steviloMest` naj vrne število *vseh* parkirnih mest (prostih in zasedenih) v garažni hiši. V sledečem primeru bi bil rezultat enak 24.



(d) `public static int najboljProstoNadstropje(boolean[][][] g)` [38–50]:

Vrne indeks nadstropja z največ prostimi mesti; če je takih indeksov več, naj vrne najmanjšega. V gornjem primeru bi bil rezultat enak 2.

(2) Podani so sledeči razredi (prikazani so samo atributi):

```
class Recept {    // kuharski recept
    private Korak[] koraki;    // koraki recepta
}
class Korak {    // korak recepta
    private Snov[] vhodi;      // jedilne snovi, ki vstopajo v korak
    private Snov[] izhodi;     // jedilne snovi, ki so rezultat koraka
    private String akcija;     // akcija, npr. "zmesaj"
    private int trajanje;      // trajanje koraka v minutah
}
class Snov {     // jedilna snov
    private String naziv;      // unikaten naziv, npr. "moka" ali "palacinke"
}
```

Dopolnite sledeče metode:

(a) `public int trajanje()` v razredu `Recept` [1–12]:

Vrne trajanje recepta v minutah (torej vsoto trajanj vseh korakov).

(b) `public int prviKorakZAkcijo(String akcija)` v razredu `Recept` [13–25]:

Vrne indeks prvega koraka s podano akcijo oziroma -1, če takšnega koraka v receptu ni.

(c) `public boolean naIzhodu(Recept recept)` v razredu `Snov` [26–37]:

Vrne `true` natanko v primeru, če snov `this` nastopa na izhodu vsaj enega koraka podanega recepta.

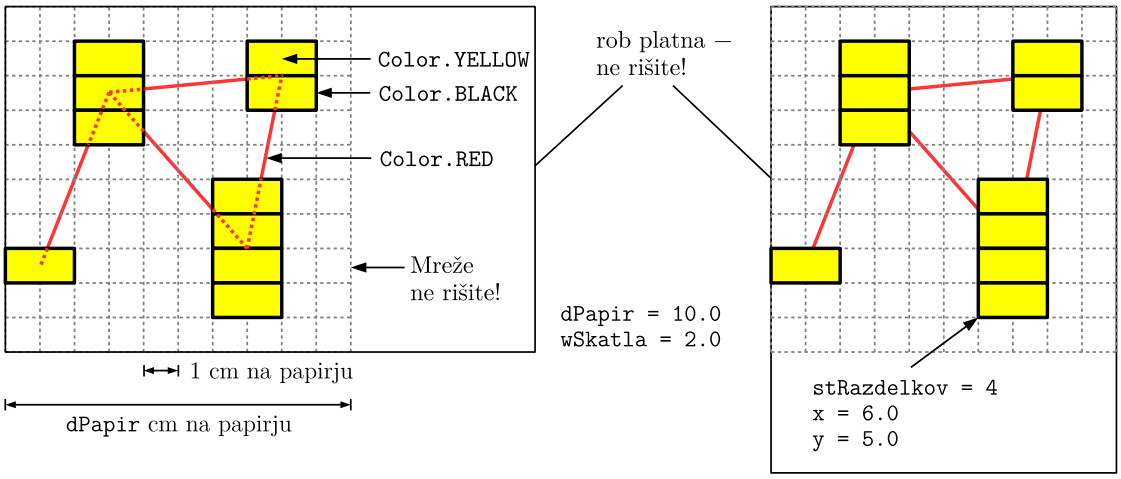
(d) `public int steviloVstopnihSnovi()` v razredu `Recept` [38–50]:

Vrne število snovi, ki se pojavljajo *samo* na vhodih v korake recepta. Lahko predpostavite, da vsaka snov nastopa na vhodu v kvečjemu enem koraku recepta.

(3) Na kvadratnem listu papirja s stranico `dPapir` cm je narisan diagram s »škatlami« in povezavami med njimi. Vse škatle so široke `wSkatla` cm, po višini pa so razdeljene na razdelke višine 1 cm. Povezave povezujejo središča škatel. Podatki o škatlah in povezavah so zbrani v tabelah `skatle` (tipa `Skatla`) in `povezave` (tipa `Povezava`):

```
private static class Skatla {
    private int stRazdelkov;   // število razdelkov (≥ 1)
    private double x, y;       // odmik (v cm) zgornjega levega kota škatle
                               //       od zgornjega levega kota lista papirja
}
private static class Povezava {
    private Skatla prva, druga;    // škatli, ki ju povezava povezuje
}
```

Dopolnite metodo `narisi` v razredu `Diagram` tako, da bo list z diagramom »preslikala« na platno. List naj bo postavljen v zgornjem levem kotu, zavzema pa naj celotno krajšo stranico platna. Zgledujte se po sledeči sliki, ki se nanaša na primer v razredu `Diagram` in na javna primera 8–9:



Dopolnite tudi sledeči metodi:

- `public double sirinaSkatle(double wp, double hp)` [1–5]:

  Na podlagi širine in višine platna vrne širino škatle v slikovnih pikah (»pikslih«) — ne v centimetrih!

- `public double[] sredisceSkatle(Skatla sk, double wp, double hp)` [6–15]:

  Vrne koordinati v »pikslih« (v obliki tabele $\{x, y\}$) središča podane škatle.

Namig: Kaj se vam splača narisati prej — škatle ali povezave?

Opombe: Škatle se med seboj ne prekrivajo. Za primere 16–25 velja `wp == hp`, za primere 16–35 pa (tudi) `povezave.length == 0`.

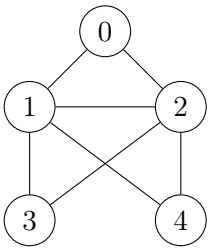
Primer izpita pri predmetu Programiranje I

Tik pred začetkom ...

- Na voljo imate 90 minut časa.
- Ni nujno, da se vam bo prva naloga zdela najlažja, tretja pa najtežja.
- Pazite, da se ne boste predolgo ukvarjali s težjimi primeri oz. podnalogami pri nalogi X, pri tem pa povsem pozabili na nalogo Y.
- Navodila za testiranje in oddajo najdete v datotekah `README.txt` v mapah, ki pripadajo posameznim nalogam.
- Na učilnico oddajte točno tisto, kar piše v navodilih. Oddajajte sproti!
- Sproti testirajte!
- Pazite na zahrbtne napake, kot so npr. zamenjava indeksov `i` in `j`, podpičje na koncu glave zanke, ...
- Najboljši zatiralec hroščev in izjem je `System.out.println`/`printf`.
- Dopolnite že pripravljene razrede. Seveda lahko vanje po potrebi dodajate tudi pomožne metode.
- Goljufe čaka Dantejev pekel.

Držimo pesti!

(1) Enostaven neusmerjen graf z vozlišči 0, 1, 2, ..., $n-1$ ($n \geqslant 2$) lahko predstavimo z *matriko sosednosti* (tabela tipa `boolean[][]` velikosti $n \times n$, v kateri element `[i][j]` pove, ali sta vozlišči i in j povezani) ali pa s *seznamom sosednosti* (nepravokotna tabela tipa `int[][]` z n vrsticami, v kateri vrstica `[i]` vsebuje indekse vozlišč, povezanih z vozliščem i). Na primer:




```
// Matrika sosednosti:
boolean[][] m = {
    {F, T, T, F, F},
    {T, F, T, T, T},
    {T, T, F, T, T},
    {F, T, T, F, F},
    {F, T, T, F, F}
}; // (T = true, F = false)
```

```
// Seznam sosednosti:
int[][] s = {
    {1, 2},
    {0, 2, 3, 4},
    {0, 1, 3, 4},
    {1, 2},
    {1, 2}
};
```

V razredu `Graf` dopolnite sledeče metode:

- `public static boolean imaEulerjevObhod(boolean[][] m)`: [1–16]

  Vrne `true` natanko v primeru, če ima graf, podan z matriko sosednosti `m`, Eulerjev obhod. To velja natanko tedaj, ko ima vsako vozlišče sodo število sosedov. Na primer, graf na gornji sliki ima Eulerjev obhod.

- `public static boolean[][] vMatriko(int[][] s)`: [17–33]

  Vrne matriko sosednosti, ki opisuje isti graf kot podani seznam sosednosti `s`.

- `public static boolean istiGraf(boolean[][] m, int[][] s)`: [34–50]

  Vrne `true` natanko v primeru, če matrika sosednosti `m` in seznam sosednosti `s` opisujeta isti graf.

(2) Zaporedje k-kotnih števil je definirano takole:

$$a_1 = 1 \xrightarrow{+k-1} a_2 \xrightarrow{+2k-3} a_3 \xrightarrow{+3k-5} a_4 \xrightarrow{+4k-7} a_5 \xrightarrow{+5k-9} a_6 \ldots$$

Na primer, zaporedje 5-kotnih števil je 1, 5, 12, 22, 35, 51, ...:

$$1 \xrightarrow{+4} 5 \xrightarrow{+7} 12 \xrightarrow{+10} 22 \xrightarrow{+13} 35 \xrightarrow{+16} 51 \ldots$$

Razred `Zaporedje` dopolnite s sledečimi konstruktorji in metodami (ter po potrebi z atributi):

- `public Zaporedje(int k)`: [1–50]

  Ustvari objekt razreda `Zaporedje`, s pomočjo katerega bo mogoče (z zaporednimi klici metode `naslednje`) tvoriti zaporedje `k`-kotnih števil.

- `public int naslednje()`: [1–50]

  Vrne naslednje število v zaporedju. Ko se metoda nad nekim objektom tipa `Zaporedje` pokliče prvič, naj vrne prvo število (to je vedno 1), ko se pokliče drugič, naj vrne drugo število v zaporedju itd.

- `public int katero()`: [17–50]

  Vrne zaporedno številko števila, ki ga bo vrnil naslednji klic metode `naslednje`.

- `public void ponastavi(int n)`: [34–50]

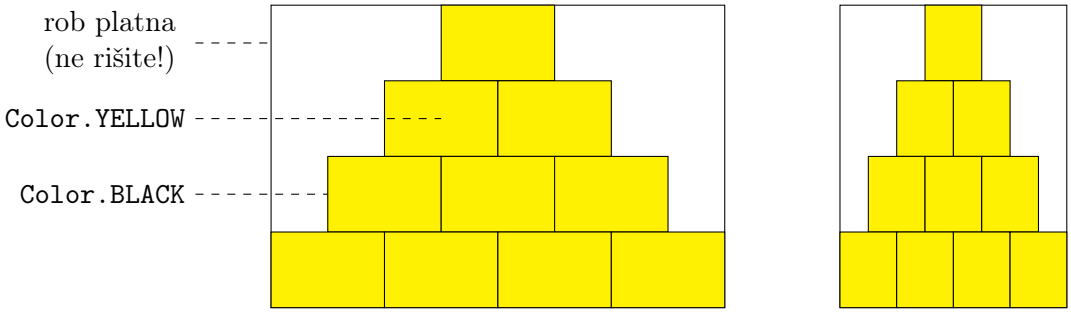
  Naredi vse, kar je treba, da bo naslednji klic metode `naslednje` vrnil `n`-to število v zaporedju. Velja $\texttt{n} \geqslant 1$.

Za lažje razumevanje si oglejmo primer uporabe razreda `Zaporedje`:

```
Zaporedje z = new Zaporedje(5);          // tvorili bomo 5-kotna števila
System.out.println( z.naslednje() );     // 1
System.out.println( z.naslednje() );     // 5
System.out.println( z.naslednje() );     // 12
System.out.println( z.katero() );        // 4 (metoda naslednje bo naslednjič
                                         //    vrnila četrto število v zaporedju)
System.out.println( z.naslednje() );     // 22
System.out.println( z.katero() );        // 5

z.ponastavi(3);
System.out.println( z.katero() );        // 3
System.out.println( z.naslednje() );     // 12
System.out.println( z.naslednje() );     // 22
```

(3) Dopolnite metodo `narisi` v razredu `Piramida` tako, da bo narisala »piramido« višine `visina`. Število `visina` (podano je kot atribut) je v vseh primerih enako najmanj 2. Sledeča slika prikazuje piramido višine 4 pri dveh različnih velikostih platna:



Piramida naj bo sestavljena iz enako velikih blokov (obrobljenih pravokotnikov) in naj po obeh dimenzijah zavzema celotno površino platna.

Dopolnite tudi sledeči metodi:

- `public double sirinaBloka(double wp, double hp)` [1–5]
- `public double visinaBloka(double wp, double hp)` [6–10]

Metodi naj vrneta širino oziroma višino posameznega bloka piramide v odvisnosti od širine (`wp`) in višine (`hp`) platna.

Izpit pri predmetu Programiranje I
3. februar 2016

Na voljo imate 90 minut časa.
Testirajte z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) V razredu `Prva` dopolnite sledeče metode:

- `public static int naslednjaPotenca2(int n)` [J1–J3, S1–S16]:

  Vrne najmanjšo potenco števila 2, ki je strogo večja od števila `n`. Na primer, za `n` = 20 naj metoda vrne 32, za `n` = 32 pa 64.

- `public static int sosedjeZRazmikom(int[] t, int r)` [J4–J7, S17–S33]:

  Vrne podatek o tem, kolikokrat je razlika med sosednjima elementoma tabele `t` enaka bodisi `r` bodisi $-$`r`. Na primer, za tabelo {10, 13, 17, 14, 17, 5, 2} in `r` = 3 naj metoda vrne 4 (ustrezni prehodi so 10 → 13, 17 → 14, 14 → 17 in 5 → 2).

- `public static int[] vsote(int[] t)` [J8–J10, S34–S50]:

  Vrne tabelo s tremi elementi, v kateri prvi element vsebuje vsoto prvega, četrtega, sedmega, ... elementa tabele `t`, drugi element vsoto drugega, petega, osmega, ... elementa tabele `t`, tretji element pa vsoto tretjega, šestega, devetega, ... elementa tabele `t`.

(2) Igralca izmenično spuščata žetone različnih barv (npr. prvi rdeče, drugi modre) v posamezne stolpce plošče z 10 vrsticami in `stStolpcev` stolpci. Igro prične prvi igralec. Dopolnite razred `Igra` s sledečimi konstruktorji in metodami:

- `public Igra(int stStolpcev)` [J1–J10, S1–S50]:

  Inicializira igro na deski s `stStolpcev` stolpci. Velja $1 \leqslant$ `stStolpcev` $\leqslant 100$.

- `public void spusti(int st)` [J1–J10, S1–S50]:

  Igralec na potezi spusti žeton v stolpec z indeksom `st` in preda potezo nasprotniku. Parameter `st` se v nobenem testnem primeru ne nanaša na stolpec, v katerem je že 10 žetonov.

- `public int naPotezi()` [J1–J2, S1–S12]:

  Vrne 1, če je na potezi prvi igralec, oziroma 2, če je na potezi drugi.

- `public int vStolpcu(int st)` [J3–J5, S13–S25]:

  Vrne število žetonov v stolpcu z indeksom `st`.

- `public int vVrstici(int vr)` [J6–J7, S26–S37]

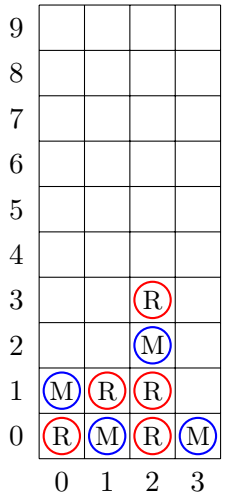
  Vrne število žetonov v vrstici z indeksom `vr`. Vrstice se štejejo **od spodaj navzgor**; spodnja vrstica ima torej indeks 0.

- `public boolean zaporedje(int igralec, int vr, int n)` [J8–J10, S38–S50]:

  Vrne `true` natanko v primeru, če se v vrstici `vr` nahaja najmanj `n` zaporednih žetonov v barvi igralca `igralec`. Velja `n` $\geqslant 1$ in `igralec` $\in \{1, 2\}$.

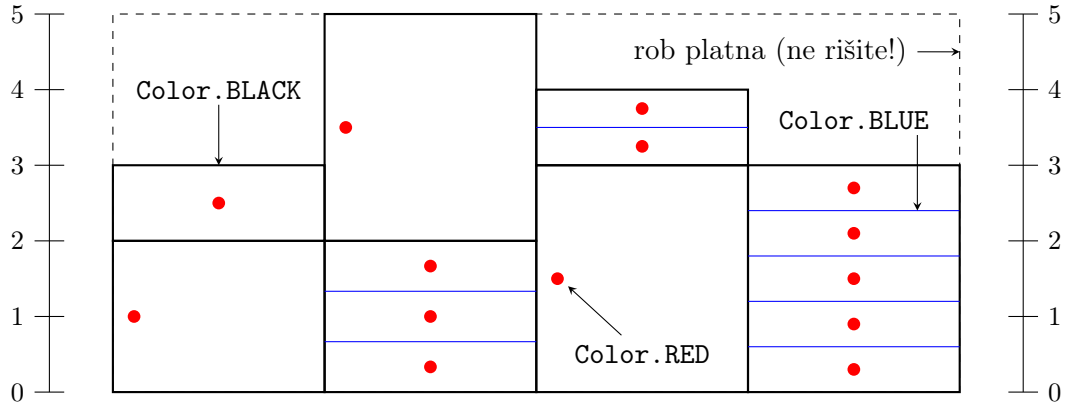
Primer (slika na desni prikazuje ploščo po izvedbi vseh klicev):

```
Igra igra = new Igra(4);
System.out.println(igra.naPotezi());  // 1
igra.spusti(2);
System.out.println(igra.naPotezi());  // 2
igra.spusti(1); igra.spusti(2);
igra.spusti(2); igra.spusti(1);
igra.spusti(3); igra.spusti(0);
igra.spusti(0); igra.spusti(2);
System.out.println(igra.naPotezi());  // 2
System.out.println(igra.vStolpcu(2));  // 4
System.out.println(igra.vStolpcu(3));  // 1
System.out.println(igra.vVrstici(1));  // 3
System.out.println(igra.vVrstici(3));  // 1
System.out.println(igra.vVrstici(4));  // 0
System.out.println(igra.zaporedje(1, 1, 2));  // true
System.out.println(igra.zaporedje(1, 2, 2));  // false
```



(3) Dopolnite metodo `narisi` v razredu `Pohistvo` tako, da bo ta na podlagi tabele `this.kosi` tipa `Kos[][]` narisala shemo pohištva po zgledu sledečega primera:

```
this.kosi: {
    { new Omara(2), new Predalnik(1, 1) },    // prvi »stolpec«
    { new Predalnik(2, 3), new Omara(3) },    // drugi »stolpec«
    { new Omara(3), new Predalnik(1, 2) },    // tretji »stolpec«
    { new Predalnik(3, 5) }                   // četrti »stolpec«
};
```



Pri vseh kosih pohištva je podana višina v relativnih enotah, pri predalnikih pa poleg tega še število predalov. Znotraj posameznih »stolpcev« so kosi nanizani od spodaj navzgor. Vsi kosi so enako široki, skupaj pa zavzemajo celotno širino platna. Višina najvišjega »stolpca« je enaka višini platna. Premer ročaja znaša 10 pik. Razdalja med levim robom vrat omare in levim robom ročaja je prav tako enaka 10 pik.

Poleg metode `narisi` dopolnite tudi sledeči metodi:

- `public double sirinaKosa(double wp, double hp)` [J1, S1–S5]:

  Vrne širino kosa pohištva za platno širine `wp` in višine `hp`.

- `public double visinaEnote(double wp, double hp)` [J2–J3, S6–S15]:

  Vrne število pik, ki ustreza enoti višine kosa pohištva.

Rešitve 1. izpitnega roka (3. februar 2016)

Prva naloga

Metoda `naslednjaPotenca2`

Želimo poiskati prvo potenco števila 2, ki je strogo večja od podanega števila `n`. Nalogo lahko rešimo tako, da pričnemo z enico ($= 2^0$), nato pa jo tako dolgo množimo z 2, dokler ne presežemo števila `n`. Dobljeno število je iskana potenca:

```
public static int naslednjaPotenca2(int n) {
    int potenca = 1;
    while (potenca <= n) {
        potenca *= 2;
    }
    return potenca;
}
```

Metoda `sosedjeZRazmikom`

Naloga je povsem premočrtna: z indeksom `i` se sprehodimo od 1 do dolžine tabele `t` minus 1, nato pa vsakokrat preverimo absolutno vrednost razlike med elementoma `t[i]` in `t[i-1]`. Če je enaka `r`, povečamo števec parov. (Ta števec na začetku seveda postavimo na 0, na koncu pa ga vrnemo.)

```
public static int sosedjeZRazmikom(int[] t, int r) {
    int stParov = 0;
    for (int i = 1;  i < t.length;  i++) {
        if (Math.abs(t[i] - t[i - 1]) == r) {
            stParov++;
        }
    }
    return stParov;
}
```

Metoda `vsote`

Na začetku izdelamo tabelo `vsote` dolžine 3, nato pa se sprehodimo po tabeli `t` in elemente sproti prištevamo k posameznim elementom tabele `vsota`. Elemente z indeksi 0, 3, 6, ... prištevamo k elementu `vsote[0]`, elemente z indeksi 1, 4, 7, ... k elementu `vsote[1]`, elemente z indeksi 2, 5, 8, ... pa k elementu `vsote[2]`. Skratka: element tabele `t` z indeksom `i` prištejemo k elementu tabele `vsote` z indeksom `i % 3`.

```
public static int[] vsote(int[] t) {
    int[] vsote = new int[3];
    for (int i = 0;  i < t.length;  i++) {
        vsote[i % 3] += t[i];
    }
    return vsote;
}
```

Druga naloga

Ključni del te naloge je določitev atributov. Potrebujemo sledeče:

- `int naPotezi`: Ta atribut ima vrednost 1, če naslednja poteza pripada prvemu igralcu, oziroma 2, če pripada drugemu.
- `int stStolpcev`: Število stolpcev igralne plošče.
- `int[][] plosca`: Gre za tabelo z 10 vrsticami in `stStolpcev` stolpci. Element na indeksu `[`i`][`j`]` ima vrednost 0, če je polje v vrstici i in stolpcu j prazno, vrednost 1, če polje vsebuje žeton prvega igralca, in vrednost 2, če polje vsebuje žeton drugega igralca. Vrstice štejemo od spodaj navzgor; spodnja vrstica ima torej indeks 0.
- `int[] zapolnjenost`: Ta atribut ni nujno potreben, a nam precej olajša reševanje naloge. Gre za tabelo s `stStolpcev` elementi. Vsak element tabele pove, koliko žetonov se trenutno nahaja v pripadajočem stolpcu igralne plošče.

Definirajmo si še konstanto `ST_VRSTIC`:

```
public class Igra {

    private static final int ST_VRSTIC = 10;

    private int[][] plosca;
    private int[] zapolnjenost;
    private int naPotezi;
    private int stStolpcev;
    ...
}
```

V konstruktorju inicializiramo vse štiri atribute:

```
public class Igra {
    ...
    public Igra(int stStolpcev) {
        this.stStolpcev = stStolpcev;
        this.plosca = new int[ST_VRSTIC][stStolpcev];
        this.zapolnjenost = new int[stStolpcev];
        this.naPotezi = 1;
    }
    ...
}
```

V metodi `spusti` moramo v ustrezni element tabele `this.plosca` vpisati številko igralca na potezi, torej `this.naPotezi`. Žeton postavimo v stolpec `st` (ki ga metoda prejme kot parameter) in v vrstico `this.zapolnjenost[st]` (tj. trenutno število žetonov v stolpcu `st`). Poleg tega moramo povečati zapolnjenost stolpca `st` in predati potezo nasprotnemu igralcu.

```
public class Igra {
    ...
    public void spusti(int st) {
        this.plosca[this.zapolnjenost[st]][st] = this.naPotezi;
        this.zapolnjenost[st]++;
        this.naPotezi = 3 - this.naPotezi;
    }
```

```
    ...
}
```

Zahvaljujoč atributoma `naPotezi` in `zapolnjenost` sta metodi `naPotezi` in `vStolpcu` resnično enostavni:

```
public class Igra {
    ...
    public int naPotezi() {
        return this.naPotezi;
    }

    public int vStolpcu(int st) {
        return this.zapolnjenost[st];
    }
    ...
}
```

V metodi `vVrstici` se sprehodimo po podani vrstici tabele `this.plosca` in preštejemo elemente z vrednostjo nad 0.

```
public class Igra {
    ...
    public int vVrstici(int vr) {
        int stZetonov = 0;
        for (int i = 0;  i < this.stStolpcev;  i++) {
            if (this.plosca[vr][i] > 0) {
                stZetonov++;
            }
        }
        return stZetonov;
    }
    ...
}
```

V metodi `zaporedje` se prav tako sprehodimo po vrstici `vr`, ki jo metoda prejme kot parameter. Med sprehodom vzdržujemo števec `k`. Če naletimo na element z vrednostjo `igralec` (to pomeni, da se v pripadajočem polju igralne plošče nahaja žeton v barvi igralca `igralec`), povečamo števec `k` za 1. V nasprotnem primeru postavimo števec `k` na 0. Če števec `k` prispe do vrednosti `n`, vemo, da smo pravkar videli `n` zaporednih žetonov v barvi igralca `igralec`. V takšnem primeru lahko takoj vrnemo `true`.

```
public class Igra {
    ...
    public boolean zaporedje(int igralec, int vr, int n) {
        int k = 0;
        for (int j = 0;  j < this.stStolpcev;  j++) {
            if (this.plosca[vr][j] == igralec) {
                k++;
                if (k == n) {
                    return true;
                }
            } else {
                k = 0;
            }
        }
```

```
        return false;
    }
}
```

Tretja naloga

Ker je število kosov po širini platna enako `this.kosi.length`, vsi kosi pa so enako široki, širine posameznega kosa ni težko izračunati:

```
public class Pohistvo extends Platno {
    ...
    public double sirinaKosa(double wp, double hp) {
        return (wp / this.kosi.length);
    }
    ...
}
```

Pri metodi `visinaEnote` pa se za odtenek bolj prepotimo. Ker se najvišji stolpec razteza čez celotno višino platna (`hp`), je višina relativne enote v pikah (»pikslih«) enaka `hp / vvMax`, kjer `vvMax` označuje višino najvišjega stolpca v relativnih enotah. To višino pa izračunamo tako, da se sprehodimo po tabeli `this.kosi` in za vsako njeno vrstico (oziroma za vsak stolpec naše sobe) seštejemo višine kosov pohištva v relativnih enotah. Maksimum določimo na način, ki smo ga že velikokrat uporabili.

```
public class Pohistvo extends Platno {
    ...
    public double visinaEnote(double wp, double hp) {
        int vvMax = 0;
        for (int i = 0;  i < this.kosi.length;  i++) {
            int vv = vsotaVisin(this.kosi[i]);
            if (vv > vvMax) {
                vvMax = vv;
            }
        }
        return hp / vvMax;
    }

    private static int vsotaVisin(Kos[] kosi) {
        int vsota = 0;
        for (int i = 0;  i < kosi.length;  i++) {
            vsota += kosi[i].vrniVisino();
        }
        return vsota;
    }
    ...
}
```

Pohištvo narišemo s sprehodom po vseh elementih tabele `this.kosi`. Širino posameznega kosa v pikah izračunamo z metodo `sirinaKosa`, višino v pikah pa pridobimo tako, da višino relativne enote v pikah (metoda `visinaEnote`) pomnožimo z višino kosa v relativnih enotah. Obenem vzdržujemo spremenljivki `x` in `y`, ki označujeta trenutni položaj na platnu. Upoštevati moramo, da se v smeri x pomikamo desno (`x` najprej postavimo na 0, nato pa ga vsakokrat povečamo za širino kosa v pikah), v smeri y pa navzgor (v okviru

posameznega stolpca sobe `y` najprej postavimo na `hp`, nato pa ga vsakokrat zmanjšamo za višino trenutnega kosa).

```
public class Pohistvo extends Platno {
    ...
    protected void narisi(Graphics2D g, double wp, double hp) {
        double x = 0.0;
        double wKos = this.sirinaKosa(wp, hp);
        double hEnota = this.visinaEnote(wp, hp);

        for (int i = 0;  i < this.kosi.length;  i++) {
            double y = hp;

            for (int j = 0;  j < this.kosi[i].length;  j++) {
                Kos kos = this.kosi[i][j];
                double hKos = kos.vrniVisino() * hEnota;
                y -= hKos;
                g.setColor(Color.BLACK);
                g.drawRect(ri(x), ri(y), ri(wKos), ri(hKos));
                kos.narisiDodatke(g, x, y, wKos, hKos);
            }
            x += wKos;
        }
    }
}
```

Metoda `narisiDodatke` nariše ročaj omare oziroma ročaje posameznih predalov in meje med njimi v predalniku. Poleg objekta za risanje metoda sprejme še koordinati zgornjega levega kota kosa ter širino in višino kosa v pikah. Ker se metoda za omare obnaša drugače kot za predalnike, je najenostavneje, če jo v razredu `Kos` deklariramo kot abstraktno, v razredih `Omara` in `Predalnik` pa jo ustrezno definiramo.

```
public class Pohistvo extends Platno {

    // odmik ročaja od levega roba omare
    private static final double ODMIK_ROCAJA = 10.0;

    // premer ročaja
    private static final double PREMER_ROCAJA = 10.0;

    ...

    public static abstract class Kos {
        ...
        public abstract void narisiDodatke(
                Graphics2D g, double x, double y, double w, double h);
    }

    public static class Omara extends Kos {
        ...
        public void narisiDodatke(Graphics2D g, double x, double y, double w, double h) {
            g.setColor(Color.RED);
            g.fillOval(ri(x + ODMIK_ROCAJA), ri(y + (h - PREMER_ROCAJA) / 2.0),
                       ri(PREMER_ROCAJA), ri(PREMER_ROCAJA));
        }
    }
```

```
    public static class Predalnik extends Kos {
        ...
        public void narisiDodatke(Graphics2D g, double x, double y, double w, double h) {
            double hPredal = h / this.stPredalov;
            double yp = y;

            for (int i = 0;  i < this.stPredalov;  i++) {
                if (i > 0) {
                    g.setColor(Color.BLUE);
                    g.drawLine(ri(x), ri(yp), ri(x + w), ri(yp));
                }
                g.setColor(Color.RED);
                g.fillOval(ri(x + (w - PREMER_ROCAJA) / 2.0),
                           ri(yp + (hPredal - PREMER_ROCAJA) / 2.0),
                           ri(PREMER_ROCAJA),
                           ri(PREMER_ROCAJA));
                yp += hPredal;
            }
        }
    }
    ...
}
```

Izpit pri predmetu Programiranje I
18. februar 2016

Na voljo imate 90 minut časa.
Testirajte z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) V razredu `Prva` dopolnite sledeče metode:

- `public static int prviPraviDelitelj(int n)` [J1–J3, S1–S16]:

  Vrne najmanjši delitelj števila `n`, ki je večji od 1. Na primer, za `n` = 20 naj metoda vrne 2, za `n` = 35 pa 5. Velja `n` $\geqslant$ 2.

- `public static boolean uravnotezeno(boolean[] t)` [J4–J7, S17–S33]:

  Vrne `true`, če je število elementov `true` v tabeli `t` enako številu elementov `false`. V nasprotnem primeru naj metoda vrne `false`.

- `public static int[] rotacija(int[] t, int r)` [J8–J10, S34–S50]:

  Vrne rotacijo tabele `t` za `r` mest v desno. Na primer, pri tabeli {10, 20, 30, 40, 50} bi klic metode za `r` = 1 vrnil tabelo {50, 10, 20, 30, 40}, za `r` = 2 bi dobili tabelo {40, 50, 10, 20, 30}, za `r` = −1 tabelo {20, 30, 40, 50, 10}, za `r` = 42 pa tabelo {40, 50, 10, 20, 30}.

(2) Knjižnica ima določeno število članov in knjižnih naslovov, za vsak naslov pa premore po 10 izvodov knjig. Tako člani kot naslovi so oštevilčeni z indeksi od 0 naprej. Dopolnite razred `Knjiznica` s sledečimi konstruktorji in metodami:

- `public Knjiznica(int stClanov, int stNaslovov)` [J1–J10, S1–S50]:

  Inicializira objekt, ki predstavlja knjižnico s `stClanov` člani in `stNaslovov` knjižnimi naslovi.

- `public void posodi(int clan, int naslov)` [J1–J10, S1–S50]:

  Knjižnica posodi članu `clan` en izvod knjige z naslovom `naslov`. Če so trenutno vsi izvodi tega naslova posojeni, se ne zgodi nič. (Parametra `clan` in `naslov` sta seveda indeksa člana in knjižnega naslova.)

- `public void pridobiNazaj(int clan)` [J5–J10, S21–S50]:

  Član `clan` vrne vse svoje izposojene knjige.

- `public int naVoljo(int naslov)` [J1–J2, S1–S12]:

  Vrne število izvodov knjige z naslovom `naslov`, ki jih ima knjižnica trenutno na voljo za posojo.

- `public int posojeni(int clan, int naslov)` [J3–J5, S13–S25]:

  Vrne število izvodov knjige z naslovom `naslov`, ki jih ima član `clan` trenutno v izposoji.

- `public int posojeni()` [J6–J7, S26–S37]:

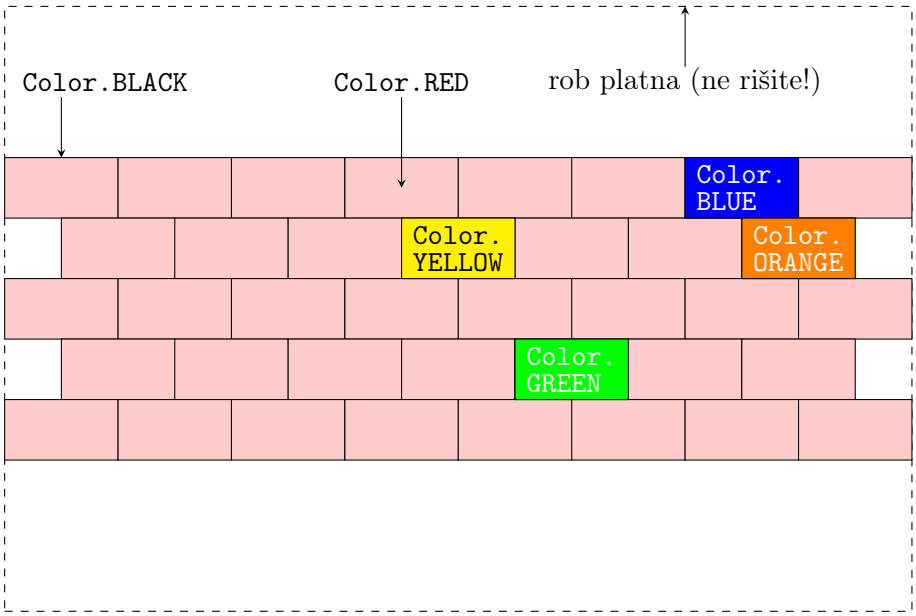
  Vrne skupno število izvodov, ki jih imajo člani knjižnice trenutno v izposoji.

- `public int najClan(int naslov)` [J8–J10, S38–S50]:

  Vrne indeks člana z največ izposojenimi izvodi knjige z naslovom `naslov`. Če je takih članov več, vrne tistega z najmanjšim indeksom.

(3) Dopolnite metodo `narisi` v razredu `Zid` tako, da bo ta na podlagi spremenljivk `this.stVrstic` in `this.stStolpcev` (tipa `int`) ter tabele `this.posebniZidaki` (tipa `Zidak[]`) narisala zid po zgledu sledečega primera:

```
this.stVrstic: 5
this.stStolpcev: 8
this.posebniZidaki: { new Zidak(1, 3, Color.YELLOW), new Zidak(0, 6, Color.BLUE),
                      new Zidak(3, 4, Color.GREEN), new Zidak(1, 6, Color.ORANGE) }
```



Zid zavzema celotno širino in polovico višine platna. Po višini se nahaja na sredini platna. Vsi zidaki so enako veliki. Vsaka druga vrstica zidu je zamaknjena za polovico širine zidaka. Pri posebnih zidakih (tistih z drugačno barvo polnila) so podani indeks vrstice, indeks stolpca in barva polnila.

Poleg metode `narisi` dopolnite tudi sledeče metode:

- `public double sirinaZidaka(double wp, double hp)` [J1, S1–S5]:

  Vrne širino zidaka za platno širine `wp` in višine `hp`.

- `public double visinaZidaka(double wp, double hp)` [J2, S6–S10]:

  Vrne višino zidaka.

- `public double yZgornjiRob(double wp, double hp)` [J3, S11–S15]:

  Vrne koordinato y zgornjega roba zidu.

- `public double[] xyZidaka(int vr, int st, double wp, double hp)` [J4–J5, S16–S25]:

  Vrne koordinati zgornjega roba zidaka v vrstici z indeksom `vr` in stolpcu z indeksom `st` v obliki tabele z dvema elementoma (prvi element je koordinata x, drugi pa y).

Rešitve 2. izpitnega roka (18. februar 2016)

Prva naloga

Metoda `prviPraviDelitelj`

Zanima nas najmanjši delitelj števila `n`, ki je večji od 1. Zaradi predpostavke `n >= 2` tak delitelj vedno obstaja. Poiščemo ga tako, da števec, označimo ga z `d`, nastavimo na 2, nato pa ga v zanki povečujemo. Pred vsakim obhodom zanke preverimo, ali je število `n` deljivo s številom `d`. Če je, zanko prekinemo in vrnemo `d`, sicer pa nadaljujemo z izvajanjem zanke.

```
public static int prviPraviDelitelj(int n) {
    int d = 2;
    while (n % d != 0) {
        d++;
    }
    return d;
}
```

Metoda `uravnotezeno`

Preveriti želimo, ali je število elementov z vrednostjo `true` v tabeli `t` enako številu elementov z vrednostjo `false`. Odgovor na to vprašanje dobimo s števcem in obhodom po tabeli:

```
public static boolean uravnotezeno(boolean[] t) {
    int stTrue = 0;
    for (int i = 0;  i < t.length;  i++) {
        if (t[i]) {
            stTrue++;
        }
    }
    return (stTrue == t.length - stTrue);
}
```

Metoda `rotacija`

V metodi `rotacija` moramo izdelati in napolniti novo tabelo, ki je glede na tabelo `t` zarotirana za `r` mest v desno. Novo tabelo bomo poimenovali `rezultat`.

Za začetek se osredotočimo na primer `0 <= r < t.length`. Hitro vidimo, da velja sledeče:

```
rezultat[0] = t[t.length - r]
rezultat[1] = t[t.length - r + 1]
...
rezultat[r - 1] = t[t.length - r + (r - 1)]

rezultat[r] = t[0]
rezultat[r + 1] = t[1]
...
rezultat[r + (t.length - r - 1)] = t[t.length - r - 1]
```

Gornja pravila lahko prepišemo v dve ločeni zanki: eno potrebujemo za prvih `r` elementov tabele `rezultat`, drugo pa za preostalih `t.length - r` elementov. Lahko pa pravila zapišemo nekoliko drugače in s tem število zank zmanjšamo na eno samo:

```
rezultat[(i + r) % t.length] = t[i]
```

To pravilo velja za vsak `i` od 0 do vključno `t.length - 1`, torej za vse elemente tabel `t` in `rezultat`. Pravilo lahko neposredno prepišemo v zanko:

```
int[] rezultat = new int[t.length];
for (int i = 0;  i < t.length;  i++) {
    rezultat[(i + r) % t.length] = t[i];
}
```

Gornja zanka deluje tudi za primer `r >= t.length`. Zakaj? Zato, ker je rotacija za `t.length` (in tudi za `2 * t.length`, za `3 * t.length` itd.) mest enakovredna rotaciji za 0 mest, rotacija za `t.length + 1` (in tudi za `2 * t.length + 1`, za `3 * t.length + 1` itd.) mest je enakovredna rotaciji za 1 mesto itd. Rotacija za `r` mest je torej enakovredna rotaciji za `r % t.length` mest.

Kako pa je z rotacijo za negativno število mest? Pravilo, da je rotacija za `r` mest enakovredna rotaciji za `r % t.length` mest, velja tudi za negativne vrednosti `r`, vendar pa vrednost izraza `r % t.length` za negativne `r` znaša med `-t.length + 1` in vključno 0, kar nam lahko povzroči težave pri dostopu do elementov tabele `rezultat`. K sreči pa obstaja preprosta rešitev: izrazu `(i + r) % t.length` prištejemo `t.length`, da v vsakem primeru pridemo na pozitivno stran številske osi, nato pa dobljeni rezultat ponovno delimo s `t.length` in izračunamo ostanek. Na ta način indeks v tabelo `rezultat` zanesljivo spravimo v interval od 0 do vključno `t.length - 1`.

Metodo `rotacija` lahko potemtakem zapišemo takole:

```
public static int[] rotacija(int[] t, int r) {
    int[] rezultat = new int[t.length];
    for (int i = 0;  i < t.length;  i++) {
        rezultat[((i + r) % t.length + t.length) % t.length] = t[i];
    }
    return rezultat;
}
```

Druga naloga

Za začetek premislimo o atributih razreda `Knjiznica`. Ni težko ugotoviti, da je smiselno hraniti število članov in število knjižnih naslovov. Pripadajoča atributa bomo poimenovali `stClanov` in `stNaslovov`. Poleg tega pa moramo za vsakega člana hraniti število izposojenih izvodov za posamezne naslove, saj si je v knjižnici možno izposoditi tudi več izvodov istega naslova. Podatke o izposojah posameznih članov bomo hranili v tabeli `izposoje` tipa `int[][]`. Vrstice te tabele pripadajo članom, stolpci naslovom, element na položaju `[c][i]` pa pove, koliko izvodov naslova `i` ima član `c` trenutno v izposoji.

Z atributi `stClanov`, `stNaslovov` in `izposoje` bi že lahko shajali, kljub temu pa nam bo koristilo, če bomo vzdrževali še število izvodov posameznih naslovov, ki jih ima knjižnica trenutno na voljo. Te podatke bomo hranili v tabeli `stIzvodovNaVoljo` tipa `int[]`, v kateri element na položaju `[i]` pove, koliko izvodov naslova `i` ima knjižnica trenutno na voljo.

Da se izognemo uporabi »grdih« fiksnih števil, bomo poleg navedenih atributov definirali še konstanto `ST_IZVODOV_NA_NASLOV`:

```
public class Knjiznica {

    private static final int ST_IZVODOV_NA_NASLOV = 10;

    private int stClanov;
    private int stNaslovov;
    private int[] stIzvodovNaVoljo;
    private int[][] izposoje;
    ...
}
```

V konstruktorju, kot smo že vajeni, inicializiramo atribute. Atributa `stClanov` in `stNaslovov` nastavimo na vrednosti, ki sta podani kot parametra konstruktorja, začetno število razpoložljivih izvodov je za vsak naslov enako `ST_IZVODOV_NA_NASLOV`, noben član pa na začetku nima ničesar v izposoji:

```
public class Knjiznica {
    ...
    public Knjiznica(int stClanov, int stNaslovov) {
        this.stClanov = stClanov;
        this.stNaslovov = stNaslovov;
        this.stIzvodovNaVoljo = new int[stNaslovov];
        for (int i = 0;  i < stNaslovov;  i++) {
            this.stIzvodovNaVoljo[i] = ST_IZVODOV_NA_NASLOV;
        }
        this.izposoje = new int[stClanov][stNaslovov];
    }
    ...
}
```

Knjižnica lahko članu `clan` izvod naslova `naslov` posodi le v primeru, če ima vsaj enega na voljo. No, če to drži, število razpoložljivih izvodov naslova `naslov` zmanjšamo za 1, obenem pa povečamo število izvodov tega naslova, ki jih ima član `clan` v izposoji:

```
public class Knjiznica {
    ...
    public void posodi(int clan, int naslov) {
        if (this.stIzvodovNaVoljo[naslov] > 0) {
            this.stIzvodovNaVoljo[naslov]--;
            this.izposoje[clan][naslov]++;
        }
    }
    ...
}
```

Ko član `clan` vrne izposojene knjige, se v knjižnici ustrezno poveča zaloga izvodov, članu `clan` pa se število izposojenih izvodov za vse knjižne naslove zmanjša na 0:

```
public class Knjiznica {
    ...
    public void pridobiNazaj(int clan) {
        for (int i = 0;  i < this.izposoje[clan].length;  i++) {
            this.stIzvodovNaVoljo[i] += this.izposoje[clan][i];
```

```
            this.izposoje[clan][i] = 0;
        }
    }
    ...
}
```

Zaradi ustrezne izbire atributov sta metoda `naVoljo` in prva metoda `posojeni` sila preprosti:

```
public class Knjiznica {
    ...
    public int naVoljo(int naslov) {
        return this.stIzvodovNaVoljo[naslov];
    }

    public int posojeni(int clan, int naslov) {
        return this.izposoje[clan][naslov];
    }
    ...
}
```

Pri drugi metodi `posojeni` moramo sešteti izposoje vseh naslovov pri vseh članih, torej vse elemente tabele `this.izposoje`:

```
public class Knjiznica {
    ...
    public int posojeni() {
        int stPosojenih = 0;
        for (int c = 0;  c < this.stClanov;  c++) {
            for (int i = 0;  i < this.stNaslovov;  i++) {
                stPosojenih += this.izposoje[c][i];
            }
        }
        return stPosojenih;
    }
    ...
}
```

Člana z največ izposojenimi izvodi naslova `naslov` določimo tako, da se sprehodimo po stolpcu z indeksom `naslov` v tabeli `this.izposoje` in v njem poiščemo prvi element z maksimalno vrednostjo:

```
public class Knjiznica {
    ...
    public int najClan(int naslov) {
        int cNaj = 0;
        for (int c = 0;  c < this.stClanov;  c++) {
            if (this.izposoje[c][naslov] > this.izposoje[cNaj][naslov]) {
                cNaj = c;
            }
        }
        return cNaj;
    }
}
```

Tretja naloga

Število zidakov po širini je enako `this.stStolpcev`, zato širina posameznega zidaka znaša ...

```
public class Zid extends Platno {
    ...
    public double sirinaZidaka(double wp, double hp) {
        return (wp / this.stStolpcev);
    }
    ...
}
```

Tudi višine zidaka ni prav nič težko izračunati: pri višini platna `hp` je zid visok `(hp / 2)` pik, zatorej je posamezen zidak visok ...

```
public class Zid extends Platno {
    ...
    public double visinaZidaka(double wp, double hp) {
        return ((hp / 2.0) / this.stVrstic);
    }
    ...
}
```

Ker zid zavzema polovico višine platna, je koordinata y njegovega zgornjega roba enaka ...

```
public class Zid extends Platno {
    ...
    public double yZgornjiRob(double wp, double hp) {
        return (hp / 4.0);
    }
    ...
}
```

Koordinato y zidaka v vrstici `vr` in stolpcu `st` dobimo preprosto tako, da koordinati y zgornjega roba zidu prištejemo `vr`-kratnik višine zidaka. Koordinato x v lihih vrsticah (oziroma pri sodih vrednostih `vr`) izračunamo kot `st`-kratnik širine zidaka, v sodih vrsticah pa dobljenemu rezultatu prištejemo še polovico širine zidaka. Navedeno pravilo bi lahko zapisali s pomočjo pogojnega stavka, gre pa tudi elegantneje:

```
public class Zid extends Platno {
    ...
    public double[] xyZidaka(int vr, int st, double wp, double hp) {
        double wZidak = this.sirinaZidaka(wp, hp);
        double hZidak = this.visinaZidaka(wp, hp);
        double xZidak = (0.5 * (vr % 2) + st) * wZidak;
        double yZidak = this.yZgornjiRob(wp, hp) + vr * hZidak;
        return new double[]{xZidak, yZidak};
    }
}
```

Pri risanju se najprej pretvarjamo, kot da posebnih zidakov sploh ni, in narišemo zid, sestavljen iz samih navadnih (rdečih) zidakov. V drugi fazi pa na ustrezna mesta postavimo

posebne zidake. Zid narišemo s standardno dvojno zanko, pri kateri moramo paziti le na zamik in manjše število zidakov v sodih vrsticah. Koda za izris posebnih zidakov pa je še enostavnejša, saj se moramo zgolj sprehoditi čez tabelo `this.posebniZidaki` in za vsak zidak s pomočjo metode `xyZidaka` izračunati njegov položaj na platnu.

```
public class Zid extends Platno {
    ...
    protected void narisi(Graphics2D g, double wp, double hp) {
        double wZidak = this.sirinaZidaka(wp, hp);
        double hZidak = this.visinaZidaka(wp, hp);
        double hZid = hp / 2.0;
        double xLevo = 0.0;
        double y = this.yZgornjiRob(wp, hp);

        for (int i = 0;  i < this.stVrstic;  i++) {
            double x = (i % 2 == 0) ? (0.0) : (wZidak / 2.0);
            for (int j = i % 2;  j < this.stStolpcev;  j++) {
                g.setColor(Color.RED);
                g.fillRect(ri(x), ri(y), ri(wZidak), ri(hZidak));
                g.setColor(Color.BLACK);
                g.drawRect(ri(x), ri(y), ri(wZidak), ri(hZidak));
                x += wZidak;
            }
            y += hZidak;
        }

        for (int i = 0;  i < this.posebniZidaki.length;  i++) {
            Zidak zidak = this.posebniZidaki[i];
            double[] xy = this.xyZidaka(
                    zidak.vrniVrstico(), zidak.vrniStolpec(), wp, hp);
            g.setColor(zidak.vrniBarvo());
            g.fillRect(ri(xy[0]), ri(xy[1]), ri(wZidak), ri(hZidak));
            g.setColor(Color.BLACK);
            g.drawRect(ri(xy[0]), ri(xy[1]), ri(wZidak), ri(hZidak));
        }
    }
}
```

Izpit pri predmetu Programiranje I
7. september 2016

Na voljo imate 90 minut časa.
Testirajte z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) V razredu `Prva` dopolnite sledeče metode:

- `public static boolean jePotenca(int n, int osnova)` [J1–J3, S1–S16]:

  Če je število `n` potenca števila `osnova`, vrne `true`, sicer pa vrne `false`. Velja $\texttt{n} \geqslant 1$ in $\texttt{osnova} \geqslant 2$.

- `public static int zadnjiManjsi(int[] t, int n)` [J4–J6, S17–S33]:

  Vrne indeks zadnjega elementa v naraščajoče urejeni tabeli `t`, ki je strogo manjši od števila `n`. Če je že prvi element tabele enak najmanj `n`, naj metoda vrne vrednost -1. Na primer, za tabelo $\{10, 20, 30, 40\}$ naj metoda pri $\texttt{n} = 21$ vrne vrednost 1, pri $\texttt{n} = 20$ vrednost 0, pri $\texttt{n} \leqslant 10$ pa vrednost -1.

- `public static int[][] zmnozki(int[] a, int[] b)` [J7–J10, S34–S50]:

  Vrne tabelo, v kateri je element v i-ti vrstici in j-tem stolpcu zmnožek i-tega elementa tabele `a` in j-tega elementa tabele `b`.

(2) Banka vzdržuje podatke o računih svojih komitentov. Račun ima unikatno številko (niz oblike SI56 XXXX XXXX XXXX XXX) in stanje — podatek o količini denarja, ki ga ima komitent na voljo. Razred `Banka` dopolnite s sledečimi metodami:

- `public void odpriRacun(String stRacuna)` [J1–J10, S1–S50]:

  Odpre (ustvari) račun s podano unikatno številko in z 0 enotami denarja. Metoda se v nobenem testnem primeru ne pokliče več kot 100-krat. V testnih primerih J1–J4 (S1–S20) se pokliče natanko enkrat, kar pomeni, da vseskozi rokujemo z istim računom.

- `public int stanje(String stRacuna)` [J1–J10, S1–S50]:

  Vrne stanje na računu s podano številko. Lahko predpostavite, da račun že obstaja. To velja tudi za ostale podnaloge.

- `public void polozi(String stRacuna, int znesek)` [J1–J10, S1–S50]:

  Doda `znesek` enot denarja na račun s podano številko. Znesek je vedno pozitiven. To velja tudi za ostale podnaloge.

- `public boolean dvigni(String stRacuna, int znesek)` [J4–J10, S16–S50]:

  Če je na računu dovolj denarja (najmanj `znesek` enot), odvzame z njega `znesek` enot denarja in vrne `true`, sicer pa ne naredi ničesar in zgolj vrne `false`.

- `public boolean prenesi(String izvor, String cilj, int zn)` [J6–J10, S26–S50]:

  Če je na računu s številko `izvor` dovolj denarja, prenese `zn` enot denarja z računa s številko `izvor` na račun s številko `cilj` in vrne `true`, sicer pa zgolj vrne `false`.

- `public String najvecPologov()` [J8–J10, S36–S50]:

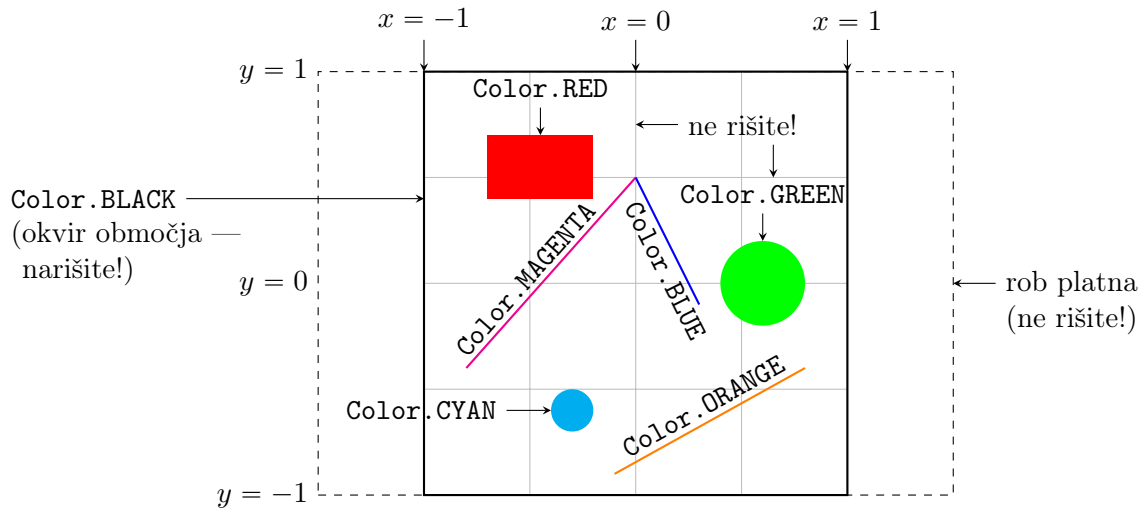
  Vrne številko računa, na katerega je bil največkrat položen ali prenešen denar. Lahko predpostavite, da obstaja natanko en tak račun. Prenos naj se upošteva le v primeru, če je bil dejansko izveden (če je metoda `prenesi` vrnila `true`).

(3) Dopolnite metodo `narisi` v razredu `Tretja` tako, da bo ta na podlagi tabele `this.elementi` tipa `Element[]` narisala tloris urbanega območja. Območje ima obliko kvadrata s koordinatami oglišč $(-1,-1)$, $(1,-1)$, $(1,1)$ in $(-1,1)$. Vsebuje različne elemente — stavbe, drevesa in cestne odseke. Elementi so v datoteki `Tretja.java` predstavljeni kot objekti sledečih razredov:

```
abstract class Element {
    private Color barva;     // barva elementa
}
class Stavba extends Element {   // stavba v obliki pravokotnika
    private double x, y;     // koordinati zgornjega levega oglišča
    private double w, h;     // širina (w) in višina (h)
}
class Drevo extends Element {    // drevo v obliki kroga
    private double x, y;     // koordinati središča
    private double r;        // polmer
}
class CestniOdsek extends Element {    // cestni odsek v obliki daljice
    private double x1, y1;   // koordinati prvega krajišča
    private double x2, y2;   // koordinati drugega krajišča
}
```

Območje naj se nahaja na sredini platna, po krajši stranici pa naj zavzema celotno površino platna. Elemente narišite v takšnem vrstnem redu, kot so navedeni v tabeli `this.elementi`. Sledi primer tabele in pripadajoče slike:

```
{ new Stavba(Color.RED, -0.7, 0.7, 0.5, 0.3),
  new Drevo(Color.GREEN, 0.6, 0.0, 0.2),
  new CestniOdsek(Color.MAGENTA, -0.8, -0.4, 0.0, 0.5),
  new CestniOdsek(Color.BLUE, 0.0, 0.5, 0.3, -0.1),
  new CestniOdsek(Color.ORANGE, 0.8, -0.4, -0.1, -0.9),
  new Drevo(Color.CYAN, -0.3, -0.6, 0.1) }
```



Poleg metode `narisi` dopolnite tudi metode `stranicaObmocja`, `xObmocja` in `yObmocja`. Metoda `stranicaObmocja` vrne dolžino stranice območja v slikovnih pikah (»pikslih«) pri podani širini in višini platna, metodi `xObmocja` in `yObmocja` pa koordinati zgornjega levega kota območja, merjeni v slikovnih pikah.

Rešitve 3. izpitnega roka (7. september 2016)

Prva naloga

Metoda `jePotenca`

Število `n` je potenca števila `osnova`, če velja `n == osnova * osnova * ... * osnova`. Zato lahko število `n` zaporedoma delimo s številom `osnova`, dokler ne pridemo do 1. Če se deljenje v nekem koraku ne izide, vemo, da `n` *ni* potenca števila `osnova`. Če prispemo do enice, pa vemo, da je.

```
public static boolean jePotenca(int n, int osnova) {
    while (n > 1 && n % osnova == 0) {
        n /= osnova;
    }
    return (n == 1);
}
```

Metoda `zadnjiManjsi`

Ker je tabela urejena, najučinkovitejši algoritem temelji na bisekciji. Vendar pa je tabela v vseh testnih primerih dovolj majhna, da se lahko po njej preprosto pomikamo, dokler ne prispemo bodisi do konca ali pa do elementa, ki je večji ali enak številu `n`. Iskani indeks je za eno manjši od indeksa elementa, na katerem se ustavimo.

```
public static int zadnjiManjsi(int[] t, int n) {
    int i = 0;
    while (i < t.length && t[i] < n) {
        i++;
    }
    return (i - 1);
}
```

Ali metoda pravilno deluje tudi za primer, ko je že prvi element tabele večji ali enak številu `n`? Da! Zanka se v tem primeru niti enkrat ne izvede, kar pomeni, da števec `i` ostane na ničli, vrnemo pa -1.

Metoda `zmnozki`

Pripravimo si tabelo `t` velikosti `a.length` krat `b.length`, nato pa se po njej sprehodimo z dvojno zanko. Vrednost elementa `t[i][j]` izračunamo kot zmnožek elementov `a[i]` in `b[j]`.

```
public static int[][] zmnozki(int[] a, int[] b) {
    int[][] t = new int[a.length][b.length];
    for (int i = 0;  i < a.length;  i++) {
        for (int j = 0;  j < b.length;  j++) {
            t[i][j] = a[i] * b[j];
        }
    }
    return t;
}
```

Druga naloga

Ker se metoda `odpriRacun` v nobenem testnem primeru ne pokliče več kot 100-krat, vemo, da bomo operirali z največ 100 računi:

```
public class Druga {

    private static final int MAKS_ST_RACUNOV = 100;
    ...
}
```

Račun ima svojo številko (ki je v resnici niz, ne celo število) in stanje (celo število). Ker gre za sestavljeni podatkovni tip, pri katerem imajo elementi različen pomen, je posamezen račun najbolj smiselno predstaviti kot objekt razreda `Racun`:

```
public class Druga {
    ...
    public static class Racun {

        private String stevilka;
        private int stanje;

        public Racun(String stevilka, int stanje) {
            this.stevilka = stevilka;
            this.stanje = stanje;
        }
    }
    ...
}
```

Banka hrani podatke o posameznih računih. Ker je maksimalno število računov vnaprej znano, nam bo tabela dobro služila:

```
public class Druga {
    ...
    public static class Banka {

        private Racun[] racuni;
        private int stRacunov;

        public Banka() {
            this.racuni = new Racun[MAKS_ST_RACUNOV];
        }
    }
}
```

Atribut `stRacunov` hrani *dejansko* število računov v tabeli `racuni`. V tabeli bo vedno zasedenih vseh `MAKS_ST_RACUNOV` celic, vendar pa bo samo prvih `stRacunov` celic vsebovalo veljavne račune (oziroma, če smo natančnejši, reference na objekte, ki predstavljajo odprte račune). Ostale celice bodo vsebovale vrednost `null`.

Metodo `odpriRacun` sprogramiramo tako, da ustvarimo nov objekt tipa `Racun` in referenco nanj shranimo v prvo prosto celico tabele `racuni`:

```
public class Druga {
    ...
    public static class Banka {
        ...
        public void odpriRacun(String stRacuna) {
            Racun racun = new Racun(stRacuna, 0);
            this.racuni[this.stRacunov] = racun;
            this.stRacunov++;
        }
    }
}
```

Upoštevamo, da mora biti stanje računa ob odprtju enako 0.

Stanje računa s podano številko pridobimo tako, da račun v tabeli najprej poiščemo, nato pa ga povprašamo o stanju:

```
public class Druga {
    ...
    public static class Banka {
        ...
        public int stanje(String stRacuna) {
            Racun racun = this.poisciRacun(stRacuna);
            return racun.vrniStanje();
        }
    }
}
```

Račun s številko `stRacuna` poiščemo tako, da se sprehodimo po tabeli računov in vsakokrat preverimo, ali je številka trenutnega računa v tabeli enaka `stRacuna`:

```
public class Druga {
    ...
    public static class Banka {
        ...
        private Racun poisciRacun(String stRacuna) {
            for (int i = 0;  i < this.stRacunov;  i++) {
                if (this.racuni[i].vrniStevilko().equals(stRacuna)) {
                    return this.racuni[i];
                }
            }
            throw new RuntimeException(
                    String.format("Ne najdem računa s številko %s ...", stRacuna));
        }
    }
}
```

Ker so vsi testni primeri zastavljeni tako, da iskani račun *vedno* obstaja, je neuspeh iskanja znak za napako v kodi. Zato raje vržemo izjemo, kot pa da bi vrnili `null`.

Manjkata nam še metodi `vrniStevilko` in `vrniStanje` v razredu `Racun`. To sta zgolj preprosta »getterja«:

```
public class Druga {
    ...
    public static class Racun {
        ...
```

```
        public String vrniStevilko() {
            return this.stevilka;
        }

        public int vrniStanje() {
            return this.stanje;
        }
    }
    ...
}
```

Kako položimo denar na račun s številko `stRacuna`? Tako, da račun najprej poiščemo, nato pa nad njim pokličemo metodo `polozi`, ki poveča stanje za podani znesek. Pri dvigu je podobno, le da se nam lahko zgodi, da na računu ni dovolj denarja. V tem primeru stanja računa ne smemo spremeniti.

```
public class Druga {
    ...
    public static class Racun {
        ...
        public void polozi(int znesek) {
            this.stanje += znesek;
        }

        public boolean dvigni(int znesek) {
            if (this.stanje < znesek) {
                return false;
            }
            this.stanje -= znesek;
            return true;
        }
    }

    public static class Banka {
        ...
        public void polozi(String stRacuna, int znesek) {
            Racun racun = this.poisciRacun(stRacuna);
            racun.polozi(znesek);
        }

        public boolean dvigni(String stRacuna, int znesek) {
            Racun racun = this.poisciRacun(stRacuna);
            return racun.dvigni(znesek);
        }
        ...
    }
}
```

Prenos je kombinacija dviga in pologa. Če je dvig uspešen (če je na izvornem računu dovolj denarja), položimo denar na ciljni račun, sicer pa zgolj vrnemo `false`.

```
public class Druga {
    ...
    public static class Banka {
        ...
        public boolean prenesi(String izvorni, String ciljni, int znesek) {
            if (this.dvigni(izvorni, znesek)) {
```

```
                this.polozi(ciljni, znesek);
                return true;
            }
            return false;
        }
        ...
    }
}
```

Če želimo ugotoviti, na kateri račun smo denar največkrat položili, moramo za vsak račun hraniti tekoče število pologov. Ker je število pologov lastnost posameznega računa, bomo v razred `Racun` dodali atribut `stPologov` in ustrezno poskrbeli zanj:

```
public class Druga {
    ...
    public static class Racun {
        ...
        private int stPologov;

        public Racun(String stevilka, int stanje) {
            ...
            this.stPologov = 0;
        }

        public int vrniSteviloPologov() {
            return this.stPologov;
        }

        public void polozi(int znesek) {
            ...
            this.stPologov++;
        }
    }
}
```

Ob vsakem pologu na račun `this` se števec pologov na ta račun poveča za 1.

Račun z največ pologi sedaj poiščemo s preprosto zanko za iskanje maksimuma:

```
public class Druga {
    ...
    public static class Banka {
        ...
        public String najvecPologov() {
            int iNaj = 0;
            for (int i = 0;  i < this.stRacunov;  i++) {
                if (this.racuni[i].vrniSteviloPologov() >
                        this.racuni[iNaj].vrniSteviloPologov()) {
                    iNaj = i;
                }
            }
            return this.racuni[iNaj].vrniStevilko();
        }
        ...
    }
}
```

Tretja naloga

Urbano območje, ki ga rišemo, je kvadrat, ki zavzema celotno krajšo stranico platna. Dolžina stranice kvadrata je potemtakem enaka dolžini krajše stranice platna:

```
public class Tretja extends Platno {
    ...
    public double stranicaObmocja(double wp, double hp) {
        return Math.min(wp, hp);
    }
    ...
}
```

Urbano območje se mora po obeh dimenzijah nahajati na sredini platna. Razlika med koordinato x levega roba območja (`xObmocje`) in koordinato x levega roba platna (0) mora biti torej enaka razliki med koordinato x desnega roba platna (`wp`) in koordinato x desnega roba območja (`xObmocje + dObmocje`, kjer `dObmocje` označuje dolžino stranice območja). Torej:

```
xObmocje - 0 == wp - (xObmocje + dObmocje)
    xObmocje == (wp - dObmocje) / 2
```

Enako velja za koordinate y.

```
public class Tretja extends Platno {
    ...
    public double xObmocja(double wp, double hp) {
        double dObmocje = this.stranicaObmocja(wp, hp);
        double xObmocje = (wp - dObmocje) / 2.0;
        return xObmocje;
    }

    public double yObmocja(double wp, double hp) {
        double dObmocje = this.stranicaObmocja(wp, hp);
        double yObmocje = (hp - dObmocje) / 2.0;
        return yObmocje;
    }
    ...
}
```

Sliko narišemo tako, da najprej narišemo okvir območja (koordinati levega zgornjega kota območja in dolžino njegove stranice pridobimo s pomočjo metod `stranicaObmocja`, `xObmocja` in `yObmocja`), nato pa po vrsti narišemo posamezne elemente iz tabele `this.elementi`:

```
public class Tretja extends Platno {
    ...
    protected void narisi(Graphics2D g, double wp, double hp) {
        double dObmocje = this.stranicaObmocja(wp, hp);
        double xObmocje = this.xObmocja(wp, hp);
        double yObmocje = this.yObmocja(wp, hp);

        g.setColor(Color.BLACK);
        g.drawRect(ri(xObmocje), ri(yObmocje), ri(dObmocje), ri(dObmocje));

        double dEnota = dObmocje / 2.0;
```

```
        double xSredisce = xObmocje + dEnota;
        double ySredisce = yObmocje + dEnota;

        for (int i = 0;  i < this.elementi.length;  i++) {
            this.elementi[i].narisi(g, dEnota, xSredisce, ySredisce);
        }
    }
    ...
}
```

Koordinate v tabeli `this.elementi` so podane v (recimo mu) matematičnem koordinatnem sistemu, sliko pa moramo narisati v koordinatnem sistemu platna. Točka (0, 0) v matematičnem koordinatnem sistemu se preslika v središče platna — v točko `(xSredisce, ySredisce)`. Velja `xSredisce = xObmocje + dEnota` in `ySredisce = yObmocje + dEnota`, kjer je `dEnota = dObmocje / 2` razdalja v koordinatnem sistemu platna, ki ustreza razdalji 1 v matematičnem koordinatnem sistemu. Mere `dEnota`, `xSredisce` in `ySredisce` posredujemo metodi `narisi` znotraj zanke `for`, da si poenostavimo risanje posameznih elementov.

Ker metodo `narisi` pokličemo nad vsakim elementom tabele `this.elementi`, jo moramo definirati že v razredu `Element`. Pri splošnem elementu lahko metoda `narisi` nastavi kvečjemu njegovo barvo; narisati ga seveda ne more.

```
public class Tretja extends Platno {
    ...
    public static abstract class Element {
        ...
        protected void narisi(Graphics2D g, double dEnota,
                              double xSredisce, double ySredisce) {
            g.setColor(this.barva);
        }
    }
}
```

Pri risanju stavbe s pomočjo metode `narisi` iz nadrazreda najprej nastavimo barvo, nato pa narišemo ustrezno velik pravokotnik na ustreznem položaju. Ker se koordinata $x = 0$ v matematičnem koordinatnem sistemu preslika v koordinato `X = xSredisce` v koordinatnem sistemu platna, dolžina 1 v matematičnem koordinatnem sistemu pa v dolžino `dEnota` v koordinatnem sistemu platna, se koordinata $x =$ `this.x` v matematičnem koordinatnem sistemu preslika v koordinato `X = xSredisce + this.x * dEnota` v koordinatnem sistemu platna. Podobno velja za koordinate y, le da moramo upoštevati, da te v matematičnem koordinatnem sistemu naraščajo navzgor, v koordinatnem sistemu platna pa navzdol.

```
public class Tretja extends Platno {
    ...
    public static class Stavba extends Element {
        ...
        public void narisi(Graphics2D g, double dEnota,
                           double xSredisce, double ySredisce) {

            super.narisi(g, dEnota, xSredisce, ySredisce);
            g.fillRect(ri(xSredisce + this.x * dEnota),
                       ri(ySredisce - this.y * dEnota),
                       ri(this.w * dEnota),
```

```
                        ri(this.h * dEnota));
        }
    }
}
```

Pri drevesu moramo upoštevati, da vrednosti `this.x` in `this.y` predstavljata koordinati središča kroga v matematičnem koordinatnem sistemu, metodi `fillOval` pa moramo podati koordinati zgornjega levega oglišča omejevalnega kvadrata kroga v koordinatnem sistemu platna. Vrednost `this.r` predstavlja polmer kroga v matematičnem koordinatnem sistemu, metodi `fillOval` pa moramo podati dolžino stranice njegovega omejevalnega kvadrata (torej premer kroga) v koordinatnem sistemu platna.

```
public class Tretja extends Platno {
    ...
    public static class Drevo extends Element {
        ...
        public void narisi(Graphics2D g, double dEnota,
                           double xSredisce, double ySredisce) {

            super.narisi(g, dEnota, xSredisce, ySredisce);
            g.fillOval(ri(xSredisce + (this.x - this.r) * dEnota),
                       ri(ySredisce - (this.y + this.r) * dEnota),
                       ri(2.0 * dEnota * this.r),
                       ri(2.0 * dEnota * this.r));
        }
    }
}
```

Cestni odsek nam sedaj ne bi smel povzročati preglavic:

```
public class Tretja extends Platno {
    ...
    public static class CestniOdsek extends Element {
        ...
        public void narisi(Graphics2D g, double dEnota,
                           double xSredisce, double ySredisce) {

            super.narisi(g, dEnota, xSredisce, ySredisce);
            g.drawLine(ri(xSredisce + dEnota * this.x1),
                       ri(ySredisce - dEnota * this.y1),
                       ri(xSredisce + dEnota * this.x2),
                       ri(ySredisce - dEnota * this.y2));
        }
    }
}
```

Izpit pri predmetu Programiranje I
15. februar 2017

Na voljo imate 90 minut časa.
Vse naloge lahko testirate z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) V razredu `Prva` dopolnite sledeči metodi:

- `public static boolean deljivoZVsotoStevk(int n)` [J1–J5, S1–S25]:

  Vrne `true` natanko v primeru, če je podano pozitivno število n deljivo z vsoto svojih števk. Primer takšnega števila je 201, saj je deljivo z $2 + 0 + 1 = 3$.

- `public static int negativnaVrstica(int[][] t)` [J6–J10, S26–S50]:

  Vrne indeks prve vrstice pravokotne tabele `t`, ki vsebuje sama negativna števila. Če takšne vrstice ni, naj metoda vrne -1. Na primer, za tabelo $\{\{-6, 0, -7\}, \{0, 0, 0\}, \{-7, -4, -21\}, \{-9, 9, -5\}\}$ naj metoda vrne 2.

(2) Objekt razreda `Prosilec` predstavlja prosilca za delovno mesto, za katero se zahteva znanje določenih jezikov na določenih nivojih. Recimo, da je na svetu le 10 jezikov; imenujmo jih jezik 0, jezik 1, ..., jezik 9. Razred `Prosilec` dopolnite s sledečimi elementi (po potrebi dodajte še svoje):

- `public Prosilec()` [J1–J10, S1–S50]:

  Ustvari prosilca, ki zna vseh 10 jezikov na nivoju 0.

- `public void seJeNaucil(int jezik, int nivo)` [J1–J10, S1–S50]:

  Zabeleži, da se je prosilec `this` naučil jezika `jezik` na nivoju `nivo`.

- `public int nivo(int jezik)` [J1–J2, S1–S10]:

  Vrne nivo, na katerem prosilec `this` obvlada jezik `jezik`.

- `public boolean izpolnjuje(Pogoj[] pogoji)` [J3–J5, J8, S11–S25]:

  Vrne `true` natanko v primeru, če prosilec `this` izpolnjuje podane pogoje. Tabela `pogoji` vsebuje pare (j_0, n_0), (j_1, n_1) itd., ki povedo, da mora prosilec obvladati jezik j_0 vsaj na nivoju n_0, jezik j_1 vsaj na nivoju n_1 itd. Izpolnjeni morajo biti vsi pogoji, ne le eden od njih.

- `public int presezek(Pogoj[] pogoji)` [J6–J7, J8, S26–S35]:

  Vrne presežek prosilca `this` glede na pogoje `pogoji`. Presežek izračunamo tako, da za jezike v tabeli `pogoji` seštejemo razlike med prosilčevim in zahtevanim nivojem znanja. Na primer, če prosilec obvlada jezik 3 na nivoju 40, jezik 5 na nivoju 60 in jezik 8 na nivoju 50, tabela `pogoji` pa vsebuje para (3, 30) in (8, 70), potem naj metoda vrne $(40 - 30) + (50 - 70) = -10$.

Objekt razreda `Razpis` vsebuje tabelo prosilcev in tabelo zahtevanih pogojev. Razred dopolnite s s sledečo metodo:

- `public int najboljsi()` [J8–J10, S36–S50]:

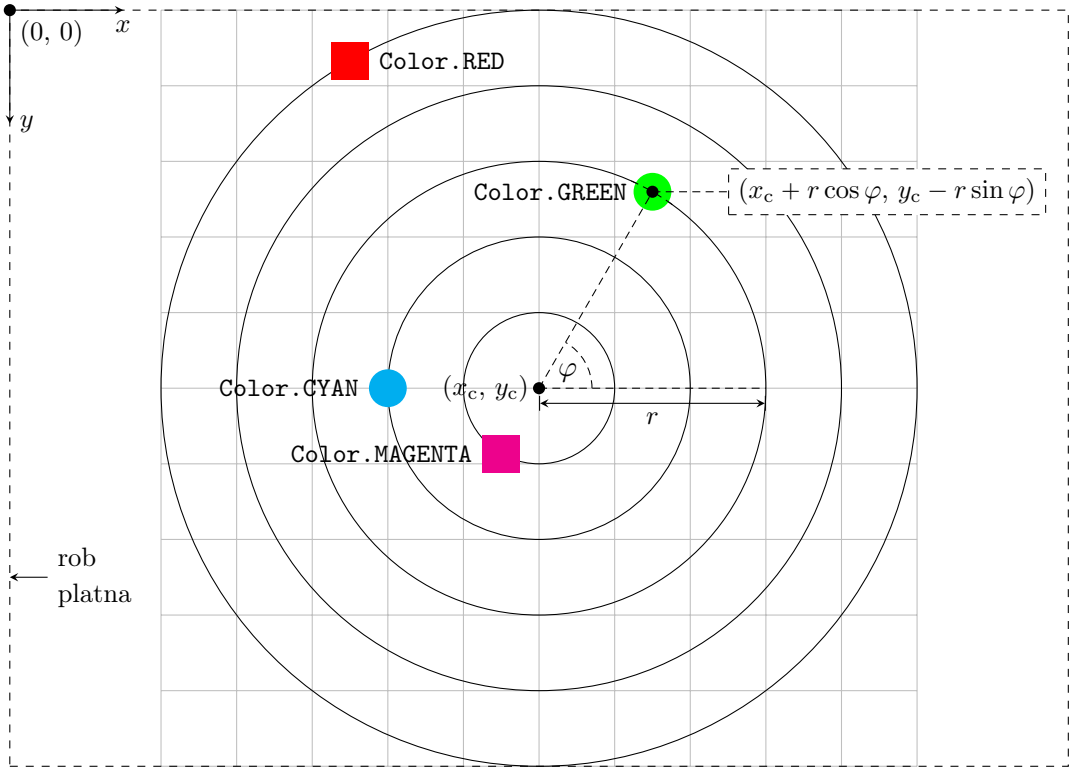
  Vrne indeks tistega prosilca v tabeli `this.prosilci`, ki ima največji presežek glede na pogoje `this.pogoji`, pri čemer upošteva samo prosilce, ki pogoje izpolnjujejo. Če jih nihče ne izpolnjuje, naj vrne -1, če je najboljših prosilcev več, pa naj vrne indeks prvega med njimi.

(3) Dopolnite metodo `narisi` v razredu `Tretja` tako, da bo na podlagi atributov `this.stOrbit` (tipa `int`) in `this.planeti` (tipa `Planet[]`) narisala orbite in planete na njih. Orbite so koncentrične krožnice s središčem v središču platna. Oštevilčene so s številkami od 1 (najbolj notranja) do `this.stOrbit` (najbolj zunanja). Premer orbite s številko i je enak (i / `this.stOrbit`)-kratniku dolžine krajše stranice platna.

Planeti so dveh vrst: *krogoni* in *kvadratoni*. Krogoni so krožne, kvadratoni pa kvadratne oblike. Za vsak planet sta podana barva in položaj njegovega središča, ki je določen s številko orbite in kotom v radianih. Premer krogona in dolžina stranice kvadratona sta vedno enaka četrtini premera orbite s številko 1.

Sledeča slika se nanaša na vrednost `this.stOrbit == 5` in tabelo `this.planeti` s sledečo vsebino:

```
{ new Krogon(    Color.GREEN,   3,  1.0 * Math.PI / 3.0),    // orbita 3, kot π/3
  new Kvadraton( Color.RED,     5,  2.0 * Math.PI / 3.0),
  new Krogon(    Color.CYAN,    2,  3.0 * Math.PI / 3.0),
  new Kvadraton( Color.MAGENTA, 1,  4.0 * Math.PI / 3.0) }
```



Narišite **samo** orbite in planete; vse ostalo na gornji sliki je zgolj v pomoč.

Dopolnite tudi sledeči metodi:

- `public double premerOrbite(int orbita, double wp, double hp)` [J1, S1–S5]:

  Vrne premer (v pikah) orbite s številko `orbita` za platno širine `wp` pik in višine `hp` pik.

- `public double[] polozajNaOrbiti(int orbita, double kot, double wp, double hp)` [J2–J3, S6–S15]:

  Vrne tabelo, ki vsebuje koordinati (v koordinatnem sistemu platna) točke na orbiti s številko `orbita` pri kotu `kot` radianov. Prvi element tabele naj vsebuje koordinato x, drugi pa y. Za računanje kosinusa in sinusa uporabite metodi `Math.cos` in `Math.sin`; obe sprejmeta kot v radianih.

Izpit pri predmetu Programiranje I
31. januar 2018

Vse naloge lahko testirate z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) Dopolnite sledeči metodi:

- `public static int zadnjiDeljiviPar(int[] t, int d)` [J1–J5, S1–S25]

  Vrne indeks zadnjega elementa v tabeli `t`, za katerega velja, da je tako element sam kot tudi njegov levi sosed deljiv s številom `d`. Če takšnega elementa v tabeli ni, naj metoda vrne -1. Na primer, pri tabeli $\{15, 3, 4, 12, 24, 9, 21, 5, 18\}$ in vrednosti `d` $= 3$ naj metoda vrne vrednost 6 (tj. indeks elementa 21).

- `public static long odstraniNicle(long n)` [J6–J10, S26–S50]

  Vrne število, ki ga dobimo, če iz števila `n` odstranimo vse ničle. Na primer, za `n` $= 450172030086$ naj metoda vrne število 45172386.

  Nalogo rešite zgolj z operacijami nad celimi števili. Uporaba operacij nad nizi, tabelami, objekti tipa `java.util.List` ipd. bo kaznovana s prepolovitvijo točk.

(2) Podani so razredi `Predmet`, `Student` in `Letnik` (prikazani so samo atributi):

```
class Predmet {
    private String naziv;
    private int kreditneTocke;   // število kreditnih točk
    private boolean strokovni;   // true: strokovni; false: prosto izbirni
    private int dan;   // dan, ko se izvajajo predavanja (1: pon., ..., 5: petek)
}
class Student {
    private String ime;
    private Predmet[] predmeti;   // predmeti, ki jih je vpisal v tem letniku
    private int[] ocene;          // ocene pri posameznih predmetih
}
class Letnik {
    private Student[] studenti;
}
```

V vseh objektih razreda `Student` sta tabeli `predmeti` in `ocene` enako dolgi, element `ocene[i]` pa predstavlja oceno, ki jo je študent dobil pri predmetu `predmeti[i]`. Ocene lahko zavzamejo vrednosti od 0 do 10. Ocena 0 pomeni, da študent še ni polagal izpita, ocene od 1 do 5 so negativne, ocene od 6 do 10 pa pozitivne.

V razredu `Student` dopolnite sledeči metodi:

- `public boolean imaProstoIzbirni()` [J1–J2, S1–S12]

  Vrne `true` natanko v primeru, če je študent `this` vpisal vsaj en prosto izbirni predmet.

- `public int[] steviloKT()` [J3–J5, S13–S25]

  Vrne tabelo z dvema elementoma. Prvi vsebuje skupno število doseženih kreditnih točk iz strokovnih, drugi pa skupno število doseženih kreditnih točk iz

prosto izbirnih predmetov. (Upoštevajte samo predmete, pri katerih je študent že opravil izpit.)

V razredu `Letnik` dopolnite sledeči metodi:

- `public int steviloNeuspesnih()` [J6–J7, S26–S37]

  Vrne število študentov, ki niso opravili niti enega izpita.

- `public int morebitnaPrekrivanja(int dan)` [J8–J10, S38–S50]

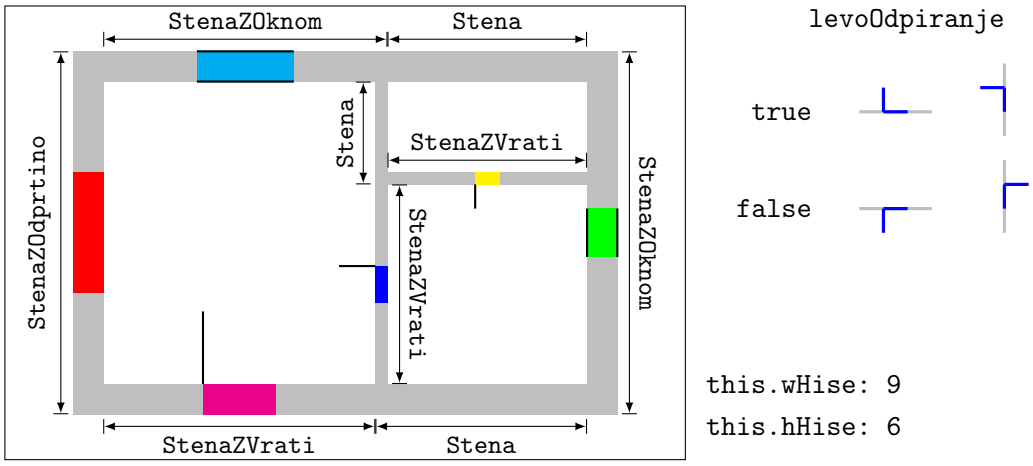
  Vrne število študentov, ki so vpisali vsaj dva predmeta, pri katerih se predavanja izvajajo na dan `dan` (1: ponedeljek, ..., 5: petek).

(3) Tloris pravokotne hiše velikosti `this.wHise`×`this.hHise` metrov lahko narišemo tako, da narišemo vse njene stene (razred `Stena`). Stene so lahko vodoravne ali navpične, za vsako steno pa poznamo njeno lego (koordinati x in y levega zgornjega oglišča), dolžino in debelino. Vse stene narišemo v barvi `Color.GRAY`.

Nekatere stene imajo odprtino (razred `StenaZOdprtino`), v katero lahko kasneje vgradimo okna ali vrata. Odprtina se vedno nahaja na sredini stene, njena barva in dolžina pa sta podani z atributoma `barvaOdprtine` in `dolzinaOdprtine`.

Pri stenah, ki imajo v odprtino vgrajeno okno (razred `StenaZOknom`) ali vrata (razred `StenaZVrati`), je treba narisati še okno oziroma vrata. Okno narišemo z dvema vzporednima črnima črtama, ki omejujeta odprtino po njeni dolžini, vrata pa narišemo s črno črto, ki prikazuje lego vrat, ko so ta odprta za 90 stopinj. Pri tem je treba upoštevati, da se vrata lahko odpirajo na levo ali na desno (atribut `levoOdpiranje`).

Sledeča slika prikazuje primer tlorisa (testni primer J10) in pomen atributa `levoOdpiranje` pri vodoravni in navpični steni:



Dopolnite metodo `narisi` tako, da bo na podlagi tabele `this.stene` narisala ustrezen tloris. Pri tem upoštevajte, da so testni podatki pripravljeni tako, da je razmerje stranic platna enako razmerju stranic hiše. Tloris naj bo narisan na sredini platna, zaseda pa naj 80% širine in 80% višine platna.

Dopolnite tudi metode `xTlorisa`, `yTlorisa` in `pikNaMeter`. Metodi `xTlorisa` in `yTlorisa` vrneta koordinati zgornjega levega kota hiše, merjeni v slikovnih pikah. Metoda `pikNaMeter` pa vrne število slikovnih pik, ki ustrezajo dolžini enega metra.

Izpit pri predmetu Programiranje I
15. februar 2018

Vse naloge lahko testirate z ukazom `tj.exe` (brez parametrov).
Veliko uspeha!

(1) Dopolnite sledeči metodi:

- `public static boolean jePodvojena(int[] t)` [J1–J5, S1–S25]

  Vrne `true` natanko v primeru, če so vsi elementi v prvi polovici tabele `t` enaki istoležnim elementom v drugi polovici. Primer takšne tabele je {7, 10, 8, 7, 10, 8}, saj je prvi element enak četrtemu, drugi petemu, tretji pa šestemu. Tabela `t` ima v vseh testnih primerih sodo število elementov.

- `public static int[] steviloDeljivih(int[][] t, int d)` [J6–J10, S26–S50]

  Vrne tabelo, v kateri i-ti element pove, koliko elementov v i-tem stolpcu tabele `t` je deljivih s številom `d`. Število elementov vrnjene tabele naj bo enako številu stolpcev tabele `t`.

(2) Igralci, označeni z indeksi od 0 do `stIgralcev` − 1, igrajo partije prek šahovskega portala. Vsaka posamezna zmaga prinese po dve točki, remi eno, poraz pa nič točk.

V razredu `Portal` dopolnite sledeče elemente:

- `public Portal(int stIgralcev)`

  Inicializira objekt, ki predstavlja šahovski portal s `stIgralcev` igralci.

- `public void zabeleziPartijo(int beli, int crni, int izid)`

  Zabeleži dejstvo, da sta igralca z indeksoma `beli` in `crni` (`beli` ≠ `crni`) odigrala partijo, v kateri je igralec `beli` igral z belimi, igralec `crni` pa s črnimi figurami. Igra se je zaključila z izidom `izid` (0: remi, 1: zmaga belega, 2: zmaga črnega).

- `public int steviloRemijev()` [J1–J2, S1–S12]

  Vrne skupno število remijev v vseh doslej odigranih partijah.

- `public int tocke(int igralec)` [J3–J5, S13–S25]

  Vrne skupno število točk, ki jih je doslej zbral igralec z indeksom `igralec`.

- `public int[] medsebojneTocke(int a, int b)` [J6–J7, S26–S37]

  Vrne tabelo z dvema elementoma: prvi vsebuje skupno število točk, ki jih je igralec z indeksom `a` zbral v igrah proti igralcu z indeksom `b`, drugi pa skupno število točk, ki jih je igralec `b` zbral v igrah proti igralcu `a`. Na primer, če je dvakrat zmagal igralec `b` (ne glede na to, ali je igral z belimi ali črnimi figurami), ena partija pa se je končala z remijem, bi metoda vrnila tabelo {1, 5}.

- `public int pariZRemijem()` [J8–J10, S38–S50]

  Vrne število različnih parov igralcev, pri katerih se je vsaj ena medsebojna partija končala z remijem. Para (i, j) in (j, i) štejemo kot en in isti par, ne kot dva različna para.

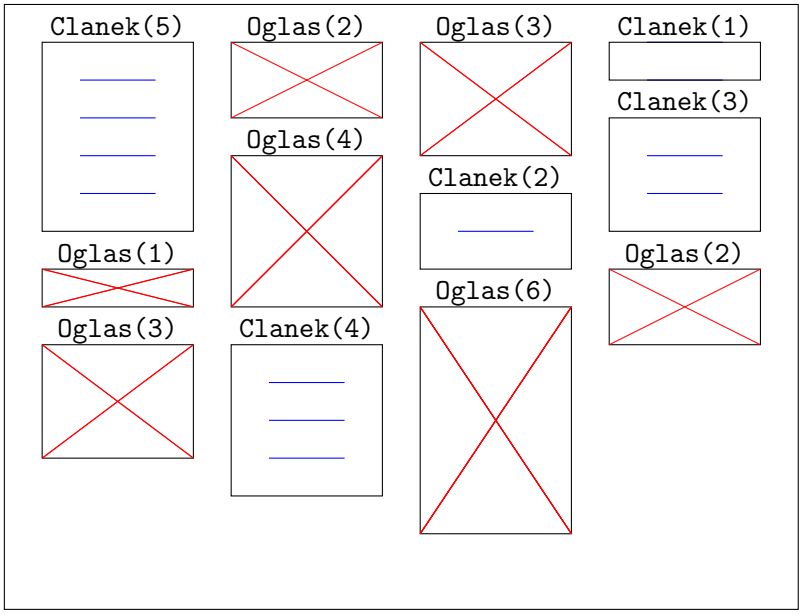
(3) Metodo `narisi` dopolnite tako, da bo narisala izgled časopisne strani. Časopisno stran si lahko predstavljamo kot matriko objav (tabela `this.objave` tipa `Objava[][]`), v kateri so vsi stolpci enako široki, objave znotraj stolpca pa niso nujno enako visoke. (Kljub temu vsi stolpci vsebujejo enako število objav.) Stolpci so široki po `this.sirinaStolpca` centimetrov, objava pa je visoka `visina` (atribut razreda `Objava`) centimetrov. Razmik med posameznimi stolpci in med posameznimi objavami znotraj stolpca znaša 1 cm, enako velik pa je tudi odmik skrajno levega oz. desnega stolpca od levega oz. desnega roba platna in odmik prve vrstice objav od zgornjega roba platna.

Objave so dveh tipov: *članek* (razred `Clanek`) in *oglas* (razred `Oglas`). Vsako objavo narišite kot pravokotnik z belim polnilom in črno obrobo. Pravokotnik za članek naj vsebuje `visina` − 1 enakomerno razmaknjenih modrih vodoravnih daljic, ki se se raztezajo čez polovico širine pravokotnika in so po širini postavljene na sredini pravokotnika. Pravokotnik za oglas naj vsebuje rdeč križ.

Število pik, ki ustreza enemu centimetru, določite tako, da se bo časopisna stran raztezala čez celotno širino platna.

Sledeča slika prikazuje videz platna za vrednost `sirinaStolpca` = 4 in tabelo `objava` s sledečo vsebino (testni primer J8):

```
{ { new Clanek(5), new Oglas(2),  new Oglas(3),  new Clanek(1) },
  { new Oglas(1),  new Oglas(4),  new Clanek(2), new Clanek(3) },
  { new Oglas(3),  new Clanek(4), new Oglas(6),  new Oglas(2)  } }
```



Poleg metode `narisi` dopolnite tudi metodo `pikNaCm`, ki vrne število pik na platnu, ki ustreza enemu centimetru na časopisni strani.

Izpit pri predmetu Programiranje I
29. avgust 2018

Vse naloge lahko testirate z ukazom `tj.exe` ali `tj_stari.exe` (brez parametrov).
Veliko uspeha!

(1) Dopolnite sledeči metodi:

- `public static int alternirajocaVsota(int n)` [J1–J5, S1–S25]

  Vrne vsoto $a_0 - a_1 + a_2 - a_3 + \ldots$, kjer a_0 predstavlja *zadnjo* (!) števko pozitivnega števila `n`, a_1 predzadnjo itd.

- `public static int[] polozajiZvezdic(char[][] t)` [J6–J10, S26–S50]

  Vrne tabelo, v kateri i-ti element podaja položaj (indeks vrstice) najnižje ležečega znaka `*` v i-tem stolpcu pravokotne tabele `t`. Če v stolpcu ni znaka `*`, naj ima pripadajoči element vrnjene tabele vrednost -1. Dolžina vrnjene tabele naj bo enaka številu stolpcev tabele `t`.

(2) Podani so razredi `TrgovskiCenter`, `Trgovina` in `Izdelek` (prikazani so samo atributi):

```
class TrgovskiCenter {
    private Trgovina[][] trgovine;    // trgovine, ki tvorijo trgovski center
}                                     //    (razporejene so v pravokotni matriki)
class Trgovina {
    private Izdelek[] izdelki;  // izdelki, ki jih trgovina prodaja
    private int[] zaloga;       // zaloga[i]: trenutna zaloga izdelka izdelki[i]
    private int saldo;          // trenutno stanje na računu trgovine
}
class Izdelek {
    private String naziv;
    private int cena;
}
```

V razredu `Trgovina` dopolnite sledeči metodi:

- `public int poisciIzdelek(String naziv)` [J1–J2, S1–S12]

  Vrne indeks izdelka z nazivom `naziv` v tabeli `this.izdelki` oziroma -1, če iskanega izdelka ni v tabeli.

- `public int prodaj(int indeks, int n)` [J3–J5, S13–S25]

  Če ima trgovina `this` na zalogi vsaj `n` izdelkov z indeksom `indeks`, jih proda `n`, sicer pa toliko, kolikor jih ima na zalogi. Metoda naj vrne prihodek od prodaje izdelkov, še pred tem pa naj poveča saldo (za prihodek) in zmanjša zalogo.

V razredu `TrgovskiCenter` dopolnite sledeči metodi:

- `public boolean vSosescini(int vr, int st, String naziv)` [J6–J7, S26–S37]

  Vrne `true` natanko v primeru, če trgovina na položaju (`vr`, `st`) (indeks vrstice, indeks stolpca) ali pa katerakoli od njenih sosed prodaja izdelek z nazivom `naziv`, četudi ga morda trenutno nima na zalogi. V vseh testnih primerih ima trgovina na položaju (`vr`, `st`) osem sosed: levo, desno, zgornjo, spodnjo in štiri diagonalne.

- `public int pohod(int[][] premiki, String naziv)` [J8–J10, S38–S50]

  Pričnemo v trgovini na položaju (0, 0), potem pa se premikamo v skladu z navodili, zapisanimi v vrsticah tabele `premiki`. V vsaki trgovini (tudi v začetni in končni) pokupimo celotno zalogo izdelkov z nazivom `naziv`. Metoda naj ustrezno posodablja zalogo in saldo v obiskanih trgovinah, vrne pa naj skupno količino denarja, ki ga porabimo na pohodu.

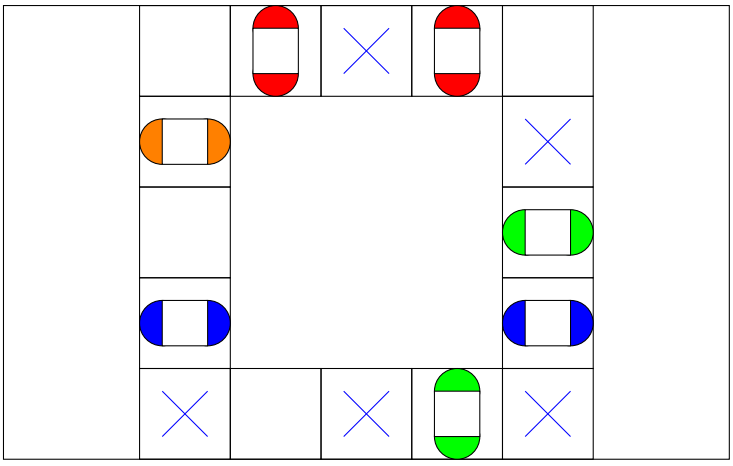
  Vsaka vrstica tabele `premiki` ima po dva elementa. Prvi podaja vrstični premik (razliko med indeksoma ciljne in trenutne vrstice), drugi pa stolpčni premik. V vseh testnih primerih so vsi premiki legalni.

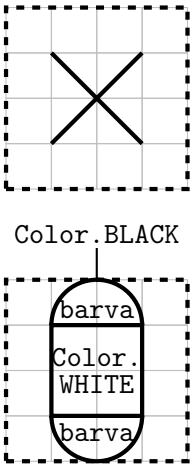
(3) Metodo `narisi` dopolnite tako, da bo narisala igralno površino različice igre Monopoly. Igralna površina ima obliko kvadratnega okvirja z `this.dolzina` polji po stranici. Podatki o posameznih poljih so nanizani v tabeli `this.polja` (tipa `Polje[]`). Prvi element tabele se nanaša na polje v zgornjem levem kotu igralne površine, nato pa si polja sledijo v smeri urinega kazalca.

Polja so treh vrst: *navadno polje* (razred `Polje`), *presenečenje* (razred `Presenecenje`) in *posest* (razred `Posest`). Vsako polje v osnovi narišite kot kvadrat z belim polnilom in črno obrobo. Polje presenečenja naj poleg tega vsebuje še križ barve `Color.BLUE`, posest pa »kapsulo« barve `barva` (atribut v razredu `Posest`). Polja v vogalih igralne površine so lahko samo navadna polja ali presenečenja.

Sledi primer, za katerega velja `this.dolzina == 5`. Pri risanju križa in kapsule si pomagajte s shemama na desni. Upoštevajte, da je kapsula pri posestih na levem in desnem robu igralne površine obrnjena za 90 stopinj.

```
this.polja: {
    new Polje(), new Posest(Color.RED), new Presenecenje(), new Posest(Color.RED),
    new Polje(), new Presenecenje(), new Posest(Color.GREEN), new Posest(Color.BLUE),
    new Presenecenje(), new Posest(Color.GREEN), new Presenecenje(), new Polje(),
    new Presenecenje(), new Posest(Color.ORANGE), new Polje(), new Posest(Color.BLUE),
};
```





Igralna površina naj zavzema celotno krajšo stranico platna, po daljši pa naj bo prikazana na sredini.

Poleg metode `narisi` dopolnite še metode `stranicaPolja` (vrne dolžino stranice polja, merjeno v pikah), `xLevo` (vrne koordinato x levega roba igralne površine) in `yZgoraj` (vrne koordinato y zgornjega roba igralne površine).