Programozás 2

– 1. zárthelyi dolgozat, pót-ZH –

2021. október 28., 18.00

1. feladat – metál az ész (1 pont)

Készítse el a `Song` osztályt, amely egy valós világbeli zeneszámot modellez. Egy zeneszámról a következő adatokat akarjuk letárolni: `title` (cím), `artist` (előadó), ill. `length` (hossz, másodpercben megadva). Miután létrehoztunk egy zeneszám objektumot, az legyen *immutable*, vagyis utólag ne lehessen módosítani.

Írja meg továbbá a `Playlist` osztályt, amely egy egyszerűsített hétköznapi lejátszási listát valósít meg. Az osztályban a zeneszámok tárolását dinamikus tömbbel oldjuk meg.

A `Playlist` osztály támogassa a következő műveleteket:

- az `add()` metódussal hozzá lehessen adni egy zeneszámot a lejátszási listához
- a `shufflePlaylist()` metódussal össze lehessen keverni a zeneszámok sorrendjét
- a `calculateTotalLength()` metódus pedig adja vissza a lejátszási lista teljes hosszát másodpercben

Használat (`Main.java`):

```
Playlist queen = new Playlist();
Song song1 = new Song("Bohemian Rhapsody", "Queen", 360);
Song song2 = new Song("The Show Must Go On", "Queen", 262);
queen.add(song1);
queen.add(song2);

Playlist metal = new Playlist();
Song song3 = new Song("Empire Of The Clouds", "Iron Maiden", 1080);
Song song4 = new Song("Fear Of The Dark", "Iron Maiden", 439);
Song song5 = new Song("Smoke on the Water", "Black Sabbath", 330);
metal.add(song3);
metal.add(song4);
metal.add(song5);

System.out.println(queen.calculateTotalLength());    // 622

metal.printPlaylist();    // a kimenet így nézzen majd ki:
/*
Lejátszási lista:
- Empire Of The Clouds (Iron Maiden)
- Fear Of The Dark (Iron Maiden)
- Smoke on the Water (Black Sabbath)
*/
```

Folyt. a köv. oldalon!

```
metal.shufflePlaylist();    // véletlenszerűen összekeveri a zeneszámok sorrendjét
metal.printPlaylist();    // egy lehetséges kimenet:
/*
Lejátszási lista:
- Empire Of The Clouds (Iron Maiden)
- Smoke on the Water (Black Sabbath)
- Fear Of The Dark (Iron Maiden)
*/
```

2. feladat – Armstrong-számok (1 pont)

Egy háromjegyű szám akkor Armstrong-szám, ha a számjegyek köbeinek összege az eredeti számot adja vissza. Például a 371 Armstrong-szám, mivel

$$3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$$

Írjon egy programot, ami parancssori argumentumként kap egy háromjegyű számot, s eldönti, hogy a szám Armstrong-szám-e vagy sem.

Ha nem pontosan egy parancssori argumentumot adtunk meg, akkor hibaüzenet.

Ha egyetlen parancssori argumentumot adtunk meg, de az nem háromjegyű, akkor szintén hibaüzenet.

Feltételezhetjük, hogy az argumentumok mindegyike pozitív szám (ezt nem kell külön levizsgálni).

Futási példák:

```
$ java Main
Hiba! Pontosan egy paraméter kell!

$ java Main 2 3 5
Hiba! Pontosan egy paraméter kell!

$ java Main 2
Hiba! Egy háromjegyű számot kell megadni!

$ java Main 278 9
Hiba! Pontosan egy paraméter kell!

$ java Main 2468
Hiba! Egy háromjegyű számot kell megadni!

$ java Main 111
Ez NEM Armstrong-szám.

$ java Main 371
Ez Armstrong-szám.
```

Egy szám tesztelését (miszerint Armstrong-szám-e vagy sem) szervezze ki egy statikus metódusba, amit a következőképpen akarunk meghívni (példa):

```
boolean result1 = MyMath.isArmstrong(371);    // true
boolean result2 = MyMath.isArmstrong(111);    // false
```

Vegye észre, hogy a metódus a számot egész típusúként (`int`) kapja meg!

Tegyen róla, hogy a `MyMath` osztályt ne lehessen példányosítani!

3. feladat – stat (1 pont)

Interaktív módon kérjen be a felhasználótól egy sztringet.

Feltételezhetjük, hogy a sztringben csakis az angol ábécé kis- és nagybetűi, számjegyek, ill. a szóköz szerepel.

Írjunk ki a képernyőre egy kis statisztikát.

Példák:

```
// az <E> jelentése: itt ütöttünk Enter-t

$ java Main
Szöveg: Hello World 2021<E>
Magánhangzók száma:  3
Mássalhangzók száma: 7
Számjegyek száma:    4

$ java Main
Szöveg: abc def 1 5 7<E>
Magánhangzók száma:  2
Mássalhangzók száma: 4
Számjegyek száma:    3
```

A statisztikát a `StringUtils.stat()` nevű metódussal állapítsuk meg.

A metódus megkapja a felhasználó által megadott sztringet, s egy háromelemű tömbben visszaadja a három egész értéket.

A tömb első eleme tartalmazza a magánhangzók számát. A tömb második eleme tartalmazza a mássalhangzók számát. A tömb harmadik eleme tartalmazza a számjegyek számát.

A kiíratás során az értékek (számok) egymás alatt jelenjenek meg, balra igazítva.

4. feladat – IP-címek (1 pont)

Tekintsük a mellékelt `ips.txt` állományt. Az állomány minden sora egy-egy IP-címet tartalmaz.

Egy IPv4 szerinti IP-cím egy 32 bites egész szám, amelyet hagyományosan négy darab egy bájtos, azaz 0 és 255 közé eső, ponttal elválasztott decimális számmal írunk le a könnyebb olvashatóság kedvéért (pl. 172.16.254.1).

Kérdés: hány darab szabályos IP-címet tartalmaz az input állomány?

Egy IP-cím akkor szabályos, ha valamennyi komponensének az értéke 0 és 255 közé esik (zárt intervallum). Például a `172.16.254.1` szabályos, viszont a `221.189.684.756` nem szabályos, mivel a harmadik komponensének az értéke (684) hibás.

Egy IP-cím tesztelését (miszerint szabályos-e vagy sem) szervezze ki egy statikus metódusba, amit a következőképpen akarunk meghívni (példa):

```
boolean ok = MyUtils.isValidIP("221.189.684.756");    // false
```

Futási példa:

```
$ java Main
12
```

A 12 helyett természetesen a helyes eredményt kell kiírni (vagyis a szabályos IP-címek számát).