Levelezős Programozás beugró (Java) **2007. december 1.**

Adott az alábbi osztály:

```
public class Autó
{
  private String rendszám;
  private int teljesítmény;
  private boolean automata;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható!
2. Írja meg a három adattaghoz tartozó lekérdező metódusokat!
3. Definiálja felül az **`Object`** osztályból örökölt **`toString()`** metódust úgy, hogy az autó mindhárom adattagját egy sorban adja vissza!
4. Definiálja felül az **`Object`** osztályból örökölt **`equals()`** metódust úgy, hogy két autó csak akkor legyen egyenlő, ha a rendszámuk megegyezik!
5. Egészítse ki az **`Autó`** osztály definícióját úgy, hogy a példányai teljesítményük alapján összehasonlíthatók legyenek!
6. Származtasson az **`Autó`** osztályból egy **`Teherautó`** osztályt! A teherautók az autó tulajdonságain kívül még egy egész értékű **`teherbírás`** tulajdonsággal rendelkeznek.
7. Egészítse ki a **`Teherautó`** osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
8. Definiálja felül az **`Autó`** osztályból örökölt **`toString()`** metódust úgy, hogy a teherautó mind a négy adattagját egy sorban adja vissza!
9. Készítsen egy **`Main`** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **`Autó`** típusú objektumot, amelynek a rendszámát parancssori argumentumként kapjuk meg, a teljesítménye 100 lóerő és automata váltós! Írja ki a képernyőre a létrehozott objektumot!
10. Egészítse ki a főprogramot egy olyan kódrészlettel, amely beolvassa a billentyűzetről 2 autó és 2 teherautó adatait (ilyen sorrendben soronként egyet-egyet), és elhelyezi őket egy négyelemű tömbben! Az autók és a teherautók egyes adatait szóköz választja el egymástól.
11. Egészítse ki az **`Autó`** osztályt egy olyan metódussal, amely igazzal tér vissza, ha az autó rendszáma szabályos, azaz pontosan 6 karakter hosszúságú, amelyből az első három (angol) betű, a második három pedig (decimális) számjegy!
12. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy autókat tartalmazó tömböt, és visszaad egy teherautókat tartalmazó kollekciót, amely azokat a tömbbeli **`Teherautó`** típusú objektumokat tartalmazza, amelyeknek a teherbírása meghaladja a 20 tonnát!
13. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy autókat tartalmazó kollekciót, és képernyőre írja a három legnagyobb teljesítményű autót! Ha a kollekció háromnál kevesebb elemet tartalmaz, akkor írja ki mindet!
14. Egészítse ki az **`Autó`** osztály definícióját olyan lehetőséggel, amelynek segítségével bármelyik osztályból lekérdezhető, hogy hány autót hoztak eddig forgalomba (azaz hány példánya létezik az **`Autó`** osztálynak)!
15. Egészítse ki az **`Autó`** osztály definícióját olyan lehetőséggel, amelynek segítségével „extrákat” (nem szériafelszereléseket) tárolhatunk minden autónál! Tegye lehetővé, hogy egy autó új extrát kapjon, illetve hogy egy létező extra törölhető legyen! Egy új autó semmilyen extrával nem rendelkezik.

Levelezős Programozás pótbeugró (Java) 2008. január 5.

Adott az alábbi osztály:

```
public class Személy
{
  protected String név;
  protected int életkor;
  private boolean férfi;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható!
2. Írja meg a három adattaghoz tartozó lekérdező és beállító metódusokat!
3. Definiálja felül az **`Object`** osztályból örökölt **`toString()`** metódust úgy, hogy a személy mindhárom adattagját egy sorban adja vissza!
4. Egészítse ki az **`Személy`** osztály definícióját úgy, hogy a példányai életkoruk alapján összehasonlíthatók legyenek!
5. Származtasson a **`Személy`** osztályból egy **`Hallgató`** osztályt! A hallgatók a személy tulajdonságain kívül még egy valós értékű **`átlag`** tulajdonsággal rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
6. Definiálja felül a **`Személy`** osztályból örökölt **`toString()`** metódust úgy, hogy a hallgató mind a négy adattagját külön sorban adja vissza!
7. Származtasson a **`Személy`** osztályból egy **`Oktató`** osztályt! Az oktatók a személy tulajdonságain kívül még egy String típusú **`tanszék`** tulajdonsággal rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
8. Készítsen egy **`Main`** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **`Személy`** típusú objektumot, amely egy Mona Liza nevű 20 éves nőt reprezentál! Írja ki a képernyőre a létrehozott objektumot!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely beolvassa a billentyűzetről 2 hallgató és 2 oktató adatait (ilyen sorrendben, soronként egyet-egyet), és elhelyezi őket egy négyelemű tömbben! A hallgatók és az oktatók egyes adatait szóköz választja el egymástól.
10. Egészítse ki a **`Hallgató`** osztályt egy olyan metódussal, amely igazzal tér vissza, ha a hallgató jó képességű, azaz az átlaga legalább 4!
11. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy személyeket tartalmazó kollekciót, és visszatér a jó képességű hallgatók életkorainak az átlagával!
12. Egészítse ki a **`Hallgató`** osztályt olyan lehetőséggel, amelynek segítségével bármely más osztályból beállítható, de nem lekérdezhető, hogy milyen átlag felett tekintünk egy hallgatót jó képességűnek! Ha szükséges, írja újra a 10. feladatban megírt metódust!
13. Készítsen egy **`Egyetem`** osztályt, amely tetszőleges számú hallgató és oktató tárolására alkalmas! Legyen lehetőség új hallgató és új oktató felvételére, valamint létező hallgató, illetve létező oktató törlésére!
14. Egészítse ki az **`Egyetem`** osztályt egy olyan metódussal, amely képernyőre írja a három legfiatalabb hallgatóját! Ha háromnál kevesebb hallgatója van az egyetemnek, akkor a metódus írja ki az összes hallgatót!
15. Egészítse ki az **`Egyetem`** osztályt egy olyan metódussal, amely az IT tanszék női oktatóinak neve végéhez hozzáfűz egy felkiáltójelet!

Levelezős Programozás beugró (Java) 2008. május 17.

Adott az alábbi osztály:

```
public class Állat
{
  protected String faj;
  protected double kor;
  protected double súly;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható!
2. Írja meg a három adattaghoz tartozó lekérdező és beállító metódusokat!
3. Definiálja felül az **`Object`** osztályból örökölt **`toString()`** metódust úgy, hogy az állat mindhárom adattagját egy sorban adja vissza „`<faj>: <kor> év, <súly> kg`” formátumban (pl. „elefánt: 3.5 év, 870.0 kg”)!
4. Egészítse ki az **`Állat`** osztály definícióját úgy, hogy a példányai súlyuk alapján összehasonlíthatók legyenek!
5. Származtasson az **`Állat`** osztályból egy **`Madár`** osztályt! A madarak az állat tulajdonságain kívül még egy logikai értékű **`röpképes`** tulajdonsággal rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
6. Definiálja felül az **`Állat`** osztályból örökölt **`toString()`** metódust úgy, hogy a madár mind a négy adattagját egy sorban adja vissza az alábbi formátumban: „`<kor> éves, <súly> kg-os {röpképes|röpképtelen} <faj>`” (pl. „2.0 éves, 10.32 kg-os röpképtelen pingvin”)!
7. Származtasson az **`Állat`** osztályból egy **`Emlős`** osztályt! Az emlősök az állat tulajdonságain kívül még egy egész értékű **`lábakSzáma`** tulajdonsággal rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
8. Készítsen egy **`Main`** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **`Állat`** típusú objektumot, amely egy 3,5 éves, 870 kg-os elefántot reprezentál! Írja ki a képernyőre a létrehozott objektumot!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely beolvassa a billentyűzetről 2 madár és 2 emlős adatait (ilyen sorrendben, soronként egyet-egyet), és elhelyezi őket egy négyelemű tömbben! A madarak és az emlősök egyes adatait szóköz választja el egymástól.
10. Egészítse ki az **`Emlős`** osztályt egy olyan metódussal, amely igazzal tér vissza, ha az állat látható nyomot hagy a homokban, azaz az egy lábra eső súlya meghaladja a 10 kg-ot!
11. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy állatokat tartalmazó kollekciót, és visszatér a legfiatalabb látható nyomot hagyó emlős fajával, vagy ha ilyen nincs, akkor egy üres sztringgel!
12. Egészítse ki az **`Emlős`** osztályt olyan lehetőséggel, amelynek segítségével bármely osztályból beállítható, de nem lekérdezhető, hogy mekkora az az egy lábra eső súly, amely felett az állat látható nyomot hagy a homokban! Ha szükséges, írja újra a 10. feladatban megírt metódust!
13. Készítsen egy **`Állatkert`** osztályt, amely tetszőleges számú állat tárolására alkalmas! Legyen lehetőség új állat felvételére, valamint létező állat törlésére!
14. Egészítse ki az **`Állatkert`** osztályt egy olyan metódussal, amely képernyőre írja az állatkert három legnehezebb egyedét! Ha háromnál kevesebb állat van az állatkertben, akkor a metódus írja ki az összes egyedet!
15. Egészítse ki az **`Állatkert`** osztályt egy olyan metódussal, amely az állatkertben található madarak fajának végéhez hozzáfűzi a „madár” szót (pl. „pingvinmadár”)!

Levelezős Programozás pótbeugró (Java) **2008. május 31.**

Adott az alábbi osztály:

```
public class Film
{
  protected String cím;
  protected int hossz;
  protected boolean korhatáros;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható!
2. Írja meg a három adattaghoz tartozó lekérdező és beállító metódusokat!
3. Definiálja felül az **`Object`** osztályból örökölt **`toString()`** metódust úgy, hogy a film mindhárom adattagját egy sorban adja vissza „`<cím>: <hossz> perces, {korhatáros|nem korhatáros}`” formátumban (pl. „Hegylakó: 112 perces, korhatáros”)!
4. Definiálja felül az **`Object`** osztályból örökölt **`equals()`** metódust úgy, hogy két film csak akkor legyen egyenlő, ha a címük megegyezik!
5. Egészítse ki a **`Film`** osztály definícióját úgy, hogy a példányai hosszuk alapján összehasonlíthatók legyenek!
6. Származtasson a **`Film`** osztályból egy **`Sorozat`** osztályt! A sorozatok a film tulajdonságain kívül még egy egész értékű **`epizódokSzáma`** tulajdonsággal rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
7. Definiálja felül a **`Film`** osztályból örökölt **`toString()`** metódust úgy, hogy a sorozat mind a négy adattagját egy sorban adja vissza az alábbi formátumban: „`<cím>: <hossz> perces, {korhatáros|nem korhatáros}, <epizódokSzáma> részes`” (pl. „Barátok közt: 13 perces, nem korhatáros, 4328 részes”)!
8. Készítsen egy **`Main`** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **`Film`** típusú objektumot, amely a 112 perc hosszú, korhatáros Hegylakó című filmet reprezentálja! Írja ki a képernyőre a létrehozott objektumot!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely beolvassa a billentyűzetről 2 film és 2 sorozat adatait (ilyen sorrendben, soronként egyet-egyet), és elhelyezi őket egy négyelemű tömbben! A filmek és a sorozatok egyes adatait szóköz választja el egymástól.
10. Egészítse ki a **`Sorozat`** osztályt egy olyan metódussal, amely igazzal tér vissza, ha a sorozat sokrészes, azaz epizódjainak száma meghaladja az 1000-et!
11. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy filmeket tartalmazó kollekciót, és visszatér a legrövidebb sokrészes sorozat címével, vagy ha ilyen nincs, akkor egy üres sztringgel!
12. Egészítse ki a **`Sorozat`** osztályt olyan lehetőséggel, amelynek segítségével bármely osztályból beállítható, de nem lekérdezhető, hogy hány rész felett tekintünk egy sorozatot sokrészesnek! Ha szükséges, írja újra a 10. feladatban megírt metódust!
13. Készítsen egy **`Téka`** osztályt, amely tetszőleges számú film tárolására alkalmas! Legyen lehetőség új film felvételére, valamint létező film törlésére!
14. Egészítse ki a **`Téka`** osztályt egy olyan metódussal, amely képernyőre írja a tékában található három leghosszabb filmet! Ha háromnál kevesebb film van a tékában, akkor a metódus írja ki az összes filmet!
15. Egészítse ki a **`Téka`** osztályt egy olyan metódussal, amely a tékában található sorozatok címének végéhez hozzáfűz egy szóközt és a „sorozat” szót (pl. „Barátok közt sorozat”)!

Levelezős Programozás beugró (Java) 2008. december 14.

Adott az alábbi osztály:

```
public class Állat
{
  protected String név;
  protected double súly;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindkét adattagjának kezdőérték adható, és írja meg az adattagokhoz tartozó lekérdező metódusokat!
2. Definiálja felül az **`Object`** osztályból örökölt **`toString()`** metódust úgy, hogy az állat adatait az alábbi formában adja vissza: „`<név> (<súly> kg)`” (pl. „Maci Laci (354.3 kg)”)!
3. Definiálja felül az **`Object`** osztályból örökölt **`equals()`** metódust úgy, hogy két állat akkor legyen egyenlő, ha a nevük megegyezik!
4. Egészítse ki az **`Állat`** osztály definícióját úgy, hogy a példányai a súlyuk alapján összehasonlíthatók legyenek! A feladat megoldásához implementálja a **`Comparable`** interfészt!
5. Származtasson az **`Állat`** osztályból egy **`Ragadozó`** osztályt! A ragadozók az állat tulajdonságain kívül még egy egész értékű **`fogakSzáma`** tulajdonsággal is rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a három adattagjának kezdőérték adható!
6. Definiálja felül az **`Állat`** osztályból örökölt **`toString()`** metódust úgy, hogy a ragadozó adatait az alábbi formátumban adja vissza: „`<név> ragadozó: <súly> kg, <fogakSzáma> foga van`” (pl. „Bagira ragadozó: 80.0 kg, 28 foga van”)!
7. Készítsen egy **`Main`** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **`Ragadozó`** típusú objektumot, amely az előző feladat példájában megadott állatot reprezentálja! Írja ki a képernyőre a létrehozott objektumot!
8. Hozzon létre a főprogramban egy állatokat tartalmazó (tetszőleges) kollekciót, amelybe vegye fel az előző feladatban létrehozott objektumot, valamint azon kívül még két újonnan létrehozott állatot!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely a billentyűzetről soronként egy-egy állatnevet olvas be mindaddig, amíg olyan nevet nem kap, amilyen nevű állat szerepel az előző feladatban feltöltött kollekcióban (feltesszük, hogy azóta a kollekció megváltozott), majd írja ki az állat súlyát!
10. Egészítse ki a **`Ragadozó`** osztályt egy olyan metódussal, amely igazzal tér vissza, ha az állat „veszélyes”, azaz ha a súlya legalább 50 kg, a fogainak száma pedig meghaladja a 40-et!
11. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy állatokat tartalmazó kollekciót, és visszatér a kollekcióban található veszélyes ragadozók számával!
12. Készítsen egy **`Állatkert`** nevű osztályt, amely értelemszerűen implementálja az alábbi interfészt:

    ```
    interface ÁllatSereglet
    {
      public void felvesz( Állat állat );
      public void eltávolít( String állatnév );
    }
    ```

    Készítsen az osztályhoz egy olyan konstruktort, amely egy paraméterként kapott, állatokat tartalmazó tömb elemeit felveszi az állatkertbe!
13. Egészítse ki az **`Állatkert`** osztályt egy olyan metódussal, amely képernyőre írja az állatkertben található állatok közül a legkisebb súlyúnak a nevét! Ha az állatkertben egyetlen állat sincs, akkor a metódus dobjon **`ÜresÁllatkertException`** kivételt! (A feladathoz a kivétel osztályát is meg kell írni.)
14. Egészítse ki az **`Állatkert`** osztályt egy olyan metódussal, amely az állatkertben található ragadozók nevét csupa nagybetűssé alakítja! Ha szükséges, egészítse ki a **`Ragadozó`** osztályt! (Segítség: a **`String`** osztály **`toUpperCase()`** metódusa visszaadja egy **`String`** objektum nagybetűs megfelelőjét.)
15. Írjon a **`Main`** osztályban egy statikus metódust, amely paraméterként megkap egy állatkerteket tartalmazó tömböt, és visszatér a tömbbeli állatkertekben található állatok átlagos súlyával!

Levelezős Programozás beugró (Java) 2009. január 4.

Adott az alábbi osztály:

```
public class Dísz
{
  protected String név;
  protected int ár;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindkét adattagjának kezdőérték adható, és írja meg az adattagokhoz tartozó lekérdező metódusokat!
2. Definiálja felül az **Object** osztályból örökölt **toString()** metódust úgy, hogy a dísz adatait az alábbi formában adja vissza (idézőjelek nélkül): „<név>, <ár> Ft” (pl. „piros üveggömb, 300 Ft”)!
3. Definiálja felül az **Object** osztályból örökölt **equals()** metódust úgy, hogy két dísz akkor legyen egyenlő, ha a nevük megegyezik, és áruk legfeljebb 100 Ft-tal tér el egymástól! (Segítség: a **Math** osztály **abs(x)** metódusa visszaadja **x** abszolút értékét.)
4. Egészítse ki a **Dísz** osztály definícióját úgy, hogy a példányai az áruk alapján összehasonlíthatók legyenek! A feladat megoldásához implementálja a **Comparable** interfészt!
5. Származtasson a **Dísz** osztályból egy **Izzósor** osztályt! Az izzósorok a dísz tulajdonságain kívül még egy egész értékű (más osztályból nem látható) **izzókSzáma** tulajdonsággal is rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a három adattagjának kezdőérték adható!
6. Definiálja felül a **Dísz** osztályból örökölt **toString()** metódust úgy, hogy az izzósor adatait az alábbi formátumban adja vissza (idézőjelek nélkül): „<izzókSzáma> darabos <név>, <ár> Ft” (pl. „15 darabos ledes karácsonyfaizzó, 2500 Ft”)! A megoldásban használja fel a **Dísz** osztály **toString()** metódusát!
7. Készítsen egy **Main** nevű osztályt, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **Izzósor** típusú objektumot, amely az előző feladat példájában megadott izzósort reprezentálja! Írja ki a képernyőre a létrehozott objektumot!
8. Hozzon létre a főprogramban egy díszeket tartalmazó (tetszőleges) kollekciót, amelybe vegye fel az előző feladatban létrehozott objektumot, valamint azon kívül még két újonnan létrehozott díszt!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely a billentyűzetről soronként egy-egy árértéket olvas be mindaddig, amíg olyan árat nem kap, amilyennél drágább dísz szerepel az előző feladatban feltöltött kollekcióban (feltesszük, hogy azóta a kollekció megváltozott), majd írja ki a dísz nevét!
10. Egészítse ki az **Izzósor** osztályt egy olyan metódussal, amely visszaadja az izzósor izzónkénti árát (pontosan, valós számként)!
11. Írjon a **Main** osztályban egy statikus metódust, amely paraméterként megkap egy díszeket tartalmazó kollekciót, és visszatér a kollekcióban található olyan izzósorok számával, amelyeknek az izzónkénti ára meghaladja a 200 Ft-ot! A megoldásban használja fel az előző feladatban megírt metódust!
12. Készítsen egy **Karácsonyfa** nevű osztályt, amely értelemszerűen implementálja az alábbi interfészt:

```
interface DíszGyűjtemény
{
  public void hozzáad( Dísz dísz ); // új díszt ad hozzá a gyűjteményhez
  public int érték();               // meghatározza a díszek összértékét
}
```

13. Készítsen a **Karácsonyfa** osztályhoz egy olyan konstruktort, amely egy paraméterként kapott, díszeket tartalmazó tömb elemeit hozzáadja a karácsonyfához!
14. Egészítse ki a **Karácsonyfa** osztályt egy olyan metódussal, amely képernyőre írja a karácsonyfán található díszek közül a legdrágábbnak a nevét! Ha a karácsonyfán egyetlen dísz sincs, akkor a metódus dobjon **DíszítetlenFaException** kivételt! (A feladathoz a kivétel osztályát is meg kell írni.)
15. Egészítse ki a **Karácsonyfa** osztályt egy olyan metódussal, amely a karácsonyfán található izzósorok nevét csupa nagybetűssé alakítja! Ha szükséges, egészítse ki az **Izzósor** osztályt! (Segítség: a **String** osztály **toUpperCase()** metódusa visszaadja egy **String** objektum nagybetűs megfelelőjét.)
16. Írjon a **Main** osztályban egy statikus metódust, amely paraméterként megkap egy karácsonyfákat tartalmazó tömböt, és visszatér a tömbbeli karácsonyfákon található díszek átlagos árával!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2009. május 23.**

Adott az alábbi osztály az **ital** csomagban:

```
public class Ital
{
  protected String név;
  protected String kiszerelés;
  protected int ár;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható, és írja meg az adattagokhoz tartozó lekérdező metódusokat!
2. Definiálja felül a **toString()** metódust úgy, hogy az ital adatait az alábbi formában adja vissza (idézőjelek nélkül): „<név>, <kiszerelés>, <ár> Ft” (pl. „Coca-Cola, 5 dl, 150 Ft”)!
3. Definiálja felül az **equals()** metódust úgy, hogy két ital akkor legyen egyenlő, ha a nevük és a kiszerelésük megegyezik!
4. Egészítse ki az **Ital** osztály definícióját úgy, hogy példányai az áruk alapján összehasonlíthatók legyenek! A feladat megoldásához implementálja a **Comparable** interfészt!
5. Származtasson az **Ital** osztályból egy nyilvános láthatóságú **SzeszesItal** osztályt, amelyet szintén az **ital** csomagban helyezzen el! A szeszesitalok az ital tulajdonságain kívül még egy valós értékű (más osztályból nem látható) **alkoholTartalom** tulajdonsággal is rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható!
6. Definiálja felül az **Ital** osztályból örökölt **toString()** metódust úgy, hogy a szeszesital adatait az alábbi formátumban adja vissza (idézőjelek nélkül): „<alkoholTartalom>% alkoholtartalmú <név>, <kiszerelés>, <ár> Ft” (pl. „11,5% alkoholtartalmú Kékfrankos, 0,75 l, 1500 Ft”)! A megoldásban használja fel az örökölt **toString()** metódust!
7. Készítsen egy **Beugró** nevű osztályt a **beugró** csomagban, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy **SzeszesItal** típusú objektumot, amely az előző feladat példájában megadott szeszesitalt reprezentálja! Írja ki a képernyőre a létrehozott objektumot!
8. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy italokat tartalmazó kollekciót és egy italnevet, képernyőre írja, hogy a megadott nevű ital milyen kiszerelésekben létezik a kollekcióban, és igazzal tér vissza, ha volt ilyen nevű ital, különben hamissal!
9. Egészítse ki a főprogramot egy olyan kódrészlettel, amely létrehoz egy italokat tartalmazó *n* elemű tömböt, ahol *n* értékét az első parancssori argumentum adja meg, majd a felhasználótól beolvassa *n* db ital adatait, létrehozza az italokat, és elhelyezi őket a tömbben! Minden második beolvasott ital legyen szeszesital!
10. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy italokat tartalmazó tömböt, és visszaad egy olyan halmazt (**java.util.HashSet**), amely a tömbben található, 10%-nál magasabb alkoholtartalmú szeszesitalokat tartalmazza! Hívja meg a metódust a főprogramban az előző feladatban létrehozott tömbbel, és tárolja el a visszakapott halmazt!
11. Egészítse ki a főprogramot egy olyan kódrészlettel, amely a billentyűzetről egy-egy nevet olvas be mindaddig, amíg olyan nevet nem kap, amilyen nevű ital szerepel az előző feladatban megkapott halmazban, majd írja ki, hogy milyen kiszerelésekben létezik ilyen nevű ital! A megoldáshoz használja fel a 8. feladatban megírt metódust!
12. Készítsen egy **Kocsma** nevű osztályt a **beugró** csomagban, amely értelemszerűen implementálja az alábbi interfészt:

```
interface ItalBolt
{
  public void hozzáad( Ital ital ); // új italt ad hozzá az italbolthoz
  public int érték();               // meghatározza az italok összértékét
}
```

13. Készítsen a **Kocsma** osztályhoz egy olyan konstruktort, amely egy paraméterként kapott, italokat tartalmazó kollekció elemeit hozzáadja az italbolthoz, de csak azokat, amelyek eddig még nem szerepeltek benne!
14. Egészítse ki a **Kocsma** osztályt egy olyan metódussal, amely képernyőre írja a kocsmában található italok közül a legdrágábbnak a nevét és az árát! Ha a kocsmában egyetlen ital sincs, akkor a metódus dobjon **ÜresKocsmaException** kivételt! (A feladathoz a kivétel osztályát is meg kell írni.)
15. Egészítse ki a **Kocsma** osztályt egy olyan metódussal, amely „5 dl”-re módosítja a kocsmában található olyan szeszesitalok kiszerelését, amelyeknek eddig „0,5 l” volt! Ha szükséges, egészítse ki a **SzeszesItal** osztályt!
16. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy kocsmákat tartalmazó tömböt, és visszatér a tömbbeli kocsmákban található italok összértékével!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2009. június 7.**

Adott az alábbi osztály az **egyetem** csomagban:

```
public class Oktató
{
  protected String név, neptunKód, tanszék;
}
```

1. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mindhárom adattagjának kezdőérték adható, és írja meg az adattagokhoz tartozó lekérdező metódusokat!
2. Definiálja felül a **toString()** metódust úgy, hogy az oktató adatait az alábbi formában adja vissza (idézőjelek nélkül): „<név> (<neptunKód>) tanszéke: <tanszék>” (pl. „Pánovics János (FF8SC5) tanszéke: Információ Technológia Tanszék”)!
3. Definiálja felül az **equals()** metódust úgy, hogy két oktató akkor legyen egyenlő, ha a Neptun-kódjuk megegyezik!
4. Egészítse ki az **Oktató** osztály definícióját úgy, hogy példányai a nevük alapján összehasonlíthatók legyenek! A feladat megoldásához implementálja a **Comparable** interfészt!
5. Származtasson az **Oktató** osztályból egy nyilvános láthatóságú **Demonstrátor** osztályt, amelyet szintén az **egyetem** csomagban helyezzen el! A demonstrátorok az oktató tulajdonságain kívül még egy szöveges (más osztályból nem látható) **szak** tulajdonsággal is rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amely a tanszéket a „fiktív tanszék” szövegre állítja, a másik három adattag értékét pedig paraméterek határozzák meg!
6. Készítsen egy **Beugró** nevű osztályt a **beugró** csomagban, amely a főprogramot tartalmazza! A főprogramban hozzon létre egy oktatókat tartalmazó halmazt (**java.util.HashSet**), majd a felhasználótól olvassa be 4 db oktató adatait, hozza létre az oktatókat, és helyezze el őket a halmazban! Minden második beolvasott oktató legyen demonstrátor (akinek a tanszéke helyett a szakját kell beolvasni)!
7. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy oktatókat tartalmazó kollekciót és egy Neptun-kódot, és eldönti, hogy van-e a megadott Neptun-kóddal rendelkező oktató a kollekcióban! Ha van, akkor írja ki a képernyőre az oktató nevét és tanszékét, ha nincs, akkor írja ki a „nincs ilyen oktató” szöveget!
8. Egészítse ki a főprogramot egy olyan kódrészlettel, amely képernyőre írja a parancssori argumentumokként kapott Neptun-kódokkal rendelkező oktatók nevét és tanszékét a 6. feladatban létrehozott halmazból! A megoldáshoz használja fel az előző feladatban megírt metódust!
9. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy oktatókat tartalmazó kollekciót, és visszaad egy olyan tömböt, amely a kollekcióban található „informatikatanár” szakos demonstrátorokat tartalmazza! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott halmazzal, és írja ki a képernyőre a visszakapott tömb elemeit!
10. Készítsen egy **Egyetem** nevű osztályt az **egyetem** csomagban, amely értelemszerűen implementálja az alábbi interfészt:

    ```
    interface Iskola
    {
      public void felvesz( Oktató oktató );  // új oktatót ad hozzá az iskolához
      public Collection<Oktató> oktatók();   // visszaadja az iskola oktatóit
    }
    ```

11. Készítsen az **Egyetem** osztályhoz egy olyan konstruktort, amely egy paraméterként kapott, oktatókat tartalmazó kollekció elemeit hozzáadja az egyetemhez, de csak azokat, amelyek eddig még nem szerepeltek benne!
12. Egészítse ki az **Egyetem** osztályt egy olyan metódussal, amely képernyőre írja az egyetemen található oktatók nevét és tanszékét a nevek szerinti ábécérendben! Ha az egyetemen egyetlen oktató sincs, akkor a metódus dobjon **ÜresEgyetemException** kivételt! (A feladathoz a kivétel osztályát is meg kell írni.)
13. Egészítse ki az **Egyetem** osztályt egy olyan metódussal, amely az egyetemen található demonstrátorok tanszékét a paraméterként megkapott sztringre módosítja! Ha szükséges, egészítse ki a **Demonstrátor** osztályt!
14. Írjon a **Beugró** osztályban egy statikus metódust, amely paraméterként megkap egy egyetemeket tartalmazó tömböt, és képernyőre írja a tömbbeli egyetemeken dolgozó oktatók számát soronként!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2009. december 19.**

1. Készítse el az **Ajándék** osztályt a **mikulás** csomagban! Az ajándékokról tudni szeretnénk a nevüket (sztring), a tömegüket (valós) és az árukat (egész). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható!
2. Definiálja felül a **toString()** metódust úgy, hogy az az ajándék adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<tömeg> kg), <ár> Ft” (pl. „nyakkendő (0,3 kg), 4000 Ft”)!
3. Definiálja felül az **equals()** metódust is: két ajándék akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik, és a tömegük legfeljebb 1 kg-mal tér el egymástól (lásd a **Math.abs()** metódust)!
4. Az ajándékok természetes rendezettsége az áruk szerint legyen értelmezve!
5. Származtasson az **Ajándék** osztályból egy **GyerekJáték** osztályt szintén a **mikulás** csomagban! A gyerekjátékok az ajándék tulajdonságain kívül még egy egész értékű, más osztályból nem látható **korhatár** tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg a korhatárhoz tartozó lekérdező metódust!
6. Definiálja felül a **toString()** metódust úgy, hogy az a gyerekjáték adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<tömeg> kg), <ár> Ft, <korhatár> éves kortól” (pl. „Monopoly (0,8 kg), 13500 Ft, 8 éves kortól”)!
7. Adott az alábbi interfész a **mikulás** csomagban:

```
public interface AjándékCsomag
{
  public int összÉrték();          // visszaadja az ajándékok összértékét
  public int nehezebb(double t);   // visszaadja a t-nél nehezebb ajándékok számát
  public Collection<GyerekJáték> gyerekjátékok();   // visszaadja a gyerekjátékokat
}
```

   Készítse el a **Puttony** osztályt a **mikulás** csomagban, amely értelemszerűen (a megjegyzéseknek megfelelően) implementálja az **AjándékCsomag** interfészt! Figyeljen arra, hogy az ajándékcsomagban lehetnek egyforma ajándékok! Készítsen az osztályhoz egy konstruktort, amelynek segítségével a puttony feltölthető egy ajándékokat tartalmazó tömb elemeivel!
8. Helyezze el a főprogramot a **télapó** csomag **Télapó** osztályában! A főprogramban olvasson be a billentyűzetről öt ajándékot, soronként egyet-egyet! Az ajándékok egyes adatait vessző választja el egymástól. Ha egy sorban három adat szerepel, akkor az egy ajándékot, ha négy, akkor pedig egy gyerekjátékot reprezentál. Hozza létre a beolvasott adatokkal rendelkező megfelelő típusú objektumokat, és helyezze el őket egy ötelemű tömbben! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()**, **hasNextXxx()**, **nextXxx()** metódusait, ahol az **Xxx** helyére a primtív típusokat kell helyettesíteni!)
9. Hozzon létre a főprogramban egy puttony objektumot, amelybe vegye fel az előző feladatban feltöltött tömbben található ajándékokat, majd írja ki a képernyőre a puttonyban található legdrágább gyerekjátékot! (Segítség: lásd a **Collections** osztály **max()** metódusát!)
10. Egészítse ki a **Puttony** osztályt egy metódussal, amely igazat ad vissza, ha a puttonyban van két egyenlő ajándék, különben hamisat! Hívja meg a metódust a főprogramban az előző feladatban létrehozott puttony objektumra, és írjon ki egy üzenetet a képernyőre az eredménynek megfelelően!
11. Egészítse ki a **Puttony** osztályt egy metódussal, amely paraméterként megkap egy egész számot, és visszaadja egy kollekcióban a puttonyban található gyerekjátékok közül azokat, amelyek korhatára nagyobb a kapott értéknél! Ha a paraméter értéke negatív, a metódus dobjon **IllegalArgumentException** kivételt! Hívja meg a metódust a főprogramban a 9. feladatban létrehozott puttony objektumra, és írjon ki egy hibaüzenetet a képernyőre, ha a metódus hívása **IllegalArgumentException** kivételt eredményez!
12. Egészítse ki a **Télapó** osztályt egy statikus metódussal, amely paraméterként megkap egy **AjándékCsomag** objektumot és egy valós értéket, majd képernyőre írja, hogy a kapott ajándékcsomagban hány olyan ajándék található, amely nehezebb a kapott értéknél! Hívja meg a metódust a főprogramban a 9. feladatban létrehozott puttony objektummal, valamint az első parancssori argumentumban megadott valós számmal! (Segítség: egy sztring valós számmá történő konvertálásához lásd a **Double** osztály **parseDouble()** metódusát!)
13. Egészítse ki a **Télapó** osztályt egy statikus metódussal, amely paraméterként megkap egy **AjándékCsomag** objektumokat tartalmazó tömböt, és visszaadja (valós számként) a tömbbeli ajándékcsomagok átlagértékét!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2010. január 9.**

1. Készítse el a **Tantárgy** osztályt az **egyetem** csomagban! A tantárgyakról tudni szeretnénk a kódjukat (sztring), a nevüket (sztring), és hogy hány kreditesek (egész). Az osztály adattagjai legyenek privát láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg a kreditszámhoz tartozó lekérdező metódust!
2. Definiálja felül a **toString()** metódust úgy, hogy az a tantárgy adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<kód> <név> (<kredit> kredites)” (pl. „ILDK302 Magas szintű programozási nyelvek 2 (5 kredites)”)!
3. Definiálja felül az **equals()** metódust is: két tantárgy akkor legyen egyenlő, ha a kódjuk (tartalmilag) megegyezik!
4. A tantárgyak természetes rendezettsége a kreditszámuk, azon belül pedig a nevük szerint legyen értelmezve!
5. Származtasson a **Tantárgy** osztályból egy **Kurzus** osztályt szintén az **egyetem** csomagban! A kurzusokról a tantárgy tulajdonságain kívül tudni szeretnénk még az alábbiakat is: előadás-e vagy gyakorlat (logikai), melyik teremben van megtartva (sztring), illetve hogy a hét melyik napján (egész 1-től 7-ig), melyik órában (egész 0-tól 23-ig) és melyik percben (egész 0-tól 59-ig) kezdődik. Az adattagok legyenek privát láthatóságúak!
6. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Ha a nap, az óra és a perc közül bármelyik is hibásan lett megadva, a konstruktor dobjon **HibásAdatException** kivételt! Készítse el a kivétel osztályát is!
7. Készítsen egy másik konstruktort is, amellyel csak a tantárgy adatai és a kurzus jellege (előadás vagy gyakorlat) adható meg tetszés szerint, a többi adattag kezdőértékei legyenek az alábbiak: a terem **null**, a nap 1 (hétfő), az óra 8, a perc pedig 0!
8. Definiálja felül a **toString()** metódust úgy, hogy az a kurzus adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<kód> <név> (<kredit> kredites) [előadás|gyakorlat], <terem> teremben, kezdete: <nap> <óra>:<perc>” (pl. „ILDK302 Magas szintű programozási nyelvek 2 (5 kredites) előadás, M419 teremben, kezdete: csütörtök 14:00”)! Ügyeljen arra, hogy a napot szövegesen, a percet pedig két karakterrel adja meg!
9. Adott az alábbi interfész az **egyetem** csomagban:

```
public interface TanterviHáló
{
  public void újTárgy(Tantárgy t);   // új tárgyat ad a hálóhoz
  public int összKredit();   // visszaadja a háló tárgyainak összkreditszámát
  public Collection<Tantárgy> tárgyak();   // visszaadja a tárgyak egy kollekcióját
}
```

   Készítse el a **PTIHáló** osztályt az **egyetem** csomagban, amely értelemszerűen (a megjegyzéseknek megfelelően) implementálja a **TanterviHáló** interfészt!
10. Helyezze el a főprogramot az **unideb** csomag **Próba** osztályában! A főprogramban olvasson be a billentyűzetről öt tantárgyat, soronként egyet-egyet! A tantárgyak egyes adatait kettőspont választja el egymástól (a sorok tehát „<kód>:<név>:<kredit>” formájúak). A beolvasott tárgyakat helyezze el egy listában, majd írja ki a képernyőre a listában található tantárgyakat a természetes rendezettségük sorrendjében! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()** metódusait, illetve a **Collections** osztály **sort()** metódusát!)
11. Ellenőrizze, hogy a program első parancssori argumentumaként megkapott kódú tantárgy szerepel-e a listában! Ha igen, írja ki a tantárgyat a képernyőre, ha nem, akkor írjon ki egy ilyen értelmű üzenetet! (Segítség: lásd a **List** interfész **indexOf()** metódusát!)
12. Egészítse ki a **Próba** osztályt egy statikus metódussal, amely paraméterként megkap egy **TanterviHáló** típusú objektumot, a kapott háló minden tantárgyához létrehoz egy elméleti és egy gyakorlati kurzust alapértelmezett teremmel és időponttal, és visszatér egy tömbbel, amely a létrehozott kurzusokat tartalmazza! Ha szükséges, egészítse ki a **Tantárgy** osztályt!
13. Hozzon létre a főprogramban egy **PTIHáló** objektumot, amelybe vegye fel a 10. feladat listájában található tantárgyakat, majd hívja meg az előző feladatban megírt metódust ezzel az objektummal, és írja ki a képernyőre a visszaadott tömbben lévő kurzusokat!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2010. május 22.**

1. Készítse el a **Számítógép** osztályt a **hardver** csomagban! A számítógépekről tudni szeretnénk a típusukat (sztring), az árukat (egész) és hogy van-e bennük WiFi (logikai). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg a logikai típusú adattaghoz tartozó lekérdező metódust!
2. Definiálja felül a **toString()** metódust úgy, hogy az a számítógép adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<típus>, <ár> Ft ([wifivel|nincs wifi])`” (pl. „Kventa Optima, 150000 Ft (nincs wifi)”)!
3. Definiálja felül az **equals()** metódust is: két számítógép akkor legyen egyenlő, ha a típusuk (tartalmilag) megegyezik!
4. A számítógépek természetes rendezettsége az áruk, azon belül pedig a típusuk szerint legyen értelmezve!
5. Származtasson a **Számítógép** osztályból egy **Laptop** osztályt szintén a **hardver** csomagban! A laptopok a számítógép tulajdonságain kívül még egy egész értékű, más osztályból nem látható készenléti idő tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható!
6. Definiálja felül a **toString()** metódust úgy, hogy az a laptop adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<típus>, <ár> Ft ([wifivel|nincs wifi]), <készenléti idő> perc`” (pl. „Gigabyte M912, 280000 Ft (wifivel), 300 perc”)!
7. Adott az alábbi interfész a **hardver** csomagban: (2 pont)

```
public interface Géppark
{
  public void felvesz(Számítógép szg);     // új számítógépet ad a gépparkhoz
  public double átlagÁr() throws ÜresGépparkException;
     // visszaadja a géppark gépeinek az átlagárát, üres géppark esetén kivételt dob
  public Collection<Számítógép> gépek();   // visszaadja a számítógépek kollekcióját
}
```

   Készítse el a **Gépterem** osztályt a **hardver** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Géppark** interfészt! A géptermeknek legyen nevük! Egészítse ki az osztályt egy konstruktorral, amellyel megadható a gépterem neve, és feltölthető a gépparkja egy kollekció elemeivel! Írja meg a kivétel osztályát is!

8. Helyezze el a főprogramot az **unideb** csomag **Próba** osztályában! A főprogramban olvassa be a billentyűzetről soronként egy-egy számítógép kettősponttal elválasztott adatait! A sorok az alábbi formájúak lehetnek:

```
S:<típus>:<ár>:[i|n]
L:<típus>:<ár>:[i|n]:<készenléti idő>
V
```

   Ha az első beolvasott karakter egy S betű, akkor a további adatok egy számítógéphez tartoznak, ha L betű, akkor pedig egy laptophoz. A bemenet végét egy V betű jelzi. A beolvasott számítógépeket helyezze el egy listában, majd írja ki a képernyőre a listában található számítógépeket a természetes rendezettségük sorrendjében! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **nextLine()**, **next()**, **nextInt()** metódusait, illetve a **Collections** osztály **sort()** metódusát!) (2 pont)

9. Ellenőrizze, hogy a program első parancssori argumentumaként megkapott típusú számítógép szerepel-e a listában! Ha igen, írja ki a számítógépet a képernyőre, ha nem, akkor írjon ki egy ilyen értelmű üzenetet! Ha nincs megadva parancssori argumentum, akkor jelenjen meg egy hibaüzenet! (Segítség: lásd a **List** interfész **indexOf()** metódusát!)
10. Egészítse ki a **Próba** osztályt egy statikus metódussal, amely paraméterként megkap egy **Géppark** típusú objektumot, és visszatér a gépparkban található legdrágább laptop típusával! Ha a gépparkban egyetlen laptop sincs, akkor a metódus dobjon **NincsLaptopException** kivételt!
11. Hozzon létre a főprogramban egy **Gépterem** objektumot, amelybe vegye fel a 8. feladat listájában található számítógépeket! A gépterem neve legyen M212! Írja ki a képernyőre a gépterem számítógépeinek az átlagárát! Hívja meg az előző feladatban megírt metódust ezzel az objektummal, és írja ki az eredményt a képernyőre! Kezelje az esetleges kivételeket! (2 pont)
12. Egészítse ki a **Próba** osztályt egy statikus metódussal, amely paraméterként megkap egy géptermeket tartalmazó tömböt, és visszatér a géptermekben található wifis laptopok számával!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2010. június 5.**

1. Készítse el a **Sportoló** osztályt a **sport** csomagban! A sportolókról tudni szeretnénk a nevüket (sztring), az életkorukat (egész) és hogy milyen sportágat űznek (sztring). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az egyes adattagokhoz tartozó lekérdező metódusokat, valamint a sportághoz tartozó beállító metódust!

2. Definiálja felül a **toString()** metódust úgy, hogy az a sportoló adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<életkor> éves), sportág: <sportág>” (pl. „Kásás Tamás (34 éves), sportág: vízilabda”)!

3. Definiálja felül az **equals()** metódust is: két sportoló akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik!

4. A sportolók természetes rendezettsége az életkoruk, azon belül pedig a nevük szerint legyen értelmezve!

5. Származtasson a **Sportoló** osztályból egy **Focista** osztályt a **foci** csomagban! A focisták olyan sportolók, akiknek a sportága csak „futball” lehet, illetve a sportoló tulajdonságain kívül még egy szöveges, más osztályból nem látható poszt tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével a sportágon kívül mindegyik adattagjának kezdőérték adható! Definiálja felül a sportághoz tartozó beállító metódust úgy, hogy ne lehessen vele módosítani a sportágat!

6. Definiálja felül a **toString()** metódust úgy, hogy az a focista adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<életkor> éves) focista, posztja: <poszt>” (pl. „Didier Drogba (32 éves) focista, posztja: csatár”)!

7. Adott az alábbi interfész a **sport** csomagban:

```
public interface Csapat
{
  public void igazol(Sportoló játékos);       // új játékost igazol a csapathoz
  public double átlagÉletkor() throws NincsJatekosException;
     // visszaadja a csapat játékosainak az átlagéletkorát,
     // üres csapat esetén kivételt dob
  public Collection<Sportoló> játékosKeret(); // visszaadja a játékosok kollekcióját
}
```

   Készítse el a **Válogatott** osztályt a **foci** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Csapat** interfészt! A válogatottakban csak focisták szerepelhetnek. Egészítse ki az osztályt egy konstruktorral, amellyel feltölthető a játékoskeret egy sportolókat tartalmazó kollekcióban található focistákkal! Írja meg a kivétel osztályát is! (2 pont)

8. Helyezze el a főprogramot a **magyarok** csomag **SportTeszt** osztályában! A főprogramban olvassa be a billentyűzetről soronként egy-egy sportoló kettősponttal elválasztott adatait! A sorok az alábbi formájúak lehetnek:

```
S:<név>:<életkor>:<sportág>
F:<név>:<életkor>:<poszt>
V
```

   Ha az első beolvasott karakter egy S betű, akkor a további adatok egy sportolóhoz tartoznak, ha F betű, akkor pedig egy focistához. A bemenet végét egy V betű jelzi. A beolvasott sportolókat helyezze el egy listában, majd írja ki a képernyőre a listában található sportolókat a természetes rendezettségük sorrendjében! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **nextLine()**, **next()**, **nextInt()** metódusait, illetve a **Collections** osztály **sort()** metódusát!) (2 pont)

9. Ellenőrizze, hogy a program első parancssori argumentumaként megkapott nevű sportoló szerepel-e a listában! Ha igen, írja ki a sportolót a képernyőre, ha nem, akkor írjon ki egy ilyen értelmű üzenetet! Ha nincs megadva parancssori argumentum, akkor jelenjen meg egy hibaüzenet! (Segítség: lásd a **List** interfész **indexOf()** és **get()** metódusait!)

10. Egészítse ki a **SportTeszt** osztályt egy statikus metódussal, amely paraméterként megkapja sportolók egy kollekcióját, és nagybetűsre módosítja a kollekcióban található sportolók sportágait! Hívja meg a metódust a főprogramban a 8. feladat listájával, majd írja ki a módosított listát a képernyőre! (Segítség: lásd a **String** osztály **toUpperCase()** metódusát!)

11. Hozzon létre a főprogramban egy **Válogatott** objektumot, amelybe vegye fel a 8. feladat listájában található focistákat! Írja ki a képernyőre a válogatott játékoskeretét, valamint a játékosok átlagéletkorát! Kezelje az esetleges ellenőrzött kivételeket hibaüzenet kiírásával!

12. Egészítse ki a **Sportoló** osztályt olyan lehetőséggel, amelynek segítségével megadható, hogy egy sportoló hány éves kortól számít idősnek! Az alapértelmezett érték legyen 40!

13. Egészítse ki a **SportTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy csapatokat tartalmazó tömböt, és visszatér a csapatokban található idős játékosok számával! Állítsa át a főprogramban a sportolók korhatárát 40-ről 35-re, majd hívja meg a metódust a 11. feladatban létrehozott válogatott objektumot tartalmazó egyelemű tömbbel, és írja ki az eredményt a képernyőre!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2010. december 18.**

1. Készítse el az **Ügynök** osztályt a **házaló** csomagban! Az ügynökökről tudni szeretnénk a nevüket (sztring), a nemüket (logikai) és hogy milyen áruval kereskednek (sztring). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az áruhoz tartozó lekérdező és beállító metódust! A többi adattaghoz nem írhat lekérdező metódust!

2. Definiálja felül a **toString()** metódust úgy, hogy az az ügynök adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név> ([férfi|nő]), áru: <áru>`” (pl. „Antunovics Antal (férfi), áru: Calgon”)!

3. Definiálja felül az **equals()** metódust is: két ügynök akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik!

4. Az ügynökök természetes rendezettsége is a nevük szerint legyen értelmezve!

5. Származtasson az **Ügynök** osztályból egy **Titkosügynök** osztályt a **világbéke** csomagban! A titkosügynök olyan ügynök, akik kizárólag a „titok” nevű „áruval” kereskedhet, illetve az ügynök tulajdonságain kívül még egy más osztályból nem látható logikai tulajdonsággal is rendelkezik, amely azt mondja meg, hogy felrázva vagy keverve issza-e a vodka-martinit. Készítsen az osztályhoz egy konstruktort, amelynek segítségével az árun kívül mindegyik adattagjának kezdőérték adható! Definiálja felül az áruhoz tartozó beállító metódust úgy, hogy ne lehessen vele módosítani az árut!

6. Definiálja felül a **toString()** metódust úgy, hogy az a titkosügynök adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név> egy [férfi|női] titkosügynök, aki [felrázva|keverve] issza a vodka-martinit`” (pl. „James Bond egy férfi titkosügynök, aki felrázva issza a vodka-martinit”)!

7. Adott az alábbi interfész a **házaló** csomagban:

```
public interface Ügynöklista
{
  public void felvesz(Ügynök ügynök);   // új ügynököt vesz fel a listára
  public int nőiÜgynökök();             // visszaadja az ügynöklistán szereplő
                                        //   női ügynökök számát
  public Collection<Ügynök> lista();    // visszaadja az ügynökök listáját
}
```

   Készítse el a **Kémszervezet** osztályt a **világbéke** csomagban, amely a megjegyzéseknek megfelelően implementálja az **Ügynöklista** interfészt! A kémszervezetekben csak titkosügynökök szerepelhetnek. Szükség szerint egészítse ki a **Titkosügynök** osztályt! Írjon az osztályhoz egy konstruktort, amellyel feltölthető az ügynöklista egy ügynököket tartalmazó kollekcióban található titkosügynökökkel! (2 pont)

8. Helyezze el a főprogramot az **mi6** csomag **ÜgynökTeszt** osztályában! A főprogramban olvassa be az `adatok.txt` nevű állományból soronként egy-egy ügynök kettősponttal elválasztott adatait! A sorok az alábbi formájúak lehetnek:

   `U:<név>:[f|F|n|N]:<áru>`
   `T:<név>:[f|F|n|N]:[f|F|k|K]`

   Ha az első beolvasott karakter egy U betű, akkor a további adatok egy ügynökhöz tartoznak, ha T betű, akkor pedig egy titkosügynökhöz. A harmadik mezőben a (kis vagy nagy) F betű férfit, az N betű nőt jelöl. A titkosügynökökhöz tartozó negyedik mezőben az F betű azt jelenti, hogy a titkosügynök felrázva, a K betű pedig azt, hogy keverve issza a vodka-martinit. A beolvasott ügynököket helyezze el egy listában, majd írja ki a képernyőre a listában található ügynököket a természetes rendezettségük sorrendjében! Kezelje az esetleges ellenőrzött kivételeket! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **nextLine()**, **next()** metódusait, illetve a **Collections** osztály **sort()** metódusát!) (2 pont)

9. Ellenőrizze, hogy a program első parancssori argumentumaként megkapott nevű ügynök szerepel-e a listában! Ha igen, írja ki az ügynököt a képernyőre, ha nem, akkor írjon ki egy ilyen értelmű üzenetet! Ha nincs megadva parancssori argumentum, akkor jelenjen meg egy hibaüzenet! (Segítség: lásd a **List** interfész **indexOf()** és **get()** metódusait!)

10. Egészítse ki az **ÜgynökTeszt** osztályt egy statikus metódussal, amely paraméterként megkapja ügynökök egy kollekcióját, és kisbetűsre módosítja a kollekcióban található ügynökök által kínált árukat! Hívja meg a metódust a főprogramban a 8. feladat listájával, majd írja ki a módosított listát a képernyőre! (Segítség: lásd a **String** osztály **toLowerCase()** metódusát!)

11. Hozzon létre a főprogramban egy **Kémszervezet** objektumot, amelybe vegye fel a 8. feladat listájában található titkosügynököket! Írja ki a képernyőre a kémszervezet ügynöklistáját, valamint a kémszervezetben szereplő női ügynökök számát!

12. Egészítse ki az **ÜgynökTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy ügynöklistákat tartalmazó kollekciót, és visszatér az ügynöklistákon szereplő azon női titkosügynökök neveit tartalmazó halmazzal, akik keverve isszák a vodka-martinit! A metódus dobjon **NoSuchElementException** kivételt, ha egy ilyen titkosügynök sincs az ügynöklistákon! Szükség szerint egészítse ki a **Titkosügynök** osztályt!

13. Hívja meg a főprogramban az előző feladatban megírt metódust a 11. feladatban létrehozott kémszervezet objektumot tartalmazó egyelemű kollekcióval, és írja ki az eredményt a képernyőre! Kezelje a **NoSuchElementException** kivételt hibaüzenet kiírásával!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2011. január 8.**

1. Készítse el a **Játékos** osztályt a **játék** csomagban! Az osztállyal valóságshow-szereplőket szeretnénk modellezni, akikről tudni szeretnénk a nevüket (sztring), az életkorukat (egész) és a foglalkozásukat (sztring). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az egyes adattagokhoz tartozó lekérdező metódusokat, valamint a névhez tartozó beállító metódust! Definiálja felül a **toString()** metódust úgy, hogy az a játékos adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<életkor> éves), foglalkozása: <foglalkozás>” (pl. „Alekosz (32 éves), foglalkozása: zenész, biztonsági őr”)!

2. Definiálja felül az **equals()** metódust is: két játékos akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik, és az életkoruk legfeljebb 1-gyel tér el egymástól! (Segítség: lásd a **Math** osztály **abs()** metódusát.)

3. A játékosok természetes rendezettsége az életkoruk, azon belül pedig a nevük szerint legyen értelmezve!

4. Származtasson a **Játékos** osztályból egy **CelebJátékos** osztályt szintén a **játék** csomagban! A celeb játékosok a játékos tulajdonságain kívül még egy más osztályból nem látható, egész értékű tiszteletdíj tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Negatív tiszteletdíj esetén a konstruktor dobjon **HibásÉrtékException** kivételt! Írja meg a tiszteletdíjhoz tartozó lekérdező metódust!

5. Definiálja felül a **toString()** metódust úgy, hogy az a celeb játékos adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „<név> (<életkor> éves), foglalkozása: <foglalkozás>, tiszteletdíja: <tiszteletdíj> Ft” (pl. „Benkő Dániel (63 éves), foglalkozása: lant- és gitárművész, tiszteletdíja: 300000 Ft”)!

6. Adott az alábbi interfész a **játék** csomagban:

```
public interface Valóságshow
{
  public void beszavaz(Játékos j);          // új játékos kerül a show-ba
  public void beszavaz(Játékos j1, Játékos j2);   // két új játékos kerül a show-ba
  public void kiszavaz(Játékos j);          // egy játékos kilép a show-ból
  public double átlagÉletkor();             // megadja a játékosok átlagéletkorát
  public Collection<Játékos> csapat();      // visszaadja a játékosok listáját
  public void kiír(String fájlnév); // kiírja a megadott fájlba a játékosok listáját
}
```

   Készítse el a **ValóVilág** osztályt a **valóvilág** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Valóságshow** interfészt! Figyeljen arra, hogy ugyanaz a játékos egy valóságshow-ban csak egyszer fordulhat elő! Legyen lehetőség a ValóVilág évadának tárolására egész számként! Készítsen az osztályhoz egy konstruktort, amelynek segítségével a show évadán kívül megadható a játékoskeret is egy játékosokat tartalmazó tömb segítségével! (2 pont)

7. Helyezze el a főprogramot az **rtl** csomag **BeleValó** osztályában! A főprogramban olvasson be a billentyűzetről öt játékost, soronként egyet-egyet! A játékosok egyes adatait pontosvessző választja el egymástól. Ha egy sorban három adat szerepel, akkor az egy egyszerű játékost, ha négy, akkor pedig egy celebet reprezentál. Hozza létre a beolvasott adatokkal rendelkező megfelelő típusú objektumokat, és helyezze el őket egy ötelemű tömbben! Kezelje az esetleges ellenőrzött kivételeket! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **nextLine()**, **next()**, **nextInt()** metódusait.)

8. Hozzon létre a főprogramban egy **ValóVilág** objektumot, amelynek évada az első parancssori argumentumban megadott szám! Töltse fel a show játékoskeretét az előző feladatban feltöltött tömbben található játékosokkal, majd írja ki a képernyőre a show-ban szereplő játékosok átlagéletkorát! (Segítség: egy sztring egész számmá történő konvertálásához lásd az **Integer** osztály **parseInt()** metódusát.)

9. Egészítse ki a **ValóVilág** osztályt egy metódussal, amely igazat ad vissza, ha legalább két olyan celeb szerepel a show-ban, akik egy paraméterként megkapott értéknél nagyobb tiszteletdíjban részesülnek, különben hamisat! Hívja meg a metódust a főprogramban az előző feladatban létrehozott **ValóVilág** objektumra, és írjon ki egy üzenetet a képernyőre az eredménynek megfelelően!

10. Egészítse ki a **ValóVilág** osztályt egy metódussal, amely paraméterként megkap egy játékosokat tartalmazó kollekciót, és beszavazza közülük a show-ba azokat, akiknek a foglalkozása „rúdtáncos”! Hívja meg a metódust a főprogramban egy újonnan létrehozott, 2011-es évadú, játékosokat nem tartalmazó **ValóVilág** objektumra a 7. feladatban feltöltött tömbből készített kollekcióval! (Segítség: lásd az **Arrays** osztály **asList()** metódusát.)

11. Egészítse ki a **BeleValó** osztályt egy statikus metódussal, amely paraméterként megkap egy **Valóságshow** objektumot, és csupa nagybetűssé alakítja a show-ban szereplő celebek nevét! Hívja meg a metódust a főprogramban a 8. feladatban létrehozott **ValóVilág** objektummal, majd írja ki a játékosokat a `vv<évad>.txt` nevű állományba! (Segítség: lásd a **String** osztály **toUpperCase()** metódusát.)

12. Egészítse ki a **BeleValó** osztályt egy statikus metódussal, amely paraméterként megkap egy **Valóságshow** típusú referenciákat tartalmazó tömböt, és mindegyik show-ból „kiszavazza” a legidősebb játékost! Hívja meg a metódust a főprogramban a 8. és a 10. feladatban létrehozott **ValóVilág** objektumokból álló tömbbel, majd írja ki a képernyőre a két valóságshow játékosait! (Segítség: lásd a **Collections** osztály **max()** metódusát.)

Magas szintű programozási nyelvek beugró (levelező tagozat) **2011. december 17.**

1. Készítse el a **`Vonat`** osztályt a **`vasút`** csomagban! Minden vonatról tudni szeretnénk az azonosítószámát (sztring, mert a vonatszám betűket is tartalmazhat), a kiindulási és célállomását (sztringek), a menetidejét percben (egész), az út hosszát km-ben (egész) és hogy tehervonat-e (logikai). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **`HibásÉrtékException`** kivételt, ha a menetidő vagy az úthossz nem pozitív! Írja meg az egyes adattagokhoz tartozó lekérdező metódusokat! Definiálja felül az **`equals()`** metódust: két vonat akkor legyen egyenlő, ha az azonosítószámuk (tartalmilag) megegyezik! Egy vonat bármelyik másik vonattal (akár **`Személyvonat`** típusúval is) lehessen egyenlő!

2. Definiálja felül a **`toString()`** metódust úgy, hogy az a vonat adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<vonatszám>: <kiindulási állomás> - <célállomás>, <úthossz> km, menetidő: [<óra> óra ]<perc> perc`”! Ha a menetidő kevesebb, mint egy óra, akkor csak a percet kell megadni. Pl.: „6007: Debrecen - Budapest-Nyugati, 221 km, menetidő: 3 óra 17 perc”.

3. A vonatok természetes rendezettsége az átlagsebességük szerint legyen értelmezve! (Ne feledje, hogy ha `a` és `b` egészek, akkor `a/b` is egész, az átlagsebesség viszont valós!)

4. Származtasson a **`Vonat`** osztályból egy **`Személyvonat`** osztályt szintén a **`vasút`** csomagban! A személyvonat olyan vonat, amelynek a logikai adattagja (tehervonat-e) csak hamis értéket vehet fel, és a vonat tulajdonságain kívül még egy más osztályból nem látható, egész értékű férőhely tulajdonsággal is rendelkezik. Készítsen az osztályhoz egy konstruktort, amelynek segítségével – a logikain kívül – mindegyik adattagjának kezdőérték adható! A konstruktor dobja tovább a **`Vonat`** osztály konstruktora által dobott esetleges kivételt!

5. Definiálja felül a **`toString()`** metódust úgy, hogy az a személyvonat adatait a 2. feladatban leírt módon adja vissza, kiegészítve még a vonat átlagsebességével km/h-ban, 2 tizedesjegyre kerekítve, a példában látható formában! Pl.: „IC609: Nyíregyháza - Budapest-Nyugati, 270 km, menetidő: 3 óra 2 perc, átlagsebesség: 89.01 km/h”. (Segítség: lásd a **`Math.rint()`** vagy a **`String.format()`** metódust.)

6. Adott az alábbi interfész a **`vasút`** csomagban:

```
public interface Menetrend
{
  // új vonatot vesz fel a menetrendbe
  public void felvesz(Személyvonat v);

  // visszaadja azoknak a vonatoknak a listáját átlagsebesség szerint csökkenő
  // sorrendbe rendezve, amelyekkel közvetlenül el lehet jutni a honnan
  // állomásról a hova állomásra
  public java.util.List<Személyvonat> közvetlen(String honnan, String hova);

  // kiírja a megadott fájlba a vonatok listáját (minden vonatot külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **`MÁVMenetrend`** osztályt a **`máv`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`Menetrend`** interfészt! Figyeljen arra, hogy ugyanaz a vonat egy menetrendben csak egyszer fordulhat elő! Legyen lehetőség a menetrend érvényességi idejének (év, hónap, nap) tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a menetrend érvényességi idejét, valamint feltölthetjük azt egy vonatokat tartalmazó kollekcióban található személyvonatokkal! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a menetrend érvényességi idejét, majd egy üres sort követően soronként az egyes vonatokat adja vissza! Az osztály adattagjai nem lehetnek publikus láthatóságúak, és nem készülhet hozzájuk lekérdező metódus! (Segítség: lásd a **`Collections.sort()`** metódust.) (2 pont)

7. Helyezze el a főprogramot a **teszt** csomag **VonatTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található vonatokat egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy vonat adatait tartalmazza az alábbi formátumban:

   `<vonatszám>;<kiindulási állomás>;<célállomás>;<menetidő>;<úthossz>[;<férőhely>]`

   Ha a vonatszám T betűvel kezdődik, akkor tehervonatról van szó, és ekkor nincs megadva a férőhely. Kezelje az esetleges ellenőrzött kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()** metódusait, valamint a **String.charAt()** metódust.) (2 pont)

8. Hozzon létre a főprogramban egy **MÁVMenetrend** objektumot, amelynek érvényességi ideje az aktuális dátumnál egy évvel későbbi dátum legyen! Töltse fel a menetrendet az előző feladatban létrehozott listában található személyvonatokkal! Olvasson be a billentyűzetről két állomásnevet, és írja ki a képernyőre azoknak a vonatoknak a listáját átlagsebességük szerint csökkenő sorrendbe rendezve, amelyekkel közvetlenül eljuthatunk az egyik állomásból a másikba vagy fordítva! (Segítség: lásd a **Collection.addAll()**, a **LocalDate.now()** és a **LocalDate.plusYears()** metódusokat.)

9. Egészítse ki a **MÁVMenetrend** osztályt egy metódussal, amely visszaadja azoknak a menetrendben szereplő vonatoknak egy kollekcióját, amelyek egy paraméterként megkapott valós értéknél kisebb átlagsebességgel rendelkeznek km/h-ban mérve! Hívja meg a metódust a főprogramban a 8. feladatban létrehozott **MÁVMenetrend** objektumra egy billentyűzetről bekért értékkel, és írja ki a kapott kollekciót a képernyőre!

10. Egészítse ki a **VonatTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy vonatokat tartalmazó kollekciót, és képernyőre írja a kollekcióban található vonatok közül a leghosszabb menetidővel és a leghosszabb úttal rendelkező tehervonat kiindulási és célállomását! Ha több maximális menetidővel vagy maximális úthosszal rendelkező tehervonat is van a kollekcióban, akkor ezek közül mindegyik kiindulási és célállomását ki kell írni. Hívja meg a metódust a főprogramban a 7. feladatban létrehozott listával!

11. Egészítse ki a **VonatTeszt** osztályt egy statikus metódussal, amely első paramétereként megkap egy **Menetrend** típusú referenciákat tartalmazó tömböt, és kiírja egy-egy állományba az egyes menetrendek vonatlistáját! Az állományok neve `<név><szám>.txt` alakú legyen, ahol a `<név>` értékét a metódus második paramétere adja meg, a `<szám>` pedig 1-ről indul és menetrendenként egyesével növekszik! Hívja meg a metódust a főprogramban a 8. feladatban létrehozott **MÁVMenetrend** objektumot tartalmazó egyelemű tömbbel és a „lista” névvel!

Magas szintű programozási nyelvek beugró (levelező tagozat) **2012. január 7.**

1. Készítse el a **Ház** osztályt a **lakás** csomagban! Minden házról tudni szeretnénk a címét (sztring), az alapterületét m^2-ben (egész), az emeleteinek számát (egész) és hogy kertes ház-e (logikai). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Ha a konstruktor 1-nél kisebb emeletszámot kap paraméterül, akkor a létrejövő ház legyen egyemeletes! Írja meg az emeletszámhoz tartozó lekérdező metódust! Definiálja felül az **equals()** metódust: két ház akkor legyen egyenlő, ha a címük (tartalmilag) megegyezik! Egy ház bármelyik másik házzal (akár **Toronyház** típusúval is) legyen összehasonlítható!
2. Definiálja felül a **toString()** metódust úgy, hogy az a ház adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`Egy <alapterület> m2 alapterületű <emelet> emeletes [kertes ]ház található itt: <cím>`” (pl. „*Egy 120 m2 alapterületű 2 emeletes kertes ház található itt: Aprajafalva, Északi fasor 12.*”)!
3. A házak természetes rendezettsége a „térfogatuk” (az alapterületük és az emeletszámuk szorzata) szerint legyen értelmezve!
4. Származtasson a **Ház** osztályból egy **Toronyház** osztályt szintén a **lakás** csomagban! A toronyház olyan ház, amely a ház tulajdonságain kívül még egy bárhonnan látható, egész értékű „emeletkorlát” osztály szintű tulajdonsággal is rendelkezik. Az emeletkorlát azt mondja meg, hogy legalább hány emeletesnek kell lennie egy toronyháznak. Az alapértelmezett értéke legyen 20! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik példány szintű adattagjának kezdőérték adható! A konstruktor dobjon **IllegalArgumentException** kivételt „*Egy toronyház legalább <emeletkorlát> emeletes!*” üzenettel, ha a paramétereként megkapott emeletszám nem éri el az emeletkorlát értékét!
5. Definiálja felül a **toString()** metódust úgy, hogy az a toronyház adatait a 2. feladatban leírt módon adja vissza, kicserélve benne a „`ház`” szót „`toronyház`”-ra (pl. „*Egy 540 m2 alapterületű 24 emeletes toronyház található itt: Középfölde, Keleti dűlő 4.*”)!
6. Adott az alábbi interfész a **lakás** csomagban:

```
public interface Háztömb
{
  // új házat vesz fel a háztömbbe
  public void felvesz(Ház ház);

  // visszaadja azoknak a toronyházaknak a listáját térfogat szerint csökkenő
  // sorrendbe rendezve, amelyek a paraméternél több emelettel rendelkeznek
  public java.util.List<Toronyház> magasToronyházak(int emelet);

  // törli a háztömbből azokat a toronyházakat, amelyeknek az emeletszáma nem éri
  // el a Toronyház emeletkorlát értékét
  public void törölKisToronyházak();

  // kiírja a megadott állományba a házak listáját (minden házat külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **Lakópark** osztályt a **napfény** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Háztömb** interfészt! Figyeljen arra, hogy ugyanaz a ház egy háztömbben csak egyszer fordulhat elő! Legyen lehetőség a lakópark nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével nevet adhatunk a lakóparknak, valamint feltölthetjük azt egy házakat tartalmazó tömb elemeivel! A **kiír()** metódust úgy implementálja, hogy az állomány első sorába a lakópark neve kerüljön! (Segítség: lásd a **Collections.sort()** metódust.) (2 pont)

7. Helyezze el a főprogramot a **teszt** csomag **LakásTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található házakat egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy ház adatait tartalmazza az alábbi formátumban:

   ```
   [T|H];<cím>;<alapterület>;<emeletszám>;[kertes|nem kertes]
   ```

   Például:

   ```
   H;Aprajafalva, Északi fasor 12.;120;2;kertes
   T;Középfölde, Keleti dűlő 4.;540;24;nem kertes
   ```

   Ha a sor egy T betűvel kezdődik, akkor toronyházról, ha H-val, akkor egyszerű házról van szó. Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()** metódusait, valamint a **String.charAt()** metódust.) (2 pont)

8. Hozzon létre a főprogramban egy **Lakópark** objektumot! A lakópark neve legyen a program második parancssori argumentuma, vagy ha az nem létezik, akkor legyen „Napfény lakópark”! Töltse fel a lakóparkot az előző feladatban létrehozott listában található házakkal! Olvasson be a billentyűzetről egy emeletszámot, és írja ki a képernyőre azoknak a toronyházaknak a listáját térfogat szerint csökkenő sorrendbe rendezve, amelyek a beolvasott értéknél több emelettel rendelkeznek! (Segítség: lásd a **Collection.toArray()** metódust.)

9. Csökkentse 5-tel a főprogramban a **Toronyház** osztály emeletkorlát tulajdonságának az értékét, majd törölje az előző feladatban létrehozott lakóparkból azokat a toronyházakat, amelyeknek az emeletszáma nem éri el ezt az új értéket!

10. Egészítse ki a **LakásTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy házakat tartalmazó kollekciót, valamint egy valós értékű, méterben megadott belmagasságot, és képernyőre írja soronként a kollekcióban található házak jellegét (egyszerű ház vagy toronyház), címét és m^3-ben mért térfogatát! Feltehetjük, hogy minden ház minden emelete a paraméterként kapott belmagassággal rendelkezik, és hogy a térfogathoz semmilyen nem ismert tényező (pl. a közfalak mérete) nem járul hozzá. A főprogramban kérjen be a billentyűzetről egy belmagasságértéket, majd hívja meg a metódust a 7. feladatban létrehozott listával és a beolvasott értékkel!

11. Egészítse ki a **LakásTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **Háztömb** típusú referenciákat tartalmazó tömböt, és kiírja egy-egy állományba az egyes háztömböket alkotó házak listáját! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és háztömbönként egyesével növekszik! Hívja meg a metódust a főprogramban a 8. feladatban létrehozott **Lakópark** objektumot tartalmazó egyelemű tömbbel!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2013. január 11.**

1. Készítse el a **Cipő** osztályt a **lábbeli** csomagban! A cipőkről tudni szeretnénk a márkájukat (sztring), a méretüket (valós), a fajtájukat (sztring) és hogy magas sarkúak-e (logikai). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **IllegalArgumentException** kivételt „*A cipőméret csak pozitív lehet!*” üzenettel, ha a paramétereként megkapott méret nem pozitív! Írja meg a mérethez tartozó lekérdező metódust! Definiálja felül az **equals()** metódust: két cipő akkor legyen egyenlő, ha minden adatuk (tartalmilag) megegyezik! Ügyeljen arra, hogy egy **Cipő** típusú objektum nem lehet egyenlő egy **Csizma** típusú objektummal, mint ahogy fordítva sem!

2. Definiálja felül a **toString()** metódust úgy, hogy az a cipő adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<márka> márkájú [magas sarkú ]<fajta>, mérete: <méret>`” (pl. „*Tisza márkájú sportcipő, mérete: 42.5*” vagy „*Celeste márkájú magas sarkú báli cipő, mérete: 38.0*”)!

3. A cipők természetes rendezettsége a méretük, azon belül a fajtájuk, azon belül pedig a márkájuk szerint legyen értelmezve!

4. Származtasson a **Cipő** osztályból egy **Csizma** osztályt szintén a **lábbeli** csomagban! A csizma a cipő tulajdonságain kívül még egy más osztályból nem látható tulajdonsággal is rendelkezik, amely azt mondja meg, hogy bélelt-e a csizma. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Definiálja felül a **toString()** metódust úgy, hogy az a cipő adataihoz fűzze hozzá, hogy „`, [nem ]bélelt`” (pl. „*Puma márkájú hótaposó csizma, mérete: 42.5, bélelt*”)!

5. Hozza létre a **CipőTétel** osztályt a **lábbeli** csomagban, amellyel egy raktárban lévő cipőket és darabszámukat szeretnénk modellezni! Ennek megfelelően legyen egy **Cipő** típusú és egy egész értékű adattagja, amelyek más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amellyel mindkét adattagjának kezdőérték adható! Írja meg a két adattaghoz tartozó lekérdező metódust! Készítsen egy olyan metódust, amellyel megnövelhető a darabszám egy paraméterként kapott értékkel! Két tétel akkor legyen egyenlő, ha a bennük tárolt cipők egyenlők! A tételek sztring reprezentációja a cipőt és a darabszám értékét is tartalmazza! (2 pont)

6. Adott az alábbi interfész a **lábbeli** csomagban:

```
public interface CipőRaktár
{
  // megadott mennyiségű cipőt vesz fel a raktárba; ha már volt ilyen cipő,
  // növeli a darabszámát, ha még nem volt, felveszi új tételként
  public void felvesz(Cipő cipő, int darab);

  // visszaadja azoknak a csizmáknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyekből van raktáron legalább a paraméterben
  // megadott darabszámú
  public java.util.List<Csizma> csizmaLista(int darab);

  // kiírja a megadott állományba a tételek listáját (minden tételt külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **Cipőbolt** osztályt az **áruház** csomagban, amely a megjegyzéseknek megfelelően implementálja a **CipőRaktár** interfészt! Figyeljen arra, hogy egy cipőboltban nem fordulhat elő két tételben ugyanaz a cipő! Legyen lehetőség a cipőbolt nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével nevet adhatunk a cipőboltnak, valamint feltölthetjük azt egy cipőket tartalmazó tömb elemeivel! Vigyázzon arra, hogy a tömbben ugyanaz a cipő többször is szerepelhet (ilyenkor annyi darab lesz raktáron, ahányszor a tömbben előfordul)! A **kiír()** metódust úgy implementálja, hogy az állomány első sorába a cipőbolt neve kerüljön! (Segítség: lásd a **List.indexOf()**, a **List.get()** és a **Collections.sort()** metódusokat.) (2 pont)

7. Helyezze el a főprogramot a **teszt** csomag **CipőTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található cipőket egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy cipő adatait tartalmazza az alábbi formátumban:

   ```
   [1|2];<márka>;<méret>;<fajta>;[magas sarkú|lapos sarkú][;[bélelt|nem bélelt]]
   ```

   Ha a sor 1-essel kezdődik, akkor cipőről, ha 2-essel, akkor csizmáról van szó. Egyszerű cipő esetén nem szerepel a béleltségre vonatkozó információ. Például:

   ```
   1;Celeste;38;báli cipő;magas sarkú
   2;Puma;42,5;hótaposó csizma;lapos sarkú;bélelt
   ```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()**, **nextDouble()** metódusait.) (2 pont)

8. Hozzon létre a főprogramban egy **Cipőbolt** objektumot! A cipőbolt neve legyen a program második parancssori argumentuma, vagy ha az nem létezik, akkor legyen „Ázsia Center”! Töltse fel a cipőboltot az előző feladatban létrehozott listában található cipőkkel! Olvasson be a billentyűzetről egy darabszámot, és írja ki a képernyőre azoknak a csizmáknak a rendezett listáját (a természetes rendezettségük sorrendjében), amelyekből legalább a beolvasott darab van raktáron a cipőboltban! (Segítség: lásd a **Collection.toArray()** metódust.)

9. Egészítse ki a **CipőTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy cipőket tartalmazó kollekciót, valamint egy valós értékű cipőméretet, és képernyőre írja soronként a kollekcióban található, a megkapott mérettel rendelkező cipők jellegét (egyszerű cipő vagy csizma) és egyéb adatait! A főprogramban kérjen be a billentyűzetről egy cipőméretet, majd hívja meg a metódust a 7. feladatban létrehozott listával és a beolvasott értékkel!

10. Egészítse ki a **CipőTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **CipőRaktár** típusú referenciákat tartalmazó tömböt, és kiírja egy-egy állományba az egyes cipőraktárakban lévő tételek listáját! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és raktáranként egyesével növekszik! Hívja meg a metódust a főprogramban a 8. feladatban létrehozott **Cipőbolt** objektumot tartalmazó egyelemű tömbbel!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2013. január 13.**

1. Készítse el a **Repülőgép** osztályt a **repülés** csomagban! A repülőgépekről tudni szeretnénk a gyártójukat (sztring), a típusukat (sztring), a hosszukat méterben (valós) és hogy sugárhajtásúak-e (logikai). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **IllegalArgumentException** kivételt „*A hossz csak pozitív lehet!*” üzenettel, ha a paramétereként megkapott hossz nem pozitív! Írja meg a gyártó és a logikai adattag lekérdező metódusát! Definiálja felül az **equals()** metódust: két repülőgép akkor legyen egyenlő, ha a gyártójuk és típusuk (tartalmilag) megegyezik! Az összehasonlítás ne különböztesse meg a kis- és nagybetűket! Egy repülőgép bármelyik másik repülőgéppel (akár **Utasszállító** típusúval is) legyen összehasonlítható! (Segítség: lásd a **String.equalsIgnoreCase()** metódust.)
2. Definiálja felül a **toString()** metódust úgy, hogy az a repülőgép adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<gyártó> <típus> [sugárhajtású ]repülőgép, hossza: <hossz> m`” (pl. „*Airbus A380-800 sugárhajtású repülőgép, hossza: 72.72 m*”)!
3. A repülőgépek természetes rendezettsége a gyártójuk, azon belül pedig a típusuk szerint legyen értelmezve! Az összehasonlítás ne különböztesse meg a kis- és nagybetűket! (Segítség: lásd a **String.compareToIgnoreCase()** metódust.)
4. Származtasson a **Repülőgép** osztályból egy **Utasszállító** osztályt szintén a **repülés** csomagban! Az utasszállítók a repülőgép tulajdonságain kívül még egy más osztályból nem látható, egész értékű férőhely tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg a férőhely lekérdező és beállító metódusait! Definiálja felül a **toString()** metódust úgy, hogy az a repülőgép adataihoz fűzze hozzá, hogy „`, <férőhely> férőhelyes`” (pl. „*Boeing 747-8I sugárhajtású repülőgép, hossza: 76.25 m, 467 férőhelyes*”)!
5. Adott az alábbi interfész a **repülés** csomagban:

```
public interface Flotta
{
  // új repülőgépet vesz fel a flottába
  public void felvesz(Repülőgép repülő);

  // visszaadja azoknak az utasszállítóknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyek legalább a paraméterben megkapott számú
  // férőhellyel rendelkeznek
  public java.util.List<Utasszállító> megfelelőGépek(int utasszám);

  // kiírja a megadott állományba a repülőgépek listáját (mindegyiket külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **Légitársaság** osztályt a **légiKözlekedés** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Flotta** interfészt! Figyeljen arra, hogy egy légitársaság több azonos márkájú és típusú repülőgéppel is rendelkezhet! Legyen lehetőség a légitársaság nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével nevet adhatunk a légitársaságnak, valamint feltölthetjük a flottáját egy repülőgépeket tartalmazó tömb elemeivel! A **kiír()** metódust úgy implementálja, hogy az állomány első sorába a légitársaság neve kerüljön! (Segítség: lásd a **Collections.sort()** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **RepülőTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található repülőgépeket egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy repülőgép adatait tartalmazza az alábbi formátumban:

   ```
   [R|U];<gyártó>;<típus>;<hossz>;[S|N][;<férőhely>]
   ```

   Ha a sor R-rel kezdődik, akkor egyszerű repülőgépről, ha U-val, akkor utasszállítóról van szó. A férőhely csak utasszállítók esetén van megadva. Az S betű azt jelenti, hogy a repülőgép sugárhajtású, az N pedig, hogy nem. Például:

   ```
   R;Boeing;747-8F;76,25;S
   U;Bombardier;Q400;32,81;N;78
   ```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()**, **nextDouble()** metódusait, valamint a **String.charAt()** metódust.) (2 pont)

7. Hozzon létre a főprogramban egy **Légitársaság** objektumot! A légitársaság neve legyen a program második parancssori argumentuma, vagy ha az nem létezik, akkor legyen „Unideb Airlines”! Töltse fel a légitársaság flottáját az előző feladatban létrehozott listában található repülőgépekkel! Olvasson be a billentyűzetről egy darabszámot, és írja ki a képernyőre azoknak az utasszállítóknak a rendezett listáját (a természetes rendezettségük sorrendjében), amelyek legalább a beolvasott darabszámú férőhellyel rendelkeznek! (Segítség: lásd a **Collection.toArray()** metódust.)

8. Egészítse ki a **RepülőTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy repülőgépeket tartalmazó kollekciót, valamint egy gyártót, és képernyőre írja soronként azoknak a kollekcióban található repülőgépeknek a jellegét (egyszerű repülőgép vagy utasszállító) és egyéb adatait, amelyeknek a gyártója a kis- és nagybetűktől eltekintve megegyezik a kapott gyártóval! A főprogramban kérjen be a billentyűzetről egy gyártót, majd hívja meg a metódust a 6. feladatban létrehozott listával és a beolvasott értékkel!

9. Egészítse ki a **Légitársaság** osztályt egy olyan metódussal, amely képernyőre írja azokat a repülőgépeket, amilyen gyártmányúból a légitársaság csak egyszerű repülőgéppel rendelkezik, utasszállítóval nem! Más szavakkal: a metódusnak ki kell listáznia a légitársaság azon repülőgépeit, amelyek nem utasszállítók, és amelyeknek a gyártójától a légitársaságnak nincs utasszállítója. A metódus a gyártók összehasonlítása során ne különböztesse meg a kis- és nagybetűket! Tesztelje a metódust a főprogramban a 7. feladatban létrehozott légitársasággal!

10. Egészítse ki a **RepülőTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **Flotta** objektumokat tartalmazó tömböt, és kiírja egy-egy állományba az egyes flottákban lévő repülőgépek listáját! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és flottánként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **Légitársaság** objektumot tartalmazó egyelemű tömbbel!

11. Egészítse ki a **RepülőTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy légitársaságokat tartalmazó kollekciót, valamint egy darabszámot, és megnöveli a kapott darabszámmal a társaságok Airbus gyártmányú utasszállító repülőgépeinek a férőhelyét (az „Airbus” szóban a kis- és nagybetűket ne különböztessük meg)! A metódus írja ki a képernyőre a módosításban érintett légitársaságok nevét (azaz azokét, amelyek rendelkeznek Airbus gyártmányú utasszállítóval), de mindegyiket csak egyszer! Ha szükséges, egészítse ki a **Légitársaság** osztályt új metódusokkal!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2013. január 14.**

1. Készítse el a **`Termék`** absztrakt osztályt a **`menza`** csomagban! A termékek számunkra lényeges tulajdonságai a megnevezésük (sztring), a mennyiségi egységük (sztring) és az egységáruk (egész). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **`equals()`** metódust: két termék akkor legyen egyenlő, ha a nevük és a mennyiségi egységük (tartalmilag) megegyezik! Definiálja felül a **`toString()`** metódust úgy, hogy az a termék adatait egymástól vesszővel elválasztva jelenítse meg, és az egységár után szerepeljen egy szóköz és a „Ft” szócska!
2. A termékek természetes rendezettsége a megnevezésük, azon belül pedig a mennyiségi egységük szerint legyen értelmezve!
3. Származtasson a **`Termék`** osztályból egy **`Étel`** és egy **`Ital`** osztályt szintén a **`menza`** csomagban! Az egyes ételekről azt is tudni szeretnénk, hogy leves-e vagy főétel, az italokról pedig azt, hogy alkoholosak-e. Az új adattagok más osztályból ne látszódjanak, de írjuk meg a lekérdező metódusaikat! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével mindegyik adattagjuknak kezdőérték adható!
4. Adott az alábbi interfész a **`menza`** csomagban:

```
public interface Étlap
{
  // új ételt vagy italt vesz fel az étlapra;
  // ha már létezett a termék, nem csinál semmit
  public void felvesz(Termék termék);

  // törli az étlapról a megadott ételt vagy italt;
  // ha ilyen nem létezik, kivételt dob
  public void töröl(Termék termék) throws NincsIlyenTermékException;

  // visszaadja az étlapon szereplő, megadott fajtájú termékek számát;
  // a fajta lehet: 1 – leves, 2 – főétel, 3 – alkoholos ital, 4 – alkoholmentes
  // ital, minden más esetben IllegalArgumentException kivételt dob
  public int termékekSzáma(int fajta);

  // kiírja a megadott állományba soronként az étlapon szereplő termékeket:
  // először az ételeket, majd az italokat, mindkét listát a természetes
  // rendezettségnek megfelelő sorrendben, a két lista között egy üres sorral
  public void kiír(String fájlnév);
}
```

   Készítse el a **`Menü`** osztályt az **`étterem`** csomagban, amely a megjegyzéseknek megfelelően implementálja az **`Étlap`** interfészt! Figyeljen arra, hogy egy menüben nem fordulhat elő kétszer ugyanaz a termék! Legyen lehetőség a menü dátumának (év, hónap, nap) tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a menü dátumát, valamint feltölthetjük azt egy termékeket tartalmazó kollekció elemeivel! Ügyeljen arra, hogy a megkapott kollekciót a **`Menü`** objektumok a későbbiekben ne tudják módosítani! Definiálja felül a **`toString()`** metódust úgy, hogy az a menü dátumát adja vissza „*éééé.hh.nn*” formában (a hónap és a nap mindig kétjegyű legyen)! A **`kiír()`** metódust úgy implementálja, hogy az állomány első sorába a menü dátuma kerüljön! (Segítség: lásd a **`Collection`** interfész **`add()`** és **`remove()`** metódusait, illetve a **`Collections.sort()`** metódust.) (3 pont)
5. Egészítse ki a **`Menü`** osztályt egy olyan metódussal, amely visszaadja egy tömbben azokat az ételeket, amelyeknek a mennyiségi egysége egy paraméterként megkapott érték! (Segítség: lásd a **`Collection.toArray()`** metódust.)

6. Helyezze el a főprogramot a **teszt** csomag **MenüTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található termékeket egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy termék adatait tartalmazza az alábbi formátumban:

   ```
   [étel|ital];<név>/<egység>/<egységár>/[leves|főétel|alkoholos|mentes]
   ```

   A sor első szava mondja meg, hogy ételről vagy italról van-e szó. A sor végén álló adat étel esetén „leves” vagy „főétel”, ital esetén „alkoholos” vagy „mentes” lehet. Például:

   ```
   étel;Bécsi szelet/adag/550/főétel
   ital;buborékos ásványvíz/0,5 l/80/mentes
   ```

   (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()** metódusait, illetve a **String.substring()** metódust.) (2 pont)

7. Hozzon létre a főprogramban egy **Menü** objektumot! A menü dátuma legyen az éppen aktuális dátum! Töltse fel a menüt az előző feladatban beolvasott termékekkel! Olvasson be a billentyűzetről egy állománynevet, és írja ki a menüt a megadott nevű állományba! (Segítség: lásd a **Calendar** osztály **getInstance()** és **get()** metódusait.)

8. Törölje a főprogramban az előző feladatban létrehozott menüből azokat az ételeket, amelyeknek a mennyiségi egysége „adag”! A megoldáshoz használja fel a **töröl()** metódust és az 5. feladatban megírt metódust!

9. Egészítse ki a **MenüTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy termékeket tartalmazó kollekciót, egyesével képernyőre írja a kollekcióban található termékeket, bekér hozzájuk egy-egy valós értékű mennyiséget, végül kiírja, hogy mennyibe kerülnek összesen a kiírt termékek a megadott mennyiségekben! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával!

10. Egészítse ki a **MenüTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **Menü** típusú objektumokat tartalmazó tömböt, és képernyőre írja azoknak a menüknek a dátumát, amelyekből összeállítható egy ebéd, azaz amelyekben van legalább egy leves, egy főétel és egy alkoholmentes ital! A megoldáshoz használja fel a **termékekSzáma()** metódust! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **Menü** objektumot tartalmazó egyelemű tömbbel!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2013. december 21.**

1. Készítse el az **Egység** absztrakt osztályt a **vendéglátás** csomagban, amellyel vendéglátóipari egységeket szeretnénk modellezni! A vendéglátóipari egységekről tárolni szeretnénk a megnevezésüket (sztring), hogy hány férőhelyesek (egész) és hogy van-e lehetőség dohányozni (logikai). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait, valamint a logikai adattag beállító metódusát! Definiálja felül az **equals()** metódust: két egység akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik! Ügyeljen arra, hogy a különböző fajta vendéglátóipari egységek nem lehetnek egyenlők egymással még akkor sem, ha a nevük megegyezik (pl. a „Kék Duna" nevű étterem nem egyenlő a „Kék Duna" nevű kocsmával)! Definiálja felül a **toString()** metódust úgy, hogy az az egység adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név>, <férőhely> férőhelyes, [nem]dohányzó`" (pl. „*Aranykakas Vendéglő, 45 férőhelyes, nemdohányzó*")!
2. Egészítse ki az **Egység** osztályt egy olyan metódussal, amelynek segítségével egy egész értékkel növelhető (vagy – negatív érték esetén – csökkenthető) a férőhelyek száma! Ha az új érték egy küszöbértéknél kisebb lenne, akkor a férőhelyek száma ne változzon, és a metódus dobjon **TúlKevésFérőhelyException** kivételt! A küszöbértéket egy publikus, osztály szintű adattagban tárolja, amelynek alapértelmezett értéke legyen 10!
3. Az egységek természetes rendezettsége a férőhelyük szerint csökkenő, azon belül pedig a megnevezésük szerint növekvő sorrendben legyen értelmezve!
4. Származtasson az **Egység** osztályból egy **Étterem** és egy **Kocsma** osztályt szintén a **vendéglátás** csomagban! Az éttermek esetén étlapot, a kocsmák esetén pedig itallapot szeretnénk még tárolni, mindkettőt egyszerű sztringek halmazaként (azaz sem az étlapokon, sem az itallapokon nem lehet két egyforma megnevezésű étel, illetve ital). Az új adattagok csak a **vendéglátás** csomag eszközeiből látszódjanak! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével a dohányzási lehetőség kivételével mindegyik adattagjuknak kezdőérték adható! Létrehozásukkor a kocsmákban legyen, az éttermekben viszont ne legyen dohányzási lehetőség!
5. Adott az alábbi, vendéglátóipari egységek nyilvántartását segítő interfész a **vendéglátás** csomagban:

```
public interface EgységAdatbázis
{
  // visszaadja az adatbázis azon egységeinek listáját a természetes
  // rendezettségük sorrendjében, amelyek legalább a paraméterben megadott számú
  // férőhellyel rendelkeznek
  public java.util.List<Egység> nagyEgységek(int férőhely);

  // visszaadja az adatbázis azon egységeinek listáját a természetes
  // rendezettségük sorrendjében, amelyek legalább az első paraméterben megadott
  // számú férőhellyel rendelkeznek, és a második paraméter szerint van vagy nincs
  // bennük dohányzási lehetőség
  public java.util.List<Egység> nagyEgységek(int férőhely, boolean dohányzó);

  // kibővíti az első paraméterben megadott nevű egységek férőhelyeit a második
  // paraméterben megadott értékkel; ha egy egység bővítése kivételt eredményez,
  // akkor az az egység ne változzon
  public void bővít(String egység, int darab);
}
```

Készítse el az **Étteremlánc** osztályt az **ipar** csomagban, amely a megjegyzéseknek megfelelően implementálja az **EgységAdatbázis** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy étteremlánc csak éttermeket tartalmazhat, de lehetnek benne azonos nevű éttermek is! Legyen lehetőség az étteremlánc nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk az étteremlánc nevét, valamint feltölthetjük azt egy vendéglátóipari egységeket tartalmazó kollekcióban szereplő éttermekkel! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban az étteremlánc nevét, majd egy üres sort követően soronként az egyes éttermeket adja vissza! (Segítség: lásd a **Collections.sort()** metódust.)

(2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található vendéglátóipari egységeket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy vendéglátóipari egység adatait tartalmazza az alábbi formátumban:

   ```
   [E|K];<név>;<férőhely>;[<termék>[/<termék>]...]
   ```

   Ha a sor első karaktere E, akkor étteremről, ha K, akkor kocsmáról van szó. Az utolsó pontosvesszőt követően nulla vagy több termék (étel, illetve ital) megnevezése található, egymástól / jellel elválasztva. Például:

   ```
   E;Aranykakas Vendéglő;45;Erőleves/Bécsi szelet/Dobostorta
   K;Budai Presszó;28;Sör/Bor/Pálinka
   K;Józsi bácsi kiskocsmája;4;
   ```

   (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()**, **hasNext()** metódusait, illetve a **String.charAt()** metódust.) (2 pont)

7. Hozzon létre a főprogramban egy **Étteremlánc** objektumot! Az étteremlánc neve legyen a második parancssori argumentum, vagy ennek hiányában „UniDeb Gourmet”! Töltse fel az étteremláncot az előző feladatban beolvasott éttermekkel! Olvasson be a billentyűzetről egy egész értéket, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében az étteremlánc azon éttermeit, amelyek legalább a beolvasott darabszámú férőhellyel rendelkeznek!

8. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy vendéglátóipari egységeket tartalmazó kollekciót, és visszaadja a kollekcióban található vendéglátóipari egységek férőhelyeinek az átlagát valós számként! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával, módosítsa a férőhelyek küszöbértékét a metódus által visszaadott átlag csonkított értékére, végül növelje meg az előző feladatban létrehozott étteremlánc „Ezüstkancsó” nevű éttermeinek férőhelyeit egy billentyűzetről beolvasott értékkel! Ellenőrizze a művelet eredményét az étteremlánc kiírásával!

9. Egészítse ki az **Egység** osztályt egy publikus logikai metódussal, amely eldönti, hogy a paramétereként megkapott termék (étterem esetén étel, kocsma esetén ital) szerepel-e az adott vendéglátóipari egység kínálatában! Egészítse ki az **Étteremlánc** osztályt is egy olyan metódussal, amely az első paramétereként megkapott nevű szöveges állományba soronként kiírja azokat az éttermeket, amelyekben kapható a második paramétereként megkapott nevű étel! A metódus kezelje az ellenőrzött kivételeket! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott étteremláncra `lista.txt` állománynévvel és egy billentyűzetről beolvasott ételnévvel!

10. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy vendéglátóipari egységeket tartalmazó kollekciót, amelyben minden 10 főnél kevesebb férőhellyel rendelkező éttermet dohányzásra alkalmassá alakít, illetve minden 10 főnél több férőhellyel rendelkező kocsmát nemdohányzóvá alakít! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával, majd ellenőrzésképpen írja ki a képernyőre soronként a lista elemeit!

11. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely első paraméterként megkap egy **Étteremlánc** típusú objektumokat tartalmazó tömböt, és képernyőre írja azokat az étteremláncokat, melyekben van legalább két olyan étterem, amely legalább a második paraméterben megadott számú férőhellyel rendelkezik, és van benne dohányzási lehetőség! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **Étteremlánc** objektumot tartalmazó egyelemű tömbbel és a férőhelyek küszöbértékének 10-zel való osztása után kapott maradékkal!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2014. január 11.**

1. Készítse el a **Növény** osztályt a **botanika** csomagban! A növényekről tárolni szeretnénk a megnevezésüket (sztring), a színeiket (sztringtömb) és hogy ehetők-e (logikai). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható, méghozzá a színek esetén egy sztringeket tartalmazó kollekcióval! A konstruktor dobjon **NincsSzínException** kivételt, ha a kollekció üres, vagy egyáltalán nincs megadva (azaz a paraméter **null**)! Írja meg az adattagok lekérdező metódusait, valamint a megnevezés beállító metódusát! Definiálja felül az **equals()** metódust: két növény akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik! Ügyeljen arra, hogy a különböző fajta növények nem lehetnek egyenlők egymással még akkor sem, ha a nevük megegyezik (pl. a „paradicsom" nevű zöldség nem egyenlő a „paradicsom" nevű gyümölccsel)! (Segítség: lásd a **Collection.toArray()** metódust.)
2. Definiálja felül a **toString()** metódust úgy, hogy az a növény adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név> ([egyszínű|többszínű]), [nem ]ehető`" (pl. „*paradicsom (egyszínű), ehető*")!
3. A növények természetes rendezettsége az „ehetőségük" (először szerepeljenek az ehető növények), azon belül pedig a megnevezésük szerint növekvő sorrendben legyen értelmezve!
4. Származtasson a **Növény** osztályból egy **Zöldség** és egy **Gyümölcs** osztályt szintén a **botanika** csomagban! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével az „ehetőség" kivételével mindegyik adattagjuknak kezdőérték adható (mindkét osztály objektumai legyenek ehetők)! Definiálja felül a két osztály **toString()** metódusait úgy, hogy a 2. feladatban leírtaknak megfelelően működjenek, de a megnevezés mögött egy szóközzel elválasztva szerepeljen a „zöldség", illetve a „gyümölcs" szó is (pl. „*paradicsom gyümölcs (egyszínű), ehető*")!
5. Adott az alábbi interfész a **botanika** csomagban:

```
public interface Növénytár
{
  // visszaadja a növénytárban szereplő olyan növények listáját a természetes
  // rendezettségük sorrendjében, amelyek színei között szerepel a paraméterként
  // megkapott szín
  public java.util.List<Növény> adottSzínűNövények(String szín);

  // visszaadja a növénytárban szereplő olyan növények listáját a természetes
  // rendezettségük sorrendjében, amelyek színei között szerepel az első
  // paraméterként megkapott szín, és ha a második paraméter igaz, akkor zöldségek,
  // különben gyümölcsök
  public java.util.List<Növény> adottSzínűNövények(String szín, boolean zöldség);

  // kiírja a megadott nevű állományba a növények listáját (mindegyiket külön
  // sorba), először a nem ehetőket, aztán az ehetőket, név szerint növekvő
  // sorrendben
  public void kiír(String fájlnév);
}
```

Készítse el a **Zöldséges** osztályt a **piac** csomagban, amely a megjegyzéseknek megfelelően implementálja a **Növénytár** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat és lekérdező metódusokat! Figyeljen arra, hogy egy zöldségesnél csak zöldségek és gyümölcsök kaphatók (más növények nem), amelyek azonban lehetnek azonos nevűek is! Legyen lehetőség a zöldséges címének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a zöldséges címét, valamint feltölthetjük azt egy növényeket tartalmazó kollekcióban szereplő zöldségekkel és gyümölcsökkel! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban a zöldséges címét, majd egy üres sort követően soronként az egyes termékeket adja vissza! (Segítség: lásd a **Collections.sort()** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található növényeket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy növény adatait tartalmazza az alábbi formátumban:

   `[0|1|2];<név>;[<szín>[/<szín>]...]`

   Ha a sor első karaktere 0, akkor egyszerű növényről, ha 1, akkor zöldségről, ha 2, akkor gyümölcsről van szó. Az egyszerű növényekről feltételezzük, hogy nem ehetők. Az utolsó pontosvesszőt követően nulla vagy több szín megnevezése található, egymástól / jellel elválasztva. Például:

   ```
   0;ezüstfenyő;ezüst/barna
   1;petrezselyem;zöld
   2;paradicsom;piros
   0;színtelen növény;
   ```

   Ha egy növény színei nincsenek megadva, akkor az a növény ne kerüljön eltárolásra, de a beolvasás ne álljon meg! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()**, **nextInt()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **Zöldséges** objektumot! A zöldséges címe legyen a második parancssori argumentum, vagy ennek hiányában „Debreceni Kispiac”! Töltse fel a zöldséges kínálatát az előző feladatban beolvasott zöldségekkel és gyümölcsökkel! Olvasson be a billentyűzetről egy színt, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a zöldségesnél kapható azon gyümölcsöket, amelyek színei között szerepel a beolvasott szín!

8. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy növényeket tartalmazó kollekciót, és visszaadja a kollekcióban található növények átlagos színszámát valós számként! Felhasználva ezt a metódust, írja ki a képernyőre soronként a 6. feladatban létrehozott listában szereplő azon növényeket, amelyek az átlagosnál több színnel rendelkeznek!

9. Egészítse ki a **Zöldséges** osztályt egy logikai metódussal, amely csupa nagybetűssé alakítja a zöldséges kínálatában azokat a termékneveket, amelyek egy paraméterként megkapott karakterrel kezdődnek, és igazat ad vissza, ha volt módosított termék! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott zöldségesre egy billentyűzetről beolvasott karakterrel! Ellenőrzésképpen írja ki a képernyőre, hogy volt-e módosított termék, valamint a módosított zöldséges objektumot is! (Segítség: lásd a **String** osztály **charAt()** és **toUpperCase()** metódusait.)

10. Egészítse ki a **Zöldséges** osztályt egy metódussal, amely paraméterként megkap egy logikai értéket, és visszaadja a zöldségesnél kapható zöldségek számát, ha a paraméter igaz, illetve a gyümölcsök számát, ha hamis! Egészítse ki a **Teszt** osztályt is egy statikus metódussal, amely paraméterként megkap egy **Zöldséges** típusú objektumokat tartalmazó kollekciót, kiírja egy-egy szöveges állományba azoknak a zöldségeseknek a kínálatát, amelyeknél több gyümölcs kapható, mint zöldség, végül visszatér a létrehozott állományok számával! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **Zöldséges** objektumot tartalmazó egyelemű kollekcióval, és írja ki a képernyőre a létrehozott állományok számát! (2 pont)

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2014. január 18.**

1. Készítse el az **Alprogram** absztrakt osztályt az **ansiC** csomagban! Az alprogramokról tárolni szeretnénk a nevüket sztringként és a paramétereik típusait sztringek egy listájaként. Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindkét adattagjának kezdőérték adható! A konstruktor dobjon **IllegalArgumentException** kivételt „Hibás alprogramnév!” üzenettel, ha a paraméterként megkapott név **null**, üres sztring, vagy nem betűvel kezdődik! Definiálja felül az **equals()** metódust: két alprogram akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik (a fajtájuktól függetlenül)! Definiálja felül a **toString()** metódust úgy, hogy az az alprogram adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név>([<típus>[,<típus>]...])`” (pl. „*strchr(const char *,int)*”)! (Segítség: lásd a **Character.isLetter()** és a **String.charAt()** metódusokat.)

2. Egészítse ki az **Alprogram** osztályt egy **kisbetűsít()** nevű logikai metódussal, amelynek segítségével kisbetűssé alakítható az alprogram neve, valamint a paramétereinek típusai! A metódus adjon vissza igazat, ha az alprogram neve vagy valamelyik típusnév tartalmazott nagybetűt (azaz a metódus módosította az alprogramot), különben pedig hamisat! (Segítség: lásd a **String.toLowerCase()** metódust.)

3. Az alprogramok természetes rendezettsége a paraméterszámuk szerint növekvő, azon belül a nevük szerint csökkenő sorrendben legyen értelmezve!

4. Származtasson az **Alprogram** osztályból egy **Eljárás** és egy **Függvény** osztályt szintén az **ansiC** csomagban! A függvények esetén azok típusát is szeretnénk még tárolni sztringként. Az új adattag csak az **ansiC** csomagban látszódjon! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével mindegyik adattagjuknak kezdőérték adható! Definiálja felül mindkét osztályban a **toString()** metódust úgy, hogy az az **Alprogram** sztring reprezentációját adja vissza, kiegészítve azt elöl az alprogram típusával (eljárás esetén ez legyen „void”) és egy szóközzel, hátul pedig egy pontosvesszővel (pl. „*char * strchr(const char *,int);*”)! A **Függvény** osztályban definiálja felül a **kisbetűsít()** metódust is úgy, hogy a függvény nevén és a paramétereinek típusain kívül a függvény típusát is alakítsa kisbetűssé! A metódus visszatérési értéke továbbra is akkor legyen igaz, ha módosította az alprogramot, azaz vagy a függvény neve, vagy a paramétereinek típusai, vagy a függvény típusa tartalmazott nagybetűt!

5. Adott az alábbi interfész a **programozás** csomagban:

```
public interface FordításiEgység
{
  // hozzáadja a paraméterként megkapott tömbben szereplő alprogramokat a
  // fordítási egységhez
  public void hozzáad(Alprogram[] alp);

  // visszaadja a fordítási egységben található olyan alprogramok listáját a
  // természetes rendezettségük sorrendjében, amelyek a második paraméterként
  // megkapott darabszámú paraméterrel rendelkeznek; ha az első paraméter igaz,
  // akkor csak a függvényeket, ha hamis, akkor csak az eljárásokat, ha pedig
  // null, akkor mindegyiket
  public java.util.List<Alprogram> alprogramok(Boolean függvény, int paraméterSzám);

  // minden „void” típusú visszatérési értékkel rendelkező függvényhez létrehoz
  // a fordítási egységben egy eljárást, amelynek a neve a függvény neve
  // kiegészítve az elején egy P betűvel, paraméterei pedig a függvény
  // paraméterei
  public void voidFüggvényekbőlEljárás();
}
```

Készítse el a **Forrásállomány** osztályt az **ansiC** csomagban, amely a megjegyzéseknek megfelelően implementálja a **FordításiEgység** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat és lekérdező metódusokat! Figyeljen arra, hogy egy forrásállomány nem tartalmazhat azonos nevű alprogramokat! Legyen lehetőség a forrásállomány nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a forrásállomány nevét, valamint feltölthetjük azt egy alprogramokat tartalmazó kollekció elemeivel! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban a forrásállomány nevét, majd egy üres sort követően soronként az egyes alprogramokat adja vissza! (Segítség: lásd a **Collections.sort()** metódust.)

(2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található alprogramokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy alprogram adatait tartalmazza az alábbi formátumban:

   ```
   [procedure|function] <név>([<par>:<típus>[,<par>:<típus>]...])[:<típus>]
   ```

   Ha a sor a „procedure” szóval kezdődik, akkor eljárásról, ha a „function” szóval, akkor függvényről van szó. A sor végén a kettőspont és az azt követő típus (az alprogram típusa) csak függvények esetén van megadva. Például:

   ```
   function strchr(s:const char *,c:int):char *
   function malloc(size:size_t):void *
   procedure free(ptr:void *)
   ```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, a beolvasás megszakítása nélkül! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **Forrásállomány** objektumot! A forrásállomány neve legyen a második parancssori argumentum, vagy ennek hiányában „teszt.c”! Vegye fel a forrásállományba az előző feladatban beolvasott alprogramokat! Olvasson be a billentyűzetről egy egész értéket, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a forrásállomány azon alprogramjait, amelyek a beolvasott darabszámú paraméterrel rendelkeznek!

8. Egészítse ki a **Teszt** osztályt egy statikus logikai metódussal, amely paraméterként megkap egy alprogramokat tartalmazó kollekciót, és kisbetűssé alakítja a kollekcióban lévő alprogramok nevét, paramétereinek típusait, valamint függvények esetén azok típusát! A metódus adjon vissza igazat, ha legalább egy alprogramot módosított a kollekcióban, különben hamisat! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával, és írja ki a képernyőre, hogy történt-e módosítás!

9. Egészítse ki az **Alprogram** osztályt egy publikus logikai metódussal, amely eldönti, hogy van-e az alprogramnak egy paraméterként megkapott típusú paramétere! Egészítse ki a **Forrásállomány** osztályt is egy olyan metódussal, amely az első paramétereként megkapott nevű szöveges állományba soronként kiírja azokat az alprogramokat, amelyeknek van a második paramétereként megkapott típusú paramétere! A metódus kezelje az ellenőrzött kivételeket! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott forrásállományra `lista.txt` állománynévvel és egy billentyűzetről beolvasott típussal!

10. Egészítse ki a **Forrásállomány** osztályt egy publikus metódussal, amely visszaadja a forrásállomány mutató visszatérési típussal rendelkező függvényeinek számát (a mutató típusok csillag karakterrel végződnek)! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott forrásállományra, és írja ki az eredményt a képernyőre! (Segítség: lásd a **String** osztály **charAt()** és **length()** metódusait.)

11. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **Forrásállomány** típusú objektumokat tartalmazó tömböt, és képernyőre írja azokat a forrásállományokat, amelyekben több egyparaméteres függvény van, mint egyparaméteres eljárás! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **Forrásállomány** objektumot tartalmazó egyelemű tömbbel!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2015. január 9.**

1. Készítse el az **`Őshüllő`** osztályt az **`őslény`** csomagban! Az őshüllőkről tárolni szeretnénk a fajukat (sztring), az életterüket (sztring) és hogy növényevők-e (logikai). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **`equals()`** metódust: két őshüllő akkor legyen egyenlő, ha a fajuk (tartalmilag) megegyezik! Definiálja felül a **`toString()`** metódust is úgy, hogy az az őshüllő adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<faj>: <élettér>[, növényevő]`” (pl. „*Ichthyosaurus communis: víz*” vagy „*Amphicoelias fragillimus: szárazföld, növényevő*”)!
2. Az őshüllők természetes rendezettsége a fajuk szerint növekvő sorrendben legyen értelmezve!
3. Származtasson az **`Őshüllő`** osztályból egy **`Dínó`** osztályt szintén az **`őslény`** csomagban! A dínók az őshüllő tulajdonságain kívül még egy valós értékű testhossz és egy egész értékű testtömeg tulajdonsággal is rendelkeznek. Az új adattagok más osztályból ne látszódjanak, de írja meg a lekérdező metódusaikat! Készítsen az osztályhoz egy olyan konstruktort, amelynek segítségével az élettér kivételével mindegyik adattagjának kezdőérték adható! A dínók élettere legyen mindig „szárazföld”! Definiálja felül a **`toString()`** metódust úgy, hogy az az őshüllő adataihoz egy szóköz után fűzze hozzá zárójelben a dínó hosszát és tömegét is a példákban látható formában (pl. „*Tyrannosaurus rex: szárazföld (12.0 m, 6800 kg)*” vagy „*Amphicoelias fragillimus: szárazföld, növényevő (58.0 m, 122400 kg)*”)!
4. Egészítse ki az **`Őshüllő`** osztályt egy olyan metódussal, amely csupa nagybetűssé alakítja az őshüllő faját, ha az egy dínó, és a tömege meghalad egy paraméterként megkapott értéket! Minden más esetben a metódus úgy módosítsa a fajt, hogy az első karakterét alakítsa nagybetűssé, a többit pedig kisbetűssé! (Segítség: lásd a **`String`** osztály **`toLowerCase()`**, **`toUpperCase()`** és **`substring()`** metódusait, valamint a **`Character.toUpperCase()`** metódust.)
5. Adott az alábbi interfész az **`őslény`** csomagban:

```
public interface ŐslényPark
{
  // új őshüllőt vesz fel a parkba
  public void felvesz(Őshüllő őshüllő);

  // visszaadja azoknak a dínóknak a rendezett listáját (a természetes rendezettség
  // sorrendjében), amelyek testhossza legalább a paraméterben megkapott érték
  public java.util.List<Dínó> nagyDínók(double minHossz);

  // visszaadja a parkban található őshüllők számát, ha a paraméter hamis, illetve
  // csak a dínók számát, ha igaz
  public int állatokSzáma(boolean csakDínók);

  // kiírja a megadott állományba az őshüllők listáját (mindegyiket külön sorba)
  public void kiír(String fájlnév);
}
```

   Készítse el a **`JurassicPark`** osztályt a **`jura`** csomagban, amely a megjegyzéseknek megfelelően implementálja az **`ŐslényPark`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a parkban több azonos fajhoz tartozó őshüllő is előfordulhat! Legyen lehetőség a park nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a park nevét, valamint feltölthetjük azt egy őshüllőket tartalmazó kollekció elemeivel! A konstruktor a paramétereként megkapott kollekcióról készítsen másolatot! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a park nevét, majd egy üres sort követően soronként az egyes őshüllőket adja vissza! (Segítség: lásd a **`Collections.sort()`** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található őshüllőket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy őshüllő adatait tartalmazza az alábbi formátumok egyikében:

   ```
   H;<faj>;<élettér>;[N|R|M|D]
   D;<faj>;[N|R|M|D];<testhossz>;<testtömeg>
   ```

   Ha a sor első karaktere `H`, akkor egyszerű őshüllőről, ha `D`, akkor dínóról van szó. Az `N`, `R`, `M`, `D` karakterek jelentése rendre: növényevő, ragadozó, mindenevő, dögevő. Például:

   ```
   H;Ichthyosaurus communis;víz;R
   D;Amphicoelias fragillimus;N;58;122400
   ```

   A beolvasás után alakítsa át a lista összes őshüllőjének faját a 4. feladatban megírt metódus segítségével, 1000 kg-os tömeghatárral! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()**, **nextDouble()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **JurassicPark** objektumot! A park neve legyen a második parancssori argumentum, vagy ennek hiányában „Jurassic Park”! Töltse fel a parkot az előző feladatban beolvasott őshüllőkkel! Olvasson be a billentyűzetről egy valós számot, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a park azon őshüllőit, amelyek hossza legalább a beolvasott érték!

8. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy őshüllőket tartalmazó tömböt, valamint egy sztringet, és képernyőre írja soronként azoknak a tömbbeli őshüllőknek a jellegét (egyszerű őshüllő vagy dínó) és egyéb adatait, amelyeknek a faja tartalmazza a megkapott sztringet! A főprogramban kérjen be a billentyűzetről egy sztringet, majd hívja meg a metódust a 6. feladatban létrehozott listából készített tömbbel és a beolvasott értékkel! (Segítség: lásd a **Collection.toArray()** és a **String.contains()** metódust.)

9. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **ŐslényPark** típusú objektumot, valamint egy őshüllőket tartalmazó kollekciót, és felveszi az őslényparkba a kollekcióban található őshüllők közül azokat a dínókat, amelyek testhossza meghaladja a kollekcióban található dínók átlagos hosszát! Ha a kollekcióban egyetlen dínó sem található, a metódus dobjon **NincsDínóException** kivételt! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **JurassicPark** objektummal és a 6. feladatban létrehozott listával, kezelve a kivételt egy hibaüzenet kiírásával! Ellenőrzésképpen írja ki a képernyőre a módosított **JurassicPark** objektumot!

10. Egészítse ki a **JurassicPark** osztályt egy olyan metódussal, amely paraméterként megkap két testtömegértéket, és visszaad egy sztringlistát, amely azon dínók fajait tartalmazza, amelyek tömege a megkapott két érték közé esik! Figyeljen arra, hogy a két testtömegérték közül bármelyik lehet nagyobb a másiknál! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **JurassicPark** objektumra két, billentyűzetről beolvasott egész számmal, és írja ki a képernyőre az eredményül kapott fajokat!

11. Egészítse ki a **Teszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **ŐslényPark** típusú referenciákat tartalmazó tömböt, és kiírja egy-egy szöveges állományba azoknak a parkoknak az őshüllőit, amelyekben több dínó van, mint egyéb őshüllő! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **JurassicPark** objektumot tartalmazó egyelemű tömbbel!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2015. január 10.**

1. Készítse el a **`Tévé`** osztályt az **`elektronika`** csomagban! A tévékről tárolni szeretnénk a márkájukat (sztring), a típusukat (sztring), a tulajdonságaikat (sztringtömb) és az árukat (egész). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható, méghozzá a tulajdonságok esetén egy sztringeket tartalmazó kollekcióval! Írja meg az adattagok lekérdező metódusait, valamint a márka beállító metódusát! Definiálja felül az **`equals()`** metódust: két tévé akkor legyen egyenlő, ha a márkájuk és a típusuk (tartalmilag) megegyezik (a fajtájuktól függetlenül)! (Segítség: lásd a **`Collection.toArray()`** metódust.)
2. Definiálja felül a **`toString()`** metódust úgy, hogy az a tévé adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<márka> <típus>[ (<tulajdonság>, ...)], <ár> Ft`”! (Pl. „*Samsung UE40H6500 (3D, smart, LED, A+ energiaosztály, full HD, 40 col), 169999 Ft*” vagy „*Orion PIF 24-D, 44899 Ft*”.)
3. A tévék természetes rendezettsége az áruk szerint csökkenő, azon belül a márkájuk, azon belül pedig a típusuk szerint növekvő sorrendben legyen értelmezve!
4. Származtasson a **`Tévé`** osztályból egy **`LEDTévé`** és egy **`SmartTévé`** osztályt szintén az **`elektronika`** csomagban! A LED-tévék esetén tárolni szeretnénk, hogy OLED technológiájúak-e, a smart tévék esetén pedig azt, hogy lehet-e rájuk új alkalmazásokat telepíteni. Az új adattagok legyenek logikai típusúak, és más osztályból ne látszódjanak, de írja meg a lekérdező metódusaikat! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével mindegyik adattagjuknak kezdőérték adható!
5. Adott az alábbi interfész az **`elektronika`** csomagban:

```
public interface TévéÜzlet
{
  // visszaadja az üzletben kapható olyan tévék listáját a természetes
  // rendezettségük sorrendjében, amelyek tulajdonságai között szerepel a
  // paraméterként megkapott tulajdonság
  public java.util.List<Tévé> adottTulajdonságúTévék(String tulajdonság);

  // visszaadja az üzletben kapható olyan modern tévék listáját a természetes
  // rendezettségük sorrendjében, amelyek tulajdonságai között szerepel a
  // paraméterként megkapott tulajdonság; egy tévé modern, ha OLED technológiával
  // készült LED-tévé, vagy olyan smart tévé, amelyre lehet új alkalmazásokat
  // telepíteni
  public java.util.List<Tévé> adottTulajdonságúModernTévék(String tulajdonság);

  // kiírja a megadott nevű állományba a tévék listáját (mindegyiket külön
  // sorba), három felcímkézett csoportban: először a LED-tévéket („LED-tévék”
  // címkével), aztán a smart tévéket („Smart tévék” címkével), végül a többit
  // („Egyéb tévék” címkével), a csoportokon belül a természetes rendezettségük
  // sorrendjében
  public void kiír(String fájlnév);
}
```

Készítse el a **`MédiaPiac`** osztályt a **`piac`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`TévéÜzlet`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a médiapiacon több azonos márkájú és típusú tévé is kapható! Legyen lehetőség a médiapiac nevének és címének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a médiapiac nevét és címét, valamint feltölthetjük azt egy tévéket tartalmazó tömb elemeivel! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a médiapiac nevét és címét, majd egy üres sort követően az egyes tévéket adja vissza a **`kiír()`** metódusnál megadott formátumban! (Segítség: lásd a **`Collections.sort()`** és az **`Arrays.asList()`** metódusokat.)

(2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **TévéTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található tévéket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy tévé adatait tartalmazza az alábbi formátumban:

   ```
   [E|L|S];<márka>;<típus>;<ár>;[[+|-];][<tulajdonság>[,<tulajdonság>]...]
   ```

   Ha a sor első karaktere `E`, akkor egyszerű tévéről, ha `L`, akkor LED-tévéről, ha `S`, akkor smart tévéről van szó. Az ár utáni + vagy − karakter csak LED-tévék és smart tévék esetén szerepel: LED-tévék esetén a + azt jelenti, hogy a tévé OLED technológiával készült, a − azt, hogy nem; smart tévék esetén a + azt jelenti, hogy a tévére lehet új alkalmazásokat telepíteni, a − azt, hogy nem. Az utolsó pontosvesszőt követően nulla vagy több tulajdonság megnevezése található, egymástól vesszővel elválasztva. Például:

   ```
   L;Vortex;LED-V32ZH8DC;54999;-;32 col,B energiaosztály,HD ready
   L;Orion;PIF 24-D;44899;-;
   S;LG;70LB650V;699899;+;LED,3D,70 col,A+ energiaosztály,Full HD
   L;Toshiba;32 W 3433DG;79999;-;smart,32 col,A energiaosztály,HD ready
   E;LG;50PB560B;119999;plazma,50 col,B energiaosztály
   ```

   (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()**, **nextInt()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **MédiaPiac** objektumot! A médiapiac neve legyen a második parancssori argumentum, vagy ennek hiányában „Debreceni Médiapiac”, a címe pedig a harmadik parancssori argumentum, vagy ennek hiányában „Debrecen, Piac u.”! Töltse fel a médiapiac kínálatát az előző feladatban beolvasott tévékkel! Olvasson be a billentyűzetről egy tulajdonságot, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a médiapiacon kapható azon tévéket, amelyek tulajdonságai között szerepel a beolvasott tulajdonság! (Segítség: lásd a **Collection.toArray()** metódust.)

8. Egészítse ki a **TévéTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy tévéket tartalmazó nem üres kollekciót, és visszaadja a kollekcióban található tévék tulajdonságainak átlagos számát valós számként! Felhasználva ezt a metódust, a főprogramban írja ki a képernyőre soronként a 6. feladatban létrehozott listában szereplő azon tévéket, amelyek az átlagosnál kevesebb tulajdonsággal rendelkeznek!

9. Egészítse ki a **MédiaPiac** osztályt egy logikai metódussal, amely csupa nagybetűssé alakítja a médiapiac kínálatában azokat a tévémárkákat, amelyek egy paraméterként megkapott karakterrel kezdődnek, és igazat ad vissza, ha volt módosított termék! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott médiapiacra egy billentyűzetről beolvasott karakterrel! Ellenőrzésképpen írja ki a képernyőre, hogy volt-e módosított termék, és ha igen, írja ki a módosított médiapiac objektumot is! (Segítség: lásd a **String** osztály **charAt()** és **toUpperCase()** metódusait.)

10. Egészítse ki a **MédiaPiac** osztályt egy logikai metódussal, amely igazat ad vissza, ha a médiapiacon több smart tévé van, mint LED-tévé! A metódus tekintse LED-tévének azokat a tévéket, amelyek vagy a típusuk szerint azok (azaz a **LEDTévé** osztály példányai), vagy a tulajdonságaik között szerepel a „LED” szó, nem megkülönböztetve a kis- és nagybetűket! Hasonlóan tekintsük smart tévének mindazokat a tévéket, amelyek vagy a típusuk szerint azok (azaz a **SmartTévé** osztály példányai), vagy a tulajdonságaik között szerepel a „smart” szó, nem megkülönböztetve a kis- és nagybetűket! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott médiapiacra, és írja ki a képernyőre az eredményt! (Segítség: lásd a **String** osztály **equalsIgnoreCase()** metódusát.)

11. Egészítse ki a **TévéTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **MédiaPiac** típusú objektumokat tartalmazó kollekciót és egy tulajdonságot, kiírja egy-egy szöveges állományba azoknak a médiapiacoknak a kínálatát, ahol legalább 3 olyan modern tévé kapható, amelyek tulajdonságai között szerepel a paraméterként megkapott tulajdonság, végül visszatér a létrehozott állományok számával! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **MédiaPiac** objektumot tartalmazó egyelemű listával és az ott beolvasott tulajdonsággal, majd írja ki a képernyőre a létrehozott állományok számát!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2015. január 17.**

1. Készítse el a **`Társasjáték`** osztályt a **`játék`** csomagban! Minden társasjátékról tárolni szeretnénk a nevét (sztring), hogy hány éves kortól ajánlott (egész), valamint hogy legalább és legfeljebb hány személy játszhatja (egészek). Az egész értékű adattagok esetében a 0 jelentse azt, hogy az adott érték nincs meghatározva! Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **`IllegalArgumentException`** kivételt „Negatív érték!” üzenettel, ha a numerikus paraméterek bármelyike is negatív! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **`equals()`** metódust: két társasjáték akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik! Definiálja felül a **`toString()`** metódust is úgy, hogy az a társasjáték adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név>[ (<korhatár> éves kortól)], játékosok száma: [<min>]-[<max>]`”! A korhatár, valamint a játékosok minimális és maximális száma csak akkor jelenjenek meg, ha nem 0 értékűek! (Pl. „*Gazdálkodj okosan! (10 éves kortól), játékosok száma: 2–6*” vagy „*Puzzle 15 db-os, játékosok száma: –*”.)
2. A társasjátékok természetes rendezettsége a korhatáruk szerint csökkenő, azon belül a nevük szerint növekvő sorrendben legyen értelmezve!
3. Származtasson a **`Társasjáték`** osztályból egy **`Kártyajáték`** osztályt szintén a **`játék`** csomagban! A kártyajátékok a társasjáték tulajdonságain kívül még egy sztring típusú paklifajta és egy egész típusú pakliméret tulajdonsággal is rendelkeznek. Az új adattagok más osztályból ne látszódjanak, de írja meg a lekérdező metódusaikat! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Definiálja felül a **`toString()`** metódust úgy, hogy az a társasjáték adataihoz egy vessző és egy szóköz után fűzze hozzá a pakli méretét és fajtáját is a példákban látható formában (pl. „*Kanaszta (10 éves kortól), játékosok száma: 4–4, 110 lapos francia kártya*” vagy „*Zsír, játékosok száma: 2–4, 32 lapos magyar kártya*”)!
4. Egészítse ki a **`Társasjáték`** osztályt egy olyan metódussal, amely megcseréli a minimális és a maximális játékosszámot, ha mindkettő meg van határozva, és az előbbi nagyobb az utóbbinál!
5. Adott az alábbi interfész a **`játék`** csomagban:

```
public interface JátékLista
{
  // új játékot vesz fel a listára
  public void felvesz(Társasjáték játék);

  // visszaadja azoknak a kártyajátékoknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyeket a paraméterként megkapott fajtájú
  // paklival kell játszani
  public java.util.List<Kártyajáték> kártyajátékok(String paklifajta);

  // visszaadja azoknak a játékoknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyeket játszhat egy paraméterként megkapott
  // életkorú személy
  public java.util.List<Társasjáték> megfelelőJátékok(int életkor);

  // kiírja a megadott állományba a játékok listáját (mindegyiket külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **`KívánságLista`** osztályt a **`játékos`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`JátékLista`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a kívánságlistán egy játék csak egyszer szerepelhet! Legyen lehetőség a kívánságlista tulajdonosának tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a tulajdonos nevét, valamint feltölthetjük a listát egy társasjátékokat tartalmazó kollekció elemeivel! A konstruktor a paramétereként megkapott kollekcióról készítsen másolatot! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a tulajdonos nevét, majd egy üres sort követően soronként az egyes játékokat adja vissza! (Segítség: lásd a **`Collections.sort()`** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **JátékTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található társasjátékokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy társasjáték adatait tartalmazza az alábbi formátumban:

   ```
   [T|K];<név>;<korhatár>;<min>;<max>[;<paklifajta>;<pakliméret>]
   ```

   Ha a sor első karaktere `T`, akkor egyszerű társasjátékról, ha `K`, akkor kártyajátékról van szó. A paklifajta és a pakliméret csak kártyajátékok esetén szerepel. Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! Például:

   ```
   T;Gazdálkodj okosan!;10;2;6
   T;Puzzle 15 db-os;0;0;0
   K;Kanaszta;10;4;4;francia;110
   ```

   A beolvasás után korrigálja az esetlegesen felcserélve megadott minimális és maximális játékosszámokat a 4. feladatban megírt metódus segítségével! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **KívánságLista** objektumot! A tulajdonosa legyen a második parancssori argumentum, vagy ennek hiányában „Petike”! Töltse fel a listát az előző feladatban beolvasott társasjátékokkal! Olvasson be a billentyűzetről egy paklifajtát, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a kívánságlistán szereplő azon kártyajátékokat, amelyeket a beolvasott fajtájú paklival kell játszani!

8. Egészítse ki a **JátékTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy társasjátékokat tartalmazó tömböt, és képernyőre írja soronként azoknak a tömbbeli játékoknak a jellegét (egyszerű társasjáték vagy kártyajáték) és egyéb adatait, amelyek bármilyen életkorban játszhatók! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listából készített tömbbel! (Segítség: lásd a **Collection.toArray()** metódust.)

9. Egészítse ki a **JátékTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **JátékLista** típusú referenciát, valamint egy társasjátékokat tartalmazó kollekciót, és felveszi a játéklistára csupa nagybetűs névvel a kollekcióban található társasjátékok közül azokat a kártyajátékokat, amelyeket 32 lapos magyar kártyával kell játszani! A kollekcióban lévő társasjátékok nem módosulhatnak! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **KívánságLista** objektummal és a 6. feladatban létrehozott listával! Ellenőrzésképpen írja ki a képernyőre a módosított **KívánságLista** objektumot! (Segítség: lásd a **String.toUpperCase()** metódust.)

10. Egészítse ki a **KívánságLista** osztályt egy olyan metódussal, amely paraméterként megkap egy egész értéket, és visszaad egy sztringlistát, benne azon kártyajátékok neveivel, amelyeket játszhatnak annyian, amennyi a paraméterként megkapott érték! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **KívánságLista** objektumra egy billentyűzetről beolvasott egész számmal, és írja ki a képernyőre az eredményül kapott kártyajátékneveket!

11. Egészítse ki a **JátékTeszt** osztályt egy statikus metódussal, amely paraméterként megkap egy **JátékLista** típusú referenciákat tartalmazó tömböt, kiírja egy-egy szöveges állományba azoknak a játéklistáknak az elemeit, amelyekben van 3 évesek által is játszható játék, végül visszatér a létrehozott állományok számával! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **KívánságLista** objektumot tartalmazó egyelemű tömbbel, és írja ki a képernyőre a létrehozott állományok számát!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2017. január 6.**

1. Készítse el a **TanulmányiVerseny** osztályt a **verseny** csomagban! Minden tanulmányi versenyről tárolni szeretnénk a megnevezését (sztring), a témájául szolgáló tantárgyat (sztring), a dátumát (év, hó, nap), valamint a kezdési és befejezési idejét (óra, perc). Feltehetjük, hogy a versenyek csak egy naptári napot ölelnek fel. A dátumot, illetve a kezdési és befejezési időt tetszőlegesen választott adattípusokkal reprezentálhatja (lásd például a **LocalDate** és a **LocalTime** osztályokat). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **equals()** metódust: két verseny akkor legyen egyenlő, ha a megnevezésük, a tantárgyuk és a dátumuk (tartalmilag) megegyezik (a fajtájuktól függetlenül)! Definiálja felül a **toString()** metódust is úgy, hogy az a verseny adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<megnevezés> (<tantárgy>), <dátum> <kezdés>-<befejezés>`”! A dátum és időpont értékeket tetszőleges formában megjelenítheti. (Pl. „*Országos Középiskolai Tanulmányi Verseny (fizika), 2016. november 17. 14:00–19:00*”.)
2. Származtasson a **TanulmányiVerseny** osztályból egy **ProgVerseny** osztályt szintén a **verseny** csomagban! A programozó versenyekről – a tanulmányi verseny tulajdonságain kívül – azt is tudni szeretnénk, hogy milyen programozási nyelveket lehet rajtuk használni (ehhez is válasszon egy megfelelő adattípust). Az új adattag más osztályból ne látszódjon, de írja meg a lekérdező metódusát! Készítsen az osztályhoz egy konstruktort, amelynek segítségével a tantárgy kivételével mindegyik adattagjának kezdőérték adható! A programozó versenyek esetén a tantárgy legyen mindig „informatika”! A konstruktor dobjon egy (tetszőleges) kivételt „Nincs megadva programozási nyelv!” üzenettel, ha a paraméterében egyetlen programozási nyelvet sem kapott! Definiálja felül a **toString()** metódust úgy, hogy az a tanulmányi verseny sztring reprezentációjához egy vessző és egy szóköz után fűzze hozzá a támogatott programozási nyelveket is a példában látható formában (pl. „*DEIK Regionális Programozó Csapatverseny (informatika), 2016. december 4. 10:00–15:00, támogatott nyelvek: C, C++, Java, C#, Pascal, Python*”)!
3. A programozó versenyek természetes rendezettsége a támogatott programozási nyelvek száma szerint csökkenő, azon belül a dátumuk és kezdési idejük szerint növekvő sorrendben legyen értelmezve!
4. Egészítse ki a **TanulmányiVerseny** osztályt egy olyan metódussal, amely megcseréli a kezdési és a befejezési időt, ha az előbbi nagyobb az utóbbinál!
5. Adott az alábbi interfész a **verseny** csomagban:

```
public interface VersenyLista
{
  // új versenyt vesz fel a listára
  public void felvesz(TanulmányiVerseny verseny);

  // visszaadja azoknak a programozó versenyeknek a rendezett listáját (a
  // természetes rendezettség sorrendjében), amelyeken használható a paraméterként
  // megkapott programozási nyelv
  public java.util.List<ProgVerseny> versenyek(String nyelv);

  // visszaadja azoknak a programozó versenyeknek a rendezett listáját (a
  // természetes rendezettség sorrendjében), amelyeken használható a paraméterként
  // megkapott összes programozási nyelv
  public java.util.List<ProgVerseny> versenyek(String[] nyelvek);
}
```

Készítse el a **VersenyÉvad** osztályt a **tanév** csomagban, amely a megjegyzéseknek megfelelően implementálja a **VersenyLista** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a versenyévadban egy verseny csak egyszer szerepelhet! Legyen lehetőség a versenyévad tanévének a tárolására (pl. 2016/2017-es tanév)! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a tanévet, valamint feltölthetjük a listát egy tanulmányi versenyeket tartalmazó kollekció elemeivel! A konstruktor a paramétereként megkapott kollekcióról készítsen másolatot, hogy az eredeti kollekció esetleges későbbi módosítása ne legyen hatással a versenyévadban tárolt versenyekre, és fordítva! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban a tanévet, majd egy üres sort követően soronként az egyes versenyeket adja vissza! (Segítség: lásd a

Collections.sort(), a **Collection.contains()**, a **Collection.containsAll()** és az **Arrays.asList()** metódusokat.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **VersenyTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található tanulmányi versenyeket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy tanulmányi verseny adatait tartalmazza az alábbi formátumban:

   `[T|P];<név>[;<tantárgy>];<év>.<hó>.<nap>;<óra>:<perc>;<óra>:<perc>[;<nyelv>[,...]]`

   Ha a sor első karaktere `T`, akkor egyszerű tanulmányi versenyről, ha `P`, akkor programozó versenyről van szó. Az év mindig 4, a hó, nap, óra és perc pedig 2 karakteres egész szám. A tantárgy csak egyszerű tanulmányi versenyeknél van megadva, a programozási nyelvek pedig csak programozó versenyek esetén szerepelhetnek. Például:

   ```
   T;Országos Középiskolai Tanulmányi Verseny;fizika;2016.11.17;14:00;19:00
   P;DEIK Regionális Programozó Csapatverseny;2016.12.04;10:00;15:00;C,C++,Java,C#
   P;Hibás prog. verseny: nincs nyelv;2017.01.06;13:00;16:00
   ```

   Kezelje a 2. feladatban dobott esetleges kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! A beolvasás után korrigálja az esetlegesen felcserélve megadott kezdési és befejezési időket a 4. feladatban megírt metódus segítségével! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **hasNext()**, **nextLine()**, **next()**, **nextInt()** metódusait. Ügyeljen arra, hogy a versenyek megnevezése tartalmazhat pont, kettőspont és vessző karaktereket is, csak pontosvesszőt nem! Ha a pont karakter mentén történő daraboláshoz reguláris kifejezést használ, akkor azt `"\\."` formában kell megadni.) (3 pont)

7. Hozzon létre a főprogramban egy **VersenyÉvad** objektumot! A tanév legyen a második parancssori argumentum, vagy ennek hiányában „2016/2017”! Töltse fel a versenyévadot az előző feladatban beolvasott versenyekkel! Olvasson be a billentyűzetről egy programozási nyelvet, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a versenyévadban szereplő azon programozó versenyeket, amelyeken használható a beolvasott nyelv!

8. Egészítse ki a **VersenyTeszt** osztályt egy metódussal, amely paraméterként megkap egy tanulmányi versenyeket tartalmazó tömböt, és képernyőre írja soronként azoknak a tömbbeli versenyeknek a fajtáját (egyszerű tanulmányi verseny vagy programozó verseny) és egyéb adatait, amelyek időtartama kevesebb, mint 3 óra! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listából készített tömbbel! (Segítség: lásd a **LocalTime.isBefore()**, a **LocalTime.plusHours()** és a **Collection.toArray()** metódusokat.)

9. Egészítse ki a **VersenyÉvad** osztályt egy olyan metódussal, amely törli az évadból azokat a versenyeket, amelyek dátuma nem illeszkedik a versenyévad tanévébe! Ha egy dátumban az évszám *x*, akkor az adott év első félévébe tartozó dátum az $x - 1/x$, a második félévbe tartozó dátum pedig az $x/x + 1$ tanévhez tartozik. (A 2017.01.06. dátum például a 2016/2017-es tanévhez tartozik.) Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **VersenyÉvad** objektumra, majd írja ki a képernyőre az eredményül kapott versenyévadot! (Segítség: ne használja a Java for-each ciklusát, ha törölni szeretne a bejárandó kollekcióból! Lásd helyette például a **Collection.iterator()** és az **Iterator.remove()** metódusokat. Lásd továbbá a **LocalDate** osztály **getYear()** és **getMonthValue()** metódusait.)

10. Egészítse ki a **VersenyTeszt** osztályt egy metódussal, amely paraméterként megkap egy **VersenyLista** típusú referenciákat tartalmazó tömböt, kiírja egy-egy szöveges állományba azokat a versenylistákat, amelyekben van legalább egy olyan programozó verseny, amelyen használható a C és a C++ nyelv is, végül visszatér a létrehozott állományok számával! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **VersenyÉvad** objektumot tartalmazó egyelemű tömbbel, és írja ki a képernyőre a létrehozott állományok számát!

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2017. január 9.**

1. Készítse el a **`Tantárgy`** osztályt az **`egyetem`** csomagban! A tantárgyakról tudni szeretnénk a kódjukat (sztring), a nevüket (sztring), és hogy hány kreditesek (egész). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **`equals()`** metódust: két tantárgy akkor legyen egyenlő, ha a kódjuk (tartalmilag) megegyezik (de egy tantárgy nem lehet egyenlő egy kurzussal még akkor sem, ha a kódjuk megegyezik)! Definiálja felül a **`toString()`** metódust is úgy, hogy az a tantárgy adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<kód> <név> (<kredit> kredites)`”! (Pl. „*ILDK302 Magas szintű programozási nyelvek 2 (5 kredites)*”.)
2. A tantárgyak természetes rendezettsége a kreditszámuk szerint csökkenő, azon belül a nevük szerint növekvő, azon belül pedig a kódjuk szerint növekvő sorrendben legyen értelmezve!
3. Származtasson a **`Tantárgy`** osztályból egy **`Kurzus`** osztályt szintén az **`egyetem`** csomagban! A kurzusokról – a tantárgy tulajdonságain kívül – tudni szeretnénk még az alábbiakat is: előadás-e vagy gyakorlat (logikai), melyik teremben van megtartva (sztring), hány órás (egész), illetve hogy a hét melyik napján (hétfő, kedd stb.) és mikor kezdődik (óra, perc). Ez utóbbi adattagokat tetszőleges adattípusokkal reprezentálhatja (lásd például a **`DayOfWeek`** és a **`LocalTime`** osztályokat)! Az új adattagok más osztályból ne látszódjanak, de írja meg a lekérdező metódusaikat! Definiálja felül az **`equals()`** metódust: két kurzus akkor legyen egyenlő, ha a kódjukon kívül a kezdési idejük (a nap, az óra és a perc) is megegyezik (tartalmilag)! Definiálja felül a **`toString()`** metódust is úgy, hogy az a tantárgy sztring reprezentációjához egy vessző és egy szóköz után fűzze hozzá a kurzus egyéb adatait is az alábbi formában: „`<óraszám> órás [előadás|gyakorlat], <terem> teremben, kezdete: <nap> <óra>:<perc>`”! (Pl. „*ILDK302 Magas szintű programozási nyelvek 2 (5 kredites), 2 órás előadás, IK-F02 teremben, kezdete: csütörtök 16:00*”)! Ügyeljen arra, hogy a napot magyarul, a percet pedig két karakterrel jelenítse meg! (Segítség: lásd a **`DayOfWeek.getDisplayName()`** metódust.)
4. Készítsen a **`Kurzus`** osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **`HibásAdatException`** kivételt „Hibás érték!” üzenettel, ha a paramétereként kapott óraszám nem pozitív, illetve a hét napja vagy a kezdési idő értéke nem megfelelő (például az óra nem 0 és 23, vagy a perc nem 0 és 59 közé esik)! Készítsen egy másik konstruktort is, amellyel csak a tantárgy adatai és a kurzus jellege (előadás vagy gyakorlat) adható meg tetszés szerint, a többi adattag alapértelmezett értékei legyenek az alábbiak: a terem `null`, az óraszám 2, a nap hétfő, a kezdési idő pedig 08:00! (Segítség: lásd a **`LocalTime.of()`** metódust.)
5. Adott az alábbi interfész az **`egyetem`** csomagban:

```
public interface TantervіHáló
{
  // új tantárgyat/kurzust ad a hálóhoz
  public void újTárgy(Tantárgy tárgy);

  // visszaadja a hálóban lévő tantárgyak/kurzusok összkreditszámát
  public int összKredit();

  // visszaadja azoknak a kurzusoknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyek időtartama meghaladja a paraméterként
  // megkapott óraszámot
  public java.util.List<Kurzus> hosszúKurzusok(int óraszám);
}
```

Készítse el a **`PTIHáló`** osztályt az **`unideb`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`TantervіHáló`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a tantervi hálóban egy tantárgy/kurzus csak egyszer szerepelhet! Legyen lehetőség a tantervi háló megnevezésének a tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a tantervi haló nevét, valamint feltölthetjük a hálót egy tantárgyakat és kurzusokat tartalmazó kollekció elemeivel! A konstruktor a paramétereként megkapott kollekcióról készítsen másolatot, hogy az eredeti kollekció esetleges későbbi módosítása ne legyen hatással a tantervi hálóban tárolt tantárgyakra, és fordítva! Definiálja felül a **`toString()`** metódust úgy, hogy az az első

sorban a háló megnevezését, majd egy üres sort követően soronként az egyes tantárgyakat és kurzusokat adja vissza! (Segítség: lásd a **`Collections.sort()`** metódust.) (2 pont)

6. Helyezze el a főprogramot az **`unideb`** csomag **`Teszt`** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található tantárgyakat és kurzusokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy tantárgy vagy kurzus adatait tartalmazza az alábbi formátumban:

   `<kód>;<név>;<kredit>[;[E|G][;<terem>;<óraszám>;<nap>;<óra>:<perc>]]`

   Ha a kreditszám után folytatódik a sor, akkor kurzusról, ha nem, akkor tantárgyról van szó. Az `E` betű előadást, a `G` gyakorlatot jelent. A nap a „`H`”, „`K`”, „`SZE`”, „`CS`”, „`P`”, „`SZO`”, „`V`” sztringek egyike. Például:

   ```
   ILDK302;Magas szintű programozási nyelvek 2;5
   ILDK302;Nem lesz a hálóban;0
   ILDK302;Magas szintű programozási nyelvek 2;5;E
   XXX;Hibás óraszámú kurzus;1;E;terem;0;V;12:34
   ILDK302;Magas szintű programozási nyelvek 2;5;G;IK-105;4;SZE;14:00
   ILDK302;Ez sem lesz a hálóban;5;G;IK-106;4;H;08:00
   ```

   A kurzusok létrehozásához használja a megfelelő konstruktort! Kezelje a 4. feladatban dobott esetleges kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **`Scanner`** osztály **`useDelimiter()`**, **`hasNextLine()`**, **`hasNext()`**, **`nextLine()`**, **`next()`**, **`nextInt()`** metódusait, valamint a **`LocalTime.parse()`** metódust.) (3 pont)

7. Hozzon létre a főprogramban egy **`PTIHáló`** objektumot, amelynek megnevezése legyen a második parancssori argumentum, vagy ennek hiányában „PTI-s tantervi háló”! Töltse fel a hálót az előző feladatban beolvasott tantárgyakkal és kurzusokkal! Olvasson be a billentyűzetről egy óraszámot, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a hálóban szereplő azon kurzusokat, amelyek időtartama meghaladja a beolvasott óraszámot!

8. Egészítse ki a **`Teszt`** osztályt egy metódussal, amely paraméterként megkap egy **`Tantárgy`** típusú referenciákat tartalmazó kollekciót, a kollekcióban található minden olyan tantárgyhoz, amely nem kurzus, létrehoz egy elméleti és egy gyakorlati kurzust alapértelmezett teremmel, óraszámmal és kezdési idővel, majd visszaad egy tömböt a létrehozott kurzusokkal! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával, és írja ki a képernyőre soronként az eredményül kapott kurzusokat!

9. Egészítse ki a **`PTIHáló`** osztályt egy olyan metódussal, amely törli a hálóból azokat a kurzusokat, amelyeknél a terem értéke `null`! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **`PTIHáló`** objektumra, majd írja ki a képernyőre az eredményül kapott hálót! (Segítség: ne használja a Java for-each ciklusát, ha törölni szeretne a bejárandó kollekcióból! Lásd helyette például a **`Collection.iterator()`** és az **`Iterator.remove()`** metódusokat.)

10. Egészítse ki a **`Teszt`** osztályt egy metódussal, amely paraméterként megkap egy **`TanterviHáló`** típusú referenciákat tartalmazó tömböt és egy kreditszámot, kiírja egy-egy szöveges állományba azokat a tantervi hálókat, amelyekben a tantárgyak/kurzusok összkreditszáma a második paraméterben megkapott kreditszám, végül visszatér a létrehozott állományok számával! Az állományok neve `lista<szám>.txt` alakú legyen, ahol a `<szám>` 1-ről indul, és állományonként egyesével növekszik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **`PTIHáló`** objektumot tartalmazó egyelemű tömbbel és egy billentyűzetről beolvasott egész számmal, majd írja ki a képernyőre a létrehozott állományok számát!

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2017. december 16.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el az **`Ital`** osztályt az **`ital`** csomagban! Az italokról tudni szeretnénk a megnevezésüket (sztring), a kiszerelésüket (sztring) és az árukat (egész). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait, valamint a kiszerelés beállító metódusát! Definiálja felül az **`equals()`** metódust: két ital akkor legyen egyenlő, ha a nevük és a kiszerelésük (tartalmilag) megegyezik (a típusuktól függetlenül)! Definiálja felül a **`toString()`** metódust is úgy, hogy az az ital adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név>, <kiszerelés>, <ár> Ft`”! (Pl. „*Coca-Cola, 5 dl, 249 Ft*”.)
2. Az italok természetes rendezettsége az áruk szerint csökkenő, azon belül a nevük szerint növekvő, azon belül pedig a kiszerelésük szerint növekvő sorrendben legyen értelmezve!
3. Származtasson az **`Ital`** osztályból egy **`SzeszesItal`** osztályt szintén az **`ital`** csomagban! A szeszesitalok – az ital tulajdonságain kívül – még egy valós értékű, más osztályból nem látható alkoholtartalom tulajdonsággal is rendelkeznek. Egészítse ki az osztályt egy konstruktorral, amelynek segítségével mind a négy adattagjának kezdőérték adható! Írja meg az új adattag lekérdező metódusát! Definiálja felül a **`toString()`** metódust úgy, hogy az a szeszesital adatait az alábbi formában adja vissza: „`<alkoholtartalom>% alkoholtartalmú <név>, <kiszerelés>, <ár> Ft`”! (Pl. „*11.5% alkoholtartalmú Kékfrankos, 0,75 l, 2490 Ft*”)! A megoldásban használja fel az örökölt **`toString()`** metódust!
4. Adott az alábbi interfész az **`ital`** csomagban:

```
public interface ItalBolt
{
  // új italt ad hozzá az italbolthoz
  public void hozzáad(Ital ital);

  // meghatározza az italok összértékét
  public int összÉrték();

  // visszaadja azoknak a szeszesitaloknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyek alkoholtartalma meghaladja a paraméterként
  // megkapott értéket
  public java.util.List<SzeszesItal> erősPiák(double limit);
}
```

Készítse el a **`Kocsma`** osztályt a **`vendéglátás`** csomagban, amely a megjegyzéseknek megfelelően implementálja az **`ItalBolt`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy a kocsmában ugyanolyan nevű és kiszerelésű ital több példányban is szerepelhet! Legyen lehetőség a kocsma megnevezésének a tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a kocsma nevét, valamint feltölthetjük a kínálatát egy italokat tartalmazó kollekció elemeivel! Figyeljen arra, hogy a paraméterként megkapott kollekció esetleges későbbi módosítása ne legyen hatással a kocsmában tárolt italokra, és fordítva! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a kocsma megnevezését, majd egy üres sort követően soronként az egyes italokat adja vissza! (2 pont)

5. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található italokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy ital adatait tartalmazza az alábbi formátumban:

   ```
   <megnevezés>;<kiszerelés>;<ár>[;<alkoholtartalom>]
   ```

   Ha az ár után folytatódik a sor, akkor szeszesitalról, ha nem, akkor alkoholmentes italról van szó. Például:

   ```
   Coca-Cola;5 dl;249
   Kékfrankos;0,75 l;2490;11,5
   Coca-Cola;1,5 l;329
   Coca-Cola;0,33 l;149
   Fanta;5 dl;249
   ```

   (2 pont)

6. Hozzon létre a főprogramban egy **Kocsma** objektumot, amelynek megnevezése legyen a második parancssori argumentum, vagy ennek hiányában „Egyetemi Büfike”! Töltse fel a kocsma kínálatát az előző feladatban beolvasott italokkal! Olvasson be a billentyűzetről egy valós számot, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a kínálatban szereplő azon szeszesitalokat, amelyek alkoholtartalma meghaladja a beolvasott értéket!

7. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy italokat tartalmazó kollekciót, egy italnevet és egy állománynevet, kiírja az állományba, hogy a megadott nevű ital milyen kiszerelésekben létezik a kollekcióban, és igazzal tér vissza, ha volt ilyen nevű ital, különben hamissal! Hívja meg a metódust a főprogramban az 5. feladatban létrehozott listával, valamint egy billentyűzetről beolvasott italnévvel és állománynévvel, majd írja a képernyőre a „Nincs ilyen nevű ital!” szöveget, ha a kollekcióban nem szerepelt ilyen nevű ital!

8. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy italokat tartalmazó tömböt, és visszaad egy olyan halmazt (**java.util.HashSet**), amely a tömbben található, 10 karakternél hosszabb nevű szeszesitalokat tartalmazza! Hívja meg a metódust a főprogramban az 5. feladatban létrehozott listából készült tömbbel, és írja ki a képernyőre soronként az eredményül kapott halmazban szereplő szeszesitalokat!

9. Egészítse ki a **Kocsma** osztályt egy olyan metódussal, amely paraméterként megkap egy logikai értéket, és visszaadja a kocsmában kapható italok átlagárát, ha a paraméter hamis, illetve a kocsmában kapható szeszesitalok átlagárát, ha a paraméter igaz! Ha a kocsmában egyetlen ital, illetve egyetlen szeszesital sincs, akkor a metódus dobjon **NincsElégPiaException** kivételt „Szomjazom!” üzenettel! (A kivétel osztályát is meg kell írni!) Hívja meg a metódust a főprogramban a 6. feladatban létrehozott kocsmára igaz értékű paraméterrel, és írja ki az eredményt a képernyőre! Kezelje a kivételt az üzenetének kiírásával!

10. Egészítse ki a **Kocsma** osztályt egy olyan metódussal, amely „5 dl”-re módosítja a kocsmában található olyan szeszesitalok kiszerelését, amelyeknek eddig „0,5 l” volt! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott kocsmára, majd írja ki a képernyőre a módosított kocsma objektumot!

11. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy kocsmákat tartalmazó tömböt, és visszatér a tömbbeli kocsmákban található italok összértékével! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott kocsmát tartalmazó egyelemű tömbbel, és írja ki a képernyőre a kapott összértéket!

Magas szintű programozási nyelvek 2 beugró (levelező tagozat) **2018. január 4.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el a **`Számonkérés`** nevű absztrakt osztályt az **`egyetem`** csomagban! Minden számonkérésről tárolni szeretnénk a megnevezését (sztring), a kezdési idejét (dátum és idő), hogy írásbeli-e vagy szóbeli (logikai), valamint az egyes feladatok pontszámait (egészeket tartalmazó tömb). (Tipp: a dátum és az idő tárolásához lásd például a **`Calendar`** vagy a **`LocalDateTime`** osztályokat!) Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait és a megnevezés beállító metódusát! Egészítse ki az osztályt egy olyan publikus metódussal, amely visszaadja a kezdési idő egy tetszőleges sztring reprezentációját, amelyben szerepel az év, a hónap, a nap, az óra és a perc (lásd a példát a következő feladatban)!
2. Egészítse ki a **`Számonkérés`** osztályt egy olyan publikus metódussal, amely megadja a számonkérésen elérhető maximális pontszámot! Definiálja felül az **`equals()`** metódust: két számonkérés akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik (a típusuktól függetlenül)! Definiálja felül a **`toString()`** metódust is úgy, hogy az a számonkérés adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név> [írásbeli|szóbeli] számonkérés (<kezdési idő>)`”! (Pl. „*Magas szintű programozási nyelvek 2 írásbeli számonkérés (2018.01.04. 13:00)*”.)
3. A számonkérések természetes rendezettsége a kezdési idejük szerint csökkenő, azon belül a nevük szerint növekvő sorrendben legyen értelmezve!
4. Származtasson a **`Számonkérés`** osztályból egy **`Zh`** és egy **`Vizsga`** osztályt szintén az **`egyetem`** csomagban! Írjon a két osztályhoz egy-egy konstruktort, amelyek segítségével mindegyik adattagjuknak kezdőérték adható, kivéve a zh-k esetén a logikai adattagot; a zh-k legyenek mindig írásbeliek! Definiálja felül mindkét osztályban a **`toString()`** metódust úgy, hogy azok az örökölt **`toString()`** által visszaadott sztringben cseréljék ki a „számonkérés” szót a „zh”, illetve a „vizsga” szóra! (Pl. „*Adatbázisrendszerek írásbeli vizsga (2018.01.12. 10:00)*”)! (Tipp: lásd a **`String.replace()`** metódust!)
5. Adott az alábbi interfész az **`egyetem`** csomagban:

```
public interface Tanulnivalók
{
  // új számonkérést ad hozzá a tanulnivalókhoz
  public void hozzáad(Számonkérés teszt);

  // visszaadja a számonkérések átlagos pontszámát; kivételt dob, ha egyetlen
  // számonkérés sincs a nyilvántartásban
  public double átlagPontszám() throws NincsTesztException;

  // visszaadja a számonkérések listáját a természetes rendezettségük sorrendjében;
  // ha az első paraméter igaz, akkor csak a vizsgákat, ha hamis, akkor csak a
  // zh-kat, ha null, akkor mindegyiket; ha a második paraméter igaz, akkor csak az
  // írásbeliket, ha hamis, akkor csak a szóbeliket, ha null, akkor mindegyiket
  public java.util.List<Számonkérés> tesztek(Boolean vizsgák, Boolean írásbelik);
}
```

Készítse el a **`FélévesTesztek`** osztályt az **`unideb`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`Tanulnivalók`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Legyen lehetőség a félév azonosítójának a tárolására (pl. „2017/2018/1. félév”)! Figyeljen arra, hogy a félév során több azonos nevű számonkérés is előfordulhat! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a félév azonosítóját, valamint a félév során előforduló számonkéréseket egy kollekció elemeivel! Figyeljen arra, hogy a paraméterként megkapott kollekció esetleges későbbi módosítása ne legyen hatással a féléves számonkérésekre, és fordítva! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a félév azonosítóját, majd egy üres sort követően soronként az egyes számonkéréseket adja vissza! (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található számonkéréseket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy számonkérés adatait tartalmazza az alábbi formátumban:

   ```
   [V|Z];<megnevezés>;<év>.<hónap>.<nap>;<óra>:<perc>;[[I|S]];<pont>[,<pont>]...
   ```

   Ha a sor első karaktere `V`, akkor vizsgáról, ha `Z`, akkor zh-ról van szó. Az év mindig 4, a hónap, nap, óra és perc pedig 2 karakterrel megadott egész szám, amelyek szabályos dátumot, illetve időt határoznak meg. Zh-k esetén a kezdési időt egyből a pontszámok követik (legalább egy), vizsgák esetén az `I` írásbeli, az `S` pedig szóbeli vizsgát jelöl. Például:

   ```
   Z;Magas szintű programozási nyelvek 2;2018.01.04;13:00;1,1,1,2,2,1,1,1,1,1,1
   V;Adatbázisrendszerek;2017.02.10;10:00;I;14,8,14,10,18,8,12,8,8
   V;Adatbázisrendszerek;2018.01.12;10:00;I;14,15,15,10,4,8,8,8,18
   V;A mesterséges intelligencia alapjai;2017.06.01;08:00;S;1
   Z;Adatbázisrendszerek;2018.01.04;10:00;1,1,1,1,1,1,1,1,1,1
   ```

   (Tipp: a kezdési idő létrehozásához lásd a **Calendar.Builder** osztályt, illetve a **LocalDateTime.of()** metódust!) (2 pont)

7. Hozzon létre a főprogramban egy **FélévesTesztek** objektumot, amelyben a félév azonosítója legyen a második parancssori argumentum, vagy ennek hiányában „2017/2018/1. félév”! Töltse fel a félév tesztjeit az előző feladatban beolvasott számonkérésekkel! Írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a félév írásbeli számonkéréseit!

8. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy számonkéréseket tartalmazó kollekciót, egy megnevezést és egy állománynevet! Ha a kollekcióban nem található számonkérés a kapott megnevezéssel, akkor a metódus adjon vissza hamis értéket! Ha található, akkor írja ki az állományba soronként a megadott névvel rendelkező számonkérések kezdési idejét és az elérhető pontszámot, majd egy külön sorba írja ki a felsorolt számonkéréseken elérhető összpontszámot, végül térjen vissza igazzal! A metódus egy magyar nyelvű hibaüzenettel jelezze, ha nem sikerült megnyitni az állományt! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott listával, valamint egy billentyűzetről beolvasott megnevezéssel és állománynévvel, majd írja a képernyőre a „Nincs ilyen nevű számonkérés!” szöveget, ha a kollekcióban nem szerepelt ilyen nevű számonkérés! Ne feledje, hogy mind a számonkérés megnevezése, mind az állománynév tartalmazhat szóköz karaktert!

9. Egészítse ki a **FélévesTesztek** osztályt egy **java.util.Set<Zh>** típusú metódussal, amely visszatér azoknak a zh-knak a halmazával, amelyeknél az elérhető pontszám nagyobb a számonkérések átlagos pontszámánál! A megoldásban használja fel az **átlagPontszám()** metódust, és dobja tovább a **NincsTesztException** kivételt, ha egyetlen számonkérés sincs a nyilvántartásban! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **FélévesTesztek** objektumra, és írja ki a képernyőre soronként az eredményül kapott halmazban szereplő számonkéréseket! Kezelje a metódus által dobott kivételt egy magyar nyelvű üzenet kiírásával!

10. Egészítse ki a **FélévesTesztek** osztályt egy olyan metódussal, amely paraméterként megkap egy évszámot, és csupa nagybetűssé alakítja azoknak a számonkéréseknek a megnevezését, amelyek kezdési ideje a paraméterként megkapott évre esik! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **FélévesTesztek** objektumra 2018-as évszámmal, majd írja ki a képernyőre a módosított objektumot! (Tipp: lásd a **String.toUpperCase()** metódust!)

11. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy **Tanulnivalók** típusú tömböt, és visszatér a tömbbeli féléves tesztekben szereplő szóbeli vizsgák darabszámával! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott **FélévesTesztek** objektumot tartalmazó egyelemű tömbbel, és írja ki a képernyőre a kapott darabszámot!

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2018. december 15.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el a **`Repülőgép`** osztályt a **`repülés`** csomagban! A repülőgépekről tudni szeretnénk a gyártójukat (sztring), a típusukat (sztring), a hosszukat méterben (valós) és hogy sugárhajtásúak-e (logikai). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **`IllegalArgumentException`** kivételt „*A hossz csak pozitív lehet!*” üzenettel, ha a paramétereként megkapott hossz nem pozitív! Írja meg a gyártó és a logikai adattag lekérdező metódusát! Definiálja felül az **`equals()`** metódust: két repülőgép akkor legyen egyenlő, ha a gyártójuk és típusuk (tartalmilag) megegyezik! Az összehasonlítás ne különböztesse meg a kis- és nagybetűket! Egy repülőgép bármelyik másik repülőgéppel (akár **`Utasszállító`** típusúval is) legyen összehasonlítható! (Segítség: lásd a **`String.equalsIgnoreCase()`** metódust.)
2. Definiálja felül a **`toString()`** metódust úgy, hogy az a repülőgép adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<gyártó> <típus> [sugárhajtású ]repülőgép, hossza: <hossz> m`” (pl. „*Airbus A380-800 sugárhajtású repülőgép, hossza: 72.72 m*”)!
3. A repülőgépek természetes rendezettsége a gyártójuk, azon belül pedig a típusuk szerint legyen értelmezve! Az összehasonlítás ne különböztesse meg a kis- és nagybetűket! (Segítség: lásd a **`String.compareToIgnoreCase()`** metódust.)
4. Származtasson a **`Repülőgép`** osztályból egy **`Utasszállító`** osztályt szintén a **`repülés`** csomagban! Az utasszállítók a repülőgép tulajdonságain kívül még egy más osztályból nem látható, egész értékű férőhely tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg a férőhely lekérdező és beállító metódusait! Definiálja felül a **`toString()`** metódust úgy, hogy az a repülőgép adataihoz fűzze hozzá, hogy „`, <férőhely> férőhelyes`” (pl. „*Boeing 747-8I sugárhajtású repülőgép, hossza: 76.25 m, 467 férőhelyes*”)!
5. Adott az alábbi interfész a **`repülés`** csomagban:

```
public interface Flotta
{
  // új repülőgépet vesz fel a flottába
  public void felvesz(Repülőgép repülő);

  // visszaadja azoknak az utasszállítóknak a rendezett listáját (a természetes
  // rendezettség sorrendjében), amelyek legalább a paraméterben megkapott számú
  // férőhellyel rendelkeznek
  public java.util.List<Utasszállító> megfelelőGépek(int utasszám);

  // kiírja a megadott állományba a repülőgépek listáját (mindegyiket külön sorba)
  public void kiír(String fájlnév);
}
```

Készítse el a **`Légitársaság`** osztályt a **`légiKözlekedés`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`Flotta`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy légitársaság több azonos márkájú és típusú repülőgéppel is rendelkezhet! Legyen lehetőség a légitársaság nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével nevet adhatunk a légitársaságnak, valamint feltölthetjük a flottáját egy repülőgépeket tartalmazó tömb elemeivel! A **`kiír()`** metódust úgy implementálja, hogy az állomány első sorába a légitársaság neve kerüljön! (Segítség: lásd a **`Collections.sort()`** vagy a **`List.sort()`** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **RepülőTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található repülőgépeket egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy repülőgép adatait tartalmazza az alábbi formátumban:

   ```
   [R|U];<gyártó>;<típus>;<hossz>;[S|N][;<férőhely>]
   ```

   Ha a sor `R`-rel kezdődik, akkor egyszerű repülőgépről, ha `U`-val, akkor utasszállítóról van szó. A férőhely csak utasszállítók esetén van megadva. Az `S` betű azt jelenti, hogy a repülőgép sugárhajtású, az `N` pedig, hogy nem. Például:

   ```
   R;Boeing;747-8F;76,25;S
   U;Bombardier;Q400;32,81;N;78
   ```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, de a kivétel előfordulása ne szakítsa meg a beolvasást és a lista feltöltését! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **next()**, **nextInt()**, **nextDouble()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **Légitársaság** objektumot! A légitársaság neve legyen a program második parancssori argumentuma, vagy ha az nem létezik, akkor legyen „Unideb Airlines”! Töltse fel a légitársaság flottáját az előző feladatban létrehozott listában található repülőgépekkel! Olvasson be a billentyűzetről egy darabszámot, és írja ki a képernyőre azoknak az utasszállítóknak a rendezett listáját (a természetes rendezettségük sorrendjében), amelyek legalább a beolvasott darabszámú férőhellyel rendelkeznek! (Segítség: lásd a **Collection.toArray()** metódust.)

8. Egészítse ki a **RepülőTeszt** osztályt egy metódussal, amely paraméterként megkap egy repülőgépeket tartalmazó kollekciót, valamint egy gyártót, és képernyőre írja soronként azoknak a kollekcióban található repülőgépeknek a jellegét (egyszerű repülőgép vagy utasszállító) és egyéb adatait, amelyeknek a gyártója a kis- és nagybetűktől eltekintve megegyezik a kapott gyártóval! A főprogramban kérjen be a billentyűzetről egy gyártót, majd hívja meg a metódust a 6. feladatban létrehozott listával és a beolvasott értékkel!

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2018. december 17.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el a **`Versenyző`** absztrakt osztályt a **`verseny`** csomagban! A versenyzőkről tudni szeretnénk az azonosítójukat (egész), a nevüket (sztring) és hogy milyen területen versenyeznek (sztring). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Írja meg az adattagok lekérdező metódusait! Definiálja felül az **`equals()`** metódust: két versenyző akkor legyen egyenlő, ha az azonosítójuk megegyezik (függetlenül a két versenyző konkrét típusától; lásd a 4. feladatot)!
2. Definiálja felül a **`toString()`** metódust úgy, hogy az a versenyző adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név> (<terület>), azonosítója: <azonosító>`” (pl. „*Lewis Hamilton (sport/Formula 1), azonosítója: 44*”)!
3. A versenyzők természetes rendezettsége a versenyzési területük szerint lexikografikusan növekvő, azon belül a nevük szerint lexikografikusan növekvő, azon belül pedig az azonosítójuk szerint növekvő sorrend szerint legyen értelmezve!
4. Származtasson a **`Versenyző`** osztályból egy **`Egyéni`** és egy **`Csapat`** osztályt szintén a **`verseny`** csomagban! Az egyéni versenyzők a versenyző tulajdonságain kívül még egy más osztályból nem látható, egész értékű életkor tulajdonsággal is rendelkeznek. A csapatok esetében legyen lehetőség a csapattagok nevének a tárolására (akárhány csapattag lehet)! Ez az adattag se legyen látható más osztályból! Készítsen a két osztályhoz egy-egy konstruktort, amelyek segítségével mindegyik adattagjuknak kezdőérték adható! Írja meg az új adattagok lekérdező metódusait! Definiálja felül a **`toString()`** metódust mindkét osztályban úgy, hogy az a versenyző adataihoz fűzze hozzá az elején, hogy „`egyéni versenyző: `”, illetve „`csapat: `”, valamint a végére az életkort, illetve a csapattagok nevét, például így: „*egyéni versenyző: Arany Tímea (ének), azonosítója: 250, kora: 19 év*”, illetve: „*csapat: USNK (ének), azonosítója: 3178, tagjai: Laskay Nimród, Filep Kund*”)!
5. Adott az alábbi interfész a **`verseny`** csomagban:

```
public interface Verseny
{
  // új versenyzőt nevez a versenybe, ha még eddig nem volt
  public void nevez(Versenyző versenyző);

  // visszaléptet egy azonosítóval megadott versenyzőt a versenyből; igazzal tér
  // vissza, ha a megadott versenyző nevezve volt a versenybe, különben hamissal
  public boolean visszalép(int azonosító);

  // visszaad egy kételemű tömböt, amelynek az első eleme a versenybe nevezett
  // egyéni versenyzők, a második eleme pedig a versenybe nevezett csapatok számát
  // adja meg
  public int[] versenyzőkSzáma();
}
```

Készítse el az **`XFaktor`** osztályt a **`dalverseny`** csomagban, amely a megjegyzéseknek megfelelően implementálja a **`Verseny`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Legyen lehetőség az X-Faktor évadának tárolására! Figyeljen arra, hogy az X-Faktor egy évadába egy versenyző csak egyszer nevezhet! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk az X-Faktor évadát, valamint a versenyzőket is egy versenyzőket tartalmazó tömb elemeivel! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban az „X-Faktor” szót és a verseny évadát adja meg (pl. „X-Faktor - 2018”), majd egy üres sort követően soronként az egyes versenyzőket sorolja fel a természetes rendezettségük sorrendjében! (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **VersenyTeszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található versenyzőket egy **ArrayList** objektumba! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy versenyző adatait tartalmazza az alábbi formátumban:

```
E;<azonosító>;<név>;<terület>;<életkor>
C;<azonosító>;<név>;<terület>;<csapattagnév>[/<csapattagnév>]...
```

Ha a sor `E`-vel kezdődik, akkor egyéni versenyzőről, ha `C`-vel, akkor csapatról van szó. Figyeljen arra, hogy a név és a terület tartalmazhat perjelet, a csapattagnevek pedig pontosvesszőt! Például:

```
E;250;Arany Tímea;ének;19
C;3178;USNK;ének;Laskay Nimród/Filep Kund
```

(2 pont)

7. Hozzon létre a főprogramban egy **XFaktor** objektumot! A verseny évada legyen a program második parancssori argumentuma, vagy ha az nem létezik, akkor legyen 2018! Nevezze be a versenybe az előző feladatban létrehozott listában található versenyzők közül azokat, akiknek a területe „ének”! Olvasson be a billentyűzetről egy azonosítószámot, és léptesse vissza a versenyből a megadott azonosítóval rendelkező versenyzőt! Ha a megadott azonosítóval nincs benevezett versenyző, akkor írjon ki egy értelemszerű üzenetet! Ellenőrzésképpen írja ki a képernyőre a módosított **XFaktor** objektumot!

8. Egészítse ki a **VersenyTeszt** osztályt egy metódussal, amely paraméterként megkap egy versenyeket tartalmazó kollekciót, valamint egy állománynevet, és kiírja a megadott nevű állományba azokat a versenyeket, amelyekbe több csapat nevezett, mint egyéni versenyző! A főprogramban kérjen be a billentyűzetről egy állománynevet, majd hívja meg a metódust a 7. feladatban létrehozott **XFaktor** objektumot tartalmazó egyelemű, tetszőleges típusú kollekcióval és a beolvasott értékkel!

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2019. december 18.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el az **`Alprogram`** absztrakt osztályt az **`ansiC`** csomagban! Az alprogramokról tárolni szeretnénk a nevüket sztringként és a paramétereik típusait sztringek egy listájaként. Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindkét adattagjának kezdőérték adható! A konstruktor dobjon **`IllegalArgumentException`** kivételt „*Hibás alprogramnév!*” üzenettel, ha a paraméterként megkapott név **`null`**, üres sztring, vagy nem betűvel kezdődik! Definiálja felül az **`equals()`** metódust: két alprogram akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik (a fajtájuktól függetlenül)! Definiálja felül a **`toString()`** metódust úgy, hogy az az alprogram adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<név>([<típus>[,<típus>]...])`” (pl. „*strchr(const char *,int)*”)! (Segítség: lásd a **`Character.isLetter()`**, a **`String.charAt()`** és a **`String.join()`** metódusokat.)
2. Egészítse ki az **`Alprogram`** osztályt egy **`kisbetűsít()`** nevű publikus logikai metódussal, amelynek segítségével kisbetűssé alakítható az alprogram neve, valamint a paramétereinek típusai! A metódus adjon vissza igazat, ha az alprogram neve vagy valamelyik típusnév tartalmazott nagybetűt (azaz a metódus módosította az alprogramot), különben pedig hamisat! (Segítség: lásd a **`String.toLowerCase()`** metódust.)
3. Az alprogramok természetes rendezettsége a paraméterszámuk szerint növekvő, azon belül a nevük szerint csökkenő sorrendben legyen értelmezve!
4. Származtasson az **`Alprogram`** osztályból egy **`Eljárás`** és egy **`Függvény`** osztályt szintén az **`ansiC`** csomagban! A függvények esetén azok típusát is szeretnénk még tárolni sztringként. Az új adattag csak az **`ansiC`** csomagban látszódjon! Készítsen a két osztályhoz olyan konstruktorokat, amelyek segítségével mindegyik adattagjuknak kezdőérték adható! Definiálja felül mindkét osztályban a **`toString()`** metódust úgy, hogy az az **`Alprogram`** sztring reprezentációját adja vissza, kiegészítve azt elöl az alprogram típusával (eljárás esetén ez legyen „void”) és egy szóközzel, hátul pedig egy pontosvesszővel (pl. „*char * strchr(const char *,int);*”)! A **`Függvény`** osztályban definiálja felül a **`kisbetűsít()`** metódust is úgy, hogy a függvény nevén és a paramétereinek típusain kívül a függvény típusát is alakítsa kisbetűssé! A metódus visszatérési értéke továbbra is akkor legyen igaz, ha módosította az alprogramot, azaz vagy a függvény neve, vagy a paramétereinek típusai, vagy a függvény típusa tartalmazott nagybetűt!
5. Adott az alábbi interfész a **`programozás`** csomagban:

```
public interface FordításiEgység
{
  // hozzáadja a paraméterként megkapott tömbben szereplő alprogramokat a
  // fordítási egységhez
  public void hozzáad(Alprogram[] alp);

  // visszaadja a fordítási egységben található olyan alprogramok listáját a
  // természetes rendezettségük sorrendjében, amelyek a második paraméterként
  // megkapott darabszámú paraméterrel rendelkeznek; ha az első paraméter igaz,
  // akkor csak a függvényeket, ha hamis, akkor csak az eljárásokat, ha pedig
  // null, akkor mindegyiket
  public java.util.List<Alprogram> alprogramok(Boolean függvény, int paraméterSzám);

  // minden „void” típusú függvényhez létrehoz a fordítási egységben egy
  // eljárást, amelynek a neve a függvény neve kiegészítve az elején egy P
  // betűvel, paraméterei pedig a függvény paraméterei
  public void voidFüggvényekbőlEljárás();
}
```

Készítse el a **Forrásállomány** osztályt az **ansiC** csomagban, amely a megjegyzéseknek megfelelően implementálja a **FordításiEgység** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy forrásállomány nem tartalmazhat azonos nevű alprogramokat! Legyen lehetőség a forrásállomány nevének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a forrásállomány nevét, valamint feltölthetjük azt egy alprogramokat tartalmazó kollekció elemeivel! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban a forrásállomány nevét, majd egy üres sort követően soronként az egyes alprogramokat adja vissza! (Segítség: lásd a **Collections.sort()** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található alprogramokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy alprogram adatait tartalmazza az alábbi formátumban:

```
[procedure|function] <név>([<par>:<típus>[,<par>:<típus>]...])[:<típus>]
```

Ha a sor a „procedure” szóval kezdődik, akkor eljárásról, ha a „function” szóval, akkor függvényről van szó. A sor végén a kettőspont és az azt követő típus (az alprogram típusa) csak függvények esetén van megadva. Például:

```
function strchr(s:const char *,c:int):char *
function malloc(size:size_t):void *
procedure free(ptr:void *)
```

Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, a beolvasás megszakítása nélkül! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **Forrásállomány** objektumot! A forrásállomány neve legyen a második parancssori argumentum, vagy ennek hiányában „teszt.c”! Vegye fel a forrásállományba az előző feladatban beolvasott alprogramokat! Olvasson be a billentyűzetről egy egész értéket, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében a forrásállomány azon alprogramjait, amelyek a beolvasott darabszámú paraméterrel rendelkeznek!

8. Egészítse ki a **Forrásállomány** osztályt egy publikus metódussal, amely visszaadja a forrásállomány mutató visszatérési típussal rendelkező függvényeinek számát (a mutató típusok csillag karakterre végződnek)! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott forrásállományra, és írja ki az eredményt a képernyőre! (Segítség: lásd a **String** osztály **endsWith()** metódusát.)

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2020. január 8.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el a **`Térkép`** osztályt a **`kartográfia`** csomagban! A térképekről tárolni szeretnénk a címüket, azaz hogy mit ábrázolnak (sztring), a valós méretekhez viszonyított méretarányukhoz tartozó arányszámot (egész) és egy névjegyzéket (sztringek listája). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! A konstruktor dobjon **`IllegalArgumentException`** kivételt „*Hibás méretarány!*” üzenettel, ha a paraméterként megkapott arányszám nem pozitív! Definiálja felül az **`equals()`** metódust: két térkép akkor legyen egyenlő, ha a címük és a méretarányuk (tartalmilag) megegyezik (a típusuktól függetlenül)! Definiálja felül a **`toString()`** metódust úgy, hogy az a térkép adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<cím>, 1:<arányszám>[ (<név>[, <név>]...)]`” (pl. „*Debrecen, 1:20000 (Piac u., Nagytemplom, DE Főépület)*”)! (Segítség: lásd a **`String.join()`** metódust.)
2. Egészítse ki a **`Térkép`** osztályt egy **`nagybetűsít()`** nevű publikus logikai metódussal, amelynek segítségével nagybetűssé alakíthatók a névjegyzékben szereplő nevek kezdőbetűi! Felteheti, hogy minden név tartalmaz legalább egy karaktert. A metódus adjon vissza igazat, ha volt kisbetűvel kezdődő név a névjegyzékben (azaz a metódus módosította a térképet), különben pedig hamisat! (Segítség: lásd a **`List`** interfész **`get()`** és **`set()`**, a **`String`** osztály **`charAt()`** és **`substring()`**, valamint a **`Character`** osztály **`isLowerCase()`** és **`toUpperCase()`** metódusait.)
3. A térképek természetes rendezettsége a címük szerint lexikografikusan növekvő sorrendben legyen értelmezve! Ha két térképnek azonos a címe, akkor közülük a részletesebb (nagyobb méretarányú) kerüljön előrébb a rendezettségi sorrendben (az 1:1000 méretarányú térkép részletesebb, mint az 1:10000 méretarányú)! Ha a méretarányok is megegyeznek, akkor a nagyobb névjegyzékkel rendelkező térkép kerüljön előrébb (azaz amelyikben több név szerepel)!
4. Származtasson a **`Térkép`** osztályból egy **`TematikusTérkép`** osztályt szintén a **`kartográfia`** csomagban! A tematikus térképekről tárolni szeretnénk még azok témáját is (sztringként). Az új adattag csak a **`kartográfia`** csomagban látszódjon! Készítsen az osztályhoz egy olyan konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Definiálja felül a **`toString()`** metódust úgy, hogy az a **`Térkép`** sztring reprezentációját adja vissza, kiegészítve azt a végén egy vesszővel, egy szóközzel és a térkép témájával (pl. „*Magyarország, 1:1100000, domborzat*”)! Definiálja felül a **`nagybetűsít()`** metódust is úgy, hogy a térkép névjegyzékében szereplő nevek kezdőbetűin kívül a térkép témájának kezdőbetűjét is alakítsa nagybetűssé! A metódus visszatérési értéke továbbra is akkor legyen igaz, ha módosította a térképet, azaz vagy a névjegyzékben szereplő nevek valamelyike, vagy a téma kisbetűvel kezdődött!
5. Adott az alábbi interfész a **`térképKiadó`** csomagban:

```
public interface TérképTár
{
  // hozzáadja a paraméterként megkapott tömbben szereplő térképeket a
  // térképtárhoz
  public void hozzáad(Térkép[] térképek);

  // visszaadja a térképtárban található olyan térképek listáját a természetes
  // rendezettségük sorrendjében, amelyek névjegyzékében legalább a második
  // paraméterként megkapott darabszámú név szerepel; ha az első paraméter igaz,
  // akkor csak a tematikus térképeket, ha hamis, akkor mindegyiket
  public java.util.List<Térkép> térképek(boolean csakTematikus, int nevekSzáma);

  // visszaadja a paraméterként megkapott címmel rendelkező térképek névjegyzékeinek
  // az unióját lexikografikusan növekvő sorrendben; ügyeljen arra, hogy egy elem
  // csak egyszer fordulhat elő a teljes névjegyzékben!
  public java.util.Collection<String> teljesNévjegyzék(String cím);
}
```

Készítse el az **Atlasz** osztályt a **kartográfia** csomagban, amely a megjegyzéseknek megfelelően implementálja a **TérképTár** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy atlasz több azonos című és méretarányú térképpel is rendelkezhet! Legyen lehetőség az atlasz címének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk az atlasz címét, valamint feltölthetjük azt egy térképeket tartalmazó (tetszőleges) kollekció elemeivel! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban az atlasz címét, majd egy üres sort követően soronként az egyes térképeket adja vissza! (Segítség: lásd a **TreeSet** osztályt, valamint a **Collections.addAll()** és a **Collections.sort()** metódusokat.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található térképeket, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy térkép adatait tartalmazza az alábbi formátumban:

   ```
   <cím>;1:<arányszám>;[<név>[:<név>]...][;<téma>]
   ```

   Ha a névjegyzék után folytatódik a sor, akkor tematikus térképről, ha nem, akkor hagyományos térképről van szó. Ha a térkép névjegyzéke üres, akkor hagyományos térkép esetén a sort egy pontosvessző zárja, míg tematikus térkép esetén a méretarányt és a témát két pontosvessző választja el egymástól. Például:

   ```
   Debrecen;1:20000;Piac u.:Nagytemplom:DE Főépület
   Föld;1:200000000;Európa:Ázsia:Afrika:Amerika:Ausztrália:Antarktisz;Édesvízkészletek
   Magyarország;1:1100000;
   Magyarország;1:1100000;;domborzat
   Kincses térkép;1:1000;házikó:kincs:nagy tölgyfa
   ```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének kiírásával, a beolvasás megszakítása nélkül! Figyeljen arra, hogy a cím és a téma tartalmazhat kettőspont karaktereket, a nevek viszont nem tartalmaznak pontosvesszőt! (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNext()**, **next()**, **nextInt()** metódusait.) (2 pont)

7. Hozzon létre a főprogramban egy **Atlasz** objektumot! Az atlasz neve legyen a második parancssori argumentum, vagy ennek hiányában „Földrajzi világatlasz”! Vegye fel az atlaszba az előző feladatban beolvasott térképeket! Olvasson be a billentyűzetről egy egész értéket, és írja ki a képernyőre soronként a természetes rendezettségük sorrendjében az atlasz azon tematikus térképeit, amelyek névjegyzékében legalább a beolvasott darabszámú név szerepel!

8. Egészítse ki az **Atlasz** osztályt egy publikus metódussal, amely visszaadja az atlaszban található, paraméterként megadott témájú tematikus térképek számát! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott atlaszra egy billentyűzetről beolvasott témával, és írja ki az eredményt a képernyőre!

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2020. december 21.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

1. Készítse el az **`Ajándék`** osztályt a **`mikulás`** csomagban! Az ajándékokról tárolni szeretnénk a nevüket (sztring), a tömegüket kg-ban (valós) és az árukat Ft-ban (egész). Az osztály adattagjai legyenek védett láthatóságúak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindhárom adattagjának kezdőérték adható! Definiálja felül az **`equals()`** metódust: két ajándék akkor legyen egyenlő, ha a nevük (tartalmilag) megegyezik, és a tömegük legfeljebb 1 kg-mal tér el egymástól (a típusuktól függetlenül)! Definiálja felül a **`toString()`** metódust úgy, hogy az az ajándék adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül):
„`<név> (<tömeg> kg), <ár> Ft`” (pl. „*nyakkendő (0.1 kg), 4000 Ft*”)! (Segítség: lásd a **`Math.abs()`** metódust.)
2. Az ajándékok természetes rendezettsége az áruk szerint csökkenő, azon belül a nevük szerint lexikografikusan növekvő sorrendben legyen értelmezve!
3. Származtasson az **`Ajándék`** osztályból egy **`GyerekJáték`** osztályt szintén a **`mikulás`** csomagban! A gyerekjátékok az ajándék tulajdonságain kívül még egy egész értékű, más osztályból nem látható, évben mért korhatár tulajdonsággal is rendelkeznek. Készítsen az osztályhoz egy olyan konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! Írja meg a korhatárhoz tartozó lekérdező metódust! Definiálja felül a **`toString()`** metódust úgy, hogy az az **`Ajándék`** sztring reprezentációját adja vissza, kiegészítve azt a végén egy vesszővel, egy szóközzel és a korhatárral: „`<név> (<tömeg> kg), <ár> Ft, <korhatár> éves kortól`” (pl. „*Monopoly (0.8 kg), 13500 Ft, 8 éves kortól*”)!
4. Adott az alábbi interfész a **`mikulás`** csomagban:

```
public interface AjándékCsomag
{
  // visszaadja az ajándékok összértékét
  int összÉrték();

  // visszaadja a t-nél nehezebb ajándékok számát
  int nehezekSzáma(double t);

  // visszaadja a gyerekjátékokat a természetes rendezettségük sorrendjében úgy,
  // hogy az egyenlő gyerekjátékok közül csak egy szerepel a kollekcióban
  java.util.Collection<GyerekJáték> gyerekjátékok();
}
```

Készítse el a **`Puttony`** osztályt a **`mikulás`** csomagban, amely a megjegyzéseknek megfelelően implementálja az **`AjándékCsomag`** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy puttonyban több egyforma ajándék is előfordulhat! Legyen lehetőség a puttony címzettjének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a puttony címzettjét, valamint feltölthetjük azt egy ajándékokat tartalmazó tömb elemeivel! Definiálja felül a **`toString()`** metódust úgy, hogy az az első sorban a puttony címzettjét, majd egy üres sort követően soronként az egyes ajándékokat adja vissza! (Segítség: lásd a **`Collection.addAll()`** és az **`Arrays.asList()`** metódusokat.) (2 pont)

5. Helyezze el a főprogramot a **télapó** csomag **Télapó** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található ajándékokat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy ajándék adatait tartalmazza az alábbi formátumban:

   ```
   <név>;<tömeg>;<ár>[;<korhatár>]
   ```

   Ha az ár után folytatódik a sor, akkor gyerekjátékról, ha nem, akkor egyszerű ajándékról van szó. Például:

   ```
   nyakkendő;0,1;4000
   5 kg-os súlyzó;5;5555
   kisautó;0,67;3999;2
   Monopoly;0,8;13500;8
   dominó;0,4;2500;4
   kisautó;1,2;12000;3
   sakk-készlet;1,1;8000
   ```

   (Segítség: lásd a **Scanner** osztály **useDelimiter()**, **hasNextLine()**, **nextLine()**, **hasNextInt()**, **next()**, **nextInt()**, **nextDouble()** metódusait.) (2 pont)

6. Hozzon létre a főprogramban egy **Puttony** objektumot! A puttony címzettje legyen a második parancssori argumentum, vagy ennek hiányában „*a világ összes gyereke*”! Töltse meg a puttonyt az előző feladatban beolvasott ajándékokkal, majd írja ki a képernyőre a puttonyban található legdrágább gyerekjátékot, vagy ha ilyen nincs, akkor a „*Nincs gyerekjáték a puttonyban.*” üzenetet! (Segítség: lásd a **Collection.toArray()** metódust.)

7. Egészítse ki az **Puttony** osztályt egy publikus metódussal, amely paraméterként megkap egy egész számot, és visszaadja egy kollekcióban a puttonyban található gyerekjátékok közül azokat, amelyek korhatára nagyobb a kapott értéknél! Ha a paraméter értéke negatív, a metódus dobjon **IllegalArgumentException** kivételt „*Negatív életkor!*” üzenettel! Hívja meg a metódust a főprogramban a 6. feladatban létrehozott puttony objektumra egy billentyűzetről beolvasott életkor értékkel, és írja ki a képernyőre az eredményül kapott kollekciót! Amennyiben a metódus hívása **IllegalArgumentException** kivételt eredményezne, akkor a kivétel üzenetét (és csak azt) jelenítse meg!

8. Egészítse ki a **Télapó** osztályt egy metódussal, amely paraméterként megkap egy **AjándékCsomag** objektumokat tartalmazó tömböt, és visszaad egy másik tömböt, amelynek az *i*-edik eleme a paraméterként megkapott tömb *i*-edik eleme által reprezentált ajándékcsomagban található ajándékok átlagértékét tartalmazza pontos valós értékként! Felteheti, hogy minden ajándék pozitív tömeggel rendelkezik. Ha a paraméterként kapott tömb valamelyik eleme **null**, vagy egy üres ajándékcsomagot reprezentál, akkor a visszaadott tömb megfelelő eleme 0 legyen! Hívja meg a metódust a főprogramban egy kételemű tömbbel, amelynek az első eleme a 6. feladatban létrehozott puttony objektum, a második eleme pedig **null**, majd írja ki a képernyőre az eredményül kapott tömböt! (Segítség: ne feledje, hogy két egész szám hányadosa egész!)

Magas szintű programozási nyelvek 2 / Programozási nyelvek 2 / Programozás 2

gyakorlati számonkérés (levelező tagozat) **2021. január 5.**

A feladatsorban szereplőkön kívül csak privát láthatóságú adattagokat és metódusokat definiálhat!

Ebben a feladatsorban kórokozókkal (patogénekkel) foglalkozunk. A kórokozóknak itt most hat fajtáját különböztetjük meg (zárójelben a fajtát jelölő betűvel): prion (P), vírus (V), baktérium (B), gomba (G), növény (N), állat (A). A kórokozókat egy olyan osztályhierarchiával reprezentáljuk, amelynek a gyökerében a **`Kórokozó`** absztrakt osztály áll, melynek hat alosztálya a fent felsorolt hat fajtát reprezentálja. Ezeknek az alosztályoknak további leszármazottai is lehetnek az egyes fajták további kategorizálásától függően. Ebben a feladatsorban csak a vírusokat és a baktériumokat vesszük górcső alá.

1. Készítse el a **`Kórokozó`** absztrakt osztályt a **`patogenetika`** csomagban! A kórokozókról tárolni szeretnénk a fajtájukat (karakter), a megnevezésüket (sztring), az általuk okozott betegség nevét (sztring) és azokat a gazdatesteket, amelyekre nézve veszélyt jelenthetnek (sztringtömb). Az osztály adattagjai más osztályból ne látszódjanak! Készítsen az osztályhoz egy konstruktort, amelynek segítségével mindegyik adattagjának kezdőérték adható! A konstruktor dobjon **`IllegalArgumentException`** kivételt „*Érvénytelen fajta!*” üzenettel, ha a paraméterként megkapott fajtajelölő karakter nem a fent felsorolt hat lehetőség egyike, illetve ugyanilyen kivételt „*Érvénytelen név!*” üzenettel, ha a kapott megnevezés **`null`** vagy üres sztring! Írja meg az okozott betegséghez és a gazdatestekhez tartozó lekérdező metódusokat! (Segítség: lásd a **`String.indexOf()`** és a **`String.isEmpty()`** metódusokat.)
2. Definiálja felül az **`equals()`** metódust: két kórokozó akkor legyen egyenlő, ha a fajtájuk és a megnevezésük (tartalmilag) megegyezik (a konkrét típusuktól függetlenül)! Definiálja felül a **`toString()`** metódust úgy, hogy az a kórokozó adatait az alábbi formában adja vissza (egy sorban, idézőjelek nélkül): „`<fajta>: <megnevezés>; okozott betegség: [-|<betegség>]; gazdatestek: [-|<gazdatest>[, <gazdatest>]...]`”! A fajtát sztringként kell megadni (lásd a fent felsorolt fajtamegnevezéseket). A metódus legyen felkészítve mind a hat lehetséges kórokozófajta sztringgé alakítására! Az okozott betegség helyén egy kötőjel szerepeljen, ha az nem ismert (azaz **`null`** vagy üres sztring), mint ahogy a gazdatestek helyén is, ha az értéke **`null`** vagy üres tömb! A gazdatesteket egy vesszővel és egy szóközzel kell elválasztani egymástól. (Pl. „*vírus: EBOV; okozott betegség: ebola; gazdatestek: ember, denevér, majmok*”)! (Segítség: lásd a **`String.join()`** metódust.)
3. A kórokozók természetes rendezettsége a fajtájuk megnevezése szerint lexikografikusan növekvő, azon belül a megnevezésük szerint lexikografikusan növekvő sorrendben legyen értelmezve!
4. Származtasson a **`Kórokozó`** osztályból egy **`Vírus`** és egy **`Baktérium`** osztályt szintén a **`patogenetika`** csomagban! A vírusok a kórokozó tulajdonságain kívül egy logikai típusú tulajdonsággal is rendelkeznek, amely megmondja, hogy koronavírusról van-e szó. A baktériumokról pedig tárolni szeretnénk azt is, hogy melyik törzsbe tartoznak (sztringként). Az új adattagok legyenek védett láthatóságúak! Készítsen a két osztályhoz egy-egy olyan konstruktort, amelyek segítségével a fajtán kívül mindegyik adattagjuknak kezdőérték adható! A fajta vírusok esetén mindig V, míg baktériumok esetén B. Írja meg az új adattagok lekérdező metódusait! Definiálja felül mindkét osztályban a **`toString()`** metódust úgy, hogy az a **`Kórokozó`** sztring reprezentációját adja vissza, kiegészítve azt a végén egy pontosvesszővel, egy szóközzel, valamint vírusok esetén a „*koronavírus*” vagy a „*nem koronavírus*” sztringek egyikével, baktériumok esetén pedig a törzzsel! Pl.: „*vírus: SARS-CoV-2; okozott betegség: COVID-19; gazdatestek: ember, tobzoska, denevér; koronavírus*” vagy „*baktérium: Escherichia coli; okozott betegség: bélfertőzés; gazdatestek: ember, melegvérű állatok; proteobaktérium*”.

5. Adott az alábbi interfész a **patogenetika** csomagban:

```
public interface KórokozóTár
{
  // hozzáadja a megkapott tömbben található kórokozókat a kórokozótárhoz
  void hozzáad(Kórokozó[] kórokozók);

  // visszaadja egy listában a természetes rendezettségük sorrendjében azokat a
  // koronavírusokat, amelyek veszélyt jelentenek a megkapott gazdatestre nézve
  java.util.List<Vírus> koronavírusokGazdában(String gazdatest);

  // visszaadja lexikografikusan növekvő sorrendben a kórokozótárban szereplő
  // kórokozók által okozott összes betegséget úgy, hogy egy betegség csak egyszer
  // szerepel a kollekcióban (null értékek és üres sztringek nélkül!)
  java.util.Collection<String> betegségek();
}
```

   Készítse el a **Kutatólabor** osztályt a **kutatóintézet** csomagban, amely a megjegyzéseknek megfelelően implementálja a **KórokozóTár** interfészt! Az osztály nem tartalmazhat nyilvános adattagokat! Figyeljen arra, hogy egy kutatólaborban egy kórokozót csak egyszer vegyen nyilvántartásba! Legyen lehetőség a kutatólabor megnevezésének tárolására! Készítsen az osztályhoz egy konstruktort, amelynek segítségével megadhatjuk a kutatólabor megnevezését, valamint feltölthetjük a labor hűtőjét egy kórokozókat tartalmazó tetszőleges kollekció elemeivel (tehát lehessen megadni akár halmazt, akár listát, akár bármilyen más kollekciót)! Definiálja felül a **toString()** metódust úgy, hogy az az első sorban a kutatólabor megnevezését, majd egy üres sort követően soronként az ott tárolt kórokozókat adja vissza a természetes rendezettségük sorrendjében! (Segítség: lásd a **TreeSet** osztályt, valamint a **Collections.addAll()** metódust.) (2 pont)

6. Helyezze el a főprogramot a **teszt** csomag **Teszt** osztályában! A főprogramban olvassa be egy szöveges állományból az abban található kórokozókat, és tárolja el azokat egy listában! Az állomány nevét a program az első parancssori argumentumában kapja meg. Ha nincs megadva parancssori argumentum, vagy a megadott nevű állomány nem létezik, a program írjon ki egy értelemszerű, magyar nyelvű hibaüzenetet, és álljon meg! Az állomány soronként egy-egy kórokozó adatait tartalmazza az alábbi formátumban:

```
<fajta>:[<megnevezés>]:[<betegség>]:[<gazdatest>[,<gazdatest>]...]:<extra>
```

   A fajta csak `V` vagy `B` lehet. `V` esetén vírusról, `B` esetén baktériumról van szó. A megnevezés, az okozott betegség és a gazdatestek mind opcionálisak, az extra adat viszont mindig meg van adva. Ez utóbbi vírusok esetén vagy egy `K` betű (ekkor koronavírusról van szó), vagy egy `N` betű (ekkor nem koronavírusról van szó), baktériumok esetén pedig a baktériumtörzset adja meg sztringként. Például:

```
V:EBOV:ebola:ember,denevér,majmok:N
V:SARS-CoV-2:COVID-19:ember,tobzoska,denevér:K
B:Escherichia coli:bélfertőzés:ember,melegvérű állatok:proteobaktérium
B:ismeretlen baci:::vadiúj törzs
```

   Kezelje az esetleges **IllegalArgumentException** kivételeket a kivétel üzenetének (és csak annak!) kiírásával, a beolvasás megszakítása nélkül! (2 pont)

7. Hozzon létre a főprogramban egy **Kutatólabor** objektumot! A kutatólabor megnevezése legyen a második parancssori argumentum, vagy ennek hiányában „*DE Kutatólabor*”! Töltse fel a labor hűtőjét az előző feladatban beolvasott kórokozókkal, majd írja ki a képernyőre soronként a természetes rendezettségük sorrendjében azokat a kutatólaborban tárolt koronavírusokat, amelyek veszélyt jelentenek egy billentyűzetről beolvasott gazdatestre nézve! (A gazdatestben szóköz is szerepelhet!)

8. Egészítse ki a **Teszt** osztályt egy metódussal, amely paraméterként megkap egy **KórokozóTár** objektumokat tartalmazó tömböt és egy betegségnevet, és minden olyan kórokozótárhoz, melyben nincs olyan kórokozó, amely a megkapott betegséget okozza, hozzáad egy új baktériumot, amelynek neve „*új baci*”, törzse „*új törzs*”, az okozott betegsége a megkapott betegség, a veszélyeztetett gazdatesteket pedig nem ismerjük! Hívja meg a metódust a főprogramban a 7. feladatban létrehozott kutatólabort tartalmazó egyelemű tömbbel és egy billentyűzetről beolvasott (akár szóközt is tartalmazó) betegséggel, majd ellenőrzésképpen írja ki a képernyőre a tömb elemeit! (Segítség: lásd a **Collection.contains()** metódust.)