Java lambda labor

Készítette: Goldschmidt Balázs, BME IIT, 2023.

Alapfeladat

A feladat a korábban már megoldott sörnyilvántartó alkalmazásunk refaktorálása lesz. A refaktorálás azt jelenti, hogy az adott modul, osztály, rendszer belső felépítésén, algoritmusain úgy módosítunk, hogy közben sem a külső felülete, sem a nyújtott funkcionalitás nem változik.

Jelen esetben az a célunk, hogy a korábbi megoldás azon elemeit, amik lambdával helyettesíthetők, lecseréljük. Kiindulásként használjuk az általunk korábban már elkészített alkalmazást.

A feladatot megoldó osztályokat a lambeer nevű csomagban (package) hozzuk létre!
Ehhez a saját korábbi megoldásunkban a csomagnevet módosítani kell.

1 Komparátorok cseréje

Ahogy az előadás-példákon is láttuk, a komparátorok használata lambda kifejezésekkel sokkal egyszerűbb, áttekinthetőbb és tisztább kódot eredményez.

a) A list parancsban a meglevő komparátorokat (*NameComparator*, *StrengthComparator*, *StyleComparator*) cseréljük le lambda kifejezésekre. Ahol a komparátort paraméterként átadjuk, ott alkalmazzunk hagyományos lambda kifejezést (*x -> expression* formában).
b) Próbáljuk ki, hogy az így lecserélt komparátorok valóban az elvárt működést adják-e.

Tipp

a *Collections.sort* metódust nevek rendezésekor például így hívhatjuk meg:

```
Collections.sort(beers,
      (b1,b2) -> b1.getName().compareTo(b2.getName())
);
```

2 Parancsok lambdásítása

Vegyük észre, hogy a parancsaink mind azonos szignatúrával (fejléccel) rendelkeznek, csak a nevük különbözik! Ha az eredeti megoldásunk a Parancs (Command) mintát követte, és minden parancsot külön osztályban valósítottunk meg, akkor ez még inkább szembeötlő.

Alakítsuk át úgy a parancsfeldolgozást, hogy a parancsokat egy *HashMap*-be tesszük, ahol a kulcs a parancs neve (amit a bemenetről kapunk), az érték pedig a futtatandó kód. Ez utóbbit lambda-kifejezésként is megadhatjuk!

a) Készítsünk egy *Command* interfészt (ha még nem lenne)

   Az interfész egy metódussal rendelkezik: *void execute(String[] args)*

b) A parancsfeldolgozás előtt hozzunk létre egy *HashMap*-et (*commands*), ami *String*, *Command* párokat tartalmazhat!

c) Töltsük fel a commands hashtáblát a parancsainkkal! A nevek adottak, az értékek legyenek a parancsimplementáló metódusok referenciái, pl:

```
commands.put("list", Main::list);
```

   Ha lenne olyan parancsunk, amit nem tettünk függvénybe, akkor azt lambda-kifejezésként adjuk át a hashmapnek!

d) A parancsfeldolgozó ciklust alakítsuk át! A *switch-case* vagy *if-else-if...* struktúra helyett annyira van szükség, hogy ha jön egy parancs, akkor a hashtáblából vegyük ki a hozzá tartozó parancsot, és hajtsuk végre az eddigi paraméterezéssel!

   Ha egy parancs hiányzik a kulcsok közül, jelezzük a felhasználónak, hogy ismeretlen parancsot adott!

e) Próbáljuk ki, hogy a parancsfeldolgozás az eddig megszokott működést nyújtja-e!

Tippek:

A parancs végrehajtása:

```
commands.get(cmd[0]).execute(cmd);
```

A létezés ellenőrzését végezhetjük külön is (*Map.contains*), de azt is vizsgálhatjuk, hogy a get *null*-t ad-e vissza. Ez utóbbi esetben a fenti sort két részre kell szedni.

3 Komparátorok név alapján

A következő két feladat célja az, hogy összetett komparátorokat készítsünk, ahol futási időben dől el, hogy milyen sorrendben kell alkalmazni az egyes kompartorokat.

Első lépésben tegyük lehetővé, hogy a komparátorokra a paraméterben megadott nevekkel tudjunk hivatkozni! Cél, hogy a listázás során a név alapján egy map-ből vegyük ki a névhez tartozó komparátort.

Javasolt megvalósítási lépések:

a) Hozzunk létre egy statikus attribútumot (comps) a komparátorok tárolására! Ez

legyen egy Map, ami String (a komparátor neve) és *Comparator<Beer>* között hoz létre leképezést

b) Egy statikus inicializáló blokkban töltsük fel a comps map-et a komparátorainkkal! Ehhez használjuk a Comparator interfész comparing metódusát!

   Kelleni fog komparátor a name, strength és style paraméterekhez.

c) A list metódusban hozzunk létre egy *Comparator<Beer>* típusú változót (cmp), ez fogja a rendezési elvet tartalmazni. Kezdőértéknek adjuk neki a comps-ban a "name" kulcshoz tartozó értéket!

   Ha a parancs paramétere szerepel a comps kulcsai között, vegyük ki a hozzá tartozó komparátort és a cmp változó mutasson erre!

d) A rendezést végezzük a cmp alapján.
e) Végül a szokott módon írassuk ki a sör-listánk elemeit.

4 Kombinált komparátorok

A feladat során azt szeretnénk elérni, hogy a *list*-ben alkalmazott rendezés legyen garantáltan konzervatív. Ez azt jelenti, hogy ha az éppen végrehajtott rendezés során két elem azonosnak számít, akkor a sorrendjüket a rendezés előtti sorrendjük határozza meg. Itt most csak a *list* parancsban zajló rendezések számítanak, a kollekció más parancsokban végrehajtott átrendezgetésének ne legyen hatása.

Azt is szeretnénk, ha a végső változatban a *list* parancs tetszőleges számú argumentummal hívva a megadott sorrendben hajtaná végre a rendezést a kiíratás előtt.

Az alapötlet a következő. Ha a listet az alábbi módon hívjuk meg:

```
list name
list strength
list style
```

akkor az utolsó esetben azt várjuk, hogy a rendezés alapja már a *style*, *strength*, *name* legyen, ebben a csökkenő erősorrendben. Ehhez a *list* futásai között el kell tárolnunk a komparátorokat és a sorrendjüket. Az egyes listázásoknál pedig először a komparátorok sorrendjét állítjuk át, majd ez alapján készítjük az összetett komparátort, ami a rendezésben a döntést hozza.

Bővítsük a list metódus képességeit! A korábbi listázások rendezési elve a sör-listánk aktuális állapotától függetlenül ne vesszen el, hanem legyen explicit módon figyelembe véve (konzervatív rendezés). Ezt kombinált komparátor készítésével érjük el.

Javasolt megvalósítási lépések:

a) Hozzunk létre egy *List<String>* statikus tagváltozót (*lparams*), ami majd a rendezési elvek sorrendjét fogja definiálni. Ebben fogjuk a paraméterek nevét a rendezés során

figyelembe vett erősségüknek megfelelően tárolni.

b) A statikus inicializáló blokkban az *lparams* dinamikus típusa legyen *LinkedList<String>*, és töltsük fel a *comps* map kulcsaival.

c) A *list* metódusban az eddigiek mellett a kapott paraméter alapján módosítsuk a komparátor-nevek sorrendjét az *lparams* listában!

   A cél, hogy az éppen kapott nevet töröljük és helyezzük a lista elejére.

d) A *list* metódusban a sorrendezést az *lparams*-ban levő nevek alapján végezzük. Ehhez össze kell fűznünk a nevek sorrendjében a komparátorainkat, és ezt a kombinált komparátort kell a rendezés során használni. Ne feledkezzünk meg a *Comparator* interfész kedves metódusairól, nagy segítségünkre lehetnek!

e) Próbáljuk ki, hogy működik-e a megoldásunk!