Java Util labor

Készítette: Goldschmidt Balázs, BME IIT, 2015.

A feladatok megoldásához felhasználandó osztályok leírásait az alábbi URL-en találja meg:

http://download.oracle.com/javase/8/docs/api/

A feladat célja egy sörnyilvántartó alkalmazás készítése, amely a szabványos bemeneten fogad parancsokat, és a szabványos kimenetre írja ki az eredményt. A parancsokat soronként olvassa és dolgozza fel. Amikor az "exit" parancsot kapja, kilép.

A feladatok megoldásához felhasználandó osztályok leírásait az alábbi URL-en találjuk meg:

http://download.oracle.com/javase/8/docs/api/

1 Tárolandó adatszerkezet kialakítása

a) Készítsünk egy sör (*Beer*) osztályt, amelynek van név (*name, String*), jelleg (*style, String*), alkoholfok (*strength, double*) attribútuma!
b) Az attribútumokat lehessen konstruktorból inicializálni!
c) Az attribútumokhoz legyenek getter metódusok!
d) A *toString()* metódus által visszaadott Stringben szerepeljen az attribútumok értéke is!
e) A főprogramban hozzunk létre 2 sört és írassuk ki őket a szabványos kimenetre!

2 Parancsok fogadása és feldolgozása

A nyilvántartás működéséhez az alkalmazásnak képesnek kell lennie arra, hogy menet közben további söröket lehessen felvenni. Ehhez az alkalmazás sorokat olvas, majd a sorokat a whitespace-ek mentén *String*-ekké töri. A töréshez használjuk a *String* osztály *split* metódusát! A *split* metódus paramétere legyen egy szóközt tartalmazó *String* ("␣"). Az így kapott tömb első *String*-je a parancs, a többi a parancs argumentuma.

Pl. ha a bemeneti sor:

```
"add Guinness stout 4.2"
```

akkor a *split* eredménye:

```
{"add", "Guinness", "stout", "4.2"}
```

a) Módosítsuk úgy a főprogramot, hogy az beolvas egy sort, majd a fenti módon széttöri, és kiírja a kapott tömb első elemét, majd azt, hogy hány elemű a tömb.
b) Módosítsuk tovább a főprogramot: az alkalmazás ciklusban olvassa a sorokat, és csak akkor lépjen ki, ha a beolvasott sor első szava "exit".

3 Kollekció alapok

Az alkalmazást bővítsük új sörök bevitelét és a meglevők listázását lehetővé tevő parancsokkal! Az első két parancsa az **add** és a **list**.

a) Az **add** parancs hatására új sör (*Beer*: név (*name*), jelleg (*style*), alkoholfok (*strength*)) jön létre, és adódik a meglevők listájához, pl*:*
   - add Guinness stout 4.2
   - add Leffe_bruin brown_ale 6.5
   - add Dr_Eher pilsner 5.2

   A sörök tárolására használjunk *ArrayListet*! Figyeljünk arra, hogy az *ArrayList* a parancsokat megvalósító metódusok számára elérhető legyen! Ha a metódusaink statikusak, akkor az *ArrayList* is legyen az.

b) A **list** paranccsal a nyilvántartás tartalma íratható ki (tetszőleges sorrendben).

Minden parancshoz egy saját függvényt kell implementálni. A függvények fejléce a következő mintát kövesse (ahol a "fun" helyett az adott parancs neve áll):

```
protected void fun(String[] cmd)
```

A parancsok feldolgozása a következő módon történjen. Készíteni kell egy *if-else if-...* sorozatot, ahol az egyes *if*-ek azt ellenőrzik, hogy a beolvasott parancs (az előző pontban előálló tömb első eleme) egy adott előre definiált paranccsal egyezik-e. Ha igen, akkor meghívja a parancsot megvalósító függvényt. Pl:

```
if ("add".equals(cmd[0])) {
      add(cmd);
}
```

4 Szerializálás

Az alkalmazásnak legyen beolvasó és kiíró parancsa (**load**, **save**), amelyek szerializáltan írják ki az sörök listáját a parancsok argumentumában megadott fájlba! Ne feledje, hogy a tároló elemeit nem kell egyesével kiíratni, elég a tárolót magát kiírni!

5 Comparator

Bővítsük az alkalmazás listázó parancsát (**list**)! A listázás sorrendjét határozza meg a parancs opcionális argumentuma:

name - név
style - jelleg
strength - alkoholfok

A feladat megoldásához készítsünk minden rendezésfajtához egy-egy *java.util.Comparator* implementációt (*NameComparator*, *StyleComparator*, *StrengthComparator*). A lista rendezéséhez használjuk a *java.util.Collections* osztály megfelelő metódusát!

Szorgalmi feladat: ha több opcionális argumentum is van, akkor a rendezés kombinált legyen, pl: *"list style name"* parancs esetén az azonos jellegű söröket névsorrendben listázzuk.

6 Szűrés

a) Bővítsük az alkalmazást név szerinti kereséssel (**search** parancs). Ez paraméterként kapjon egy String-et, és csak azokat a söröket listázza ki, amelyek neve egyezik a paraméter értékével. Használjuk a for-each ciklust az elemeken való iteráláshoz!
b) Bővítsük az alkalmazást szabadszöveges kereséssel (**find** parancs). Ez paraméterként kapjon egy String-et, és csak azokat a söröket listázza ki, amelyek nevében szerepel a paraméter értéke! Pl. ha *alma*, *körte*, *barack* nevű söreink vannak, akkor a "**find a**" parancs kilistázza az *alma* és a *barack* nevű söröket.

7 Törlés

Az alkalmazáshoz készítsen törlő parancsot (**delete**), ami a neve alapján töröl egy sört. Használjunk iterátort a kereséshez és a törléshez!

Szorgalmi feladat: a kereséshez használja a *java.util.Collections.binarySearch* metódust az API leírásban megadott módon.

8 Szűrés 2

A *search* és a *find* parancsok is kaphassanak extra paramétert, ami alapján keresnek:

name - név
style - jelleg
strength - alkoholfok (*search*: pontos egyezés, *find*: legalább ekkora érték kell)
weaker - alkoholfok (csak *find* esetén: maximum ekkora érték lehet)

Pl. *"find style pils"* ereménye minden olyan sör, aminek a jellegében szerepel a *"pils"* sztring.

---

9 PQueue osztály

Implementáljon generikus **PQueue<T extends Comparable>** osztályt!
Az implementációban az elemek tárolására használjon **ArrayList**-et, a maximumkiválasztáshoz használja a **java.util.Collections** osztályt!
A **PQueue** interfésze az alábbi metódusokat valósítsa meg:

- *konstruktor* - létrehoz egy üres prioritási sort (**P**riority **Queue**-t)
- **void push(T t)** - t elemet a sorba helyezi
- **T pop()** - a sorban található legnagyobb elemet visszaadja és törli a sorból
- **T top()** - a sorban található legnagyobb elemet visszaadja, de nem törli
- **int size()** - visszaadja a sorban tárolt elemek számát
- **void clear()** - törli a sorban levő elemeket

A *pop* és a *top* metódusok dobjanak (általunk definiált) **EmptyQueueException**-t!

Készítsen egy **Test** osztályt, ami a **main** metódusában kipróbálja egy *String*gel parametrizált *PQueue* összes metódusát!

10 For-each

Tegye lehetővé, hogy az előző feladatban elkészített osztályra meg lehessen hívni a Java *for-each* parancsát. Forduljon le az alábbi kódrészlet, ami kiírja a sor tartalmát nagyság szerint csökkenő sorrendben(4,3,2,1). A feladathoz biztosítani kell egy olyan objektumot, ami megvalósítja a *java.lang.Iterable* interfészt. Az iterátor sorban halad, először a legnagyobb elemet adja vissza (ha szükséges, használja a *Collections* megfelelő metódusát). A megoldáshoz nem szükséges saját *Iterator* osztályt készíteni!

```
PQueue<Integer> s = new PQueue<Integer>();
s.push(1); s.push(2); s.push(3); s.push(4);
for (Integer i : s) {
    System.out.println(i);
}
```

Az előző feladat **Test** osztályának **main** metódusában próbálja ki a fenti kódrészletet!