Dodatne naloge – Vmesniki in lambde

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svojo rešitev preverite z množico testnih razredov in pripadajočih izhodnih datotek.

1 Zadnje ujemanje

Napišite razred `ZadnjeUjemanje` z metodo

```
public static <T> T zadnji(Collection<T> zbirka, Predicate<T> pogoj)
```

ki vrne zadnji element zbirke, ki izpolnjuje podani pogoj, oziroma `null`, če noben element zbirke ne izpolnjuje pogoja. Lahko predpostavite, da je vrstni red elementov v zbirki natančno določen (ta lastnost med drugim velja za zbirke tipov `List` in `NavigableSet`).

2 Kumulativa

Napišite razred `Kumulativa` z metodo

```
public static <T> List<T> kumulativa(List<T> seznam, BinaryOperator<T> operator)
```

ki vrne kumulativo podanega seznama glede na podani dvojiški operator (označimo ga s $\circ$). Če podani seznam vsebuje elemente $s_0, s_1, \ldots, s_{n-1}$, potem kumulativo tvorijo elementi $t_0, t_1, \ldots, t_{n-1}$, pri čemer je $t_0 = s_0$ in $t_i = t_{i-1} \circ s_i$ za vsak $i \in \{1, \ldots, n-1\}$. Na primer, kumulativa seznama [3, 7, −2, 9, 6] glede na operacijo seštevanja je seznam [3, 10, 8, 17, 23].

Metodo `kumulativa` nato uporabite v metodi

```
public static <T> List<T> doslejNajvecji(List<T> seznam, Comparator<T> primerjalnik)
```

ki za podani seznam vrne nov seznam iste dolžine, v katerem je i-ti element enak največjemu (glede na podani primerjalnik) izmed prvih i elementov podanega seznama. Na primer, pri seznamu [3, 7, −2, 9, 6] in običajni urejenosti števil bi dobili rezultat [3, 7, 7, 9, 9].

3 Urejanje po vrednostih funkcije

Napišite razred `UrejanjePoFunkciji` z metodo

```
public static <T, R extends Comparable<R>> void urediPoFunkciji(
        List<T> seznam, Function<T, R> funkcija)
```

ki podani seznam uredi glede na vrednosti podane funkcije na elementih seznama. Če za funkcijo f in elementa seznama p in q velja $f(p) < f(q)$, potem se mora v urejenem seznamu element p nahajati pred elementom q. V metodi si pomagajte s klicem metode `sort` iz razreda `List`.

Na primer, če seznam $[-3, 5, -7, 8, 2, -6]$ uredimo glede na funkcijo $f(x) = |x|$, dobimo seznam $[2, -3, 5, -6, -7, 8]$, urejanje glede na funkcijo $f(x) = -x$ pa nam dá seznam $[8, 5, 2, -3, -6, -7]$.

Zapis `R extends Comparable<R>` v seznamu tipskih parametrov pove, da mora biti tip `R` podtip tipa `Comparable<R>` (če je `R` razred, mora implementirati vmesnik `Comparable<R>`). Generični tip `R` lahko potemtakem nadomestimo s tipom `Integer`, `String` ali `Cas`, ne pa s tipom `Object` ali `Oseba`.

Metodo `urediPoFunkciji` nato uporabite še v sledečih metodah:

- `public static void urediPoAbsolutniVrednosti(List<Integer> stevila)`

  Podani seznam uredi po absolutnih vrednostih posameznih elementov.

- `public static void urediPoDolzini(List<String> nizi)`

  Podani seznam uredi po dolžinah posameznih nizov.

- `public static <T extends Comparable<T>> List<Integer> vrstniRed(List<T> seznam)`

  Vrne nov seznam, v katerem i-ti element vsebuje indeks elementa podanega seznama, ki bi se v naravno urejeni različici podanega seznama nahajal na i-tem mestu. Na primer, za seznam [bojan, darja, eva, ana, cvetko] naj metoda vrne seznam [3, 0, 4, 1, 2].

4 Obrat primerjalnika

Napišite razred `ObratPrimerjalnika` z metodo

```
public static <T> Comparator<T> obrni(Comparator<T> primerjalnik)
```

ki vrne primerjalnik, glede na katerega sodi objekt a pred objekt b natanko tedaj, ko objekt b glede na podani primerjalnik sodi pred objekt a.

Metodo `obrni` nato uporabite v metodi

```
public static <T> void urediPadajoce(List<T> seznam, Comparator<T> primerjalnik)
```

ki podani seznam padajoče uredi glede na podani primerjalnik. Pomagajte si tudi z metodo `sort` iz razreda `List`.

5 `Comparable` → `Comparator`

Napišite razred `ComparableVComparator` z metodo

```
public static <T extends Comparable<T>> Comparator<T> pretvori()
```

ki vrne primerjalnik, ki objekta primerja glede na njuno naravno urejenost. Zapis `T extends Comparable<T>` v seznamu tipskih parametrov pove, da mora biti tip `T` podtip tipa `Comparable<T>` (če je `T` razred, mora implementirati vmesnik `Comparable<T>`).

6 Kombinacija primerjalnikov

Napišite razred `KombinacijaPrimerjalnikov` z metodo

`public static <T> Comparator<T> kombinacija(List<Comparator<T>> primerjalniki)`

ki vrne primerjalnik, ki par elementov tipa `T` primerja s prvim primerjalnikom v podanem nepraznem seznamu in vrne rezultat primerjave, če se elementa po tem kriteriju razlikujeta, sicer pa elementa primerja še z drugim primerjalnikom v podanem seznamu. Če se tokrat razlikujeta, vrne rezultat primerjave, sicer pa ju primerja še s tretjim primerjalnikom itd. Če se elementa po nobenem primerjalniku ne razlikujeta, je končni rezultat metode enak 0.

7 Iterator po tabeli

Razred `OvojnikTabele` ...

```
public class OvojnikTabele {
    private Object[] tabela;

    public OvojnikTabele(Object[] tabela) {
        this.tabela = tabela;
    }
}
```

... dopolnite tako, da se bo zanka v sledeči kodi sprehodila po elementih tabele `t`:

```
Object[] t = ...;
OvojnikTabele tab = new OvojnikTabele(t);
for (Object element: tab) {
    ...
}
```

8 Sprehod s preskoki

Napišite razred `SprehodSPreskoki<T>`, ki omogoča sprehajanje (z zanko for-each) po podanem seznamu s podanim korakom. Razred naj vsebuje konstruktor

`public SprehodSPreskoki(List<T> seznam, int korak)`

pri čemer parameter `seznam` predstavlja seznam, po katerem se bomo sprehajali, neničelni parameter `korak` pa podaja korak sprehoda (k). Če je korak k pozitiven, potem sprehod po seznamu z zanko for-each pričnemo z elementom na indeksu 0, nato pa po vrsti (dokler ne prispemo do konca seznama) obiščemo elemente na indeksih k, $2k$, $3k$ itd. Če je korak k negativen, pa sprehod pričnemo na zadnjem elementu seznama, nato pa indeks po vrsti zmanjšujemo za $|k|$, dokler ne prispemo do začetka seznama. Na primer, koda

```
List<Integer> seznam = List.of(20, 21, 22, 23, 24, 25, 26, 27, 28, 29);
SprehodSPreskoki naprej = new SprehodSPreskoki(seznam, 4);
for (int element: naprej) {
    System.out.print(element + " ");
}
System.out.println();
SprehodSPreskoki nazaj = new SprehodSPreskoki(seznam, -3);
```

```
for (int element: nazaj) {
    System.out.print(element + " ");
}
System.out.println();
```

bi izpisala

```
20 24 28
29 26 23 20
```

9 Kombinacija iteratorjev

Iterator ima k *korakov*, če potrebuje k klicev metode `next`, da prispe do konca svojega sprehoda. *Kombinacija* iteratorjev I_0, I_1, ..., I_{n-1}, od katerih imajo vsi po k korakov, je iterator I z nk koraki, pri katerem i-ti klic metode `next` (za $i \in \{0, \ldots, nk-1\}$) vrne trenutni element pod iteratorjem $I_{i \bmod n}$ in premakne ta iterator na naslednji element.

Napišite razred `KombinacijaIteratorjev<T>` z metodo

```
public static <T> Iterator<T> kombinacija(List<Iterator<T>> iteratorji)
```

ki vrne kombinacijo iteratorjev iz podanega seznama, pri čemer lahko predpostavite, da imajo vsi iteratorji enako število korakov.