Általános tudnivalók

Ebben az ismertetésben az osztályok, valamint a minimálisan szükséges metódusok leírásai fognak szerepelni. A feladatmegoldás során fontos betartani az elnevezésekre és típusokra vonatkozó megszorításokat, illetve a szövegek formázási szabályait.

Segédfüggvények létrehozhatóak, a feladatban nem megkötött adattagok és elnevezéseik is a feladat megoldójára vannak bízva. Törekedjünk arra, hogy az osztályok belső reprezentációját a *lehető legjobban védjük*, tehát csak akkor engedjünk, és csak olyan hozzáférést, amelyre a feladat felszólít, vagy amit azt osztályt használó kódrészlet megkíván!

A beadott megoldásodnak működnie kell a mellékelt tesztprogramokkal, de ez nem elégséges feltétele az elfogadásnak. A megírt forráskód legyen kellően általános és újrafelhasználható!

Használható segédanyagok: Java dokumentáció, legfeljebb egy üres lap és toll. Ha bármilyen kérdés, észrevétel felmerül, azt a felügyelőknek kell jelezni, *NEM* a diáktársaknak!

A feladat összefoglaló leírása

A feladatban egy raktárkészlet nyilvántartását készítjük el.

A programhoz tartozik egységtesztelő és két mintafájl (termék mintafájl és szett mintafájl), amely az egyes osztályok funkcionalitását teszteli, illetve a várható pontszámot mutatja.

A feladat részletes ismertetése

Termék (11 pont)

Valósítsuk meg a `raktar.Termek` osztályt, amelynek az a feladata, hogy ábrázolja a(z egyedi) termékek adatait.

- Három rejtett adattagja van: az `azonosito`, amely egy egész szám, a `nev`, amely egy szöveg, és az `ar`, amely egy egész szám. *(1 pont)*
- Az osztály egyetlen konstruktora legyen rejtett, és három paramétert várjon: a termék azonosítóját, a nevét és az árát, és egyszerűen tárolja el ezeket. *(1 pont)*
- Az osztálynak `createTermek` néven legyen egy nyilvános, osztályszintű metódusa, amely egy szöveget kap, és egy `Termek` objektumot ad vissza.

  Ha a paraméterben kapott szöveg helyes, akkor a következő formátumban tartalmazza a termék adatait: `"azonosító,név,ár"`. A metódusnak ellenőriznie kell, hogy az input megfelelő formátumú-e (pontosan 3 része van, amelyek vesszővel vannak elválasztva, az első és a harmadik rész ténylegesen számot tartalmaz, a második rész pedig ne legyen üres). Ha a formátum nem megfelelő, akkor a metódus adjon vissza `null`-t. Ha a formátum megfelelő, akkor a metódusnak még ellenőriznie kell az ár értékét is: ha az ár 0 vagy negatív szám, akkor szintén adjon vissza `null`-t. Helyes adatok esetén a metódus a hozza létre objektumot, majd adja azt vissza. *(4 pont)*

  Segítség:

    - Használjuk a `String` osztály `split` metódusát az adatok feldarabolásához.
    - Számmá konvertáláshoz használjuk az `Integer` osztály `parseInt` metódusát. Ez `NumberFormatException` kivételt vált ki, ha a paraméter nem alakítható számmá.
- Az adattagokhoz tartozzanak nyilvános ,,getter" műveletek (`getAzonosito`, `getNev` és `getAr` néven). *(1 pont)*
- Az osztálynak legyen egy nyilvános `kedvezmenyAd` nevű metódusa, amely egy egész számot vár. Ha a kapott szám legalább 0 és legfeljebb 100, akkor a metódus csökkentse a termék árát a megadott százalékkal (tehát 20%-os kedvezmény esetén legyen az új ár az eredeti 80%-a), majd kerekítse egészre (mindig lefelé kerekítsen). Ha a kapott szám rossz, akkor ne változtassa meg az árat. *(2 pont)*
- Tartalmazzon az osztály saját nyilvános `toString` megvalósítást. Az eredmény legyen `"Azonosító: Név (Ár Ft)"` formátumú. *(2 pont)*

Szett (8 pont)

Valósítsuk meg a `raktar.Szett` osztályt, amelynek az a feladata, hogy szetteket tartson nyilván. Egy szett több darabot tartalmaz egy egyedi termékből és kedvezmény tartozik hozzá.

- Hat rejtett adattagja van: *(1 pont)*
    - `azonosito`, amely egy egész szám, a *szett* azonosítója
    - `nev`, amely egy szöveg
    - `ar`, amely egy egész szám, és a *szett* árát tartalmazza
    - `alapTermekAzonosito`, amely egy egész szám és azt az (egyedi) terméket azonosítja, amelyet a szett tartalmaz
    - `alapTermekDarabszam`, amely egy egész szám, és azt tartalmazza, hogy hány darabos a szett (hány darabot tartalmaz az egyedi termékből)
    - `szettKedvezmeny`, amely egy egész szám, és azt tartalmazza, hogy a szett árába hány százalék kedvezmény van beépítve
- Az osztály tartalmazzon egy privát konstruktort: `Szett(int azonosito, String nev, int alapTermekAzonosito, int alapTermekDarabszam, int szettKedvezmeny)`: a konstruktor az tárolja el az adatokat, az `ar` adattagot pedig állítsa be `-1`-re. *(1 pont)*
- Legyen egy osztályszintű nyilvános `createSzett` metódus, amely egy paramétert vár, mely szövegesen tartalmazza a szett adatait vesszővel elválasztva:`"azonosito,nev,alapTermekAzonosito,alapTermekDarabszam,szettKedvezmeny"`. A metódusnak ellenőriznie kell az adatokat: a paraméterben legyen meg mind az 5 adat (se több, se kevesebb), a név ne legyen üres szöveg, a többi kötelezően szám legyen, az `alapTermekDarabszam` 0-nál nagyobb, a `szettKedvezmeny` pedig legalább 0 és legfeljebb 100 lehet. Ha az adatok nem helyesek, akkor adjon vissza `null`-t, ha pedig helyesek, akkor adja vissza a létrehozott objektumot. *(3 pont)*

  Segítség: itt is használjuk a `split` és `parseInt` metódusokat.
- Hozzuk létre a következő ,,getter" műveleteket: `getAzonosito`, `getNev`, `getAr` és `getAlapTermekAzonosito`. *(1 pont)*

- Legyen egy `arKiszamol` metódus, amely paraméterként megkapja a szettben szereplő alaptermék árát (egész szám), és eltárolja a szett árát.

  Ha a kapott paraméter 0, vagy annál kisebb, akkor metódus ne módosítsa az árat. Ha viszont a kapott szám 0-nál nagyobb, akkor a metódus ez alapján kiszámolja a szett árát: alaptermek ára szorozva a mennyiséggel, majd ebből még lejön a kedvezmény; az eredményt kerekítse egészre (mindig lefelé). *(1 pont)*

- Tartalmazzon az osztály saját `toString` megvalósítást. Az eredmény legyen `"Azonosító: Név (Ár Ft)"` formátumú. Az azonosító, név és ár a *szett* azonosítója, neve és ára.*(1 pont)*

Raktár (16 pont)

Valósítsuk meg a `raktar.Raktar` osztályt, amely egyedi termékeket és szetteket tárol.

- Az osztály tartalmazzon két rejtett, dinamikusan növekvő gyűjteményt az egyedi termékek és a szettek nyilvántartására. *(1 pont)*
- Az osztály nyilvános, paraméter nélküli konstruktora inicializálja a gyűjteményeket. Ezek kezdetben üresek.
- Készítsük el a nyilvános `termekekSzama` és `szettekSzama` nevű metódusokat, melyek visszaadják, hogy a raktár hány egyedi terméket, illetve hány szettet tárol. *(1 pont)*
- Készítsünk egy nyilvános `getTermek` és egy `getSzett` nevű metódust, melyek egy egész számot várnak és visszaadják az adott indexű termék, illetve szett referenciáját, vagy `null`-t, ha nem volt olyan sorszámú elem. A sorszámozást 0-tól kezdjük. *(1 pont)*
- Készítsünk egy nyilvános `termekKeres` nevű metódust, amely egy azonosítót (egész szám) kap, és visszatér azzal az egyedi termékkel, amelynek ez az azonosítója. Ha nem talál ilyen terméket, akkor `null`-lal tér vissza. *(2 pont)*
- Legyen egy nyilvános `termekBeolvas` nevű metódus, mely annak a fájlnak a nevét kapja, amely tartalmazza az egyedi termékek adatait. Ha a beolvasás során kivétel keletkezik (nem létezik a fájl vagy nem olvasható), akkor a metódus engedje ki a keletkező kivételt.

  A metódus olvassa végig a fájlt, és tárolja el az adatokat az egyedi termékek számára létrehozott gyűjteményben.

  A fájl minden egyes sorában tartalmaz egy terméket. Helyes sor esetén a formátum a `raktar.Termek` osztály `createTermek` metódusa által elvárt. Ha a sor hibás (a`createTermek` metódus nem tudott belőle objektumot létrehozni), akkor a sort egyszerűen hagyjuk figyelmen kívül. Ha a formátum helyes, akkor az eltárolás előtt még azt is ellenőrizzük le, hogy az osztály tartalmaz-e már terméket ugyanezzel az azonosítóval - amennyiben igen, úgy szintén hagyjuk figyelmen kívül a sort (elég az egyedi termékek között végignézni). Ha a termék ebből a szempontból is helyes (tehát még nem tartunk nyilván terméket ezzel az azonosítóval), akkor a terméket tároljuk ez az egyedi termékek számára létrehozott gyűjteményben. *(5 pont)*

- Legyen egy nyilvános `szettBeolvas` nevű metódus, amely annak a fájlnak a nevét kapja, amely tartalmazza a szettek adatait. Ha a beolvasás során kivétel keletkezik (nem létezik a fájl vagy nem olvasható), akkor a metódus engedje ki a keletkező kivételt.

  A metódus olvassa végig a fájlt, és tárolja el az adatokat a szettek számára létrehozott gyűjteményben.

A fájl szerkezete a következő: a fájl minden sora egy szettet tartalmaz, melynek formája a `raktar.Szett` osztály `createSzett` metódusa által elvárt (ezt tehát nem kell ellenőrizni). A szetteknél feltételezhetjük, hogy a sor minden szempontból helyes: az azonosítója egyedi és az `alapTermekAzonosito` azonosítóval már szerepel termék az egyedi termékek között, és a szett még nem szerepel az eddigi szettek között.

A sor által megadott adatokkal hozzuk létre a szettet. A korábban eltárolt egyedi termékek közül keressük ki a szett alaptermékének árát (`termekKeres` metódus). Ennek segítségével számoltassuk ki, és állítsuk be a szett árát (`arKiszamol` metódus). Végül az elkészült objektumot tároljuk el az osztály szettek számára létrehozott gyűjteményében. *(4 pont)*

- Tartalmazzon az osztály saját `toString` megvalósítást: sortörésekkel elválasztva sorolja fel az egyedi termékek, utánuk a szettek szöveges ábrázolását (a felsorolás végén is legyen sortörés). Használjuk az termékek és szettek `toString` metódusát. *(2 pont)*

Pontozás

A tesztelő által adott pontszám csak becslésnek tekinthető, a gyakorlatvezető levonhat pontokat, vagy adhat részpontokat.

A végső pont az elméleti rész (összesen 7 pont szerezhető) és a gyakorlati rész (összesen 35 pont szerezhető) pontjainak összege.

Ponthatárok:

- 0 - 13: elégtelen (1)
- 14 - 20: elégséges (2)
- 21 - 27: közepes (3)
- 28 - 34: jó (4)
- 35 - 42: jeles (5)

Jó munkát!