Java tečaj

3. dio – Razredi i objekti (2). Kolekcije (1).

Zadaci za vježbu

V2

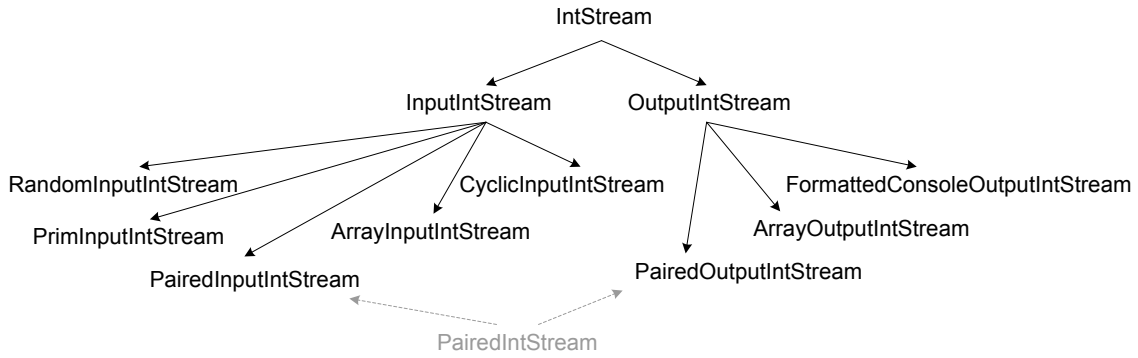
Zagreb, 2010.

1. Razredi i objekti (2). Kolekcije (1).

U nastavku slijedi niz zadataka vezanih uz treći dio tečaja. Za sve zadatke definiran je traženi izlaz programa koji treba poštivati. Ako tijekom razvoja programa koristite dodatne ispise, prije predaje zadataka te ispise treba ukloniti. Sve zadatke potrebno je smjestiti u paket `hr.fer.zemris.java.tecaj_3`. Potom zapakirajte kompletan Eclipse projekt u ZIP arhivu (ne RAR, ne TAR, ne TGZ, ne …, već ZIP), i to ćete uploadati u skladu s uputom.

1.1. Zadatak: Tokovi cijelih brojeva

Danas je u modernim programskim jezicima često u uporabi apstrakcija toka podataka. Tok možemo vizualizirati kao cijev koja spaja izvor podataka te potrošača podataka. S jedne strane, izvor (generator, proizvođač) podatke "gura" u tu cijev. Podatci putuju do drugog kraja, i tamo ih potrošač (ponor podataka) čita, točno onim redoslijedom kojim ih je proizvođač gurnuo.
Jedan od tipičnih primjera uporabe ovakve apstrakcije jest API komuniciranje s udaljenim aplikacijama (primjerice uporabom protokola TCP), ili čak obično čitanje / pisanje u datoteke.
Pri tome se tipično razlikuju dva kraja iste cijevi: stranu u koju upisujemo podatke tipično modeliramo razredom koji u imenu sadrži nešto tipa *Writer* ili *OutputStream*, dok stranu iz koje čitamo podatke tipično modeliramo razredom koji u imenu sadrži nešto tipa *Reader* ili *InputStream*.
U ovoj zadaći zabavljat ćemo se tokovima cijelih brojeva (podataka tipa `int`). Izgradit ćemo sljedeću hijerarhiju razreda:



Tok podataka može biti u dva stanja: otvoren ili zatvoren. Zatvoren tok ne može se koristiti za čitanje odnosno pisanje, pa će sve operacije koje se pokušaju napraviti nad zatvorenim tokom izazvati iznimku `IOException`. Sve ove razrede potrebno je smjestiti u paket `hr.fer.zemris.java.tecaj_3.intstreams`.

Pa krenimo redom. Razred `IntStream` je apstraktni razred, koji sadrži sljedeće apstraktne metode:

- `boolean isOpen();` javna metoda koja vraća `true` ako je tok otvoren; inače vraća `false`.

- `void close();` javna metoda koja zatvara tok.
- `int process(int[] buffer, int offset, int count, boolean blocking);` javna metoda koja prima spremnik, poziciju početnog elementa te broj elemenata koje treba obraditi. Ako je tok izlazni, pozivom ove metode pozivatelj je u `buffer` upisao brojeve koje treba "ugurati" u tok; ako je tok izlazni, pozivatelj nam je pripremio spremnik gdje ćemo smjestiti pročitane brojeve. Metoda vraća broj brojeva koje je obradila (zapisala iz spremnika u tok ili pročitala iz toka i spremila u spremnik). Taj broj sigurno neće biti veći od argumenta `count`. Taj broj može biti i nula, ako iz nekog razloga niti jedan podatak nije mogao biti obrađen, a argument `blocking` je postavljen na `false`. Argument `blocking` postavljen na `true` naređuje metodi `process` da mora obraditi barem jedan element (ne mora sve). Ako se metoda pozove s `blocking` postavljenim na `true`, a nije u mogućnosti ispuniti zahtjev da mora obraditi barem jedan element, tada će izazvati iznimku `FutileBlockingException` (izvedite ovu iznimku iz `IOException`, i smjestite je u paket `hr.fer.zemris.java.tecaj_3.intstreams`).

Sve ove metode bacaju `IOException`.

Razred `InputIntStream` također je apstraktni razred koji dodaje tri metode zajedno s njihovim implementacijama (implementacije se oslanjaju na metodu `process`):

- `int read();` je javna metoda koja čita jedan broj iz toka i vraća ga kao rezultat. Ako trenutno nema brojeva u toku, metoda blokira (ili izaziva iznimku ako blokiranje nema smisla).
- `int read(int[] buffer, int offset, int count, boolean blocking);` je javna metoda koja čita najviše `count` brojeva i smješta ih u predano polje počev od pozicije `offset`. Za detalje pogledati metodu `process`.
- `int read(int[] buffer, boolean blocking);` je javna metoda koja čita najviše onoliko brojeva koliko je polje veliko i smješta ih u predano polje (od početka). Za detalje pogledati metodu `process`.

Sve ove metode bacaju `IOException`.

Razred `RandomInputIntStream` je razred koji tražene brojeve stvara posredstvom slučajnog mehanizma. Jednom kada ga implementirate, napišite sljedeći primjer (smjestite ga u paket `hr.fer.zemris.java.tecaj_3.primjeri`) – razred `Primjer1`, čija je `main` metoda poprilici nešto ovakvoga (popraviti što ne valja):

```
int brojBrojeva = Integer.parseInt(args[0]);
InputIntStream iis = new RandomInputIntStream();
for(int i = 1; i <= brojBrojeva; i++) {
  System.out.println(i+". slučajni broj je: "+iis.read());
}
```

Razred `CyclicInputIntStream` ima javni konstruktor koji prima polje cijelih brojeva, interno ga kopira i pamti, te prilikom poziva funkcija za čitanje vraća te brojeve (nakon zadnjega ponovno se vraća na prvi, drugi, itd.). Primjerice (`Primjer2`):

```
InputIntStream iis = new CyclicInputIntStream(
                       new int[] {1,2}
                     );
for(int i = 0; i < 6; i++) {
  System.out.print(iis.read()+", ");
}
```

će ispisati: `1, 2, 1, 2, 1, 2, .`

Razred `PrimInputIntStream` ima javni konstruktor koji prima gornju ogradu, te prilikom poziva funkcija za čitanje vraća prim-brojeve manje od zadane granice (ispravnim redoslijedom). Ako se vrate svi prim-brojevi manji od zadane granice, pokušaj dohvata sljedećeg treba rezultirati bacanjem `EOFException`-a. Primjerice (`Primjer3`):

```
InputIntStream iis = new PrimInputIntStream(100);
for(int i = 0; i < 6; i++) {
  System.out.print(iis.read()+", ");
}
```

će ispisati: `1, 2, 3, 5, 7, 11, .`

Razred `ArrayInputIntStream` ima javni konstruktor koji prima polje cjelih brojeva, pamti njegovu kopiju te prilikom poziva funkcija za čitanje vraća brojeve iz tog polja, ispravnim redosljedom. Ako se vrate svi brojevi, pokušaj dohvata sljedećeg treba rezultirati bacanjem `EOFException`-a.

Razred `OutputIntStream` također je apstraktni razred koji dodaje tri metode zajedno s njihovim implementacijama (implementacije se oslanjaju na metodu `process`):

- `void write(int value);` je javna metoda koja zapisuje `value` u tok. Ako pisanje trenutno nije moguće, metoda blokira (ili izaziva iznimku ako blokiranje nema smisla).
- `int write(int[] buffer, int offset, int count, boolean blocking);` je javna metoda koja zapisuje najviše `count` brojeva u tok (čita ih iz predanog polja počev od pozicije `offset`). Za detalje pogledati metodu `process`.
- `int write(int[] buffer, boolean blocking);` je javna metoda koja u tok zapisuje najviše onoliko brojeva koliko je predano polje veliko (čita ih iz predanog polja, od početka). Za detalje pogledati metodu `process`.

Sve ove metode bacaju `IOException`.

Razred `FormattedConsoleOutputIntStream` ima javni konstruktor koji prima prefiks i sufiks kao string, te prilikom poziva funkcija za pisanje brojeve ispisuje na System.out. Primjer (`Primjer4`):

```
OutputIntStream ois =
       new FormattedConsoleOutputIntStream("[","]\n");
for(int i = 0; i < 3; i++) {
  ois.write(i);
}
```

će ispisati:

```
[0]
[1]
[2]
```

Razred `ArrayOutputIntStream` ima javni konstruktor koji prima inicijalnu veličinu polja, te prilikom poziva funkcija za pisanje brojeve nadodaje u to polje. Kada se polje potroši, realocira se novo dvostruke veličine, i tako koliko god puta trebalo. Razred dodaje jednu javnu metodu:

- `int[] toIntArray();` koja vraća novo polje veliko točno onoliko koliko je doista zapisano brojeva, i koje je popunjeno tim brojevima.

Primjer (`Primjer5`):

```
ArrayOutputIntStream aois =
       new ArrayOutputIntStream(3);
for(int i = 0; i < 4; i++) {
  aois.write(i);
}
int[] sadrzaj = aois.toIntArray();
System.out.println("Veličina: "+sadrzaj.length);
System.out.println("Sadržaj: "+Arrays.toString(sadrzaj));
```

će ispisati:

```
Veličina: 4
Sadržaj: [0,1,2,3]
```

Razred `PairedIntStream`, `PairedInputIntStream` i `PairedOutputIntStream` usko su povezani. Razred `PairedIntStream` sadrži interni spremnik brojeva fiksne veličine koji dobiva preko konstruktora (spremnik izvesti kao obično polje). Taj spremnik koristi kao strukturu *red*. Sam konstruktor odmah stvara i pamti po jedan primjerak internog razreda `PairedInputIntStream` te `PairedOutputIntStream` koji čitaju odnosno zapisuju u taj spremnik. Kako ovi razredi ne trebaju raditi u višedretvenom okruženju, u slučaju da se pozove operacija pisanja nad punim redom u blokirajućem načinu, ili da se pozove operacija čitanja nad

praznim redom u blokirajućem načinu, metoda treba baciti `FutileBlockingException`. Razred `PairedIntStream` nudi i dvije javne metode:

- `InputIntStream getInputIntStream();` koja vraća referencu na stvoreni primjerak razreda `PairedInputIntStream` (dakle, ta metoda NE stvara novi primjerak)
- `OutputIntStream getOutputIntStream();` koja vraća referencu na stvoreni primjerak razreda `PairedOutputIntStream` (dakle, ta metoda NE stvara novi primjerak).

Pojam interni razred znači da je to razred koji je deklariran unutar drugog razreda. Tako razredi `PairedInputIntStream` i `PairedOutputIntStream` nisu smješteni u svoje vlastite .java datoteke, već su napisani kao ugniježđeni razredi razreda `PairedIntStream`.

Primjer (`Primjer6`):

```
PairedIntStream paired = new PairedIntStream(2);
InputIntStream iis = paired.getInputIntStream();
OutputIntStream ois = paired.getOutputIntStream();
ois.write(1);
ois.write(2);
// više ne smijemo zvati write jer bismo dobili iznimku!
System.out.println("Čitam: "+iis.read());
// sada smo oslobodili jedno mjesto, može još jedan write:
ois.write(3);
System.out.println("Čitam: "+iis.read());
System.out.println("Čitam: "+iis.read());
System.out.println("Čitam: "+iis.read());
```

Ovo će rezultirati ispisom:

```
Čitam: 1
Čitam: 2
Čitam: 3
```

pa iznimkom `FutileBlockingException`, jer `read()` čita u blokirajućem modu. Da smo zvali neku od read funkcija koje primaju polje i vraćaju broj pročitanih znakova, i da smo tražili neblokirajuću operaciju, ne bismo dobili iznimku, već bi povratna vrijednost takve funkcije bila nula.

Napišite i implementirajte sve navedene razrede i primjere.

1.2. Zadatak: Kolekcije

Tekstualna datoteka sadrži sljedeće podatke o studentima:

- jmbag
- prezime
- ime
- grupa

Svaki redak sadrži podatke za jednog studenta. Unutar retka, podatci su razdvojeni znakom tab.

Modelirajte informacije o studentu zasebnim razredom `Student`, koji je neizmjenjiv. Razred smjestite u paket `hr.fer.zemris.java.tecaj_3.barkod`. Pažljivo implementirajte metode `hashCode` i `equals` (usporedbu radite samo temeljem elemenata `jmbag`, `prezime` i `ime`). Redefinirajte i metodu `toString` (format je propisan primjerom na kraju ovog dokumenta).

Proučite sučelja `java.lang.Comparable` te `java.util.Comparator`.

- http://java.sun.com/javase/6/docs/api/java/lang/Comparable.html
- http://java.sun.com/javase/6/docs/api/java/util/Comparator.html

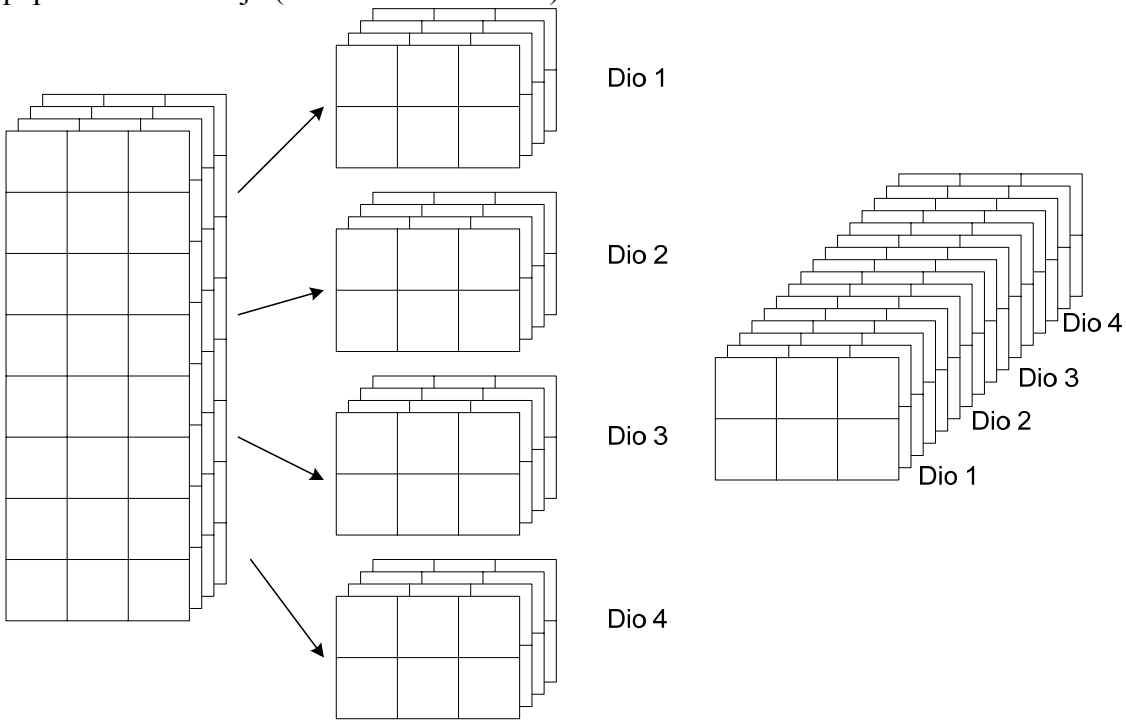
Definirajte prirodan poredak za razred `Student`, i to tako da se najprije provjere prezimena, pa imena pa tek tada matični brojevi. Redoslijed bi trebao biti rastući. Za usporedbu "naših" slova koristite odgovarajući `Collator` za hrvatski `Locale`. Ta dva objekta stavite kao privatne statičke konstante razreda `Student`:

```
private final static Locale HR_LOCALE = new Locale("HR");
private final static Collator HR_COLLATOR =
                              Collator.getInstance(HR_LOCALE);
```

U knjižarama se mogu nabaviti papiri dimenzija A4 koji se sastoje od niza naljepnica. Papir je organiziran tako ima *m* redaka i *n* stupaca (primjerice, 8 redaka i 3 stupca, što ukupno daje 24 naljepnice po papiru).

Želimo napisati program koji će pomoći u štampanju naljepnica za studente. Pravila su sljedeća. Za svakog studenta mora se štampati cijeli broj redaka (npr. uz 8x3 u redu je tražiti 3 naljepnice po sudentu, ili 6 naljepnica po studentu, ali nije dozvoljeno tražiti 4 naljepnice po studentu). Dodatno, sve naljepnice jednog studenta moraju se nalaziti na istom papiru, slijedno. To znači da je, uz pretpostavku da je papir dimenzija 8x3, dozvoljeno tražiti primjerice 6 naljepnica po studentu (tada po papiru idu 4 studenta), ali nije dozvoljeno tražiti 9 naljepnica po studentu (jer bi na papiru tada ostalo ili neiskorištenih 6 naljepnica, ili ako bismo tu započeli novog studenta, njegove naljepnice bi se morale preliti na sljedeći papir – još tri, a to nije dozvoljeno).

Prilikom štampanja naljepnica, štampanje se obavlja jednostrano, i potom naštampani papiri idu na rezanje (vidi sliku u nastavku).



Slika prikazuje slučaj u kojem su papiri dimenzija 8x3, a za svakog studenta se štampa po 6 naljepnica (lijevi dio slike). Nakon toga se svi naštampani papiri naljepnica posebnim strojem prerežu u četiri dijela (sredina slike), i dijelovi se slože jedan na drugoga (desni dio slike).

Vaš je zadatak osigurati da ovako dobiven konačni bunt naljepnica bude sortiran.

Dakle, napišite program `IspisNaljepnica` koji prima sljedeće parametre:

```
IspisNaljepnica datoteka m n k
```

gdje jek broj naljepnica za jednog studenta. Za primjer sa slike program bismo pozvali ovako:

```
IspisNaljepnica C:\studenti.txt 8 3 6
```

Pseudokod metode `main` mora odgovarati ovome:

```
lista_studenata lista = ucitaj_listu_studenata(datoteka)
lista_studenata zaIspis = novaPrilagodenaLista(lista,m,n,k)
ispisi(zaIspis,m,n,k)
```

Ako nešto ne valja s parametrima (primjerice, nije zadovoljeno neko od opisanih ograničenja koja vrijede na *m*, *n* i *k*), metode `novaPrilagodenaLista` i `ispisi`

trebaju baciti `IllegalArgumentException` uz odgovarajući opis. U metodi `main` to uhvatiti, ispisati prikladnu poruku i završiti izvođenje programa. Liste iz pseudokoda svakog studenta sadrže samo jednom (znači, ne za svaku naljepnicu po jednog). Metoda `ispisi` ispisuje elemente liste od prvog prema zadnjem (ne radi nikakve operacije nad listom – za to se je trebala pobrinuti metoda `novaPrilagodenaLista`). Očekivani format ispisa vidljiv je iz sljedećeg primjera.

Datoteka c:\studenti.txt:

```
0012345678      Perić           Pero            G1
0023456789      Ivić            Ivana           G2
0034567890      Matić           Mirjana         G1
0045678901      Antić           Ante            G1
0056789012      Barić           Barbara         G2
0067890123      Žabčić          Stjepan         G1
```

Uz poziv programa:

```
IspisNaljepnica C:\studenti.txt 8 3 12
```

očekivani ispis je:

```
(0045678901) Antić, Ante [G1]
(0034567890) Matić, Mirjana [G1]
-- nova stranica
(0056789012) Barić, Barbara [G2]
(0012345678) Perić, Pero [G1]
-- nova stranica
(0023456789) Ivić, Ivana [G2]
(0067890123) Žabčić, Stjepan [G1]
```

Redak koji sadrži ispis studenta rezultat je poziva metode `toString` nad studentom. Uočimo, metoda ispiši također ne ispisuje za svaku naljepnicu po jedan redak, već samo ispisuje studente koji će dobiti svoje naljepnice na tom papiru, korektnim redoslijedom.

Evo još jednog primjera.

Datoteka c:\studenti.txt:

```
0012345678      Perić           Pero            G1
0023456789      Ivić            Ivana           G2
0034567890      Matić           Mirjana         G1
0045678901      Antić           Ante            G1
0056789012      Barić           Barbara         G2
0067890123      Žabčić          Stjepan         G1
0078901234      Dandić          Dante           G2
0089012345      Ćanić           Tihana          G2
```

Uz poziv programa:

```
IspisNaljepnica C:\studenti.txt 6 3 6
```

očekivani ispis je:

```
(0045678901) Antić, Ante [G1]
(0078901234) Dandić, Dante [G2]
(0012345678) Perić, Pero [G1]
-- nova stranica
(0056789012) Barić, Barbara [G2]
(0023456789) Ivić, Ivana [G2]
(0067890123) Žabčić, Stjepan [G1]
-- nova stranica
(0089012345) Ćanić, Tihana [G2]
(0034567890) Matić, Mirjana [G1]
```

Prilikom rješavanja zadatka zabranjeno je direktno koristiti polja. Umjesto toga, oslonite se na uporabu kolekcija iz *Java Collection Framework*-a (i to isključivo onih koje su po tipu `Set` ili `List`).

Ako detektira da datoteka sa studentima sadrži duplikat zapisa, metoda ucitaj treba baciti iznimku `IllegalArgumentException` (uz prikladan opis). Glavi program treba uhvatiti tu iznimku, ispisati odgovarajuću poruku i završiti s izvođenjem.

Naputci:

- lista može sadržavati bilo kakve objekte – čak i druge liste, skupove i sl. Primjerice, lista koja sadrži listu skupova stringova deklarirat ćemo ovako

  ```
  List<List<Set<String>>> lista =
                  new LinkedList<List<Set<String>>>();
  ```

- za sortiranje elemenata liste koristiti prikadnu metodu razreda `java.util.Collections` (ako elementi imaju prirodan poredak, onda onu koja prima samo listu; ako nemaju, onda onu koja prima dodatni komparator koji se može na mjestu implementirati kao anonimni razred).