Zenész találkozó

A Művészetek Házában **zenész**-találkozót szerveznek. A résztvevők kétféle kategóriában regisztrálhatnak: vagy **klasszikus zenész**ként vagy **utcazenész**ként. Minden zenészt jellemez a találkozóig változtathatatlan *neve*, egy egyedi azonosító (klasszikus zenészeknél "K"-val, utcazenészeknél "U"-val kezdődjön), illetve az eddigi nyilvános *szerepléseinek száma*. A klasszikus zenésznek a regisztráció során azt is meg kell adnia, hogy *melyik filharmonikus zenekar* tagja (egyetlen String). Mivel a rendezvényig ki kell nyomtatni a résztvevőket azonosító kártyákat, ezért az utóbbi sem változhat. Ezeken a kártyákon (toString()) a zenész neve, azonosítója olvasható, klasszikus zenész esetén zenekarának neve is.

Mindegyikőjük képes *szerepel()*ni. A metódus során a nyilvános szereplések száma eggyel nő.

És bár elsősorban a zene öröme élteti őket, de azért nekik is fizetniük kell a megélhetésért, így ők is kapnak *fizetés()*t, de nem egyforma módon. A klasszikus zenészek fizetése egy országosan egységes *alapbér*ből, plusz a fellépések számának *X*-szereséből tevődik össze, ahol X szintén minden egyes klasszikus zenészre ugyanúgy érvényes pénzösszeg. Ebből a bérből lejön a szintén egyforma *K*% adókulcsú adó.

Az utcazenészek fizetése bizonytalanabb, ők nem kapnak alapbért, fizetésük kizárólag a *kalapozás()*ból jön össze, viszont nem fizetnek adót. Ugyanakkor a diákok kivételével mindenkinek egy országosan egyforma mértékű *helypénz*t is kell fizetnie. Kalapozás során a kalap addigi tartalma növekszik a metódus paraméterében lévő értékkel. Fizetésük a kalap tartalma, amiből persze lejön az esetleges helypénz. (Így akár veszteséges is lehet.)

A **vezérlő** osztályban regisztrálhatnak a zenészek (*zeneszek.txt* fájl). A diákoknak igazolást kell kapniuk, ezért a beolvasáskor minden utcazenésztől "megkérdezik", hogy diák-e (a választ egy véletlen szám generátor segítségével adja meg).

Majd elkezdődik a zenésztalálkozó. A találkozó során minden zenész szerepel, sorban egyik a másik után. Ha utcazenész következik, akkor ő kalapozik is, mégpedig úgy, hogy minden egyes társa orra elé „odarakja” a kalapot. (Annyiszor hívjuk meg, ahány társ van.) A társak véletlenszerűen dobnak bele 0 és Z közötti összeget.

Az örömzene után megvitatják az anyagiakat is. Állapítsa meg, hogy kik járnak jobban, vagyis melyik a nagyobb, a klasszikus zenészek átlagfizetése vagy az utcazenészek aznapi átlagfizetése.

Ki(k) kalapozta(k) össze a legtöbb pénzt?

Néhány megoldásrészlet:

Az ősosztály most is lehet absztrakt, és mivel a fizetés kiszámítása teljesen eltérő a két utódosztályban, ezért ezt az értékeket visszaadó metódus is lehet absztrakt. Most arra is mutatunk példát, hogy szituációtól függően eltérő lehet a `toString()` értéke. Más lesz ez az érték, ha a fizetéssel együtt vagy a nélkül akarjuk kiíratni a zenészek adatait. És mivel ezek a kiíratások az összes zenészre vonatkoznak, ezért a `toString()` értékét meghatározó logikai változó statikus lesz.

Mielőtt rátérnénk a kódokra, beszéljük meg az egyedi azonosítók kérdését. Arról már volt szó, hogy hogyan lehet egyedi azonosítót rendelni egy osztály példányaihoz (Állatverseny 2 feladat), de most külön szeretnénk sorszámozni a klasszikus- és az utcazenész példányokat. Ezt természetesen meg lehetne úgy oldani, hogy mindkét utódosztályban bevezetünk egy-egy azonosítót, és a már tanult módon generáljuk ezeket az értékeket. Az eddig elmondottaknak van azonban egy szépséghibája: ha az azonosítóhoz tartozó gettereket is csak az utód osztályokban írjuk meg, akkor kényelmetlenül tudnánk csak hivatkozni rájuk. Ez kikerülhető úgy, hogy az ősosztályban bevetetjük az absztrakt `getAzonosito()` metódust. A kérdést tehát meg lehetne oldani a már tanult módon, de most egy másik megoldási utat szeretnénk választani, amelyben két tervezési mintára is mutatunk példát.

Létrehozunk egy azonosító-generáló osztályt. Ezt úgy valósítjuk meg, hogy a *singleton* tervezési minta alapján egyetlen példányt lehessen létrehozni belőle. Maga az azonosító-gyártás pedig a *factory* mintát követi.

Azt, hogy csak egyetlen példányt lehessen létrehozni, úgy tudjuk elérni, hogy egyrészt privát láthatóságúra deklaráljuk a konstruktort, így azt kívülről nem lehet elérni, csak az osztályon belülről. A statikus `getInstance()` metódus pedig azt garantálja, hogy biztosan csak egyetlen példány jöjjön létre.

A `getEgyediId()` metódus a factory tevezési mintát veszi alapul, és a feltételtől függően vagy ilyen, vagy olyan azonosítót gyárt.

```
private static IdGenerator peldany = null;

    private IdGenerator() {
    }

    public static IdGenerator getInstance() {
        if (peldany == null) {
            peldany = new IdGenerator();
        }

        return peldany;
    }
```

```
    private int kovetkezoKlasszikusZeneszID = 0;
    private int kovetkezoUtcaZeneszID = 0;

    public String getEgyediId(Zenesz zenesz) {
        String egyediId = null;
            if(zenesz instanceof KlasszikusZenesz){
                kovetkezoKlasszikusZeneszID++;
                egyediId = "K" + kovetkezoKlasszikusZeneszID;
            }
            if(zenesz instanceof UtcaZenesz){
                kovetkezoUtcaZeneszID++;
                egyediId = "U" + kovetkezoUtcaZeneszID;
            }
        return egyediId;
    }
}
```

Ezek után az osztályok getterek és setterek nélküli része (a statikus változóhoz kell getter/setter, a `nev`, `szereplesSzam`, `azonosito` változókhoz csak getter.):

```
public abstract class Zenesz {

    private String nev;
    private int szereplesSzam;
    private static boolean fizeteshez = false;

    private String azonosito;

    public Zenesz(String nev, int szereplesSzam) {
        this.nev = nev;
        this.szereplesSzam = szereplesSzam;
        azonosito = IdGenerator.getInstance().getEgyediId(this);
    }

    /** szerepléskor a szereplések száma eggyel növekszik ...3 lines */
    public void szerepel(){
        szereplesSzam ++;
    }

    public abstract int fizetes();

    public String getAzonosito() {
        return azonosito;
    }

    /** Attól függ, hogy milyen a kiíratás, hogy mikor hívjuk meg ...4 lines */
    @Override
    public String toString() {
        if(!fizeteshez)return nev + " (" + this.getAzonosito() + ")";
        return nev + ", " + fizetes() + " Ft";
    }
```

```java
public class KlasszikusZenesz extends Zenesz{

    private String zenekarNev;

    private static int alapBer;
    private static int szereplesBerSzorzo;
    private static double adoSzazalekLab;

    public KlasszikusZenesz( String nev, int szereplesSzamm, String zenekar) {
        super(nev, szereplesSzamm);
        this.zenekarNev = zenekar;
    }

    @Override
    public int fizetes() {
        return (int) ((alapBer +
                 szereplesBerSzorzo*this.getSzereplesSzam())*
                                                 (100-adoSzazalekLab)/100);
    }

    @Override
    public String toString() {
        if(!Zenesz.isFizeteshez())return super.toString() + ", " + zenekarNev;
        return super.getNev() + ", " + zenekarNev + ", " + fizetes() + " Ft";
    }
```

```java
public class UtcaZenesz extends Zenesz{

    private boolean diak = false;
    private int fizetes = 0;

    private static int helyPenz;

    public UtcaZenesz(String nev, int szereplesSzam) {
        super(nev, szereplesSzam);
    }

    public void kalapozik(int penz){
        fizetes += penz;
    }

    @Override
    public int fizetes() {
        if(diak) return fizetes;
        return fizetes - helyPenz;
    }
```

A vezérlés néhány részlete:

```
public class Main {

   private List<Zenesz> zeneszek = new ArrayList<>();

    private final String ADAT_ELERES = "/adatok/zeneszek.txt";
    private final String CHAR_SET = "UTF-8";
    private final int ALAP_FIZETES = 100000;
    private final int SZEREPLES_BER_SZORZO = 2;
    private final int SZJA_SZAZALEK = 15;
    private final int HELYPENZ = 1000;
    private final int KALAP_DIJ_HATAR = 4000;
    private final double DIAK_ESELY = 0.5;

    public static void main(String[] args) {
          new Main().start();
    }

    private void start() {
        statikusAdatok();
        adatBevitel();
        kiiratas("A zenész találkozó résztvevői:\n");
        zeneles();
        fizetesek();
        leggazdagabbUtcazenesz();
    }
```

```
private void adatBevitel() {
    try(InputStream ins = this.getClass().getResourceAsStream(ADAT_ELERES);
        Scanner sc = new Scanner(ins, CHAR_SET)) {

        String[] adatok;
        String sor;

        while (sc.hasNextLine()) {
            sor = sc.nextLine();
            try {
                if (!sor.isEmpty()) {
                    adatok = sor.split(";");
                    switch (adatok.length) {
                        case 3:
                            zeneszek.add(new KlasszikusZenesz(adatok[0],
                                    Integer.parseInt(adatok[1]), adatok[2]));
                            break;
                        case 2:
                            UtcaZenesz zenesz = new UtcaZenesz(adatok[0],
                                    Integer.parseInt(adatok[1]));
                            if (Math.random() < DIAK_ESELY) {
                                zenesz.setDiak(true);
                            }
                            zeneszek.add(zenesz);
                            break;
                        default:
                            throw new Exception();
                    }
                }
            } catch (Exception e) {
                System.out.println("Hibás a " + sor + " adatsor!");
            }
        }
    } catch (Exception ex) {
        System.out.println("Hibás fájlmegadás");
        System.exit(-1);
    }
}

private void zeneles() {
    for (Zenesz zenesz : zeneszek) {
        zenesz.szerepel();
        if(zenesz instanceof UtcaZenesz){
            for (Zenesz tars : zeneszek) {
                if(!zenesz.equals(tars)){
                    ((UtcaZenesz) zenesz).kalapozik((int) (Math.random()*
                                                    KALAP_DIJ_HATAR));
                }
            }
        }
    }
}
```