Dodatne naloge – Razredi in objekti I

Splošna navodila

V tem sklopu nalog uporabljamo nekoliko drugačen pristop kot pri prejšnjih. Pri vseh nalogah zahtevamo realizacijo nekega razreda in njegovih javno dostopnih konstruktorjev in metod. Glede ostalih komponent (atributi, morebitne privatne metode oz. konstruktorji itd.) ste svobodni, seveda pa upoštevajte splošne smernice objektno usmerjenega programiranja. Svoje rešitve boste lahko preverili z množico testnih razredov in pripadajočih izhodnih datotek. Testni razredi so zasnovani tako, da učinkovitost vaše rešitve ni bistven dejavnik. Glavno merilo naj bo tokrat »eleganca« rešitve, ne njena optimalnost v smislu prostorsko-časovne učinkovitosti. Pri elegantni rešitvi se, denimo, metode pogosto kličejo med seboj.

1 Razreda `Posta` in `Pismo`

Naloga

V skladu s sledečimi navodili napišite razreda `Posta` in `Pismo`.

Razred `Posta`

Razred `Posta` definirajte tako, da bo vsak njegov objekt predstavljal neko pošto s poštno številko (npr. 1000) in nazivom (npr. Ljubljana). Razred naj vsebuje sledeče konstruktorje in metode:

- `public Posta(int stevilka, String naziv)`

  Ustvari nov objekt tipa `Posta`, ki predstavlja pošto s podano poštno številko in nazivom.

- `public int vrniStevilko()`

  Vrne poštno številko pošte `this`.

- `public String vrniNaziv()`

  Vrne naziv pošte `this`.

- `public String toString()`

  Za pošto `this` vrne niz sledeče oblike:

  stevilka␣*naziv*

  Na primer:

  `1000␣Ljubljana`

Razred `Pismo`

Razred `Pismo` definirajte tako, da bo vsak njegov objekt predstavljal neko pismo s sledečimi podatki:

- izvorna pošta (npr. 1000 Ljubljana);
- ciljna pošta (npr. 2000 Maribor);
- podatek o tem, ali je pismo priporočeno ali navadno;
- razdalja (v kilometrih) med izvorno in ciljno pošto.

V razredu definirajte sledeče javno dostopne konstruktorje in metode:

- `public Pismo(Posta izvorna, Posta ciljna,`
  `boolean jePriporoceno, int razdalja)`

  Ustvari nov objekt tipa `Pismo`, ki predstavlja pismo s podano izvorno in ciljno pošto, »priporočenostjo« (`true`: priporočeno; `false`: navadno) ter razdaljo (v kilometrih) med izvorno in ciljno pošto.

- `public String toString()`

  Za pismo `this` vrne niz oblike

  izvorna_pošta␣`->`␣*ciljna_pošta*␣`(`*razdalja*␣`km)`␣`[`*vrsta*`]`

  pri čemer je *vrsta* bodisi `P` (priporočeno) ali pa `N` (navadno). Na primer:

  `1000␣Ljubljana␣->␣2000␣Maribor␣(130␣km)␣[P]`

- `public boolean izviraOd(Posta posta)`

  Vrne `true` natanko v primeru, če je pošta `posta` izvorna pošta za pismo `this`.

- `public boolean staIzvorInCiljIsta()`

  Vrne `true` natanko v primeru, če ima pismo `this` isto izvorno in ciljno pošto (npr. če je pismo poslano s pošte 1000 Ljubljana na pošto 1000 Ljubljana).

- `public boolean imaIstiCiljKot(Pismo pismo)`

  Vrne `true` natanko v primeru, če ima pismo `this` isto ciljno pošto kot pismo `pismo`.

- `public static boolean imataIstiCilj(Pismo p1, Pismo p2)`

  Vrne `true` natanko v primeru, če imata obe podani pismi isto ciljno pošto.

- `public int cena()`

  Vrne ceno (v stotinih) oddaje pisma `this`. Za navadno pismo se cena izračuna glede na razdaljo: za razdaljo od 0 do vključno $(r-1)$ km je cena enaka c stotinov, za razdaljo od r do vključno $(2r-1)$ km znaša $2c$ stotinov, za razdaljo od $2r$ do vključno $(3r-1)$ km znaša $3c$ stotinov itd. Ceno priporočenega pisma izračunamo tako, da ceni navadnega pisma prištejemo priporočnino p stotinov, ki je neodvisna od razdalje. Konstante r, c in p se nastavijo z metodo, ki jo predstavljamo v naslednji alineji.

- `public static void nastaviKonstanteZaCeno(`
  `int enotaRazdalje, int enotaCene, int priporocnina)`

  Konstante r, c in p, ki se uporabljajo za izračun cene oddaje pisma (gl. prejšnjo alinejo), nastavi na vrednosti `enotaRazdalje`, `enotaCene` in `priporocnina` (v tem vrstnem redu).

- `public boolean jeDrazjeOd(Pismo pismo)`

  Vrne `true` natanko v primeru, če je cena pisma `this` večja od cene pisma `pismo`.

- `public static Pismo vrniDrazje(Pismo p1, Pismo p2)`

  Vrne tisto pismo izmed `p1` in `p2`, ki ima večjo ceno. Če imata obe pismi enako ceno, naj vrne pismo `p2`.

- `public Pismo izdelajPovratno()`

  Ustvari in vrne nov objekt tipa `Pismo`, ki predstavlja povratnico pisma `this`. Povratnica ima enake podatke kot pismo `this`, le izvorna in ciljna pošta sta med seboj zamenjani.

Testni primer 11

Testni razred:

```
public class Test11 {

    public static void main(String[] args) {
        Posta lj = new Posta(1000, "Ljubljana");
        Posta mb = new Posta(2000, "Maribor");
        Posta ce = new Posta(3000, "Celje");
        System.out.println(lj.vrniNaziv());
        System.out.println(mb.vrniStevilko());
        System.out.println(ce.toString());

        Pismo.nastaviKonstanteZaCeno(10, 3, 20);
        Pismo lj2ce = new Pismo(lj, ce, true, 75);
        Pismo mb2lj = new Pismo(mb, lj, false, 130);
        Pismo ce2ce = new Pismo(ce, ce, true, 0);
        System.out.println(lj2ce.izviraOd(mb));
        System.out.println(ce2ce.staIzvorInCiljIsta());
        System.out.println(lj2ce.imaIstiCiljKot(ce2ce));
        System.out.println(Pismo.imataIstiCilj(lj2ce, mb2lj));
        System.out.println(lj2ce.cena());
        System.out.println(mb2lj.cena());
        System.out.println(mb2lj.jeDrazjeOd(lj2ce));
        System.out.println(Pismo.vrniDrazje(mb2lj, lj2ce).toString());
        System.out.println(mb2lj.izdelajPovratno().toString());
    }
}
```

Izhod:

```
Ljubljana
2000
3000 Celje
false
true
true
false
44
42
false
```

```
1000 Ljubljana -> 3000 Celje (75 km) [P]
1000 Ljubljana -> 2000 Maribor (130 km) [N]
```

2 Razred `Ulomek`

Naloga

Napišite razred `Ulomek` tako, da bodo njegovi objekti predstavljali posamezne okrajšane ulomke. Razred `Ulomek` naj vsebuje sledeče konstruktorje in metode:

- `public Ulomek(int a, int b)`

  Ustvari nov objekt tipa `Ulomek`, ki predstavlja okrajšano različico ulomka `a`/`b`. Ulomek p/q je okrajšan natanko tedaj, ko velja $q > 0$ in $\gcd(p, q) = 1$. Na primer, okrajšana različica ulomka $15/5$ je ulomek $3/1$, okrajšana različica ulomka $10/(-20)$ pa je ulomek $-1/2$. Lahko predpostavite, da sta števili `a` in `b` različni od 0.

- `public String toString()`

  Vrne okrajšani ulomek `this` v obliki niza *števec*/*imenovalec*, npr. `3/1` ali `-1/2`.

- `public boolean jeEnakKot(Ulomek u)`

  Vrne `true` natanko v primeru, če sta ulomka `this` in `u` enaka.

- `public Ulomek negacija()`

  Ustvari in vrne nov objekt, ki predstavlja nasprotno vrednost ulomka `this` (torej $-x$, če je x dani ulomek).

- `public Ulomek obrat()`

  Ustvari in vrne nov objekt, ki predstavlja obratno vrednost ulomka `this` (torej $1/x$, če je x dani ulomek).

- `public Ulomek vsota(Ulomek u)`
  `public Ulomek razlika(Ulomek u)`
  `public Ulomek zmnozek(Ulomek u)`
  `public Ulomek kolicnik(Ulomek u)`

  Ustvari in vrne nov objekt, ki predstavlja vsoto, razliko, zmnožek oziroma količnik ulomka `this` in ulomka `u`.

- `public Ulomek potenca(int eksponent)`

  Vrne potenco ulomka `this` na podani eksponent. Eksponent ni nujno pozitiven!

- `public boolean jeManjsiOd(Ulomek u)`

  Vrne `true` natanko v primeru, če je ulomek `this` manjši od ulomka `u`.

Testni primer 11

Testni razred:

```
public class Test11 {

    public static void main(String[] args) {
        Ulomek a = new Ulomek(-30, -40);
        Ulomek b = new Ulomek(24, -18);
        Ulomek c = new Ulomek(15, 20);

        System.out.println(a.toString());
        System.out.println(a.jeEnakKot(c));
        System.out.println(a.negacija().toString());
        System.out.println(a.obrat().toString());
        System.out.println(a.vsota(b).toString());
        System.out.println(a.razlika(b).toString());
        System.out.println(a.zmnozek(b).toString());
        System.out.println(a.kolicnik(b).toString());
        System.out.println(a.potenca(2).toString());
        System.out.println(b.potenca(-3).toString());
        System.out.println(a.jeManjsiOd(b));
    }
}
```

Izhod:

```
3/4
true
-3/4
4/3
-7/12
25/12
-1/1
-9/16
9/16
-27/64
false
```

Opomba

Metode, ki jih morate realizirati, so zasnovane tako, da ne spreminjajo stanja objekta `this`. Če boste to načelo uporabili za vse metode vašega razreda `Ulomek`, bodo objekti tipa `Ulomek` *nespremenljivi* (angl. immutable). Stanja nespremenljivega objekta po izdelavi ne moremo več spreminjati. Nespremenljivi objekti imajo vrsto dobrih lastnosti in lahko programerju prihranijo marsikatero skrb:

`http://www.javapractices.com/topic/TopicAction.do?Id=29`

Na primer, sledeči potencialno zahrbtni pojavi so možni zgolj pri spremenljivih objektih, saj nespremenljivi po definiciji sploh ne morejo imeti metod vrste »setter«:

```
Oseba os1 = new Oseba("Jože", "Gorišek", "Ruše");    // ime, priimek, kraj
Oseba os2 = os1;
os1.nastaviKraj("Maribor");
os1.izpisi();   // Jože Gorišek, Maribor
os2.izpisi();   // Jože Gorišek, Maribor (past!)
```

3 Razred `Datum`

Naloga

Napišite razred `Datum` tako, da bodo njegovi objekti predstavljali veljavne datume po gregorijanskem koledarju. Hraniti želimo datume od 1. januarja 1583 (torej od leta, ko se je uveljavil gregorijanski koledar) do 31. decembra 2999. Razred `Datum` naj vsebuje sledeče metode:

- `public static Datum ustvari(int dan, int mesec, int leto)`

  Če podani parametri predstavljajo veljaven datum (`leto` med 1583 in 2999, `mesec` od 1 do 12, `dan` od 1 do števila dni v izbranem mesecu izbranega leta), naj metoda ustvari in vrne nov objekt razreda `Datum`, sicer pa naj vrne `null`. Pri preverjanju veljavnosti datuma upoštevajte pravilo za prestopna leta, ki se glasi takole: leto je prestopno, če je deljivo s 400 ali pa če je deljivo s 4, vendar ni deljivo s 100. Leta 1700, 1800, 1900, 2100, 2200, 2300, 2500, ... tako niso prestopna, leta 1600, 2000, 2400, ... pa so.

  Zakaj smo se v tem primeru odločili za statično konstrukcijsko metodo namesto za konstruktor? Ali je konstruktor kljub tej metodi smiselno napisati? Če da, naj bo javno dostopen ali privaten?

- `public String toString()`

  Vrne predstavitev datuma v obliki niza `DD.MM.LLLL`, npr. `15.07.2014` ali `05.11.1975`.

- `public boolean jeEnakKot(Datum datum)`

  Vrne `true` natanko v primeru, če objekt `this` predstavlja isti datum kot objekt `datum`.

- `public boolean jePred(Datum datum)`

  Vrne `true` natanko v primeru, če je datum `this` kronološko pred datumom `datum`. Na primer, datum 30.09.2014 je pred datumom 27.10.2014, ta pa je pred datumom 20.01.2015.

- `public Datum naslednik()`

  Ustvari in vrne nov objekt razreda `Datum`, ki predstavlja neposrednega naslednika datuma `this`. Na primer, naslednik datuma 28.02.2012 je datum 29.02.2012, njegov naslednik pa je datum 01.03.2012. Naslednik datuma 28.02.2014 je datum 01.03.2014. Če datum nima veljavnega naslednika (edini tak datum je 31.12.2999), naj metoda vrne `null`.

- `public Datum predhodnik()`

  Ustvari in vrne nov objekt razreda `Datum`, ki predstavlja neposrednega predhodnika datuma `this`. Na primer, predhodnik datuma 01.03.2012 je datum 29.02.2012, predhodnik datuma 01.03.2014 pa je datum 28.03.2014. Če datum nima veljavnega predhodnika (edini tak datum je 01.01.1583), naj metoda vrne `null`.

- `public Datum cez(int stDni)`

  Izračuna (in vrne kot nov objekt tipa `Datum`) datum, ki je za `stDni` oddaljen od datuma `this`. Parameter `stDni` je lahko tudi negativen; v tem primeru metoda izračuna datum, ki je `-stDni` pred datumom `this`. Če ciljni datum pade pred datum 01.01.1583 ali za datum 31.12.2999, naj metoda vrne `null`.

- `public int razlika(Datum datum)`

  Vrne razliko (v številu dni) med datumoma `this` in `datum`. Če je datum `this` pred datumom `datum`, je razlika seveda negativna.

Testni primer 11

Testni razred:

```
public class Test11 {

    public static void main(String[] args) {
        Datum a = Datum.ustvari(29, 2, 2016);
        Datum b = Datum.ustvari(1, 1, 2017);

        System.out.println(a.toString());
        System.out.println(a.jeEnakKot(b));
        System.out.println(a.jePred(b));
        System.out.println(a.naslednik().toString());
        System.out.println(b.predhodnik().toString());
        System.out.println(a.cez(365).toString());
        System.out.println(b.cez(-365).toString());
    }
}
```

Izhod:

```
29.02.2016
false
true
01.03.2016
31.12.2016
28.02.2017
02.01.2016
```

Opomba

Statična konstrukcijska metoda (`ustvari` v našem primeru) se imenuje *tovarna* (angl. factory method). Tovarne so namenjene nadzorovani izdelavi objektov in se zato uporabljajo v kombinaciji s privatnimi konstruktorji. V tej nalogi smo tovarno izdelali zato, ker želimo zagotoviti, da objekti razreda `Datum` predstavljajo samo veljavne datume. Samo s konstruktorjem tega ne bi mogli doseči, saj konstruktor ne more preprečiti izdelave objekta.

4 Dopolnitve razreda `Oseba` (⋆)

Naloga

V razred `Oseba`, ki smo ga napisali na vajah in ki ga najdete tudi v mapi s testnimi primeri, dodajte sledeče metode:

- `public int ocetovskaGeneracijskaRazlika(Oseba os)`

  Vrne očetovsko generacijsko razliko med osebama `this` in `os` (OGR(`this`, `os`)). Vrednost $OGR(A, B)$ za osebi A in B je definirana takole:

  - Če sta osebi A in B identični (če gre za isto osebo), velja $OGR(A, B) = 0$.
  - Če je oseba A očetovski prednik osebe B, potem $OGR(A, B)$ izračunamo kot število očetov na liniji od B do A. (Če je oseba A oče osebe B, velja $OGR(A, B) = 1$, če je oseba A oče očeta osebe B, velja $OGR(A, B) = 2$ itd.)
  - Če je oseba B očetovski prednik osebe A, velja $OGR(A, B) = -OGR(B, A)$.
  - Če nobeden od zgornjih pogojev ni izpolnjen, naj metoda namesto vračila vrednosti (torej namesto stavka `return`) sproži izjemo tipa `IllegalArgumentException`:

    ```
    throw new IllegalArgumentException();
    ```

- `public boolean jePrednikOd(Oseba os)`

  Vrne `true` natanko v primeru, če je oseba `this` prednik osebe `os`. Oseba A je prednik osebe B, če je izpolnjen eden od sledečih pogojev:

  - $A = B$ (vsaka oseba je sam svoj prednik).
  - A je prednik B-jevega očeta.
  - A je prednik B-jeve matere.

- `public void nastejPrednike()`

  Izpiše vse prednike osebe `this`. Vsak prednik naj se izpiše v svoji vrstici, in sicer na sledeči način:

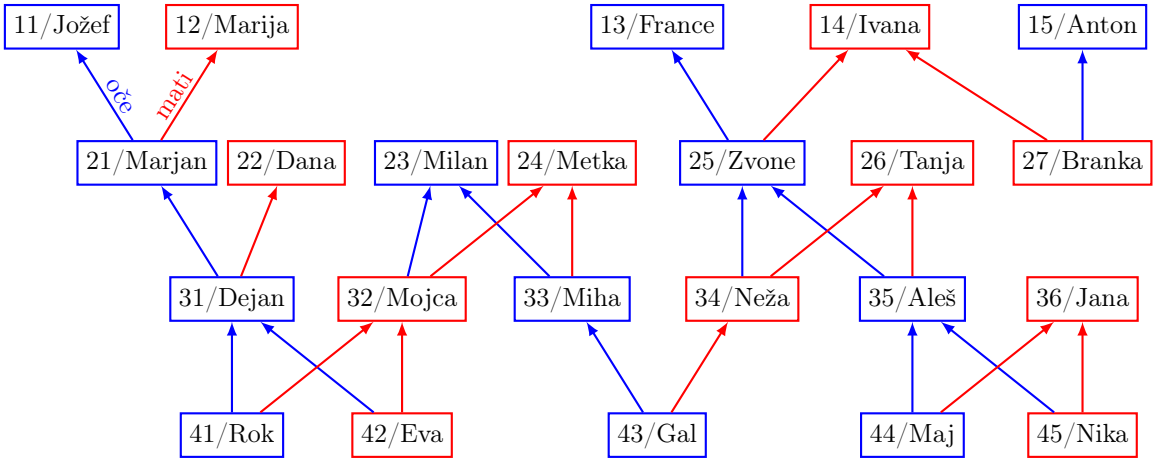
  veriga`:␣`*prednik*

  Pri tem je *veriga* niz, ki ponazarja starševsko verigo od osebe `this` do prednika, izpisanega v tekoči vrstici, *prednik* pa niz, ki ga za danega prednika vrne metoda `toString`. Niz *veriga* izdelajte po sledečem zgledu: niz `this` predstavlja osebo `this`, niz `this.oce` predstavlja očeta osebe `this`, niz `this.oce.mati` predstavlja mater očeta osebe `this` itd. Najprej naj se na opisani način izpišejo vsi predniki po očetovi strani, nato pa še vsi predniki po materini strani. Enako pravilo naj se rekurzivno uporabi za posamezne prednike. Za primer s slike 1 bi klic `os43.nastejPrednike()` izpisal sledeče:

  ```
  this: Gal Smole [M] (2009)
  this.oce: Miha Smole [M] (1978)
  this.oce.oce: Milan Smole [M] (1953)
  this.oce.mati: Metka Smole [Z] (1953)
  this.mati: Neža Smole [Z] (1980)
  this.mati.oce: Zvone Kotnik [M] (1956)
  this.mati.oce.oce: France Kotnik [M] (1932)
  this.mati.oce.mati: Ivana Kotnik [Z] (1931)
  this.mati.mati: Tanja Kotnik [Z] (1954)
  ```

- `public boolean jeSorodnikOd(Oseba os)`

  Vrne `true` natanko v primeru, če sta osebi `this` in `os` sorodnika. Osebi A in B sta sorodnika, če obstaja oseba C, ki je prednik tako osebe A kot osebe B. (Tudi



Slika 1: *Starševski odnosi med osebami v testnem razredu.*

sorodstvo je možno definirati na eleganten rekurzivni način. Odkrijte ga sami!)

V primeru z vaj (slika 1) sta osebi `os41` in `os43` (Rok in Gal) sorodnika. Sorodnika sta tudi osebi `os24` in `os42` (Metka in Eva), osebi `os33` in `os34` (Miha in Neža) pa nista.

Testni primer 11

Testni razred:

```
public class Test11 {

    public static void main(String[] args) {
        Oseba os11 = new Oseba("Jožef", "Pogačnik", 'M', 1921, null, null);
        Oseba os12 = new Oseba("Marija", "Pogačnik", 'Z', 1928, null, null);
        Oseba os13 = new Oseba("France", "Kotnik", 'M', 1932, null, null);
        Oseba os14 = new Oseba("Ivana", "Kotnik", 'Z', 1931, null, null);
        Oseba os15 = new Oseba("Anton", "Zajc", 'M', 1922, null, null);
        Oseba os21 = new Oseba("Marjan", "Pogačnik", 'M', 1946, os11, os12);
        Oseba os22 = new Oseba("Dana", "Pogačnik", 'Z', 1950, null, null);
        Oseba os23 = new Oseba("Milan", "Smole", 'M', 1953, null, null);
        Oseba os24 = new Oseba("Metka", "Smole", 'Z', 1953, null, null);
        Oseba os25 = new Oseba("Zvone", "Kotnik", 'M', 1956, os13, os14);
        Oseba os26 = new Oseba("Tanja", "Kotnik", 'Z', 1954, null, null);
        Oseba os27 = new Oseba("Branka", "Zajc", 'Z', 1952, os15, os14);
        Oseba os31 = new Oseba("Dejan", "Pogačnik", 'M', 1973, os21, os22);
        Oseba os32 = new Oseba("Mojca", "Pogačnik", 'Z', 1977, os23, os24);
        Oseba os33 = new Oseba("Miha", "Smole", 'M', 1978, os23, os24);
        Oseba os34 = new Oseba("Neža", "Smole", 'Z', 1980, os25, os26);
        Oseba os35 = new Oseba("Aleš", "Kotnik", 'M', 1982, os25, os26);
        Oseba os36 = new Oseba("Jana", "Kotnik", 'Z', 1981, null, null);
        Oseba os41 = new Oseba("Rok", "Pogačnik", 'M', 2003, os31, os32);
        Oseba os42 = new Oseba("Eva", "Pogačnik", 'Z', 2006, os31, os32);
        Oseba os43 = new Oseba("Gal", "Smole", 'M', 2009, os33, os34);
        Oseba os44 = new Oseba("Maj", "Kotnik", 'M', 2010, os35, os36);
        Oseba os45 = new Oseba("Nika", "Kotnik", 'Z', 2012, os35, os36);
```

```
        System.out.println( os13.ocetovskaGeneracijskaRazlika(os45) );
        System.out.println( os42.ocetovskaGeneracijskaRazlika(os21) );
        System.out.println( os24.jePrednikOd(os42) );
        System.out.println( os14.jePrednikOd(os33) );
        os41.nastejPrednike();
        System.out.println( os41.jeSorodnikOd(os43) );
        System.out.println( os42.jeSorodnikOd(os45) );
        System.out.println( os13.jeSorodnikOd(os44) );
    }
}
```

Izhod:

```
3
-2
true
false
this: Rok Pogačnik (M), 2003
this.oce: Dejan Pogačnik (M), 1973
this.oce.oce: Marjan Pogačnik (M), 1946
this.oce.oce.oce: Jožef Pogačnik (M), 1921
this.oce.oce.mati: Marija Pogačnik (Z), 1928
this.oce.mati: Dana Pogačnik (Z), 1950
this.mati: Mojca Pogačnik (Z), 1977
this.mati.oce: Milan Smole (M), 1953
this.mati.mati: Metka Smole (Z), 1953
true
false
true
```

5 Tabela s poljubnim številom dimenzij (⋆)

Naloga

Napišite razred `Ptabela`, čigar objekt predstavlja neko (hiper-)pravokotno celoštevilsko tabelo s *poljubnim* številom dimenzij (»p-tabela«). Razred naj ponuja sledeče konstruktorje in metode:

- `public Ptabela(int[] dimenzije)`

  Ustvari p-tabelo s podanimi velikostmi dimenzij in jo napolni z ničlami. Lahko predpostavite, da je dolžina tabele `dimenzije` enaka najmanj 1.

  Na primer, stavek

  `Ptabela p = new Ptabela(new int[]{3, 4, 2});`

  bi ustvaril p-tabelo velikosti $3 \times 4 \times 2$.

- `public void nastavi(int[] indeksi, int vrednost)`

Element p-tabele `this`, določen s podano tabelo indeksov, nastavi na podano vrednost. Lahko predpostavite, da je dolžina tabele `indeksi` enaka številu dimenzij p-tabele `this` in da so vsi indeksi veljavni.

Na primer, stavek

```
p.nastavi(new int[]{1, 3, 0}, 25);
```

bi nastavil element na poziciji `[1][3][0]` (druga »stran«, četrta vrstica, prvi element znotraj vrstice) na vrednost 25.

- `public int vrni(int[] indeksi)`

  Vrne vrednost elementa p-tabele `this`, določenega s podano tabelo indeksov. Lahko predpostavite, da je dolžina tabele `indeksi` enaka številu dimenzij p-tabele `this` in da so vsi indeksi veljavni.

- `public Ptabela podtabela(int[] indeksi)`

  Vrne nov objekt tipa `Ptabela`, ki predstavlja kopijo pod-p-tabele p-tabele `this`, določene s podano tabelo indeksov. Lahko predpostavite, da je dolžina tabele `indeksi` manjša od števila dimenzij p-tabele `this` in da so vsi indeksi veljavni.

  Na primer, če objekt `p` predstavlja p-tabelo velikosti $3 \times 4 \times 2$, potem bi klic

  ```
  p.podtabela(new int[]{2})
  ```

  vrnil kopijo tretje »strani« p-tabele `p`. Vrnjeni objekt bi torej predstavljal p-tabelo velikosti 4×2. Klic

  ```
  p.podtabela(new int[]{2, 0})
  ```

  pa bi vrnil kopijo prve vrstice tretje »strani« p-tabele `p`.

- `public String toString()`

  Vrne predstavitev p-tabele `this` v obliki niza. Enodimenzionalna p-tabela z d_1 elementi naj bo predstavljena z nizom `[`a_0`, `a_1`, ..., `a_{d_1-1}`]`, kjer so $a_0, a_1, \ldots, a_{d_1-1}$ elementi p-tabele. P-tabela velikosti $d_1 \times d_2 \times d_3 \times \ldots \times d_n$ naj bo predstavljena z nizom `[`S_0`, `S_1`, ..., `S_{d_1-1}`]`, kjer so $S_0, S_1, \ldots, S_{d_1-1}$ nizi, ki predstavljajo posamezne pod-p-tabele velikosti $d_2 \times d_3 \times \ldots \times d_n$.

Testni primer 8

Testni razred:

```
import java.util.Random;

public class Test08 {

    public static void main(String[] args) {
        // delali bomo s tabelo 3 × 4 × 2
        int a = 3, b = 4, c = 2;

        System.out.println("Inicializacija:");
        Ptabela p = new Ptabela(new int[]{a, b, c});
        System.out.println(p.toString());
        System.out.println("----------");
```

```
        System.out.println("Polnjenje z elementi od -99 do 99:");
        Random random = new Random(12345);
        for (int i = 0;  i < a;  i++) {
            for (int j = 0;  j < b;  j++) {
                for (int k = 0;  k < c;  k++) {
                    p.nastavi(new int[]{i, j, k}, random.nextInt(199) - 99);
                }
            }
        }
        System.out.println(p.toString());
        System.out.println("----------");

        System.out.println("Elementi in podtabele:");
        System.out.println("p[2][0][1] = " + p.vrni(new int[]{2, 0, 1}));
        System.out.println("p[2][1] = " + p.podtabela(new int[]{2, 1}).toString());
        System.out.println("p[1] = " + p.podtabela(new int[]{1}).toString());
        System.out.println("p = " + p.podtabela(new int[]{}).toString());
    }
}
```

Izhod:

```
Inicializacija:
[[[0, 0], [0, 0], [0, 0], [0, 0]],
 [[0, 0], [0, 0], [0, 0], [0, 0]],
 [[0, 0], [0, 0], [0, 0], [0, 0]]]
----------
Polnjenje z elementi od -99 do 99:
[[[-95, 17], [24, 57], [-43, -72], [-2, 43]],
 [[27, 13], [52, 69], [64, -31], [60, 73]],
 [[1, 61], [-8, 23], [-56, 36], [45, -71]]]
----------
Elementi in podtabele:
p[2][0][1] = 61
p[2][1] = [-8, 23]
p[1] = [[27, 13], [52, 69], [64, -31], [60, 73]]
p = [[[-95, 17], [24, 57], [-43, -72], [-2, 43]],
     [[27, 13], [52, 69], [64, -31], [60, 73]],
     [[1, 61], [-8, 23], [-56, 36], [45, -71]]]
```

Opomba: Zavoljo večje preglednosti smo izpis tridimenzionalnih p-tabel umetno prelomili. Metoda `toString` naj tovrstnih prelomov ne vstavlja.

Namig

Bi lahko p-tabelo simulirali s pomočjo enodimenzionalne tabele?